Journal of Statistical Physics, Vol. 52, Nos. 3/4, 1988

Fourier Acceleration of Iterative Processes in Disordered Systems

Ghassan George Batrouni[1] **and Alex Hansen**[2,3]

Received October 23, 1987; revision received April 22, 1988

Technical details are given on how to use Fourier acceleration with iterative processes such as relaxation and conjugate gradient methods. These methods are often used to solve large linear systems of equations, but become hopelessly slow very rapidly as the size of the set of equations to be solved increases. Fourier acceleration is a method designed to alleviate these problems and result in a very fast algorithm. The method is explained for the Jacobi relaxation and conjugate gradient methods and is applied to two models: the random resistor network and the random central-force network. In the first model, acceleration works very well; in the second, little is gained. We discuss reasons for this. We also include a discussion of stopping criteria.

KEY WORDS: Fourier acceleration; critical slowing down; relaxation; conjugate gradient algorithm; disordered systems.

1. INTRODUCTION

The necessity to solve large linear systems of equations occurs frequently in physics. This implicitly involves the inversion of large, sparse matrix systems. Direct methods, such as Gauss elimination and its sophisticated variants for sparse matrices,[1] can in some instances be useful, but in general they require a computational labor (CPU time) more than proportional to the number of equations to solve, N. An alternative approach is to use iterative (relaxation) methods.[2-5] These methods typically require a labor of order N per iteration, but the number of iterations needed to

[1] Department of Physics, Boston University, Boston, Massachusetts 02215.
[2] Groupe de Physique des Solides de l'École Normale Supérieure, F-75231 Paris, France.
[3] Present address: Institut für Theoretische Physik, Universität zu Köln, D-5000 Cologne 41, Federal Republic of Germany.

0022-4715/88/0800-0747$06.00/0

attain a fixed accuracy grows as a fractional power of N, a problem usually referred to as *critical slowing down*. In this paper we describe an algorithm for speeding up two very different iterative methods, the Jacobi relaxation[2] and conjugate gradient methods,[2,6] by reducing the critical slowing down. The acceleration algorithm was earlier described in connection with the Jacobi method by Batrouni *et al.*[7] This paper is meant to provide technical details on how to implement the Fourier acceleration idea[8] in practice, and to elaborate some tricks we have learned while applying it.

We discuss these relaxation methods and the Fourier acceleration algorithm in the context of solving transport equations for disordered media, which forms the basis of a large class of problems of considerable current interest.[9] Let us point out here that in our experience the conjugate gradient method, with or without Fourier acceleration, is the most efficient iteration method to handle these problems.

We first discuss the two specific models on which we will demonstrate the Fourier acceleration ideas, the random resistor network[4] and the elastic central-force network, both in the vicinity of their respective percolation critical points where the critical slowing down becomes especially severe due to the underlying fractal geometry of the networks. Next we discuss how the critical slowing down is related to anomalous diffusion in fractal geometries, by using the Jacobi relaxation method. We then introduce Fourier acceleration in connection with the Jacobi method, and finally use it with the conjugate gradient method.

2. THE MODELS

In order to deal with concrete examples, we will work with two models used in the study of disordered media: The random resistor network[11] and the random central-force network.[12,13] Both of these models have attracted much recent attention because of their strange scaling properties in the vicinity of the percolation critical point. Aside from their interesting physical properties, they also demonstrate clearly the strengths and the weaknesses of the Fourier acceleration method.

We describe the random resistor model first. Imagine a two-dimensional square lattice of size L (times the lattice constant) in the x direction and size $L+2$ in the y direction (Fig. 1). The lattice is periodic in the x direction. There are two types of bonds connecting neighboring nodes: They are either resistors with a unit resistance or infinite resistance. The

[4] See ref. 10 for a detailed demonstration of the use of *direct* matrix methods to find the current distribution in the random resistor networks.

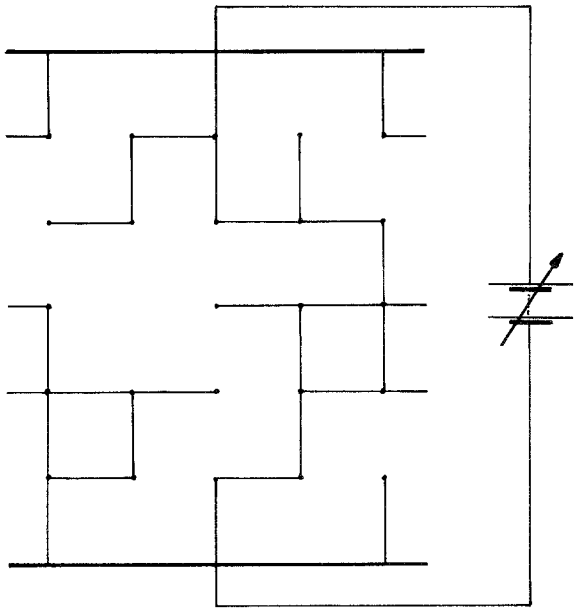

Fig. 1. A random resistor network. Here L is 5, and the lattice is periodic in the horizontal direction. The solid bonds represent unit resistors present with probability p, and the broken bonds represent an infinite resistance. A fixed voltage difference is kept between the top and bottom rows, and the problem is to calculate the currents flowing in the network.

probability of having a unit resistance at a given bond is p and the probability of having an infinite resistance at that bond is $1-p$. The horizontal bonds (i.e., in the bonds in the x direction) along the bottom and top rows have zero resistance. Now, the problem is to create a voltage difference between the bottom and top row and calculate the resulting currents flowing through each bond in the network. Let V_i denote the voltage at the ith node. Then Kirchhoff's equations are

$$\sum_j \sigma_{ij}(V_i - V_j) = 0 \tag{1}$$

where V_j are the voltages at the nearest neighbor nodes to node i, and σ_{ij} is 1 if there is a unit resistor between nodes i and j, and 0 otherwise. The index i runs over nodes that do *not* belong to the top or bottom rows, that is, the internal nodes. The index j, however, runs over all nodes. Physically, this set of equations is only a statement of current conservation at each node, and they are to be solved with the constraint that the voltages at the nodes belonging to the upper and lower rows are kept fixed. In order to write this set of equations as a matrix equation, move all voltages referring

to nodes belonging to the upper and lower rows to the right-hand side of Eq. (1) (see, e.g., ref. 14, Chapter 6). This equation may then be written as

$$\sum_j D_{ij} V_j = B_i \tag{2}$$

where the indices i and j only run over the internal nodes. The vector B_i is different from zero only if the node i is connected to a node belonging to the lower or upper rows through a unit resistor, and in that case it is equal to the voltage on the upper or lower rows. B_i is a vector with dimension L^d, where d is the dimension of the lattice of the underlying models; here $d=2$. The D_{ij} is a matrix with $L^d \times L^d$ elements corresponding to what is left of the sum on the left-hand side of Eq. (1). The solution we are seeking is of course

$$V_j = \sum_i (D^{-1})_{ij} B_i \tag{3}$$

Before using the conjugate gradient or any other method to find the solution Eq. (3), the matrix D_{ij} may be simplified by using some rather trivial facts, namely that current will not flow in dead arms or disconnected clusters of resistors. A subroutine that goes through the network after it has been generated and removes the disconnected clusters save, computer time. Algorithms also exist which remove the dead arms connected to the current-carrying backbone.[5] However, these algorithms are rather complicated to implement and CPU time-consuming enough that it is not obvious that they are useful in this context.

Let us now describe the second model we use, the random central-force network. In this case we use a square lattice with one diagonal direction present (Fig. 2). This is a distorted version of the triangular lattice used in refs. 12 and 13, but the critical properties of this lattice are the same as in the usual triangular lattice. The reason for this choice of lattice is that it is easier to see the connection between this model and the random resistor network in this case. The two types of resistors in the random resistor network are replaced in this model by two types of central-force springs, i.e., springs which are free to rotate about the nodes. This corresponds to the forces on each spring in the network always pointing along the axis of the spring; there are no angular forces. A given bond has

[5] A general and efficient algorithm to identify the current-carrying subset of bonds is the "burning algorithm" of Herrmann *et al.*[15] A more efficient but less general method (being restricted to two dimensions) is given in Roux and Hansen.[16] Methods which are well known in the computer science literature are given in Aho *et al.*[17] We have not tested how their efficiency compares to the methods of Herrmann *et al.*, and Roux and Hansen.

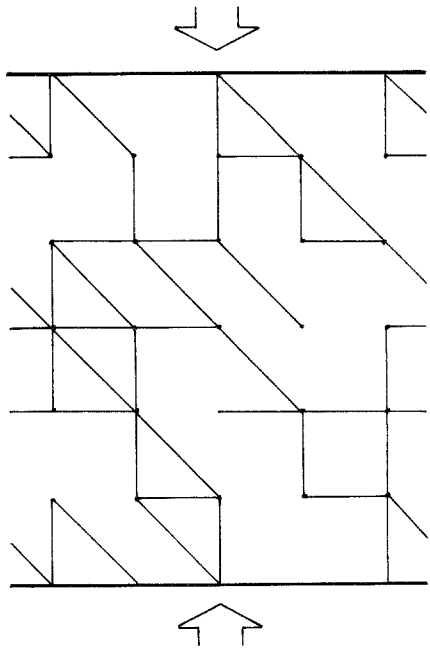

Fig. 2. A random central-force network. Here L is 5, and the lattice is periodic in the horizontal direction. The solid bonds represent springs with unit spring constants present with probability p, and the broken bonds represent infinitely soft springs present with probability $1-p$. The forces on each bond are always in the direction of the bond. A fixed displacement between the top and bottom rows is kept, and the problem is to calculate the forces carried by each bond in the network. Notice how difficult it is to determine whether this configuration is rigid or not in comparison to determining whether the configuration in Fig. 1 conducts or not.

either a finite spring constant (i.e., unit spring constant in the x or y directions, or a spring constant equal to $1/\sqrt{2}$ along the diagonal) with probability p, or zero spring constant with probability $1-p$. The problem is now to impose a given displacement between the top and bottom rows by pulling, pushing, or shearing the top row relative to the bottom row and calculate the resulting forces in the network. We define $\mathbf{R}_i$ as the displacement of the ith node and σ_{ij} as the rigidity matrix of the bond between nodes i and j. The set of (linearized) equations describing the balancing of the forces in the network, and which correspond to the Kirchhoff equations (1) in the random resistor network, are

$$\sum_j \sigma_{ij}(\mathbf{R}_i - \mathbf{R}_j) = 0 \tag{4}$$

Let us now write these equations in component form and with a small generalization: The diagonal forms an angle $\alpha = 45°$ with the x axis, but for

later use we will write α explicitly in these equations. The neighboring node to node i in the positive x direction is denoted $i1$, and the other neighboring nodes in anticlockwise manner are successively denoted $i2$, $i4$, $i8$, $i16$, and $i32$. Let us remark here that we are keeping the notation close to the one that is natural on the computer. The positive x direction is denoted direction 1. Then follows in anticlockwise manner direction 2, the positive diagonal direction, direction 4, the positive y direction, direction 8, the negative x direction, direction 16, the negative diagonal direction, and direction 32, the negative y direction. In this way, the bond structure of the lattice may be packed in a very compact manner as the sum of the directions where there are bonds from each node. Let $\sigma(i)$ denote the array containing these sums. Then, for example, the logical operation $(\sigma(i).\text{and}.4)$ will be true if there is a bond between node i and its neighbor in the positive y direction, and false if not. In the notation of Eq. (5), we have $\sigma_{i,i4} = (\sigma(i).\text{and}.4)/4$. Thus, we have

$$\begin{aligned}
&\sigma_{i,i1}(x_i - x_{i1}) + \sigma_{i,i2}[\cos^2\alpha\,(x_i - x_{i2}) + \sin\alpha\cos\alpha\,(y_i - y_{i2})] \\
&\quad + \sigma_{i,i8}(x_i - x_{i8}) + \sigma_{i,i16}[\cos^2\alpha\,(x_i - x_{i16}) \\
&\quad + \sin\alpha\cos\alpha\,(y_i - y_{i16})] = 0 \qquad (5a) \\
&\sigma_{i,i2}[\sin\alpha\cos\alpha\,(x_i - x_{i2}) + \sin^2\alpha\,(y_i - y_{i2})] + \sigma_{i,i4}(y_i - y_{i4}) \\
&\quad + \sigma_{i,i16}[\sin\alpha\cos\alpha\,(x_i - x_{i16}) + \sin^2\alpha\,(y_i - y_{i16})] \\
&\quad + \sigma_{i,i32}(y_i - y_{i32}) = 0 \qquad (5b)
\end{aligned}$$

As in the random resistor network problem, we move everything that has to do with the fixed top and bottom row to the right-hand side of Eq. (4), or in component form, of Eqs. (5a) and (5b). This leads to a matrix equation corresponding to Eq. (2) in the random resistor problem,

$$\sum_j \mathbf{D}_{i,j}\mathbf{R}_j = \mathbf{B}_i \qquad (6)$$

The only difference between this equation and Eq. (2) is that the element $\mathbf{D}_{i,j}$ is a 2×2 matrix here, and the vector components $\mathbf{R}_i$ and $\mathbf{B}_i$ are themselves two-component vectors. The total number of elements in the matrix $\mathbf{D}_{i,j}$ to be inverted is thus $(2L)^d \times (2L)^d$. As in the previous model, it is efficient to have a subroutine remove disconnected clusters after the lattice has been generated. However, unlike the random resistor network, there is no geometric method to determine which bonds belonging to the cluster spanning the lattice from bottom to top row will carry a nonzero force (thus defining the force-carrying backbone).[12]

It is possible to interpret the central-force elastic network in terms of two *interacting* random resistor networks. Suppose that in the elastic network we impose a displacement of the top row which is an equal mixture of shear and compression, i.e., equal in the x and y directions. Now, in the limit $\alpha \to 90°$, Eq. (5a) reduces to

$$\sigma_{i,i1}(x_i - x_{i1}) + \sigma_{i,i2}(x_i - x_{i2}) + \sigma_{i,i8}(x_i - x_{i8}) + \sigma_{i,i16}(x_i - x_{i16}) = 0 \qquad (7a)$$

which is identical to the Kirchhoff equations for a square random resistor network, where the two directions are given by the old x direction (i.e., the 1 and 8 directions) and by the direction of the diagonal (i.e., the 2 and 16 directions), and where the x coordinate is to be interpreted as the potential. Equation (5b) becomes

$$\sigma_{i,i4}(y_i - y_{i4}) + \sigma_{i,i32}(y_i - y_{i32}) = 0$$

which can be interpreted as the equation for L one-dimensional random resistor networks in which the y coordinate is to be interpreted as a potential. In the same way, in the limit $\alpha \to 0°$, Eq. (5b) reduces to

$$\sigma_{i,i2}(y_i - y_{i2}) + \sigma_{i,i4}(y_i - y_{i4}) + \sigma_{i,i16}(y_i - y_{i16}) + \sigma_{i,i32}(y_i - y_{i32}) = 0 \qquad (7b)$$

which may be interpreted as the Kirchhoff equations for a square random resistor network, where the two directions are given by the old y direction (i.e., the 4 and 32 directions) and by the direction of the diagonal (i.e., the 2 and 16 directions), and where the y coordinate acts as the potential. Equation (5a) describes in this limit L one-dimensional random resistor networks. At intermediate values for the angle α, the set of equations (5a) and (5b) may be interpreted as two coupled anisotropic random resistor networks where the two potentials are given by the x and y coordinates. The term "anisotropic" refers to the factors $\cos^2 \alpha$ and $\sin^2 \alpha$ that can be thought of as rescaling the resistors in the 2 and 16 directions. The terms proportional to $\sin \alpha \cos \alpha$ play the role of interaction terms between the two random resistor networks.

Both of these models exhibit critical behavior when p, the probability that a resistor or a spring is present, is in the vicinity of p_c, the percolation critical point (see ref. 19 for a general introduction to this field). The random resistor network will conduct in the thermodynamic limit ($L \to \infty$) if $p > p_c$ ($= 1/2$ for the square lattice used here), and will not conduct if $p \leqslant p_c$. The elastic central-force network will be rigid (i.e., have a nonzero elastic modulus) in the thermodynamic limit only if $p > p_c$ ($= 0.642 \pm 0.002$ for the square lattice with one diagonal used here[13]). At p_c the current-carrying backbone in the random resistor network and the force-carrying

backbone in the elastic central-force network will be fractal. This means that the number of bonds N_B in the backbone scales with some nontrivial exponent, the fractal dimension d_B, with respect to the size of the lattice L: $N_B \propto L^{d_B}$. For the random resistor network one finds[20] $d_B = 1.62 \pm 0.02$, and for the elastic central-force network one finds[13] $d_B = 1.64 \pm 0.03$. The backbone, as already mentioned, is a subset of bonds belonging to the "incipient infinite cluster." Note that the incipient infinite cluster also contains dead arms (i.e., bonds not carrying any current or force). This incipient infinite cluster also turns out to be fractal, i.e., the number of bonds belonging to it scales as $N \propto L^{d_f}$, where for the random resistor network $d_f = 91/48 \approx 1.90$.[19] The fractal dimension of the incipient infinite cluster in the central-force network is not known, but presumably it is not very different from that of the random resistor network. The question of whether the central-force network and the random resistor network are in the same univerality class is still controversial, and it is possible that the dimension of the incipient infinite cluster in the two models is different. It turns out that diffusive processes are anomalously slow on fractal structures such as these[21]: the rms distance from the starting point for a free random walker on such structures gows as t^{d_w}, where the random-walk dimension $d_w \geqslant 2$. For the random resistor network one has[22] $d_w = 2.87 \pm 0.05$ for walks on the incipient infinite cluster. The corresponding d_w for the elastic central-force network is 5.0 ± 0.1.[23] The random-walk dimension of the backbone in the random-resistor network is smaller than that of the incipient infinite cluster itself, but in such a way that the difference $d_w - d_f$ is invariant whether the two exponents d_w and d_f are calculated from the backbone or the incipient infinite cluster.[24] Thus, on the backbone of the random resistor network, $d_w = 2.59 \pm 0.05$.

We will assume in the following discussion that we are close to or at p_c whenever we refer to the two models.

3. THE JACOBI RELAXATION METHOD AND ITS ACCELERATION

The general idea behind the iterative methods used to solve equations such as Eqs. (2) and (6) is to guess an initial distribution V_i^0 (or $\mathbf{R}_i^0$) and then approach the solution $V_i^\infty = V_i$ by repeated use of some set of operations $V_i^{k+1} = I(V_i^k)$. Each iteration will typically require of the order of L^d operations. (If the backbone has been isolated by some method, it is possible to do with only L^{d_B} operations per iteration.) The number of iterations needed to find the solution to within some desired accuracy will also scale with some power of L; this is known as *critical slowing down.* This power is related to the random walk dimension d_w. This connection is

most easily understood in connection with the Jacobi relaxation method,[2] which we now discuss in the context of the random resistor network. This method can be understood directly in terms of physical processes: Assume that each node in the random resistor network is connected to a common ground through capacitors C. The Kirchhoff equations (1) and (2) then become

$$C\, dV_i/dt = \sum_j \sigma_{ij}(V_i - V_j) = \sum_j D_{ij} V_j - B_i \tag{8}$$

with the same boundary conditions as in Eq. (1). Suppose now that we charge the capacitors and then let the system evolve in time according to Eq. (8). The capacitors will discharge themselves, and as $t \to \infty$, a steady state $dV_i/dt = 0$ is reached which is the solution to the original problem, Eq. (1). Numerically, this can be done by a simple Euler time-stepping procedure,

$$\begin{aligned} V_i^{m+1} &= V_i^m + \varepsilon \sum_j \sigma_{ij}(V_i^m - V_j^m) \\ &= V_i^m + \varepsilon \left(\sum_j D_{ij} V_j^m - B_i \right) \end{aligned} \tag{9}$$

where ε is the size of the time step (divided by C). Equation (9) constitutes the Jacobi method. Notice that Eq. (8) is formally identical to a diffusion equation on a geometrically equivalent network to the random resistor network, and where the matrix D_{ij} acts as the diffusion kernel. Thus, in light of the above discussion of the scaling of time scales associated with diffusive processes on fractals, one expects the time scale governing Eq. (8) to scale with the random walk exponent d_w. This means that in order to bring nodes a distance ξ apart into mutual equilibrium, a time $t \propto \xi^{d_w}$ is needed. Thus, in order to reach the steady-state distribution of voltages given by Eq. (1) to within a given accuracy, one needs $N \propto L^{d_w}$ iterations. This is an example of (severe) critical slowing down. Physically, what is happening is that the large-scale structures of the network evolve much slowly than the small-scale structures.

One way to speed up the relaxation process of Eq. (8) would be to choose a larger time step ε. However, there is an upper limit to the size of ε for the iteration process to remain numerically stable. This limit is proportonal to the convergence time for the *fastest* developing components of the voltage distribution. Therefore, the largest possible ε is very small compared to the convergence time for the slowest developing components.

The idea behind the Fourier acceleration method when utilized in connection with the Jacobi scheme is to choose a *scale-dependent* time step ε_{ij}

in such a way that each length scale in the problem evolves with the same speed in Eq. (8), in the numerical analysis literature this is known as *preconditioning*.[2] A perfect choice for such a time step would of course be $(D^{-1})_{ij}$, all scales would then evolve equally fast by reaching the solution, Eq. (3), in one time step. Unfortunately, this is not practical as it requires previous knowledge of the solution. However, O'Shaughnessy and Procaccia[25] have suggested an ensemble-averaged form for $(D^{-1})_{ij}$,

$$\langle (D^{-1})_{ij} \rangle \propto |i-j|^{d_w - d_f} \tag{10}$$

where $|i-j|$ is the Euclidean distance between the nodes i and j. Notice that the exponent here, $d_w - d_f$, is independent of whether these quantities refer to the backbone or the incipient infinite cluster of the random resistor network, as pointed out at the end of Section 2. For a given realization of a network, $\langle (D^{-1})_{ij} \rangle$ is nonzero for all nodes whether they belong to the incipient infinite cluster or not. It turns out that Eq. (10) is an excellent choice for the time step[3] ε_{ij}, i.e., $\varepsilon_{ij} = |i-j|^{d_w - d_f}$. Thus, Eq. (9) becomes

$$V_i^{m+1} = V_i^m + \sum_n{}' \varepsilon_{in} \left(\sum_j D_{nj} V_j^m - B_n \right) \tag{11}$$

where the sum $\sum'$ runs only over nodes belonging to the incipient infinite cluster. The restriction of the sum over n in this equation is important. The reason for this is that Eq. (11) without the restriction in the sum over n finds the solution to the following equation:

$$\sum_n \varepsilon_{in} \left(\sum_j D_{nj} V_j^m - B_n \right) = 0 \tag{12}$$

This equation has the same solution V_i as Eq. (2) only if they span an L^d-dimensional vector space. But this cannot be so, since *only a subset* L^{d_f} of the nodes belongs to the incipient infinite cluster. They only span an L^{d_f}-dimensional vector space. Thus, only by restricting the sum in Eqs. (11) and (12) to the nodes that belong to the incipient infinite cluster will one have that $\sum_n' \varepsilon_{in} (\sum_j D_{nj} V_j^m - B_n) = 0$ has the same solution as $(\sum_j D_{ij} V_j^m - B_i) = 0$.

Each iteration of Eq. (11) requires a minimum of $L^{d_f} \times L^{d_f}$ operations as compared to L^{d_f} for Eq. (9). If the choice we have made for the time step ε_{in} is able to remove the critical slowing down completely so that the number of iterations necessary to reach a given accuracy *does not scale with* L, one will have to compare $L^{2d_f} (\approx L^{3.8}$, or $L^{3.2}$ if the backbone has been extracted) operations [Eq. (11)] to $L^{d_f + d_w}$ ($\approx L^{4.8}$, or $\approx L^{4.2}$ for the backbone) operations [Eq. (9)] to judge whether this method is useful or

not. We have apparently gained at best a little more than one power of L if the critical slowing down is completely removed (in practice it is never completely removed), but it is possible to do much better by invoking Fourier transforms. The reason for this is the convolution theorem. Let $A_{i,j} = A_{i-j}$ be an $(L^d \times L^d)$-dimensional matrix, and let B_i be a vector with L^d components, both defined on the incipient infinite cluster. We define the (d-dimensional) Fourier transforms of these quantities as follows:

$$a_k = F(A)_k = \sum_j e^{i(k_x j_x + k_y j_y)} A_j \tag{13a}$$

and

$$b_k = F(B)_k = \sum_j e^{i(k_x j_x + k_y j_y)} B_j \tag{13b}$$

where the sum over j runs over both the x and y components of it j_x and j_y (assuming that $d=2$), and k is the wave vector. Convolution is defined as $(A * B)_i = \sum_j A_{i-j} B_j$, and the convolution theorem states that

$$F(A * B)_i = F(A)_i F(B)_i \tag{14}$$

We see that one sum over the nodes of the lattice has been saved by doing the Fourier transforms before the product of the matrix and the vector. In other words, direct computation of the convolution $(A * B)_i$ will need $L^{d_f} \times L^{d_f}$ operations, while computing it as $F^{-1}(F(A)\,F(B))$ takes work of order $L^d \log L$ when using fast-Fourier transform algorithms.[26] Thus, the introduction of the fast-Fourier transforms makes the algorithm much more efficient. Now, in Eq. (11) we see that the costly part of each iteration is caused by a sum such as the one occurring on the left-hand side of Eq. (14); identify $\varepsilon_{ij} = |i-j|^{d_w - d_f}$ with $A_{i,j}$ and $\sum_j D_{ij} V_j^k - B_i)$ with B_i. It would therefore be very efficient if this sum could be done in Fourier space. There is only one problem: The sum is restricted to the conducting incipient infinite cluster, so how can one do Fourier transforms on something as irregular as an incipient infinite cluster? This obstacle is circumvented in the following manner: Lift the restriction on the sum over n in Eq. (11) to get

$$V_i^{m+1} = V_i^m + \sum_n \varepsilon_{in} \left(\sum_j D_{nj} V_j^m - B_n \right) \tag{15}$$

Furthermore, let i run over all nodes in the network whether they belong to the incipient infinite cluster or not. This equation will relax to a set of V_i which are different than the voltages given by the Kirchhoff equations (2), as explained after Eq. (11). However, suppose now that we are just starting

the iteration process, $m=0$, and we have chosen all voltages not on the incipient infinite cluster to be zero as initial condition. Then Eq. (15) will update the nodes *belonging* to the incipient infinite cluster correctly, while those not on it will be updated according to

$$V_i^1 = \sum_n{}' \varepsilon_{in} \left(\sum_j D_{nj} V_j^0 - B_n \right) \tag{16}$$

The restriction on n does not make any difference during this iteration, since the nodes outside the incipient infinite cluster have zero voltage. It is during the second iteration that nodes *on* the incipient infinite cluster are affected by the lifting of the restriction on the sum. During this iteration, the nonzero voltages on the nodes outside the incipient infinite cluster that developed during the first iteration will affect the development of the voltages of the nodes on the incipient infinite cluster. The remedy for this problem is simple. Use Eq. (15) to iterate the system, but after each iteration sweep through the lattice *reinitialize* all nodes outside the incipient infinite cluster back to zero. We can now use the convolution theorem on Eq. (15). The Fourier transform of ε_{ij} is

$$\varepsilon_k = k^{-d+d_f-d_w} \tag{17}$$

and Eq. (15) becomes

$$V_i^{m+1} = V_i^m + F^{-1}\left(\varepsilon_k F\left(\sum_j D_{kj} V_j^m - B_k \right) \right) \tag{18}$$

where F^{-1} denotes an inverse Fourier transform. In words, what we do is the following: We Fourier transform $\sum_j D_{kj} V_j^m - B_k$ and multiply it by Eq. (17). Then, this product is inversely Fourier transformed again, and added *only* to the nodes belonging to the incipient infinite cluster. In this way, each iteration will require $2L^d \log L$ operations and *if* the critical slowing down has been removed completely, this will also be the scaling of the entire relaxation process with respect to the lattice size L. Thus, ideally one should compare $L^2 \log L$ operations to $L^{4.8}$, or $L^{4.2}$ if the backbone has been identified, for the unaccelerated Jacobi relaxation method on the two-dimensional random resistor network. How well this works out in practice is given in ref. 7, where the backbone was identified before the Jacobi relaxation procedure was attempted. There it was found that the critical slowing down was reduced from 2.59 to 1.2 and the CPU time scaled as $L^{3.2} \log L$ as compared to $L^{4.2}$ for the unaccelerated algorithm. (The boundary conditions in this work were slightly different from those depicted in Fig. 1: Instead of bus bars at the lower and upper edges of the network,

only one node at the upper edge and one node at the lower edge were connected to the exterior current source.) Note that since ε_k is needed every iteration, we compute it once at the beginning, and store it in an array. This saves a lot of time.

Let us now leave the random resistor network in order to discuss the elastic central-force problem. The corresponding equation for this problem to Eq. (11) is

$$\mathbf{R}_i^{m+1} = \mathbf{R}_i^m + \sum_n{}' \varepsilon_{in} \left(\sum_j \mathbf{D}_{nj} \mathbf{R}_j^m - \mathbf{B}_n \right) \tag{19}$$

and the connection between this equation and Eq. (6) should be evident. But the problem is now how to choose the space-dependent time step ε_{ij}. There exists no equivalent to the O'Shaughnessy and Procaccia average Eq. (10). We will therefore make a guess here. First, we notice that the trick we used to lift the restriction on the sum in Eq. (11) can be used here also, we reinitialize all nodes not belonging to the connected cluster of springs to zero between each iteration. (We are avoiding the word "backbone" here since there is no procedure to determine which springs will carry a force within the cluster that will contain the rigid backbone.) We can then Fourier transform *both* components of the displacements $\mathbf{R}_i^k$ to get

$$\mathbf{R}_i^{m+1} = \mathbf{R}_i^m + \mathbf{F}^{-1} \left(\varepsilon_k \mathbf{F} \left(\sum_j \mathbf{D}_{kj} \mathbf{R}_j^m - \mathbf{B}_k \right) \right) \tag{20}$$

The most general form of the Fourier transform of the average of the kernel $\mathbf{D}_{ij}$, $\langle \mathbf{D}_{k,0} \rangle$, is

$$\langle \mathbf{D} \rangle = f(k)[\mathbf{1} + g(k)\mathbf{k} \otimes \mathbf{k}/|\mathbf{k}|^2] \tag{21}$$

where the symbol $\otimes$ denotes dyadic multiplication. In a full lattice, i.e., $p = 1$, $f(k)$ is k^2, and $g(k)$ is always less than 1, as a result of the Poisson ratio always being less than 1. Now, the Fourier transform of the space-dependent time step ε_{ij} may be chosen as the inverse of Eq. (21) as was done for the random resistor network:

$$\varepsilon_k = f(k)^{-1} \{ \mathbf{1} - [g(k)/(1 + g(k)]\, \mathbf{k} \otimes \mathbf{k}/|\mathbf{k}|^2 \} \tag{22}$$

We notice that whatever the function $g(k)$ is, the term proportional to $\mathbf{k} \otimes \mathbf{k}/|\mathbf{k}|^2$ in Eq. (22) that makes ε_k not diagonal is never greater than 1. Thus, the off-diagonal term in Eq. (22) is always smaller than or, at worst equal to, the diagonal term proportional to $\mathbf{1}$. Since this off-diagonal term will result in $L^d \times L^d$ operations in Eq. (20), as compared to L^d for the

diagonal term, we will need to ignore it. Furthermore, we assume that the function $f(k)$ is a power law in k, as was done in the random resistor network:

$$\varepsilon_k = \mathbf{1} k^{-\beta} \tag{23}$$

where β is some exponent to be determined by trial and error, i.e., by finding the β that works best. The Fourier-accelerated Jacobi relaxation algorithm for the elastic central-force network is then given by

$$\mathbf{R}_i^{m+1} = \mathbf{R}_i^m + \mathbf{F}^{-1}\left(k^{-\beta}\mathbf{F}\left(\sum_j \mathbf{D}_{kj}\mathbf{R}_j^m - \mathbf{B}_k\right)\right) \tag{24}$$

where the nodes not belonging to the connected cluster of springs are reinitialized to zero. The exponent β that seems to work best is 2.2. Let us here emphasize that one may choose anything for the function ε_k; the correct solution will be found. The problem is to choose one that reduces the computer time needed.

4. THE CONJUGATE GRADIENT METHOD AND ITS FOURIER ACCELERATION

We are now at the point where it is natural to describe the conjugate gradient method. The most efficient version of this method for the two problems we are discussing here is the one invented by Hestenes and Stiefel.[27] Excellent discussions of this method are found in refs. 2 and 6; thus, we will only briefly sketch it here. As usual, we start by discussing the random resistor network. The idea behind the conjugate gradient method is to minimize a function

$$H(V_i^m) = \sum_k \sum_j (V_k^m D_{kj} V_j^m)/2 - \sum_k V_k^m B_k \tag{25}$$

The minimum found is the solution to the Kirchhoff equations, Eq. (3). In order to make it easier to grasp the workings of the conjugate gradient method, it is customary to first outline the *steepest descent* method.[2,6,14] Here one constructs a series of voltage vectors V_i^m by a succession of one-dimensional minimizations along the direction of the local gradient,

$$H(V_i^{m+1}) = \min_u H(V_i^m + u r_i^m) \tag{26}$$

where $-r_i^m$ is the local gradient,

$$r_i^m = -\partial H(V_i^m)/\partial V_i^m \tag{27}$$

The convergence of this method is forbiddingly slow if the contours of constant H are highly elliptical rather than circular, which will be the case if the ratio of the largest and smallest eigenvalues of D is large, a situation which is typical. The reason for this is that the gradient at the $(m+1)$th iteration will be orthogonal to the gradient at the mth iteration as a result of the one-dimensional minimalizations along each new gradient direction. Thus, one is trying to approach the minimum of a highly elliptical valley, i.e., the minimum of the function H, by a path consisting of steps at right angles, and this results in the need for many steps, or iterations. However, notice that if the function H describes a *circular* valley, this method will converge in a maximum of L^d steps.

The problem of stepping at right angles in the steepest descent method is caused by using the gradient directions $-r_i^m$ as directions in which one searches for the new minimum. This suggests that one should use a *different* set of search directions p_i^m. The ideal choice of these new search directions would be such that at the $(m+1)$th step one finds the minimum of the function H over the subspace spanned by the search vectors already constructed, $p_i^0, p_i^1, p_i^2, \ldots, p_i^m$:

$$H(V_i^{m+1}) = \min_{\text{space spanned by } p_i^0, \ldots, p_i^m} H(V) \tag{28}$$

This is a minimalization over an $(m+1)$-dimensional space, and thus too complicated for a practical numerical method. However, if one could construct a set of search vectors $p_i^0, p_i^1, p_i^2, \ldots, p_i^m$ with the property that the one-dimensional minimalization

$$H(V_i^{m+1}) = \min_u H(V_i^m + up_i^m) \tag{29}$$

also solves the $(m+1)$-dimensional minimalization problem, Eq. (28), one would have an effective procedure with the same convergence properties as the steepest descent method in a circular valley. Such a procedure to construct effectively such a set of search vectors exists, and is called the *conjugate gradient* method. The search vectors constructed in this method are mutually conjugate with respect to D_{ij}, i.e., they obey the property

$$\sum_i \sum_j p_i^k D_{i,j} p_j^1 = 0 \qquad \text{for} \quad k \neq 1 \tag{30}$$

This property has given the conjugate gradient method its name. These vectors turn out to be linear combinations of the gradient vectors r_i^m constructed for each V_i^m. These gradient vectors are as, in the steepest descent method, mutually orthogonal,

$$\sum_i r_i^k r_i^1 = 0 \qquad \text{for} \quad k \neq 1 \tag{31}$$

Another way of understanding how the conjugate gradient method works is to notice that the vectors p_i play the role of the vectors r_i in the steepest descent method, but in a different metric $D_{i,j}$ [see Eq. (30)]. In this metric, the quadratic form (25) *always* describes a *circular* valley. Thus, the conjugate gradient method is basically the steepest descent method in a different metric chosen so that the search for the minimum is done in a *circular* valley. Thus, the maximum number of iterations necessary to find the minimum by the conjugate gradient method (ignoring roundoff errors) is, as for the steepest descent method in a circular valley, equal to the number of dimensions of the space of voltages, and this is L^d.

One should note that the property of *exact* convergence in a maximum of L^d steps is not true in practice, due to roundoff errors. Another point is that even if this property was true, L^d iterations could be too many to tolerate. So the value of the conjugate gradient method is *not* the finite-termination property, but rather that it is a good *iterative* procedure, i.e., one stops well before L^d iterations. It can be shown that its critical slowing down is never worse than the *square root* of the corresponding Jacobi procedure.[2]

Let us now present the algorithm itself, i.e., how to implement the minimalization of Eq. (29) in practice. (For a detailed derivation, see section 10.2 of ref. 2.) Start by choosing some initial set of voltages V_i^0. Calculate

$$p_i^0 = r_i^0 = B_i - \sum_j D_{i,j} V_j^0 \tag{32}$$

If it turns out that $p_i^0 = 0$ (for all i of course), the initial guess was in fact the correct solution. Otherwise, for $m = 1, 2, \ldots$, compute

$$a^m = \left(\sum_i r_i^m r_i^m \right) \Big/ \left(\sum_j \sum_k p_j^m D_{j,k} p_k^m \right) \tag{33}$$

$$V_i^{m+1} = V_i^m + a^m p_i^m \tag{34}$$

$$r_i^{m+1} = r_i^m - a^m \sum_j D_{i,j} p_j^m \tag{35}$$

$$b^m = \left(\sum_i r_i^{m+1} r_i^{m+1} \right) \Big/ \left(\sum_j r_j^m r_j^m \right) \tag{36}$$

and

$$p_i^{m+1} = r_i^{m+1} + b^m p_i^m \tag{37}$$

The natural error criterion is the norm of r_i^m. Let ε be the desired accuracy. Then stop the iteration of Eqs. (33)–(37) when

$$\left(\sum_j r_j^m r_j^m \right) \leqslant \varepsilon^2 \tag{38}$$

It is necessary to store four vectors, each with N components, namely V_i^m, r_i^m, p_i^m, and $\sum_j D_{i,j} p_j^m$ during the iteration of these equations. We notice here in passing that when implementing this scheme in practice, it is very useful to test if the orthogonality relations (30) and (32) are fulfilled. This is a quick method to check whether the conjugate gradient method functions properly. Let us mention here that we noticed a case where the conjugate gradient method seemingly functioned properly, finding the correct solution to the Kirchhoff equations. But when testing the properties of Eqs. (31) and (32), it became clear that there was a bug in the program. When this bug was identified and removed, the speed of the convergence of the method went up by a factor of 5.

It is basically no different to solve the elastic central-force problem from the random resistor problem with the conjugate gradient method. The problem now is to minimize the function

$$H(\mathbf{R}_i^m) = \left(\sum_j \mathbf{D}_{kj}\mathbf{R}_j^m - \mathbf{B}_k\right)\mathbf{D}_{kn}^{-1}\left(\sum_j \mathbf{D}_{nj}\mathbf{R}_j^m - \mathbf{B}_n\right)\Big/2 \tag{39}$$

This is done by an equivalent set of equations to Eqs. (32)–(37). In order to save the reader some time, we explicitly give the equations to be iterated here. Start by choosing some initial set of displacements $\mathbf{R}_i^0$. Calculate

$$\mathbf{p}_i^0 = \mathbf{r}_i^0 = \mathbf{B}_i - \sum_j \mathbf{D}_{i,j} \cdot \mathbf{R}_j^0 \tag{40}$$

(We are explicitly marking the "dot" products of the two-component vectors constituting the components of the vectors over the entire lattice.) If $\mathbf{p}_i^0 = 0$, the initial guess was the solution. Otherwise, for $m = 1, 2, \ldots$, compute

$$a^m = \left(\sum_i \mathbf{r}_i^m \cdot \mathbf{r}_i^m\right)\Big/\left(\sum_j \sum_k \mathbf{p}_j^m \cdot \mathbf{D}_{j,k} \cdot \mathbf{p}_k^m\right) \tag{41}$$

$$\mathbf{R}_i^{m+1} = \mathbf{R}_i^m + a^m \mathbf{p}_i^m \tag{42}$$

$$\mathbf{r}_i^{m+1} = \mathbf{r}_i^m - a^m \sum_j \mathbf{D}_{i,j} \cdot \mathbf{p}_j^m \tag{43}$$

$$b^m = \left(\sum_i \mathbf{r}_i^{m+1} \cdot \mathbf{r}_i^{m+1}\right)\Big/\left(\sum_j \mathbf{r}_j^m \cdot \mathbf{r}_j^m\right) \tag{44}$$

and

$$\mathbf{p}_i^{m+1} = \mathbf{r}_i^{m+1} + b^m \mathbf{p}_i^m \tag{45}$$

As an error criterion, we may choose in this case

$$\left(\sum_j \mathbf{r}_j^m \cdot \mathbf{r}_j^m\right) \leqslant \varepsilon^2 \tag{46}$$

We would like to raise one word of caution at this point. If the x and y axes of the elastic problem are chosen *not* to be orthogonal (such as when dealing with a triangular lattice instead of a square one with one diagonal), care must be taken to incorporate this nonorthogonality when performing "dot" products. This may seem a rather obvious point, but it is a fact that may be hidden when trying to pack these vectors as efficiently as possible into the memory of the computer. [We speak here from (unpleasant) experience.]

We now turn to the problem of Fourier accelerating this algorithm. Let us point out here that this technique is a special case of what is known in the computer-science literature as *preconditioning* the conjugate gradient method (see, e.g., Section 10.3 of ref. 2 and ref. 28). As the "philosophy" of the conjugate gradient method differs from the Jacobi method, so does the "philosophy" of Fourier acceleration of the two methods. Fourier acceleration of the conjugate gradient method will *not* reduce the number of iterations needed to reach the exact solution. Its aim is, however, to reduce the number of iterations needed to get *close enough* to the exact solution for one's purpose. The conjugate gradient method proceeds by picking a sequence of directions that eventually leads to the solution. Fourier acceleration will make the conjugate gradient method pick a different sequence, which (hopefully) will lead to the vicinity of the exact solution much faster.

As usual we start with the random resistor network. The idea is to solve an equivalent matrix problem to Eq. (2) *in Fourier space*[29]:

$$\sum_j \underline{D}_{ij} \underline{V}_j = \underline{B}_i \tag{47}$$

where

$$\underline{D}_{kl} = (\sqrt{\varepsilon_k^\gamma})\, F(D_{kl} F^{-1}(\sqrt{\varepsilon_l^\gamma})) \tag{48}$$

$$\underline{B}_k = (\sqrt{\varepsilon_k^\gamma})\, F(B_k) \tag{49}$$

and

$$V_k = F^{-1}((\sqrt{\varepsilon_k^\gamma})\, \underline{V}_k) \tag{50}$$

ε_k is the Fourier transform of the space-dependent time step defined in Eq. (17). The optimum γ is curiously not equal to 1, which is the optimum

choice when using the Jacobi relaxation method; the optimum choice here seems to be 0.5. The reason for this, or rather, why $\gamma = 1$ is *not* the best choice, lies in the critical slowing down properties of the conjugate gradient method being very different from those of the Jacobi method. These properties are very complicated and at this time not well understood. The reason for working in Fourier space is the same as for the Jacobi method: in Fourier space, ε_k is diagonal, thus greatly reducing the number of operations necessary per iteration for the matrix multiplications.

The conjugate gradient algorithm itself, Eqs. (32)–(37), is modified in the following manner: Start with Eq. (32),

$$p_i^0 = B_i - \sum_j D_{i,j} V_j^0$$

As before, if $p_i^0 = 0$, we have found the solution at the initial guess. If not, calculate

$$\underline{p}_k^0 = \underline{r}_k^0 = (\sqrt{\varepsilon_k^\gamma})\, F(p_k^0) \tag{51}$$

We can now start the iteration process. For $m = 1, 2, \ldots$, compute the vector

$$p_i^m = F^{-1}((\sqrt{\varepsilon_i^\gamma})^{-1}\, \underline{p}_i^m) \tag{52}$$

Then form the vector

$$\sum_l \underline{D}_{k,l}\, \underline{p}_l^m = (\sqrt{\varepsilon_k^\gamma})\, F\left(\sum_l D_{k,l}\, p_l^m\right) \tag{53}$$

The rest now follows Eqs. (33)–(37) closely:

$$\underline{a}^m = \left(\sum_k \underline{r}_k^m \underline{r}_k^m\right) \Big/ \left(\sum_l \sum_n \underline{p}_l^m \underline{D}_{l,n} \underline{p}_n^m\right) \tag{54}$$

$$\underline{V}_k^{m+1} = \underline{V}_k^m + \underline{a}^m \underline{p}_k^m \tag{55}$$

$$\underline{r}_k^{m+1} = \underline{r}_k^m - \underline{a}^m \sum_l \underline{D}_{k,l}\, \underline{p}_l^m \tag{56}$$

$$\underline{b}^m = \left(\sum_k \underline{r}_k^{m+1} \underline{r}_k^{m+1}\right) \Big/ \left(\sum_l \underline{r}_l^m \underline{r}_l^m\right) \tag{57}$$

and

$$\underline{p}_k^{m+1} = \underline{r}_k^{m+1} + \underline{b}^m\, \underline{p}_k^m \tag{58}$$

We take as an error criterion

$$\left[\sum_k (\underline{r}_k^m \underline{r}_k^m)/\varepsilon_k^\gamma\right] \Big/ L^d \leqslant \varepsilon^2 \tag{59}$$

which makes it identical to the one used for the *unaccelerated* conjugate gradient method, Eq. (38). When the error criterion, Eq. (59), is satisfied, $\underline{V}_k^m$ is converted into the solution V_i^m by Eq. (50).

We see that the number of operations per iteration has increased by $3L^d$ [for the multiplications by $(\sqrt{\varepsilon_k^\gamma})$ in Eqs. (52), (53), and (59)] and by $2L^d \log L$ [for the two FFTs in Eqs. (52) and (53)]. The number of iterations necessary to reach a given accuracy, however, has been drastically reduced. The comparison between runs made with and without Fourier acceleration of the conjugate gradient method on the same ensemble of random resistor networks is shown in Table I. The number of operations per iteration for the Fourier-accelerated conjugate gradient method scales as $L^2 \log L$. If the critical slowing down was completely removed (which it in practice never will be), this would also be the scaling of the number of operations necessary to reach a given accuracy. Without Fourier acceleration the number of operations scales as $L^{d_f} \times L^{d_w/2} \approx L^{3.3}$. How this works out in practice for the random resistor network without identifying the backbone is shown in Table I. The critical slowing down is reduced from $L^{1.5}$ to $L^{1.0}$, and thus one finds that the total number of operations scales as $L^{3.0} \log L$. In Fig. 3 we show the development of the error criterion, Eq. (38) or (59), for a 32×32 lattice with and without Fourier acceleration. In ref. 30 the conjugate gradient method was tested with and without Fourier acceleration on the backbone of the random resistor network. In this case the critical slowing down was reduced from $L^{1.3}$ to $L^{0.3}$. Thus, in this case the total number of operations scales as $L^{2.3} \log L$ with Fourier acceleration versus $L^{2.9}$ without Fourier acceleration. (The boundary conditions differed in this case from those

Table I. The Average Number of Iterations to Relax Random Resistor Networks of Sizes $L = 16$, 32, and 64 with and without Fourier Acceleration to within Machine Precision, $\epsilon = 10^{-10}$, Defined in Eqs. (38) and (59), by the Conjugate Gradient Method[a]

L	CG	FaCG
16	161	91
32	455	185
64	1287	392

[a] Ten lattices of each size were generated. CG, Conjugate gradient method; FaCG, Fourier-accelerated conjugate gradient method.

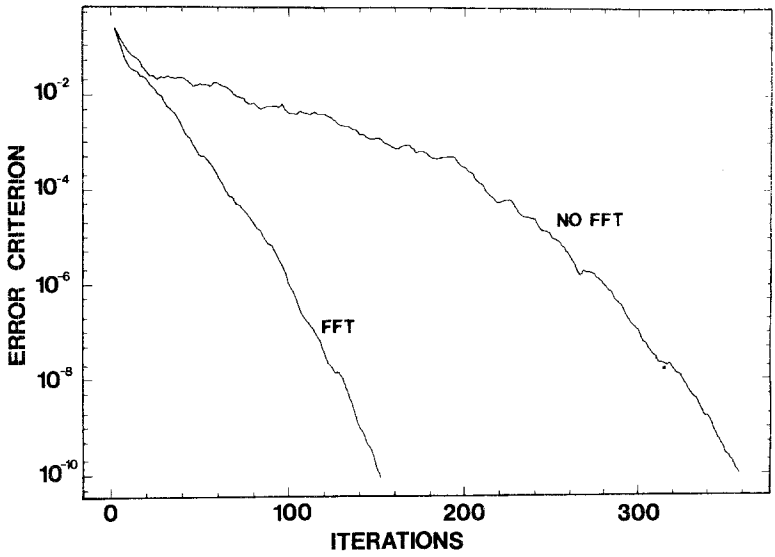

Fig. 3. The development of the error criterion ε, Eq. (38) or (59), for a 32×32 random resistor network lattice with and without Fourier acceleration of the conjugate gradient method.

depicted in Fig. 1: only one node at the upper edge and one node at the lower edges were connected to the exterior current source.)

We now turn to Fourier accelerating the conjugate gradient method when used to relax the elastic central-force network. This is done in essentially the same way as for the random resistor network; one solves an equivalent problem in Fourier space (we present the equations explicitly even though they are very similar to those for the random resistor network):

$$\sum_j \underline{\mathbf{D}}_{ij} \underline{\mathbf{R}}_j = \underline{\mathbf{B}}_i \tag{60}$$

where

$$\underline{\mathbf{D}}_{kl} = (\sqrt{\varepsilon_k^\gamma})\, \mathbf{F}(\mathbf{D}_{kl} \mathbf{F}^{-1}(\sqrt{\varepsilon_l^\gamma})) \tag{61}$$

$$\underline{\mathbf{B}}_k = (\sqrt{\varepsilon_k^\gamma})\, \mathbf{F}(\mathbf{B}_k) \tag{62}$$

and

$$\mathbf{R}_k = \mathbf{F}^{-1}((\sqrt{\varepsilon_k^\gamma}) \underline{\mathbf{R}}_k) \tag{63}$$

The equations constituting the conjugate gradient methods are implemented in the following way: Start with Eq. (32),

$$\mathbf{p}_i^0 = \mathbf{B}_i - \sum_j \mathbf{D}_{i,j} \cdot \mathbf{R}_j^0$$

If $\mathbf{p}_i^0 = 0$, $\mathbf{R}_i^0$ is the solution. If not, calculate

$$\underline{\mathbf{p}}_k^0 = \underline{\mathbf{r}}_k^0 = (\sqrt{\varepsilon_k^\gamma})\,\mathbf{F}(\mathbf{p}_k^0) \tag{64}$$

We can now start the iteration process. For $m = 1, 2, \ldots$, compute the vector

$$\mathbf{p}_i^m = \mathbf{F}^{-1}((\sqrt{\varepsilon_i^\gamma})^{-1}\,\underline{\mathbf{p}}_i^m) \tag{65}$$

Then form the vector

$$\sum_l \underline{\mathbf{D}}_{k,l} \cdot \underline{\mathbf{p}}_l^m = (\sqrt{\varepsilon_k^\gamma})\,\mathbf{F}\left(\sum_l \mathbf{D}_{k,l} \cdot \mathbf{p}_l^m\right) \tag{66}$$

The rest now follows Eqs. (41)–(45) closely:

$$\underline{a}^m = \left(\sum_k \underline{\mathbf{r}}_k^m \cdot \underline{\mathbf{r}}_k^m\right) \Big/ \left(\sum_l \sum_n \underline{\mathbf{p}}_l^m \cdot \underline{\mathbf{D}}_{l,n} \cdot \underline{\mathbf{p}}_n^m\right) \tag{67}$$

$$\underline{\mathbf{R}}_k^{m+1} = \underline{\mathbf{R}}_k^m + \underline{a}^m \underline{\mathbf{p}}_k^m \tag{68}$$

$$\underline{\mathbf{r}}_k^{m+1} = \underline{\mathbf{r}}_k^m - \underline{a}^m \sum_l \underline{\mathbf{D}}_{k,l} \cdot \underline{\mathbf{p}}_l^m \tag{69}$$

$$\underline{b}^m = \left(\sum_k \underline{\mathbf{r}}_k^{m+1} \cdot \underline{\mathbf{r}}_k^{m+1}\right) \Big/ \left(\sum_l \underline{\mathbf{r}}_l^m \cdot \underline{\mathbf{r}}_l^m\right) \tag{70}$$

and

$$\underline{\mathbf{p}}_k^{m+1} = \underline{\mathbf{r}}_k^{m+1} + \underline{b}^m \underline{\mathbf{p}}_k^m \tag{71}$$

An error criterion identical to the one used in Eq. (46) for the unaccelerated conjugate gradient method is

$$\left[\sum_k (\underline{\mathbf{r}}_k^m \cdot \underline{\mathbf{r}}_k^m)/\varepsilon_k^\gamma\right] \Big/ L^{2d} \leqslant \varepsilon^2 \tag{72}$$

When the error criterion (72) is satisfied, $\underline{\mathbf{R}}_k^m$ is converted into the solution $\mathbf{R}_i^m$ by using Eq. (63).

We found that $\gamma = 0.7$ gave the best results. However, as can be seen in Table II, the results are not as good as the random resistor case. See also Fig. 4, which shows the development of the error criterion, Eq. (46) or (72), for a 32×32 random elastic central-force network with and without Fourier acceleration. We find that the number of iterations necessary to relax the system to a within a given accuracy scales as L^2. The reason for this meager result is that the elastic central-force network behaves as two coupled random resistor networks, as discussed earlier. We are trying to

Table II. The Average Number of Iterations Necessary to Relax Random Elastic Central-Force Networks of Sizes $L=16$, 32, and 64 with and without Fourier Acceleration to within Machine Precision, $\epsilon=10^{-10}$, Defined in Eqs. (46) and (72), by the Conjugate Gradient Method[a]

L	CG	FaCG
16	285	201
32	1076	677
64	4565	2817

[a] Ten lattices of each size were generated. CG, Conjugate gradient method; FaCG, Fourier-accelerated conjugate gradient method.

accelerate the relaxation of these two coupled networks by utilizing the average of the inverse diffusion kernel for each of them, $\langle (D^{-1})_{i,j} \rangle$. This does not work very well, since the coupling between the networks is completely left out of the acceleration function. A better approach may be to accelerate this system as a single interacting theory rather than two free

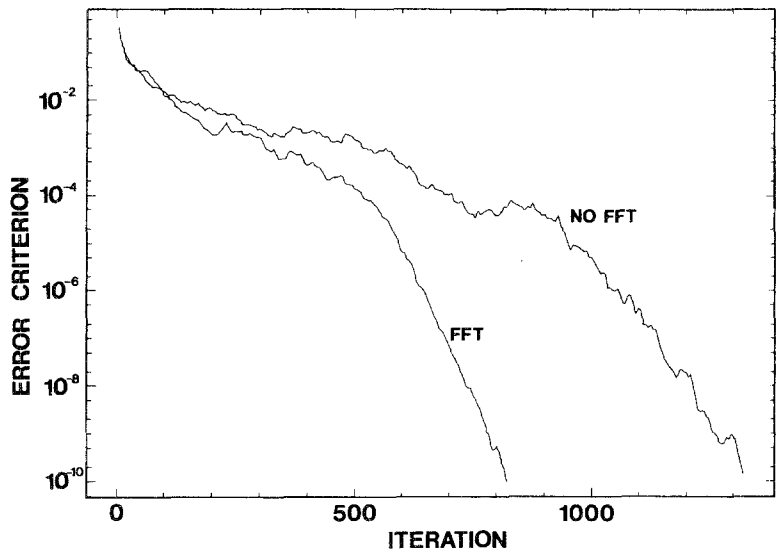

Fig. 4. The development of the error criterion ε, Eq. (46) or (72), for a 32×32 random elastic central-force network with and without Fourier acceleration of the conjugate gradient method.

ones. One possibility is to use a more complicated accelerating function ε_k, but the possible chices are limited; it is important that it is *diagonal* in order to avoid too many operations to perform the necessary matrix multiplications. The situation is the same as for the use of this technique in lattice gauge theory[8]: If one uses a free Green's function as an accelerating function, and the system one is trying to accelerate is strongly interacting, good results are not always obtainable. We should point out that in all cases we have examined, Fourier acceleration succeeded in substantially decreasing the number of iterations needed for convergence. However, the savings in CPU time (which is what counts) have not always been large. One reason for this is that saving a few iterations will be worthwhile only if the CPU time per iteration is large. If this is not so, the overhead in performing the FFTs could be more than the time saved by saving only a few iterations. As an example, we found that for the random resistor network, the Fourier-accelerated conjugate gradient method takes about 1.5 times as much CPU time per iteration as the unaccelerated one. So, if the accelerated conjugate gradient method needs 2/3 of the number of iterations needed by the conjugate gradient method, the CPU time would be the same in both methods. To save CPU time, the accelerated conjugate gradient method should cut the number of iterations down to *less* than 2/3 that of the conjugate gradient method (which it does by a large margin).

5. STOPPING CRITERIA

We end this paper by a short discussion of criteria for when an iteration procedure has found an acceptable solution. This very important question is closely related to the problem of critical slowing down. We discuss it using the random resistor network as a practical example. Figure 5 shows the convergence of the Euclidean norm of the voltage vector in the Jacobi method without Fourier acceleration,

$$V^m = \left[\sum_i (V_i^m)^2 \right]^{1/2} \tag{73}$$

Specifically, we plotted

$$\Delta'_{m+1} = \delta_{m+1}/V^m \tag{74}$$

where

$$\delta_{m+1} = \left[\sum_i (V_i^{m+1} - V_i^m)^2 \right]^{1/2} \tag{75}$$

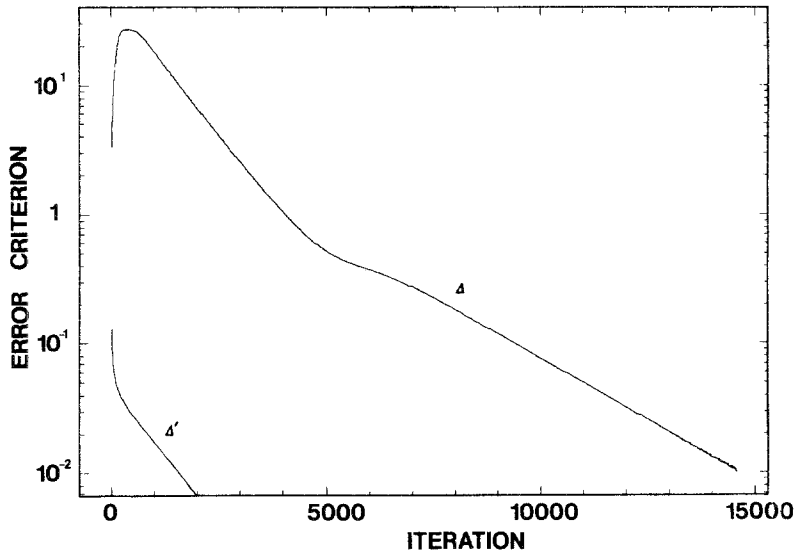

Fig. 5. A plot of Δ' as defined in Eq. (74) and Δ as defined in Eq. (79) versus iteration number for an $L=8$ size random central-force network when using the Jacobi relaxation method. The convergence of the iteration method is geometric; we estimate λ to be 0.9997.

is the norm of the change of the voltage vector from iteration to iteration. The convergence is clearly geometric (the signal of this is the straight line in the semilog plot in Fig. 5), which is to say that the voltage vector is approaching the solution ($m=\infty$) as a geometric series:

$$V_i^m = V_i^\infty + C_i \lambda^m \tag{76}$$

where V_i^∞ is the exact solution and $\lambda \leqslant 1$ is the convergence rate. The worse the critical slowing down in the system, the closer λ is to 1; in Fig. 5 we have $\lambda = 0.9997$. This is a dangerous situation, making the Δ' defined in Eq. (74) unacceptable as an error criterion. To understand this, combine Eqs. (74)–(76) to get

$$\Delta'_{m+1} = \left(\sum_i C_i^2 \right)^{1/2} |\lambda^m (1-\lambda)| / V^m \equiv C\lambda^m (1-\lambda)/V^m \tag{77}$$

Let us now introduce the "true" error, namely the norm of the distance between the correct solution V_i^∞ and V_i^m,

$$\Delta_{m+1} = \left[\sum_i (V_i^\infty - V_i^m)^2 \right]^{1/2} \Big/ V^m \tag{78}$$

Now, combining this equation with Eq. (76) (assuming that there is geometric convergence), we get

$$\Delta_{m+1} = \left(\sum_i C_i^2 \right)^{1/2} \lambda / V^m \equiv C \lambda^m / V^m \tag{79}$$

Now, comparing Eqs. (77) and (79), we see that using Δ'_m as an error criterion, *underestimates* the true error Δ_m by an amount $1-\lambda$, and when critical slowing down is present, this is a *very* small factor, often of the order 10^{-3} or smaller for the systems we are discussing here, as can be seen in Fig. 5. Thus, when critical slowing down is present and the convergence is geometric, one has to use the true error Δ_m rather than Δ'_m as an error criterion. It is therefore important to find a way to express Δ_m by quantities known at a certain iteration *without* knowledge of the correct solution. This is accomplished by approximating it by the following formula[29]:

$$\Delta_{m+1} = \Delta'_{m+1} / |1 - \delta_m / \delta_{m+1}| \tag{80}$$

The idea behind this equation is that when the geometric convergence sets in, $\delta_m / \delta_{m+1} = \lambda$ and thus the factor $1-\lambda$ in Eq. (77) is canceled. It is Δ_m given in Eq. (80) that is plotted in Fig. 5.

The stopping criterion ε we have used in the conjugate gradient method [see Eqs. (38), (46), (59), and (72)] differs slightly from Δ' defined in Eq. (74): it is the norm of the residual vector $r_i = B_i - \sum_j D_{ij} V_j$. However, the problems discussed above will also be present with this stopping criterion. There is geometric convergence also in the conjugate gradient method, even if the convergence is quite noisy, as can be seen in Figs. 3 and 4. However, if we estimate the convergence factor λ from these figures, we find $\lambda = 0.87$ for the random resistor network (Fig. 3) and $\lambda = 0.97$ for the random central-force network (Fig. 4). Thus, the error criterion ε gives a correct estimate of the true error for these runs.

ACKNOWLEDGMENTS

We thank G. R. Katz, G. P. Lepage, M. Nelkin, and J. Vannimeus for numerous useful discussions on this subject. S. Roux deserves special thanks for his many excellent suggestions on how to implement these ideas in the most efficient way. We also thank H. J. Herrmann and D. Stauffer for urging us to write something (hopefully) readable ("or at least detailed enough to be useful") on this subject. The computatins were done at the École Normale Supérieure on an FPS-164 computer supported by GRECO 70 (Expérimentation Numérique), and on an IBM 3090 at IBM Bergen Scientific Center; A.H. is grateful for the hospitality shown there. The authors acknowledge support from the NSF (G.G.B.) and from CEN-Saclay through a Joliot-Curie Fellowship (A.H.).

REFERENCES

1. G. Forsythe and C. B. Moler, *Computer Solutions of Linear Algebraic Systems* (Prentice-Hall, Englewood Cliffs, New Jersey, 1967); J. A. George, *SIAM J. Numerical Analysis* **10**:345 (1973); A. J. Hoffman, N. S. Martin, and D. J. Rose, *SIAM J. Numerical Analysis* **10**:364 (1973).
2. G. H. Golub and C. F. van Loan, *Matrix Computations* (North Oxford, London, 1986).
3. R. S. Varga, *Matrix Iterative Analysis* (Prentice-Hall, Englewood Cliffs, New Jersey, 1962).
4. D. M. Young, *Iterative Solutions of Large Linear Systems* (Academic Press, New York, 1971).
5. L. A. Hagelman and D. M. Young, *Applied Iterative Methods* (Academic Press, New York, 1981).
6. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis* (Springer-Verlag, New York, 1980), p. 572.
7. G. G. Batrouni, A. Hansen, and M. Nelkin, *Phys. Rev. Lett.* **57**:1336 (1986).
8. G. Parisi, in *Progress in Gauge Field Theory*, G. 't Hooft *et al.*, eds. (Plenum Press, New York, 1984); G. G. Batrouni, G. R. Katz, A. S. Kronfeld, G. P. Lepage, B. Svetitsky, and K. Wilson, *Phys. Rev. D* **32**:2736 (1985).
9. H. E. Stanley and N. Ostrowsky, eds., *On Growth and Form* (Martinus Nijhoff, Boston, 1986).
10. S. Kirkpatrick, in *Ill-Condensed Matter*, R. Balian, R. Maynard, and G. Toulouse, eds. (North-Holland, Amsterdam, 1979).
11. R. Rammal, C. Tannous, P. Breton, and A.-M. S. Tremblay, *Phys. Rev. Lett.* **54**:1718 (1985); L. de Arcangelis, S. Redner, and A. Coniglio, *Phys. Rev. B* **31**:4725 (1985).
12. S. Feng and P. N. Sen, *Phys. Rev. Lett.* **52**:216 (1984); A. R. Day, R. R. Tremblay, and A.-M. S. Tremblay, *Phys. Rev. Lett.* **56**:2501 (1986).
13. A. Hansen and S. Roux, preprint.
14. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes* (Cambridge University Press, Cambridge, 1986).
15. H. J. Herrmann, D. C. Hong, and H. E. Stanley, *J. Phys. A* **17**:L261 (1984).
16. S. Roux and A. Hansen, *J. Phys. A* **20**:L1281 (1987).
17. A. Aho, J. Hopcroft, and R. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, Massachusetts, 1974).
18. J. Hopcroft and R. E. Tarjan, *Commun. ACM* (Algorithm 447) **16**:372 (1973).
19. D. Stauffer, *Introduction to Percolation Theory* (Francis and Taylor, London, 1985).
20. H. J. Herrmann and H. E. Stanley, *Phys. Rev. Lett.* **53**:1121 (1984).
21. R. Rammal and G. Toulouse, *J. Phys. Lett.* (Paris) **44**:L13 (1983).
22. S. Havlin, D. Movshovitz, B. Trus, and G. H. Weiss, *J. Phys. A* **18**:L719 (1985).
23. S. Roux and A. Hansen, *J. Phys. (Paris)* **49**:897 (1988).
24. H. E. Stanley and A. Coniglio, *Phys. Rev. B* **29**:522 (1984).
25. B. O'Shaughnessy and I. Procaccia, *Phys. Rev. Lett.* **54**:455 (1985).
26. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes* (Cambridge University Press, Cambridge, 1986), Chapter 12.
27. M. R. Hestenes and E. Stiefel, *Nat. Bur. Stand. J. Res.* **49**:409 (1952).
28. P. Concus, G. H. Golub, and D. P. O'Leary, in *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, eds. (Academic Press, New York, 1976).
29. G. R. Katz, Ph.D. thesis, Cornell University, Ithaca, New York (1986).
30. A. Hansen, Ph.D. thesis, Cornell University, Ithaca, New York (1986).