NIST Internal Report
NIST IR 8591

Quasistatic Crack Formation in Multiphase Materials Driven by Internal Phases Expansion Mechanisms: Application to Cement-Based-Materials

Osamah H.A. Dehwah
Edward J. Garboczi
Nicos S. Martys
Stephanie S. Watson

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8591

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Internal Report
NIST IR 8591

Quasistatic Crack Formation in Multiphase Materials Driven by Internal Phases Expansion Mechanisms: Application to Cement-Based-Materials

Osamah H.A. Dehwah
PREP Associate, Materials and Structural Systems Division
Engineering laboratory
Department of Civil & Systems Engineering
Johns Hopkins University

Edward J. Garboczi
Applied Chemicals and Materials Division
Material Measurement laboratory

Nicos S. Martys
Materials and Structural Systems Division
Engineering laboratory

Stephanie S. Watson
Materials and Structural Systems Division
Engineering laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8591

May 2026

U.S. Department of Commerce
Howard Lutnick, Secretary

National Institute of Standards and Technology
Craig Burkhardt, Acting Under Secretary of Commerce for Standards and Technology and Acting NIST Director

NIST IR 8591
May 2026

Publication History
Approved by the NIST Editorial Review Board on 2025-09-19

How to Cite this NIST Technical Series Publication
Dehwah OHA, Garboczi EJ, Martys NS, Watson SS (2026) Quasistatic Crack Formation in Multiphase Materials Driven by Internal Phases Expansion Mechanisms: Application to Cement-Based-Materials. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Internal Report (IR) NIST IR 8591. https://doi.org/10.6028/NIST.IR.8591

Author ORCID iDs
Osamah H.A. Dehwah: 0009-0000-8183-5824
Edward J. Garboczi: 0000-0002-8973-1330
Nicos S. Martys: 0000-0002-7884-6028
Stephanie S. Watson: 0009-0003-2003-5592

Contact Information
Osamah.Dehwah@nist.gov

Abstract

Internal crack formation and propagation are often found in concrete undergoing expansion of some internal phase or phases, driven by deterioration mechanisms such as pyrrhotite oxidation or alkali-silica reaction. This study employs a 2-D finite element-based coarse-grained quasistatic approach to model crack formation and propagation when selected internal phases expand. Idealized cases of composite materials are first analyzed. The computational model is then applied to microstructure images of cement-based materials to demonstrate its effectiveness. This model enables the testing of various hypotheses regarding internal degradation mechanisms in composite materials, aiding in the development of a conceptual framework and identifying key elements of degradation. An analysis of crack initiation site location, number, and size was conducted, and crack propagation simulations were generated. A semi-empirical relationship between mechanical properties and crack area was proposed. This 2-D model provides a valuable tool to help understand internal cracking due to various deterioration mechanisms driven by expansive material phases.

Keywords

Crack nucleation and initiation; Crack Propagation; Concrete degradation; Expansive phases; Quasi-brittle fracture mechanics.

Table of Contents

List of Figures

Acknowledgments

This work was funded by Congressionally-appropriated Scientific and Technical Research and Services (STRS) to assess pyrrhotite in concrete. Osamah H.A. Dehwah was supported through Professional Research Experience Program (PREP) under award No. PREP0002340. Nicos S. Martys was supported through Domestic Guest Researcher – Research and Science (DGRRS) under Agreement No. 1333ND25FNB73008.

Author Contributions

Osamah H.A. Dehwah: Conceptualization, Methodology, Software, Data curation, Investigation, Writing- Original draft preparation. **Edward J. Garboczi**: Software, Investigation, Writing- Reviewing and Editing. **Nicos S. Martys**: Conceptualization, Software, Supervision, Methodology, Investigation, Writing- Original draft. **Stephanie S. Watson**: Supervision, Project administration, Writing- Reviewing and Editing.

1. Introduction

The deterioration and hence durability of concrete materials are primarily governed by microstructure and complex interactions between the material's microstructure and environment [1]–[3]. A primary form of damage that affects a concrete structure is cracking driven by the interaction of internal or external species with one or more material phases in the concrete [4]–[7]. Crack formation and propagation (crack growth) are critical factors influencing the performance of both unreinforced and reinforced concrete structures [8]–[11], and are considered as primary causes of structural damage and deterioration [12]. Moreover, the deterioration of concrete structures leads to significant maintenance costs [13].

A well-known mechanism for crack formation in concrete is the expansion of internal phases, which produces internal pressure [14]. This process significantly reduces load-bearing capacity [15], causes stiffness loss in structural elements, and hinders the ability of these elements to function as a fully integrated unreinforced or reinforced concrete cross-section [16]–[19]. Further, cracks can increase the permeability of concrete, allowing for enhanced ingress of moisture [20] and deleterious substances like chloride ions [21]. Such damage may lead to failures or catastrophic events in concrete structures [22], [23].

Understanding crack formation and propagation in concrete structures due to physical/chemical deterioration mechanisms is therefore of critical importance. These mechanisms include pyrrhotite oxidation [24], [25], external sulfate attack (ESA) [26], alkali-silica reaction (ASR) [6], alkali-carbonate reaction (ACR) [27], thermal expansion [28], and freeze-thaw [29]. In these cases, a size change due to expansion of internal phases induces a strain/stress in surrounding phases. When these phases reach their critical strain/stress limits in local areas, cracks can be generated. Subsequent strain and stress generated by these mechanisms can then cause cracks to propagate.

The application of analytical modeling to concrete crack propagation is generally constrained to specific cases, such as particular boundary conditions and the assumption that cracks are uniformly distributed throughout the damaged solid [30]. Given the limitations of analytical modeling for crack propagation in concrete, numerical modeling offers substantial advantages. Depending on the application requirements and problem scale, various numerical techniques may be employed, such as the finite element method (FEM) [31]–[35], boundary element method (BEM) [36], [37] and discrete element method (DEM) [38]. Most studies utilized FEM [30]–[35] for external loading applications, mainly for notched beams, where cracks are deliberately introduced and the primary emphasis is usually on a single dominant crack. Numerical models are validated by existing experimental data and used to predict new experimental results [31]–[38].

In contrast, this study addresses crack generation in addition to crack propagation in concrete structures that results from concrete deterioration due to the development of internal stresses, which are caused by factors such as pyrrhotite oxidation, ASR, aging, or any other similar degradation mechanism. A 2-D numerical FEM model of the evolution of crack networks in cement-based-materials due to internal expansion of various phases was developed. Analysis of crack widths and initiation sites was conducted, and crack networks generated. Using various

real 2-D microstructures, elastic material properties (e.g., elastic, shear and bulk modulus) were determined for reasonable values of cement paste and aggregate elastic moduli. This program serves as a valuable tool to qualitatively relate microstructure to physical/chemical degradation in the production of crack networks that affect concrete durability.

2. Numerical Model Development

The simulation begins with a representation of the multiphase material, such as concrete, which is discretized into individual (square) pixels. This 2-D digital image is usually taken from a real material that has undergone physically or chemically induced cracking, although any microstructure image or artificial digital image can be used. If cracks were originally present in the material, the cracks are removed. Each pixel represents a small, localized region of the microstructure that is a single material phase. Specific material properties, such as Young's modulus, shear modulus, bulk modulus, Poisson's ratio, and failure tensile strain/stress are assigned to each pixel based on the phase it represents. This pixelated approach enables the simulation to account for the distinct mechanical behavior of different phases, such as aggregates, cement paste, and voids, thereby providing a more accurate depiction of the material's overall response under internal physically or chemically induced degradation.

Certain phases undergo expansion due to various mechanisms, including thermal effects and chemical reactions that result in the expansion of the phase. The amount these phases expand is controlled by assigning them an intrinsic strain. Another way of thinking of this is to treat the phase as chemically-reacted to form a new phase with a different (larger) equilibrium area, which the phase tries to achieve at the expense of a rise in local stresses and strains in the surrounding material. Once the failure stress or strain is reached at a specific pixel, that pixel is converted into a crack pixel, where the Young's modulus is reduced to zero. The microstructure is updated to reflect the presence of the crack, and the stress field is recalculated while keeping all other parameters constant. At each cycle of the model, the pixels with the highest stresses (strains) are identified, and those meeting the critical stress (strain) threshold are subsequently converted into crack pixels. This process is iteratively repeated.

2.1. Quasistatic approach

Concrete is classified as a quasi-brittle material, characterized by a gradual decrease in tensile stress after reaching its tensile strength, while tensile strain or displacement continues to increase [30]. A coarse-grained quasistatic approach is adopted to model crack formation due to the inherent challenges in accurately resolving the fine temporal and spatial scales associated with crack nucleation and propagation. Detailed representation of these processes requires high-fidelity simulations to capture the rapid dynamic evolution of stress fields and material responses. This level of detail is not needed to qualitatively relate the physically or chemically driven expansion of certain phases to the generation of crack networks. The quasistatic framework, based on linear elasticity, strikes a balance between capturing the essential mechanics of crack development while maintaining computational tractability to treat representative areas of microstructure. The assumption of linear elasticity is a simplifying assumption, but not as much as might be expected, given the random microstructure. A multiscale approach is an alternative method that offers the potential to capture both small-scale and large-scale phenomena in crack formation [39], [40]. However, the system sizes that could be studied using this approach, especially in 3D, would still be significantly constrained due to the substantial computational resources required, limiting its practical application for large-scale systems.

The stress field is determined by solving the governing elastic equations through a variational principle that states that at equilibrium the elastic potential energy is an extremum [41]. In real cases such as the present model, the extremum is a minimum. The elastic equations are discretized using the FEM method where each pixel is a bi-linear, four-node quadrilateral finite element, with shape functions generated by linear interpolation of the displacement field in both directions, x and y. While spurious shear terms may arise under pure bending, potentially leading to artificially increased stiffness, such strain states are rare in the random microstructures considered, particularly when the system is not heavily cracked and elastic moduli remain above 10 % to 20 % of the uncracked value. At the heavily cracked limit, intact bridges between phases may experience localized bending, but these typically span multiple elements, and it is unclear whether such bridges would sustain pure bending modes or exhibit excessive stiffness. Previous program tests have shown that this element formulation yields accurate bulk and shear moduli in complete systems and systems with a small number of defects [41].

The resulting system of linear equations is solved iteratively using a conjugate gradient solver. This solver efficiently optimizes the displacement field so that the gradient of the elastic energy is close to zero with respect to further changes in displacements, allowing for computationally efficient convergence to the equilibrium state, where the local strain and stress fields are accurately determined.

For a composite microstructure composed of different phases, such as in concrete, the following steps outline the process to model the evolution of crack network using the finite element code for a given input file:

1. Each phase is assigned a corresponding Young's modulus and Poisson's ratio.
2. Phases expected to undergo internal expansion are assigned a fixed predefined internal strain and designated as expansion sites.
3. The FEM code utilizes this information to compute the stress/strain distribution within the microstructure.
4. Based on the solution, for each pixel the principal stresses are computed.
5. Regions of high principal stress (or strains) are identified. For a predetermined number of cracks N, the N pixels exhibiting the highest stress levels, exceeding a critical stress (or strain) threshold, are converted into crack pixels with zero modulus.
6. Once these new crack pixels are incorporated into the microstructure, the effective elastic, shear, and bulk moduli, as well as Poisson's ratio, are recalculated for the entire system.
7. Using the updated microstructure, the process is repeated to follow the evolution of cracks over subsequent iterations. The number of iterations is set to 99.
8. After each iteration, an updated microstructure is generated. This updated microstructure can be used to calculate the elastic properties.

The flowchart illustrating the FEM process is presented in Fig. 1.

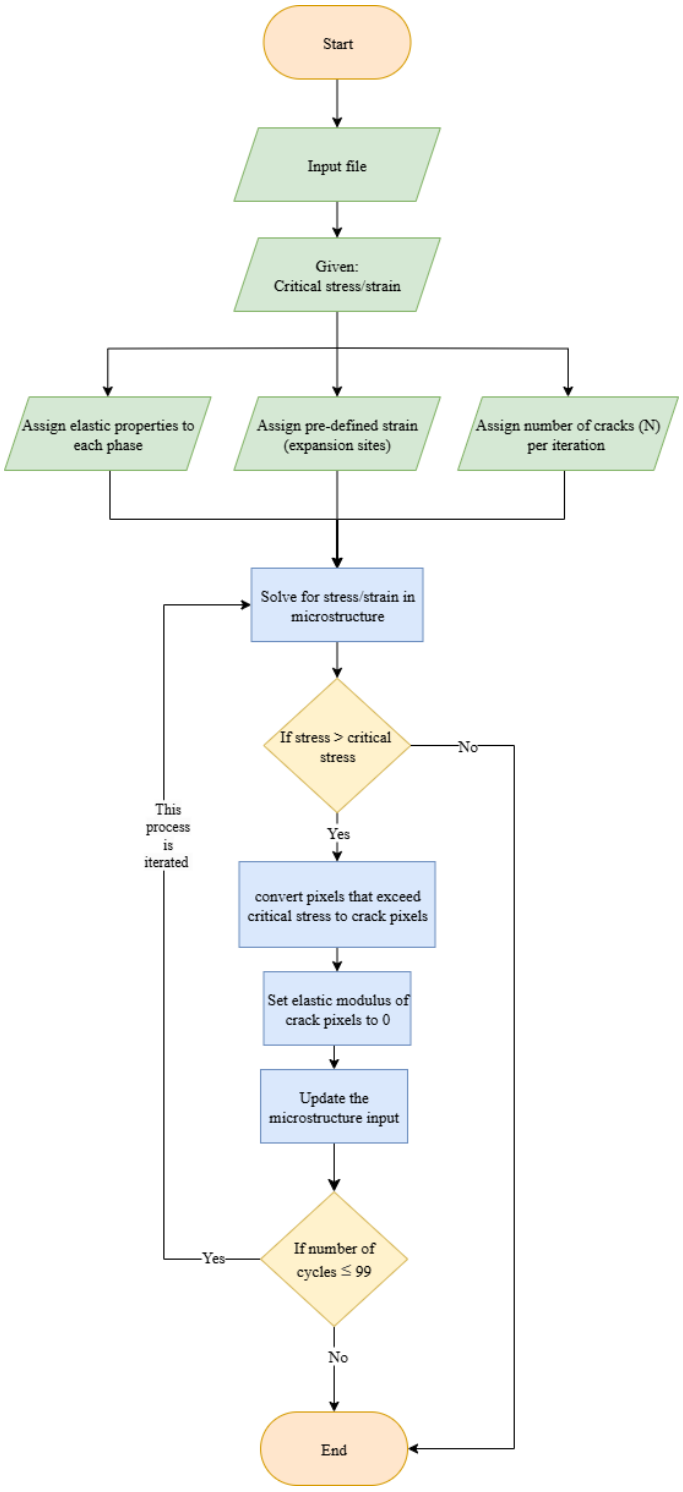


Fig. 1. FEM process overview

2.2. Formulation

The fundamental concept of this model is that a variational principle governs linear elastic problems, allowing these problems to be framed in terms of energy minimization. For a given microstructure subjected to applied fields, such as external forces or boundary conditions, the resulting elastic displacement distribution is determined by the extremization of the total stored elastic energy. This can be achieved when the gradient of the energy with respect to the problem variable, i.e., elastic displacement, vanishes (reaches zero), signifying equilibrium.

The variable E_n represents the energy term used in this study. To minimize E_n, which is a function of the displacements of each node, denoted u_m, the corresponding partial derivatives must be set to zero for all values of m, as expressed by the following equation:

$$\frac{\partial E_n}{\partial u_m} = 0 \qquad (1)$$

During the solution, the relaxation process, which is the sum of the squares of all elements of the gradient vector, where the m^{th} element corresponds to the partial derivative in Eq. (1), is computed. The solution is considered achieved when this sum falls below a predefined threshold, ensuring that the condition in Eq. (1) is approximately satisfied for all m. This threshold must be selected to be sufficiently small to ensure that further relaxation does not result in significant changes in the calculated strains and stresses within the finite elements (pixels).

The objective of this FEM analysis is to express the total energy of the system solely in terms of the elastic displacements at the nodes, thereby discretizing the continuum. Several approaches exist for achieving this; here, a bi-linear interpolation scheme, is employed. In 3D, a cubic voxel is the basic finite element that uses a tri-linear scheme [42]. Specifically, the energy for a single pixel is formulated as an integral over that pixel, which is then combined with the energies of all pixels to construct a global energy function for the system. After performing the integral, the pixel energy is expressed as a quadratic function of the nodal displacements. Consequently, the global energy becomes a quadratic function of the nodal displacements. This global energy is then minimized with respect to the nodal displacements using a conjugate gradient scheme. The elastic energy that is stored obeys a variational principle and given by:

$$E_n = \frac{1}{2}\int \varepsilon_{pq}\, C_{pqrs}\, \varepsilon_{rs}\, d^3r \qquad (2)$$

The strain tensor ε_{pq} and the elastic moduli tensor C_{pqrs} are expressed in full tensorial form, where $p, q, r, s = 1,2,3$, and the integral is computed over the area of each individual pixel or element. The total energy of the system is then obtained by summing the contributions from all pixels.

Given that the strain tensor is symmetric, Voigt notation is typically used to simplify the representation. In the 2-D case, $p, q, r, s = 1,2$ and the strain is represented as a 3-component vector containing the independent strain components (ε_{xx}, ε_{yy}, ε_{xy}). Accordingly, the elastic moduli tensor C_{pqrs} is reduced to $C_{\alpha\beta}$, where:

$$\varepsilon_{pp} = \frac{\partial u_p}{\partial x_p} \tag{3}$$

Therefore:

$$\varepsilon_{pq} = \frac{\partial u_p}{\partial x_q} + \frac{\partial u_q}{\partial x_p} \tag{4}$$

The governing equation, Eq. (2), will be solved using FEM, with the complete derivation provided in [41]. Note, to determine the effective moduli, a selected strain is applied via the periodic boundary conditions. A periodic displacement at the right-hand side is equal to the displacement at the far left-hand side plus the appropriate strain times the system size in that direction. For a strain applied in this way, the average stress throughout the system is computed, which gives the effective moduli.

2.3. Modulus Computations in 2-D

Establishing a direct analytical relationship between 2-D and 3-D elasticity presents certain challenges. Engineers typically approach problems in three dimensions, and adapting this perspective to 2-D analysis requires assumptions of either plane strain, absence of $\mathcal{Z}$-direction strains, or plane stress, absence of $\mathcal{Z}$-direction stresses. However, it is feasible to formulate the equations of elasticity directly in 2-D, independently of but analogous to the 3-D framework. This method is commonly employed in FEM programs, yielding results that resemble a plane strain approach. When applying these programs in a plane strain or plane stress context, the appropriate moduli can be substituted for the 2-D moduli. The subscripts denote the dimensionality of the modulus, where K represents the bulk modulus and G the shear modulus. For isotropic elasticity, the full elastic moduli tensor C_{ij} can be defined by two independent constants, typically selected as either K and G, or E and ν (Young's modulus and Poisson's ratio, respectively). In linear elasticity, there are analytical relationships between these two pairs of constants, as only two constants remain independent for isotropic materials. These relations are different for 2-D and 3-D. In 2-D [41]:

$$K_2 = \frac{E_2}{2\,(1-\nu_2)} \tag{5}$$

$$G_2 = \frac{E_2}{2(1+\nu_2)} \tag{6}$$

Periodic boundary conditions are used in the FEM model.

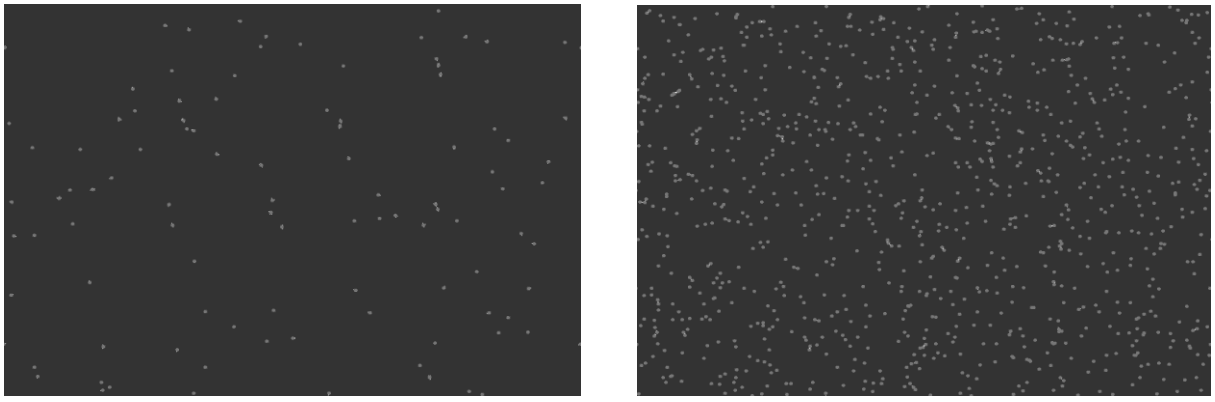
3. FEM Model

Several aspects of the simulations are discussed in this section. For a typical 2-D system (using 844 x 648 pixels in this simulation as this was the size of the original scanning electron microscope (SEM) image), the wall clock time to reach a stress field solution ranged from 10 to 30 min, with longer times observed as the system approached the formation of a percolating crack spanning the entire structure. A potential solution to accelerate the simulation is to run it on multiple processors (parallel computation). To determine the sensitivity of crack formation, the density of expansion sites and the number of cracks per iteration were studied in the following sections.

3.1. Density of Expansion Sites

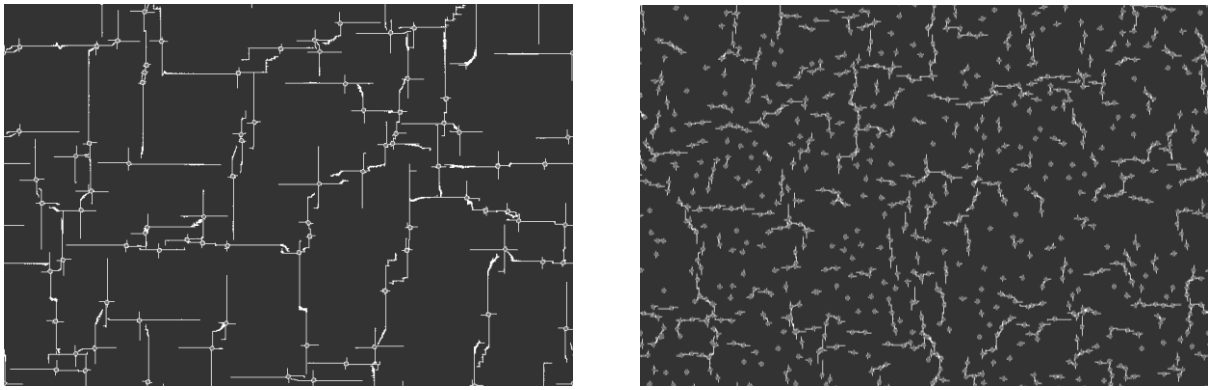
Initially, a simple two-phase model microstructure was considered: an expansive phase (1), which is assigned an initial strain, and a second phase (2), the matrix phase, with no initial strain. The expansive phase is modeled as a pixelated disk with a radius of three pixels. The number of phase (1) disks and how they are distributed can be varied to investigate the influence of these variables on crack network development. One can think of this simple model as a real case where the expansive phase is highly divided amongst the non-expanding phase. The density of expansion sites refers to the number of these disks per area and generally serve as the locations where cracks originate. Crack formation behavior varies with the distribution and density of these expansion sites although only the density of expansion sites was varied in this report. The disks were randomly placed by choosing center coordinates utilizing a pseudo-random number generator.

A comparison of two different crack site densities is shown in Fig. 2, where systems with 100 sites (disks) and 1000 sites, representing 0.5 % and 5 % of the total area, respectively, are illustrated. The corresponding crack formation and propagation patterns from these sites are depicted in Fig. 3. As observed, lower densities of expansion sites result in longer crack lengths, while higher densities lead to a shorter crack propagation, demonstrating an inverse relationship between site density and crack length. As disk area coverage increases, with disk size fixed, the average distance between disks goes as roughly (disk area fraction)$^{-1/2}$. This would only be strictly true for ordered arrangements of disks on a square lattice but one could assume that this would roughly hold for random configurations. One might expect that the average crack length, as seen in Fig. 3, would follow something like this relationship. This would mean that the average crack length in Fig. 3 (b) would be $10^{-1/2} \approx 3$ times smaller than the cracks in Fig. 3 (a). Of course, this is an approximation; a more accurate estimate would involve averaging crack lengths over multiple microstructure realizations.



(a) 100 Sites (0.5 % Area) (b) 1000 Sites (5 % Area)

Fig. 2. Crack Sites for (a) 100 Sites; (b) 1000 Sites



(a) 100 Sites (0.5 % Area) (b) 1000 Sites (5 % Area)

Fig. 3. Crack propagation for (a) 100 Sites; (b) 1000 Sites

The relationship between Young's modulus and crack area is presented in Fig. 4. An interesting result was observed in that increasing the number density of expansion sites (number of expansive sites) contributed to maintaining a higher modulus in the macroscopic sample. This occurred because cracks tended to propagate over shorter distances, stopping once they reached neighboring expansion sites. Cracks tend to remain localized when the number of expansion sites is higher, whereas a smaller number of expansion sites results in longer and more extensive cracks, leading to severe damage throughout the system. The longer cracks will lower the Young's modulus more rapidly in comparison to the shorter cracks, which dead-end at an expansion site nearby. However, this finding should not be interpreted as suggesting that an increased number of expansion sites (e.g., higher pyrrhotite content) enhances resistance to concrete degradation. Pyrrhotite expansion is coupled with sulfate diffusion, leading to internal sulfate attack (ISA), which accelerates concrete deterioration and ultimately results in structural failure [43]. This finding is likely a byproduct of the simple 2-D model and therefore cannot be used to explain results in real-world systems. However, a potential application is to update the input file to account for other deterioration mechanisms including ISA.

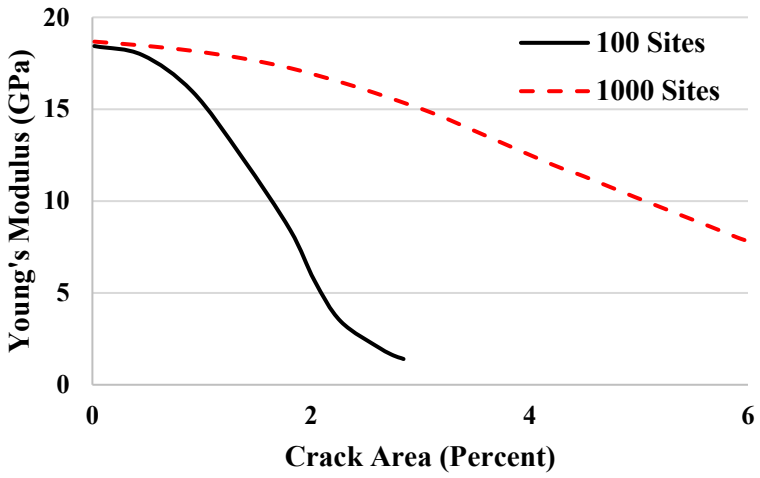


Fig. 4. Young's Modulus vs. Crack Area for 100 and 1000 sites of cracks

3.2. Real Image Simulation

As stated above, the model can be applied to a real image that has been converted into a digital format. For the purpose of this work, the image is stored in the ".pbm" format, an ASCII-based graphic format that is easily interpreted by the program. Fig. 5 provides an example of such an image, derived from a polished cross-section of pavement concrete [44]. Thresholding is applied to the image, in such a way that the image becomes an n-phase image, with each phase having its own gray-scale label for all its pixels. This digital image will be referenced throughout the manuscript as a representation of a composite material with multiple phases, such as concrete. Black is matrix, light gray = no-expansive inclusions, and dark gray = expansive inclusions.

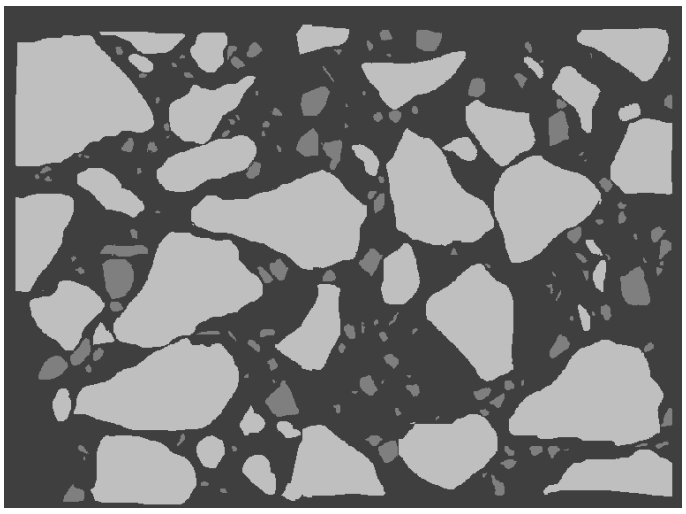


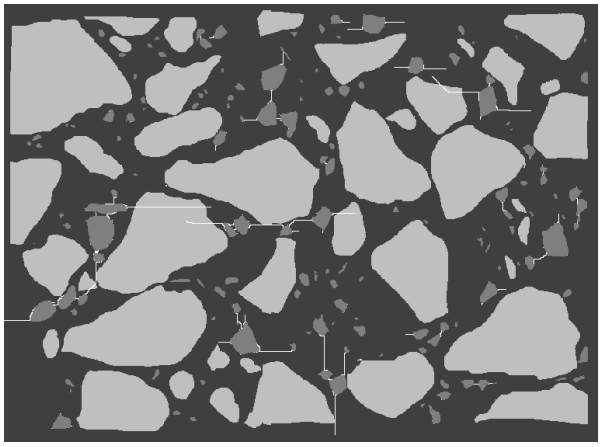
Fig. 5. A digital image of a polished cross-section of concrete [44] **with segmented phases produced using image processing on the original SEM image.**

3.3. Number of Cracks Per Iteration

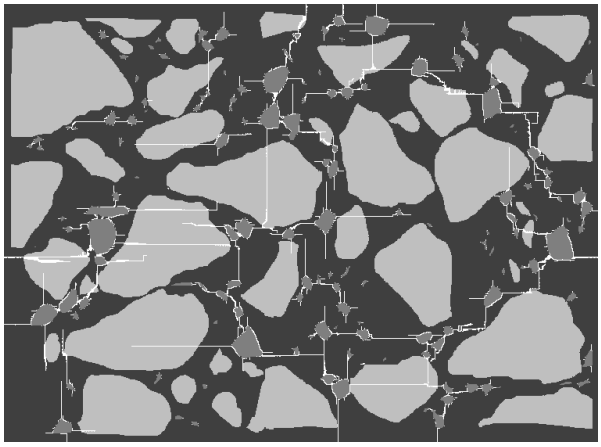
Crack formation is influenced by the number of pixels transformed into cracks in each iteration. An advantage of the developed model is the flexibility in selecting the crack number, which is the number of cracks that is generated per iteration in the microstructure and represented by N. Considering the respective roles of the number of cracks introduced per iteration and the threshold parameter, both jointly govern the evolution of the cracking pattern. While N sets the upper bound on how many candidate pixels may be selected at each iteration, the threshold determines which of those candidates are eligible to crack based on local stress. As a result, when the threshold is high, many of the N candidates may fail to meet the condition, effectively reducing the number of cracks introduced regardless of how large N is. This implies that increasing N beyond a certain point becomes immaterial: the threshold acts as a gating mechanism, and only pixels exceeding it can contribute to crack propagation.

A comparison is made across different crack numbers per iteration. This means that if N crack pixels are generated in each iteration, these N pixels have the highest values of principal tensile stress (strain). Fig. 6 illustrates four scenarios: (a) 1 crack, (b) 10 cracks, (c) 100 cracks, and (d) 400 cracks per iteration.

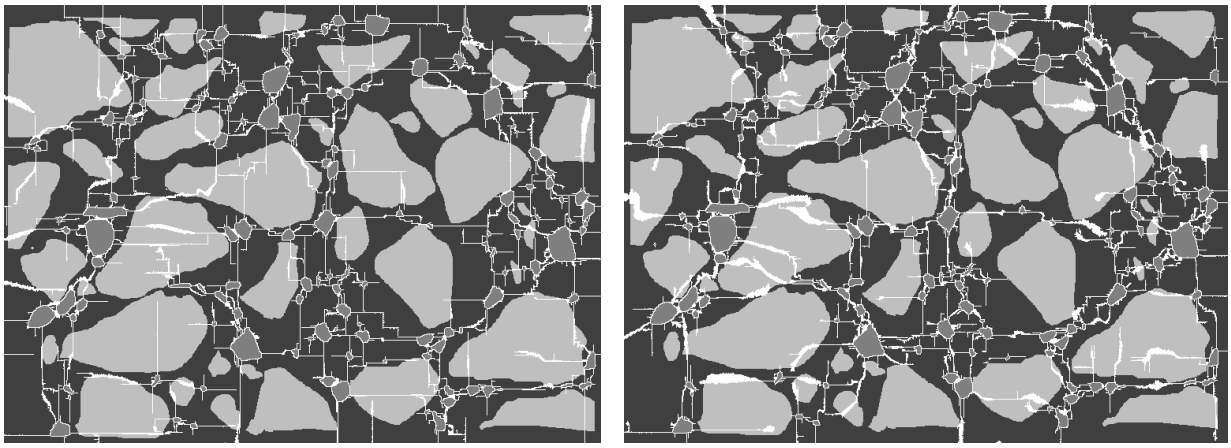
An increase in the number of cracks per iteration resulted in wider cracks. To provide a more quantitative basis for estimating crack width, several algorithms were applied, all yielding consistent results. For a given crack network, each pixel identified as part of a crack was evaluated by counting the number of connected crack pixels in the horizontal (left and right) and vertical (up and down) directions. The smaller of the two counts was taken as the local crack width. This process was repeated for all pixels within the crack network. To ensure accuracy, the results were corrected for redundant counting so that each crack width was measured only once across the network. Fig. 7 presents the calculated average crack widths corresponding to different numbers of cracks introduced per solution iteration.



(a) 1 crack per iteration



(b) 10 cracks per iteration



(c) 100 crack per iteration (d) 400 cracks per iteration

Fig. 6. Crack propagation for different number of cracks per iteration

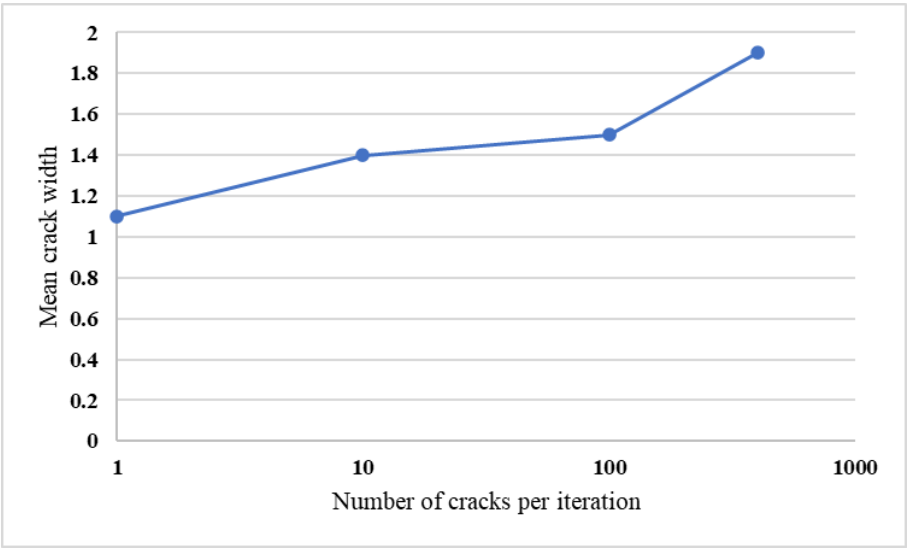
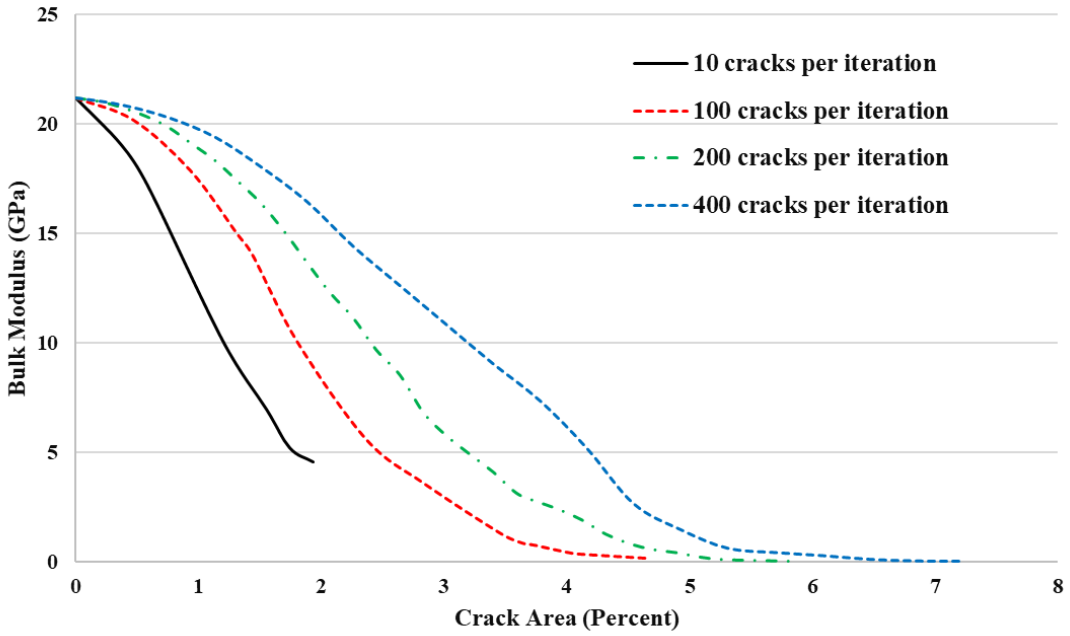
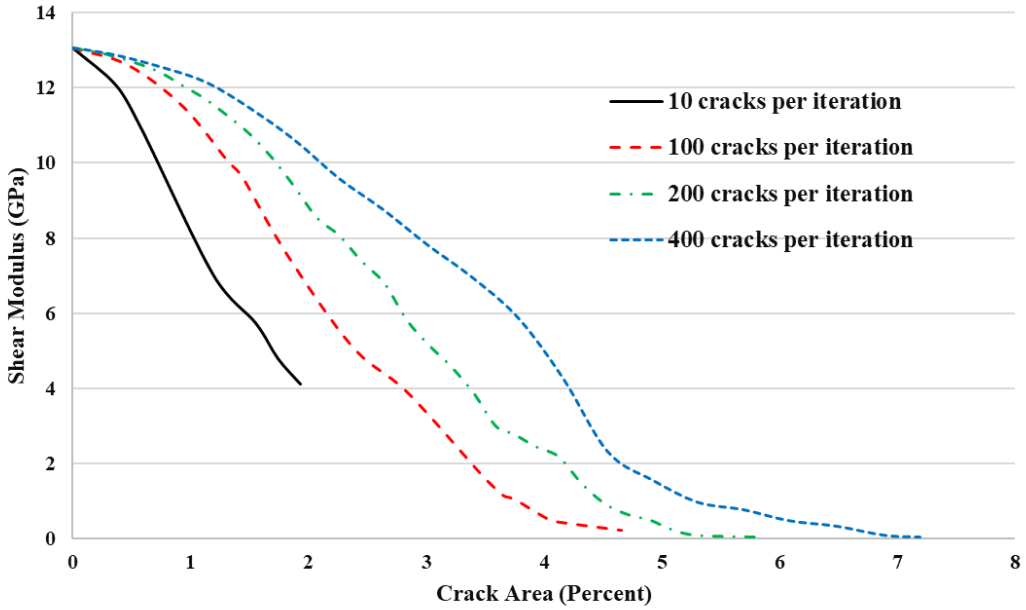


Fig. 7. Mean crack width per iteration

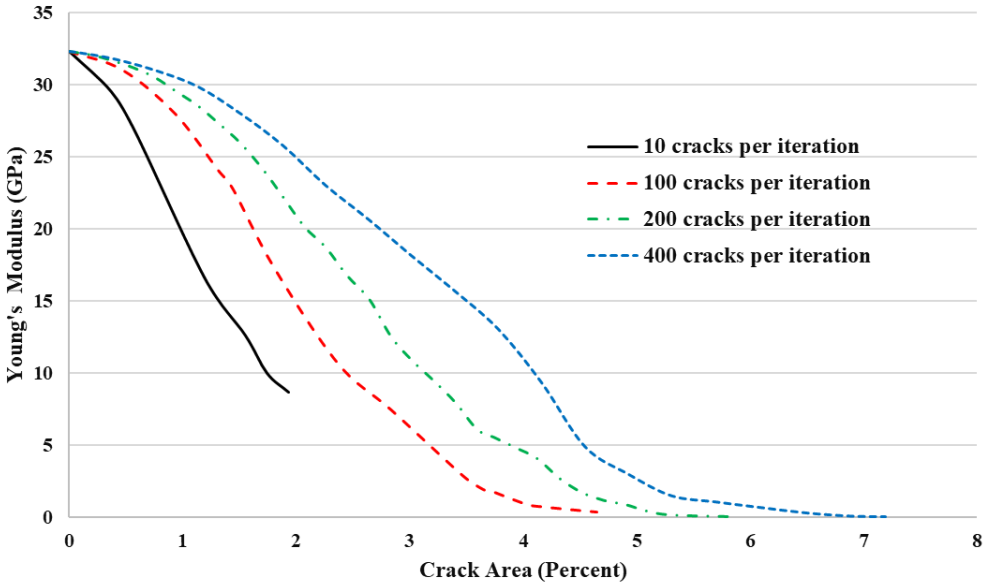
When limiting the simulation to one crack per iteration, the cracks appeared finer and more dispersed. The longer and thinner cracks observed when N=1 likely result from the stress singularity at crack tips. Limiting the simulation to one crack per iteration means that, on each step, only pixels already at existing crack tips—where stresses are highest—are likely to fracture. This configuration also led to the most rapid reduction in modulus relative to total crack area as depicted in Fig. 8. The underlying reason is that, for a given crack area, thinner cracks are, on the average, longer, thereby more significantly reducing the overall modulus of the macrostructure. For instance, in Fig. 8 (c), a Young's modulus of 15 GPa was reached after approximately 50, 115, 210, and 380 iterations for 400, 200, 100, and 10 cracks, respectively. This indicates the elastic modulus is highly dependent on the length of the crack as it begins to span a wide distance of the microstructure. This is the same result as was seen earlier in the simple model with uniform expansive disks.



(a) Bulk Modulus



(b) Shear Modulus

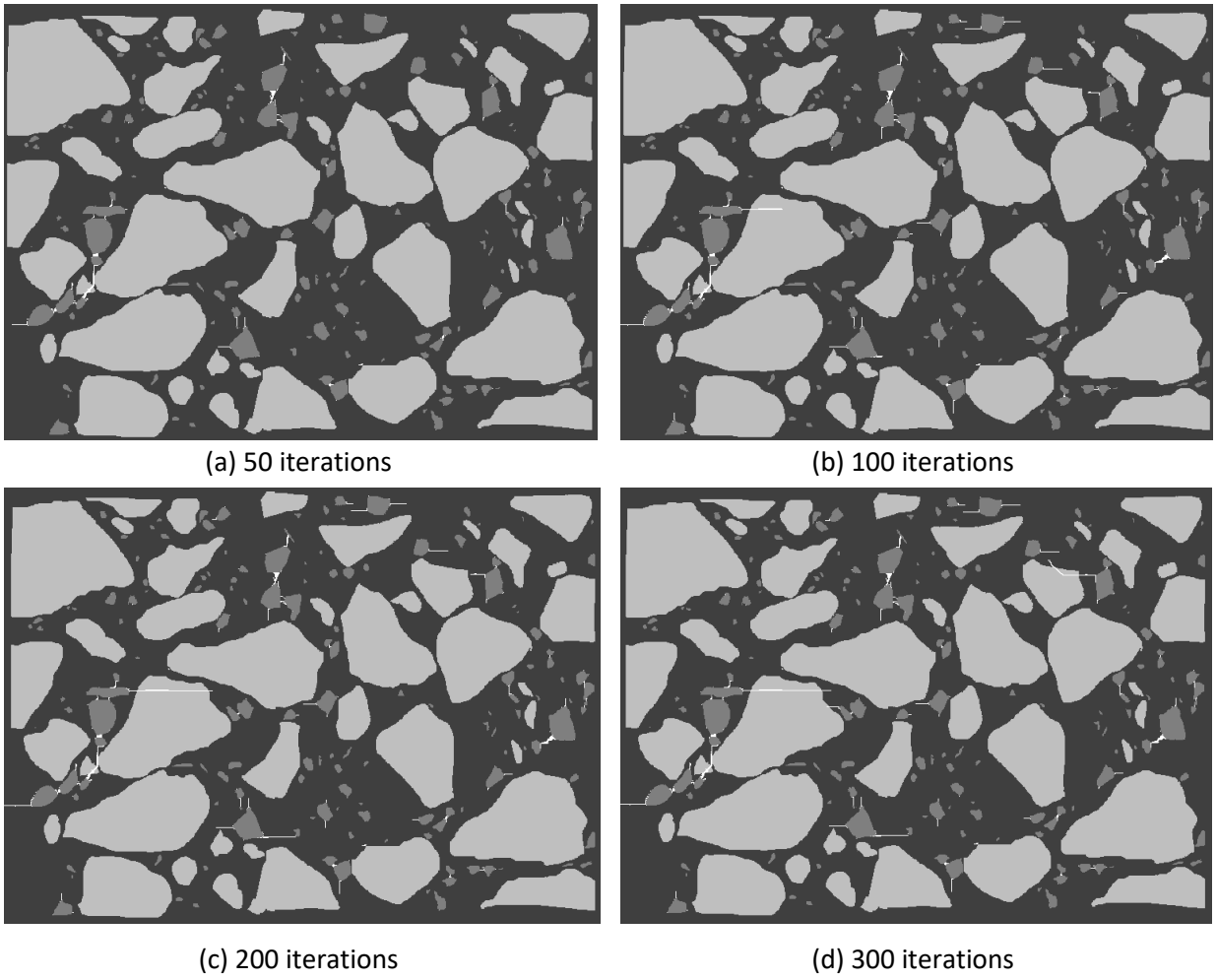


(c) Young's Modulus

Fig. 8. (a) Bulk, (b) Shear and (c)Young's Modulus Vs. Crack Area

In the initial state of the simulations, a fixed strain is assigned to certain phases. This strain can arise from various physical mechanisms, such as thermal expansion, volumetric changes due to chemical reactions, or crystal formation.

It is likely that using fewer cracks per iteration may best represent the crack microstructure of very stiff materials like concrete. However, finding such a solution is time-consuming. As shown in Fig. 6, simulating ten cracks per iteration generated a microstructure similar to that obtained with one crack per iteration. The crack propagation for ten cracks per iteration is illustrated in Fig. 9, showing the crack network at four stages: (a) 50 iterations, (b) 100 iterations, (c) 200 iterations, and (d) 300 iterations. Note, under periodic boundary conditions, at the pixel scale, the microstructure of the degraded composite structure is not affected using a periodic mesh. However, at the system level, a "neighbor" can be located on the opposite side of the image, potentially causing cracks to propagate, after a very large number of iterations, across the image boundary from one side to the other.



(a) 50 iterations (b) 100 iterations

(c) 200 iterations (d) 300 iterations

Fig. 9. Crack propagation for ten cracks per iteration at (a) 50, (b) 100, (c) 200, and (d) 300 iterations.

3.4. Anisotropy in Crack Formation

It was observed that there is a tendency for the cracks to form horizontally or vertically, which is an artifact of the finite element solution being performed on a square lattice. This anisotropic effect is more pronounced at lower crack number, as shown in Figs. 6 (a) and (b). At higher crack number, the effect is reduced because the local stress field promotes the conversion of pixels adjacent to existing cracks, increasing the likelihood of crack paths developing in directions other than purely horizontal or vertical. Although alternative strategies for crack selection that allow for more isotropic crack growth could be incorporated into the code, they are beyond the scope of this study.

3.5. Sources of Variability

The finite element code employed in this study represents the displacement field using a piecewise linear approximation, which would make the normal strain and stress continues at interfaces, but they are not forced to be. The local stress field is solved approximately using an

iterative conjugate gradient algorithm. A very small tolerance on the norm of the residual is specified to ensure that the final solution is effectively independent of the tolerance itself. Since the appropriate tolerance value is sensitive to the system size, careful selection is required. In this work, a typical tolerance on the order of $nx \times ny \times 10^{-12} = 5.47 \times 10^{-7}$ has been used, where nx and ny denote the number of elements in the x and y directions, respectively.

When evaluating the dependence of Young's modulus on crack evolution in a typical composite material, significant variability can arise solely from the initial microstructural configuration. This variability is driven by several factors, including differences in the mechanical properties of the individual phases and their spatial distribution within the material.

This study primarily focuses on the development of the model. A follow-up study will apply this framework to a real case study i.e., concrete materials, involving internal phase expansion due to chemical reaction (pyrrhotite oxidation). Further, it will explore the sensitivity of the results to convergence criteria and mesh resolution.

4. FEM Program

This section provides an overview of the FEM program, beginning with a description of its requirements and guidelines for users. A detailed program manual is then presented, including the names of the input and output files, along with a summary of the function and purpose of each file.

4.1. FEM Program Guidance

The program was developed in Fortran 77 [41], with standard compilation and storage requirements. The stress analysis code can be compiled using gfortran, with the crucial requirement that double precision is employed. This is necessary due to the solution algorithm involving operations on large arrays, including the multiplication of large and small values, and the computation of determinants of large matrices. Without double precision, numerical inaccuracies will arise. The program is divided into three components: the main program, restart module, and mechanical properties module. Each is explained in detail in the following section.

The corresponding compilation commands are as follows:

A. The main FEM program: “gfortran -fdefault-real-8 -o thermal2d-critical-stress-cracking thermal2d-critical-stress-cracking.f”
B. The FEM program for restart: “gfortran -fdefault-real-8 -o thermal2d-stress-cracking_restart thermal2d-stress-cracking_restart.f”
C. The FEM program for mechanical properties: “gfortran -fdefault-real-8 -o thermal2d-moduli thermal2d-moduli.f”

4.2. Input Data Files

For the initial run, the main code "thermal2d-critical-stress-cracking" is used, and it is executed only once. The input file, "ab4.in," represents the initial microstructure configuration. The output consists of 99 files named "fort.10XX," where "XX" ranges from 01 to 99. The file "fort.1099" contains the final microstructure. Additionally, GIF files are generated for each iteration, visually representing the evolving microstructures and corresponding crack patterns associated with the “fort.10XX” files. The code also produces an output file, "phasemod_values," which records the modulus for each phase type.

The simulation framework is configured to iterate through 99 solution cycles. In each cycle, the simulation identifies regions where the stress or strain exceeds a predefined critical threshold. A user-defined number of the highest stress (or strain) values that surpass this threshold, specified by ‘strcrit’ in the code, are then selected and converted into crack pixels by assigning a modulus of zero to those locations. After this modification, the simulation restarts and evaluates the stress distribution based on the updated microstructure. This process is repeated for the specified number of iterations.

To continue the simulation beyond 99 cycles, the microstructures generated can be used as input for the "restart" code, titled "thermal2d-stress-cracking_restart," to further investigate crack propagation. To do so, the “fort.1099”, or another selected fort file "fort.10XX", is copied to "micro.in" to serve as the input for the restart code. Additionally, the restart code utilizes the "phasemod_values" file to define the properties of each phase.

The macroscopic moduli of the microstructures are determined using the "thermal2d-moduli" code. This code calculates the bulk modulus, Young's modulus, shear modulus, and Poisson's ratio of the entire microstructure. To compute these properties for a specific microstructure, the corresponding "fort.10XX" file should be copied to "micro.in2" as input. The code also relies on the "phasemod_values" file to define the material properties for each phase.

The following sets of codes were utilized in the simulations discussed above [41]. These codes were subsequently modified to incorporate crack formation based on a specified critical stress or strain condition. The modifications were implemented according to a protocol proposed to describe quasistatic crack formation. The main scripts for the FEM programs are provided in Appendix A, and the digital code repository can be accessed and downloaded via [45].

4.3. FEM program manual (Input/Output file names):

A. To start the code: For gifs from 00 to 99

- ‘ab4.mic’ (input file describing microstructure)
- ‘thermal2d-critical-stress-cracking.f’ (the main FEM program for the crack simulation)
- ‘phasemod_values’ (output file that has the modulus of each phase)
- ‘fort.10XX’ (output describing microstructures as the simulation progresses)
- ‘00.gif’ (output: gif image of the initial microstructure without cracks)
- ‘XX.gif’ (output gif images of the evolving microstructures corresponding to the fort.10XX files)

B. To restart the code: For gifs from 99 to 198.

- ‘phasemod_values’ (input of modulus of each phase)
- ‘micro.in’ (input: starting image used to continue the simulation. Typically copy the fort.1099, or another selected fort file, to ‘micro.in’)
- ‘thermal2d-stress-cracking_restart.f’ (the FEM program for restarting the crack simulation)
- ‘fort.10XX’ (output describing microstructures as the simulation progresses)
- ‘XX.gif’ (output gif images of the evolving microstructures corresponding to the fort.10XX files)

C. Physical/Mechanical properties:

This file is used to obtain the results for the microstructure physical and mechanical properties, such as Young's modulus, shear modulus, bulk modulus, and Poisson's ratio.

- ‘micro.in2’ (input microstructure configuration image of the desired iteration. Typically copy the desired ‘fort.10XX’ to ‘micro.in2’)
- ‘thermal2d-moduli.f’ (The FEM program used to calculate the physical/mechanical properties)
- ‘ab4-elas-mod.out’ (output: it has the phase area fractions and physical/mechanical properties, based on formulations from section 2.2)

5. Conclusions

Physically or chemically induced expansion of internal phases, leading to the growth of crack networks is an important mechanism in the degradation of concrete. Such mechanisms are reflective of the presence of deleterious phases in concrete and the interaction of the microstructure and environment. In this study, a 2-D quasistatic simulation approach was developed to qualitatively model the growth and mapping of cracks resulting from the expansion of internal phases such as pyrrhotite, ASR, thermal expansion, and freeze-thaw. The following conclusions were found:

- A qualitative numerical model for simulating the development of internal-expansion-driven crack networks in cement-based materials was developed using the FEM. An analysis was conducted on crack widths and initiation sites, and corresponding crack propagation simulations were generated.
- The density of expansion sites does not directly correlate with a reduction in modulus. Lower densities of expansion sites result in longer crack lengths, whereas higher densities lead to shorter crack propagation, indicating an inverse relationship between site density and crack length, perhaps approximately an inverse square root relationship. An intriguing result observed in this study was that increasing the number density of expansion sites (i.e., crack origins) contributed to maintaining a higher modulus in the macroscopic sample. This was due to the tendency of cracks to propagate to neighboring expansion sites, hence the crack length was shorter and did not percolate through the system resulting in smaller decrease of the stiffness, i.e., elastic modulus. Furthermore, increasing the number of cracked pixels per iteration results in crack networks made up of wider, on average, cracks. This is a consequence of the fact that pixels near the highest principal tensile stress pixels generally have high stresses as well because the stress field is continuous.
- A notable feature of this modeling approach is its ability to utilize real microstructure images, enabling the comparison of simulated crack network formations with those observed experimentally. Moreover, the model is currently being applied to a real-world case study involving the expansion of a concrete foundation due to pyrrhotite. The implementation includes a mesh sensitivity analysis, evaluation of convergence criteria, and consideration of other key mechanisms to assess the performance of the model.
- While this work provides a foundation for qualitatively exploring degradation mechanisms in composite materials, extending the model to three dimensions and incorporating additional degradation effects would be necessary to develop it into a more quantitative predictive tool.

References

[1] Z. Khaji and M. Fakoor, “Strain energy release rate in combination with reinforcement isotropic solid model (SERIS): A new mixed-mode I/II criterion to investigate fracture behavior of orthotropic materials,” *Theor. Appl. Fract. Mech.*, vol. 113, no. January, p. 102962, 2021, doi: 10.1016/j.tafmec.2021.102962.

[2] J. Kováčik, L. Marsavina, and E. Linul, “Poisson’s ratio of closed-cell aluminium foams,” *Materials (Basel).*, vol. 11, no. 10, pp. 1–11, 2018, doi: 10.3390/ma11101904.

[3] E. M. Craciun, “Energy criteria for crack propagation in pre-stressed elastic composites,” *Solid Mech. its Appl.*, vol. 154, pp. 193–237, 2008, doi: 10.1007/978-1-4020-8772-1_7.

[4] G. L. Golewski, “The Phenomenon of Cracking in Cement Concretes and Reinforced Concrete Structures: The Mechanism of Cracks Formation, Causes of Their Initiation, Types and Places of Occurrence, and Methods of Detection—A Review,” *Buildings*, vol. 13, no. 3. 2023. doi: 10.3390/buildings13030765.

[5] A. E. Idiart, C. M. López, and I. Carol, “Chemo-mechanical analysis of concrete cracking and degradation due to external sulfate attack: A meso-scale model,” *Cem. Concr. Compos.*, vol. 33, no. 3, pp. 411–423, 2011, doi: 10.1016/j.cemconcomp.2010.12.001.

[6] M. Alnaggar, G. Cusatis, and G. Di Luzio, “Lattice Discrete Particle Modeling (LDPM) of Alkali Silica Reaction (ASR) deterioration of concrete structures,” *Cem. Concr. Compos.*, vol. 41, pp. 45–59, 2013, doi: 10.1016/j.cemconcomp.2013.04.015.

[7] S. Multon and A. Sellier, “Expansion modelling based on cracking induced by the formation of new phases in concrete,” *Int. J. Solids Struct.*, vol. 160, pp. 293–306, 2019, doi: 10.1016/j.ijsolstr.2018.11.001.

[8] F. U. A. Shaikh, “Effect of Cracking on Corrosion of Steel in Concrete,” *Int. J. Concr. Struct. Mater.*, vol. 12, no. 1, 2018, doi: 10.1186/s40069-018-0234-y.

[9] H. W. Reinhardt and O. Mielich, “A fracture mechanics approach to the crack formation in alkali-sensitive grains,” *Cem. Concr. Res.*, vol. 41, no. 3, pp. 255–262, 2011, doi: 10.1016/j.cemconres.2010.11.008.

[10] H. Kagimoto, Y. Yasuda, and M. Kawamura, “ASR expansion, expansive pressure and cracking in concrete prisms under various degrees of restraint,” *Cem. Concr. Res.*, vol. 59, pp. 1–15, 2014, doi: 10.1016/j.cemconres.2014.01.018.

[11] A. Danish, M. A. Mosaberpanah, and M. U. Salim, “Past and present techniques of self-healing in cementitious materials: A critical review on efficiency of implemented treatments,” *J. Mater. Res. Technol.*, vol. 9, no. 3, pp. 6883–6899, 2020, doi: 10.1016/j.jmrt.2020.04.053.

[12] D. Gardner, R. Lark, T. Jefferson, and R. Davies, “A survey on problems encountered in current concrete construction and the potential benefits of self-healing cementitious materials,” *Case Stud. Constr. Mater.*, vol. 8, no. October 2017, pp. 238–247, 2018, doi: 10.1016/j.cscm.2018.02.002.

[13] W. Khaliq and M. B. Ehsan, “Crack healing in concrete using various bio influenced self-healing techniques,” *Constr. Build. Mater.*, vol. 102, pp. 349–357, 2016, doi: 10.1016/j.conbuildmat.2015.11.006.

[14] W. Puatatsananon and V. Saouma, “Chemo-mechanical micromodel for alkali-silica reaction,” *ACI Mater. J.*, vol. 110, no. 1, pp. 67–77, 2013, doi: 10.14359/51684367.

[15] Z. Suchorab, M. Franus, and D. Barnat-Hunek, “Properties of fibrous concrete made with plastic optical fibers from E-Waste,” *Materials (Basel).*, vol. 13, no. 10, pp. 1–25, 2020, doi: 10.3390/ma13102414.

[16] A. Abolhasani, H. Nazarpour, and M. Dehestani, “Effects of silicate impurities on fracture behavior and microstructure of calcium aluminate cement concrete,” *Eng. Fract. Mech.*, vol. 242, no. Decemeber 2020, p. 107446, 2021, doi: 10.1016/j.engfracmech.2020.107446.

[17] H. Li, H. Sun, W. Zhang, H. Gou, and Q. Yang, “Study on mechanical properties of self-compacting concrete and its filled in-line multi-cavity steel tube bundle shear wall,” *Energies*, vol. 12, no. 18, 2019, doi: 10.3390/en12183466.

[18] M. Szelag, “Evaluation of cracking patterns in cement composites-from basics to advances: A review,” *Materials*, vol. 13, no. 11. 2020. doi: 10.3390/ma13112490.

[19] A. Hillerborg, M. Modéer, and P. E. Petersson, “Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements,” *Cem. Concr. Res.*, vol. 6, no. 6, pp. 773–781, 1976, doi: 10.1016/0008-8846(76)90007-7.

[20] O. H. A. Dehwah, H. Hamidane, and Y. Xi, “Characterization of Effective Moisture Diffusivity Based on Pore Structure of Concrete, submitted to,” *Sci. Rep.*, vol. 14, no. 18450, 2024, doi: 10.1038/s41598-024-66300-w.

[21] O. H. A. Dehwah and Y. Xi, “Theoretical model for the coupling effect of moisture transport on chloride penetration in concrete,” *Cem. Concr. Res.*, vol. 177, no. January, 2024, doi: 10.1016/j.cemconres.2024.107431.

[22] G. S. Saini, O. Erge, P. Ashok, and E. van Oort, “Well Construction Action Planning and Automation through Finite-Horizon Sequential Decision-Making,” *Energies*, vol. 15, no. 16, 2022, doi: 10.3390/en15165776.

[23] M. Fakoor, R. Rafiee, and S. Zare, “Equivalent reinforcement isotropic model for fracture investigation of orthotropic materials,” *Steel Compos. Struct.*, vol. 30, no. 1, pp. 1–12, 2019, doi: 10.12989/scs.2019.30.1.001.

[24] R. Zhong, X. Ai, Y. Yao, J. Wang, and K. Wille, “Effects of the expansion mechanisms on the pyrrhotite-induced deterioration of concrete foundations,” *Case Stud. Constr. Mater.*, vol. 20, no. November 2023, p. e02830, 2024, doi: 10.1016/j.cscm.2023.e02830.

[25] M. Mengason and S. Watson, “Pyrrhotite reference material synthesis,” *NIST Tech. Note - NIST-TN.2336*, 2025, doi: https://doi.org/10.6028/NIST.TN.2336.

[26] M. Belghali, K. Ayed, M. Ezziane, and N. Leklou, “The influence of sulfate attack on existing concrete bridges: Study of external sulfate attack (ESA),” *Constr. Build. Mater.*, vol. 483, no. November 2024, 2025, doi: 10.1016/j.conbuildmat.2025.141751.

[27] W. Li, M. Deng, L. Mo, D. K. Panesar, and Z. Mao, “Alkali carbonate reaction (ACR): Investigations on mechanism of dedolomitization of dolomite in dolostones,” *Constr. Build. Mater.*, vol. 351, no. April, 2022, doi: 10.1016/j.conbuildmat.2022.128942.

[28] H. Zhou, X. Tian, and J. Wu, “Cracking and thermal resistance in concrete: Coupled thermo-mechanics and phase-field modeling,” *Theor. Appl. Fract. Mech.*, vol. 130, no. December 2023, p. 104285, 2024, doi: 10.1016/j.tafmec.2024.104285.

[29] D. Lv *et al.*, “Freeze–Thaw Damage Characteristics of Concrete Based on Compressive Mechanical Properties and Acoustic Parameters,” *Materials (Basel).*, vol. 17, no. 5, 2024, doi: 10.3390/ma17051010.

[30] S. T. Yang, K. F. Li, and C. Q. Li, “Numerical determination of concrete crack width for corrosion-affected concrete structures,” *Comput. Struct.*, vol. 207, pp. 75–82, 2018, doi: 10.1016/j.compstruc.2017.07.016.

[31] J. Roesler, G. H. Paulino, K. Park, and C. Gaedicke, “Concrete fracture prediction using bilinear softening,” *Cem. Concr. Compos.*, vol. 29, no. 4, pp. 300–312, Apr. 2007, doi: 10.1016/j.cemconcomp.2006.12.002.

[32] J. H. Hanson and A. R. Ingraffea, “Using numerical simulations to compare the fracture toughness values for concrete from the size-effect, two-parameter and fictitious crack models,” *Eng. Fract. Mech.*, vol. 70, no. 7–8, pp. 1015–1027, 2003, doi: 10.1016/S0013-7944(02)00163-7.

[33] Y. Theiner and G. Hofstetter, “Numerical prediction of crack propagation and crack widths in concrete structures,” *Eng. Struct.*, vol. 31, no. 8, pp. 1832–1840, 2009, doi: 10.1016/j.engstruct.2009.02.041.

[34] T. Qiong, H. F. Isleem, and H. R. Karimi, “Effect of length-to-height ratio on fracture properties of asymmetrical single-edge notched beam (ASENB) specimen made of ceramic under full range mixed mode I/II loading state,” *Theor. Appl. Fract. Mech.*, vol. 134, no. PA, p. 104711, 2024, doi: 10.1016/j.tafmec.2024.104711.

[35] V. Nguyen-Van, P. Tran, J. Liu, M. Van Tran, and Y. M. Xie, “Extended finite element multiscale modelling for crack propagation in 3D-printed fibre-reinforced concrete,” *Addit. Manuf.*, vol. 81, no. January, p. 104019, 2024, doi: 10.1016/j.addma.2024.104019.

[36] M. H. Aliabadi and A. L. Saleh, “Fracture mechanics analysis of cracking in plain and reinforced concrete using the boundary element method,” *Eng. Fract. Mech.*, vol. 69, pp. 267–280, 2002, doi: doi.org/10.1016/S0013-7944 (01)00089- 3.

[37] A. H. Chahrour and M. Ohtsu, “BEM Analysis of Crack Propagation in Concrete Based on Fracture Mechanics,” *Bound. Elem. Methods*, no. 4, pp. 59–66, 1992, doi: 10.1007/978-3-662-06153-4_7.

[38] B. Beckmann, K. Schicktanz, D. Reischl, and M. Curbach, “DEM simulation of concrete fracture and crack evolution,” *Struct. Concr.*, vol. 13, no. 4, pp. 213–220, 2012, doi: 10.1002/suco.201100036.

[39] E. A. B. Koenders, E. Schlangen, and K. Van Breugel, “Multi-Scale Modelling : the Delftcode,” *Int. RILEM Symp. Concr. Model. – CONMOD’08*, no. May, pp. 251–258, 2008.

[40] Z. Qian, E. Schlangen, G. Ye, and K. Van Breugel, “Multiscale lattice fracture model for cement-based materials,” *4th Int. Conf. Comput. Methods*, pp. 1–8, 2012.

[41] E. J. Garboczi, “Finite element and finite difference programs for computing the linear electric and elastic properties of digital images of random materials.,” *NIST Interagency/Internal Report (NISTIR) - 6269*. p. 198, 1998. [Online]. Available: https://www.nist.gov/publications/finite-element-and-finite-difference-programs-computing-linear-electric-and-elastic

[42] H. Zhang, B. Šavija, S. C. Figueiredo, and E. Schlangen, “Experimentally validated multi-scale modelling scheme of deformation and fracture of cement paste,” *Cem. Concr. Res.*, vol. 102, no. July, pp. 175–186, 2017, doi: 10.1016/j.cemconres.2017.09.011.

[43] A. S. Carey and S. S. Watson, “Review of Potential Mitigation Methodologies for Existing Concrete Structures Containing Pyrrhotite-Bearing Aggregates,” *J. Build. Eng.*, p. 102547, 2025, doi: 10.1016/j.jobe.2025.113408.

[44] P. E. Stutzman, D. S. Bright, and E. J. Garboczi, “Finite Element Stress Computations Applied to Images of Damaged Concrete: A Possible New Diagnostic Tool for Concrete Petrography,” in *Twenty-Third International Conference on Cement Microscopy*, 2001. [Online]. Available: http://fire.nist.gov/bfrlpubs/build01/PDF/b01025.pdf

[45] O. H.A. Dehwah, E. J. Garboczi, N. S. Martys, and S. S. Watson, “2D Finite Element Based Model of Quasistatic Crack Formation Driven by Driven by Expansion of Internal Phases in Multi-phase Materials,” National Institute of Standards and Technology (NIST), Gaithersburg, MD, Version 1.0.0, (Accessed: [2-18-2026]), 2025. doi: https://doi.org/10.18434/mds2-4002.

Appendix A. The Code

A. To start the code: for gifs from 00 to 99

- 'thermal2d-critical-stress-cracking.f' (the main FEM program for the crack simulation)

```
c                Copyright Notice
c
c This software was developed at the National Institute of Standards
c and Technology (NIST) by employees of the Federal Government in the
c course of their official duties. Pursuant to title 17 Section 105
c of the United States Code this software is not subject to copyright
c protection and is in the public domain.
c
c This is an experimental system.  NIST assumes no responsibility
c whatsoever for its use by other parties, and makes no guarantees,
c expressed or implied, about its quality, reliability, or any other
c characteristic.
c
c We would appreciate acknowledgment if the software is used.
c
c
c
c Contact: Osamah H. Dehwah  osamah.dehwah@nist.gov
c        NIST
c        Engineering Laboratory
c        Materials and Structural Systems Division
c        Infrastructure Materials Group



c This version scans each time and deletes pixels whose
c maximum principal strain exceeds a critical value, strcrit.
c After each round, program eliminates small isolated aggregate and mortar clusters
c Does not check expansive phase as this never cracks
c***** pmax(m,1)=max principal STRAIN, pmax(m,2)=m

c This version also puts some randomness into matrix, to see if that
c makes the cracks have more of a random direction rather than following
c the i-j directions.

c PERIODIC BOUNDARY CONDITIONS - no displacements held at constant
c  ************************  thermal2d.f  ************************
c  BACKGROUND
```

```
c  Program adjusts dimensions of unit cell,
c  [(1 + macrostrain) times dimension],
c  in response to phases that have a non-zero eigenstrain and
c  arbitrary elastic moduli tensors.
c  All three macrostrains can adjust their values (2-d program), and are
c  stored in the last two positions in the displacement vector u,
c  as listed below. Periodic boundaries are maintained.
c  In the comments below, (USER) means that this is a section of code
c  that the user might have to change for  his particular problem.
c  Therefore the user is encouraged to search for this string.

c  PROBLEM AND VARIABLE DEFINITION

c  The problem being solved is the minimization of the elastic energy
c  1/2 uAu + bu + C + Tu + Y, where b and C are also functions of the
c  macrostrains.
c  The small array zcon computes the thermal strain energy associated
c  with macrostrains (C term), T is the thermal energy term linear in the
c  displacements (built from ss), b is the regular energy term linear in the
c  displacements, u is the displacements including the macrostrains, gb
c  is the energy gradient vector, h,Ah are auxiliary vectors,
c  dk is the single pixel stiffness matrix, pix is the phase
c  identification vector (pix=1 for phase 1, etc.), and ib is the
c  integer matrix for mapping labels from the 1-9 nearest neighbor
c  labelling to the 1-d system labelling.
c  The array prob(i) contains the volume fractions of the i'th phase,
c  strxx, etc. are the three independent area averaged stresses
c  (Voigt notation), sxx, etc. are the three independent area
c  averaged strains (Voigt notation, not counting the thermal strains),
c  cmod(i,3,3) gives the elastic modulus tensor
c  of the i'th phase, and dk(i,4,2,4,2) is the stiffness matrix of the i'th
c  phase.  The parameter nphase gives the number of phases being considered
c  in the problem, and is set by the user.

c  DIMENSIONS

c  The main arrays of the problem, u, gb, h, Ah, b, and T, are dimensioned
c  as (nx*ny)+2, which is the number of nodal displacements plus two for
c  the macrostrains.
c  The program currently assumes that the number of different phases is
c  100, since phasemod, eigen, and ss (the moduli, eigenstrains, and
c  auxiliary eigenstrain variable for each phase)
c  and dk are dimensioned to have at most 100 different kinds.  This
c  is easily changed.  The parameter nphase gives the number of phases
```

```
c  expected in the problem, and is user-specified.  All major arrays
c  are passed to subroutines via simple common statements.

c  NOTE ON USE OF PROGRAM:  Program is set up to allow the macrostrains,
c  which control the overall size of the system, to be dynamic
c  variables, which are adjusted in order to minimize the overall
c  energy.  That means that if there are no eigenstrains specified
c  for any of the phases, the overall strain will always relax to
c  zero.  If it is desired to simply apply a fixed strain, with no
c  eigenstrains, then in subroutines Energy and Dembx, one must
c  zero out the elements of gb (in energy and in dembx) that
c  correspond to the macrostrains.  This is easily done.
c  This will fix the gradients of the macrostrains to always to be
c  zero, so that they will not change, so the applied strain (initial
c  values of the macrostrains) will remain fixed.

c  STRONGLY SUGGESTED:  READ MANUAL BEFORE USING PROGRAM!!!

c  (USER)  Change these dimensions and in other subroutines at same time.
c  For example, search and replace all occurrences throughout the
c  program of "(402" by "(1602", to go from a 20 x 20 system to
c  a 40 x 40 system.
c The u and similar arrays are dimensioned (nx times ny) + 2.

real u(546914,2),gb(546914,2),b(546914,2),pmax(546914,2)
real h(546914,2),Ah(546914,2),T(546914,2)
real C,dk(100,4,2,4,2)
real cmod(100,3,3),ss(100,4,2),eigen(100,3)
real zcon(2,2,2,2),pk(3,4,2)
real phasemod(100,2),prob(100)
integer in(9),jn(9),kn(9)
integer*4 ib(546914,9)
integer*2 pix(546914)

common/list1/strxx,stryy,strxy
common/list2/h,Ah
common/list3/ib
common/list4/pix
common/list5/dk,b,C,zcon,Y
common/list6/u
common/list7/gb
common/list8/cmod,T,eigen
common/list10/phasemod,nphase,ss
common/list11/sxx,syy,sxy
```

```
common/list12/pmax
common/list13/ncrack

c  (USER)  Unit 9 is the microstructure input file, unit 7
c  is the results output file.
c Unit 8 collects the macrostrains as a function of relaxation steps
c in case this is of interest
open (9,file='ab4.mic')
open (7,file='ab4-crack-data-critical-stress-006.out')

c (USER) nx and ny are the size of the lattice
c note that the dimension of the arrays is (nx times ny) +2
nx=844
ny=648
c       ns=total number of sites = nx * ny
ns=nx*ny
write(7,9010) nx,ny,ns
9010  format(' nx= ',i4,'  ny= ',i4,' ns = ',i8)
call srand(-189)

c  Add two more entries in the displacement vector for the 3 macrostrains,
c  u(ns+1,1) = exx, u(ns+1,2) = eyy, u(ns+2,1) = exy, u(ns+2,2) = never used
nss=ns+2

c  (USER) nphase is the number of phases being considered in the problem.
c  The values of pix(m) will run from 1 to nphase.
c  add extra zero phase for cracks = 4.
nphase=3+1+10

c (USER) value of critical strain (in this case stress) that determines if pixel is cracked
c       strcrit=0.009
strcrit=0.05
write(7,*) ' value of critical strain = ',strcrit
write(7,*)
call flush(7)
c  (USER) gtest is the stopping criterion, compared to gg=gb*gb.
c  If gtest=abc*ns, when gg < gtest, the rms value per pixel
c  of gb is less than sqrt(abc).
c  The value of gtest must be adjusted for each microstructure to
c  enable valid results.
gtest=1.e-12*(nx*ny)
write(7,*) 'relaxation criterion gtest = ',gtest

c  (USER)
```

```
c  The parameter phasemod(i,j) is the bulk (i,1)and shear (i,2) moduli of
c  the i'th phase. These can be
c  input in terms of Young's modulus E (i,1) and Poisson's ratio nu (i,2).
c  The program, in do 1144 loop, then changes them to bulk and shear
c  moduli, using relations for isotropic elastic moduli.
c  For anisotropic moduli tensors, one can directly input the whole tensor
c  cmod in subroutine femat, and skip this part.
c  If you wish to input in terms of bulk (i,1) and shear (i,2) moduli,
c  then simply comment out do 1144 loop.

c mortar (moduli are in GPa). Poisson's ratio is unitless.
phasemod(1,1)=50.0
phasemod(1,2)=0.25
c expansive phase (e.g. pyrrotite, aggregate)
phasemod(2,1)=80.
phasemod(2,2)=0.2
c regular aggregate
phasemod(3,1)=80.
phasemod(3,2)=0.2
c cracked material
phasemod(4,1)=0.0
phasemod(4,2)=0.2
do 178 i=5,14
phasemod(i,1)=50.0+(rand(0)-0.5)*16.0
phasemod(i,2)=0.25
178     continue

c convert E and nu to K and G
do 1144 i=1,nphase
save=phasemod(i,1)
phasemod(i,1)=phasemod(i,1)/2./(1.-phasemod(i,2))
phasemod(i,2)=save/2./(1.+phasemod(i,2))
1144    continue

open(777,file="phasemod_values")
do i=1,nphase
write(777,*)phasemod(i,1),phasemod(i,2)
end do
close(777)

c  (USER) input eigen (thermal) strains for each phase.
c  (1=xx, 2=yy, 3=xy).
c   A positive value means that the phase wants to expand, a negative
c   value means the phase wants to shrink.
```

```
eigen(1,1)=0.0
eigen(1,2)=0.0
eigen(1,3)=0.0
eigen(2,1)=0.01
eigen(2,2)=0.01
eigen(2,3)=0.0
eigen(3,1)=0.0
eigen(3,2)=0.0
eigen(3,3)=0.0
eigen(4,1)=0.0
eigen(4,2)=0.0
eigen(4,3)=0.0
do 47 i=5,14
eigen(i,1)=eigen(1,1)
eigen(i,2)=eigen(1,2)
eigen(i,3)=eigen(1,3)
47      continue

c  Construct the 9 neighbor table, ib(m,n)

c  First construct the 9 neighbor table in terms of delta i and delta j
c  information (see Table 3 in manual)
in(1)=0
in(2)=1
in(3)=1
in(4)=1
in(5)=0
in(6)=-1
in(7)=-1
in(8)=-1
in(9)=0

jn(1)=1
jn(2)=1
jn(3)=0
jn(4)=-1
jn(5)=-1
jn(6)=-1
jn(7)=0
jn(8)=1
jn(9)=0

c  Now construct neighbor table according to 1-d labels
c  Matrix ib(m,n) gives the 1-d label of the n'th neighbor (n=1,9) of
```

```
c  the node labelled m.
do 1020 j=1,ny
do 1020 i=1,nx
m=nx*(j-1)+i
do 1004 n=1,9
i1=i+in(n)
j1=j+jn(n)
if(i1.lt.1) i1=i1+nx
if(i1.gt.nx) i1=i1-nx
if(j1.lt.1) j1=j1+ny
if(j1.gt.ny) j1=j1-ny
m1=nx*(j1-1)+i1
ib(m,n)=m1
1004  continue
1020  continue

ioutnum=1000
c initialize pmax
do 5555 m=1,ns
pmax(m,1)=0.0
pmax(m,2)=m
5555  continue
c  Compute the average stress and strain, as well as the macrostrains (overall
c  system size and shape) in each microstructure.
c  (USER) npoints is the number of microstructures to use.
npoints=99

c  Read in an initial microstructure in subroutine ppixel, and set up pix(m)
c  with the appropriate phase assignments.
call ppixel(nx,ny,ns,nphase)
call assig(ns,nphase,prob)
do 8055 i=1,nphase
write(7,9065) i,prob(i)
8055 continue
c  output elastic moduli (bulk and shear) for each phase

c make image (00.pgm) of uncracked microstructure
call image(ns,0,nx,ny)
c ncc is the number of cracked pixels generated on one iteration of micro
c ncctot is the running total number of cracked pixels
ncc=0
ncctot=0
do 8000 micro=1,npoints
c this statement stops the fracture process if no new cracks have
```

```
c been identified on a given iteration
c do loop just iterates until micro reaches npoints then
c program finishes
if(ncc.eq.0.and.micro.gt.1) goto 8000
c  Count and output the area fractions of the different phases
call assig(ns,nphase,prob)
do 8050 i=1,nphase
write(7,9065) i,prob(i)
9065 format(' Area fraction of phase ',i3,'  is ',f10.8)
8050 continue
c  output elastic moduli (bulk and shear) for each phase
write(7,*) '  Phase Moduli'
do 111 i=1,nphase
write(7,9020) i,phasemod(i,1),phasemod(i,2)
9020 format(' Phase ',i3,' bulk = ',f12.6,' shear = ',f12.6)
111 continue
c  output thermal strains for each phase
write(7,*) '  Thermal Strains'
do 119 i=1,nphase
write(7,9029) i,eigen(i,1),eigen(i,2),eigen(i,3)
9029    format('Phase ',i3,'  ',3f6.2)
119 continue
call flush(7)

c  (USER) Set inital macrostrains of computational cell
u(ns+1,1)=0.0
u(ns+1,2)=0.0
u(nss,1)=0.0
u(nss,2)=0.0
c Apply homogeneous macroscopic strain as the initial condition
c to displacement variables
do 1050 j=1,ny
do 1050 i=1,nx
m=nx*(j-1)+i
x=float(i-1)
y=float(j-1)
u(m,1)=x*u(ns+1,1)+y*u(nss,1)
u(m,2)=x*u(nss,1)+y*u(ns+1,2)
1050 continue

c  Set up the finite element stiffness matrices,the constant, C,
c  the vector, b, required for the energy. b and C depend on the macrostrains.
c  When they are updated, the values of b and C are updated too via
c  calling subroutine femat.
```

```
c  Only compute the thermal strain terms the first time femat is called,
c  (iskip=0) as they are unaffected by later changes (iskip=1) in
c  displacements and macrostrains.
c  Compute initial value of gradient gb and gg=gb*gb.
iskip=0
call femat(nx,ny,ns,iskip)
call energy(nx,ny,ns,utot)
gg=0.0
do 100 m2=1,2
do 100 m=1,nss
gg=gg+gb(m,m2)*gb(m,m2)
100     continue
write(7,9042) utot,gg
9042 format(' energy = ',e15.8,'  gg=  ',e15.8)
call flush(7)

c  Relaxation loop
c  (USER) kmax is the maximum number of times that dembx will be called,
c  with ldemb conjugate gradient steps performed during each call.
c  The total number of conjugate gradient steps allowed for a given elastic
c  computation is kmax*ldemb.
kmax=400
ldemb=1000
ltot=0
ncrack=200
C       ncrack=10

do 5000 kkk=1,kmax

c  Call dembx to implement conjugate gradient routine
write(7,*) 'Going into dembx, call no. ',kkk
call dembx(nx,ny,ns,Lstep,gg,gtest,ldemb,kkk)
ltot=ltot+Lstep
c  Call energy to compute energy after dembx call. If gg < gtest, this
c  will be the final energy.  If gg is still larger than gtest, then this
c  will give an intermediate energy with which to check how the
c  relaxation process is coming along. The call to energy does not
c  change the gradient or the value of gg.
c  Need to first call femat to update the vector b, as the value of the
c  components of b depend on the macrostrains.
iskip=1
call femat(nx,ny,ns,iskip)
call energy(nx,ny,ns,utot)
write(7,9043) utot,gg,ltot
```

```
9043    format(' energy = ',e15.8,' gg=  ',e15.8,' ltot = ',i6)
call flush(7)

c  If relaxation process is finished, jump out of loop
if(gg.lt.gtest) goto 444
c  Output stresses, strains, and macrostrains as an additional aid in judging
c  how well the relaxation process is proceeding.
iswitch=0
call stress(nx,ny,ns,iswitch)
write(7,*) ' stresses:  xx,yy,xy'
write(7,*) strxx,stryy,strxy
write(7,*) ' strains:  xx,yy,xy'
write(7,*) sxx,syy,sxy

write(7,*) ' macrostrains in same order'
2745  format(2i8,3f15.8)
call flush(7)
call flush(8)
write(7,*) 'avg = ',(u(ns+1,1)+u(ns+1,2))/2.

5000    continue

444   iswitch=1
write(7,*) ' going into stress iswitch=1'
call flush(7)
write(7,*) u(ns+1,1),u(ns+1,2),u(ns+2,1)
call stress(nx,ny,ns,iswitch)
write(7,*) ' out of stress iswitch=1'
call flush(7)
write(7,*) ' stresses:  xx,yy,xy'
write(7,*) strxx,stryy,strxy
write(7,*) ' strains:  xx,yy,xy'
write(7,*) sxx,syy,sxy
call flush(7)

write(7,*) ' macrostrains in same order'
write(7,*) u(ns+1,1),u(ns+1,2),u(ns+2,1)
write(7,*) 'avg = ',(u(ns+1,1)+u(ns+1,2))/2.
call flush(7)
c      call order(ns)
c subroutine crack identifies pixels that exceed the
c critical strain = strcrit
call crack(ns,nx,ny,strcrit,micro,ncc)
ncctot=ncctot+ncc
```

```
write(7,*) ' running total # of cracked pixels = ',ncctot
c     write(7,*) ' done with order'
write(7,*) ' done with crack'
call flush(7)
call burn(nx,ny,ns)
write(7,*) ' done with burn'
call flush(7)

ioutnum=ioutnum+1
do 9000 j=1,ny
do 9900 i=1,nx
m=(j-1)*nx+i
write(ioutnum,*)pix(m)
9900  end do
9000  end do
close(ioutnum)

c  make image (00+.pgm) of cracked microstructure
call image(ns,micro,nx,ny)
write(7,*) ' done with image',micro
call flush(7)

bulk=(strxx+stryy)/(sxx+syy)/2.
shear=strxy/sxy
young=4.*bulk*shear/(bulk+shear)
poisson=(bulk-shear)/(bulk+shear)
write(7,*) ' bulk modulus = ',bulk
write(7,*) ' shear modulus = ',shear
write(7,*) ' Young modulus = ',young
write(7,*) ' Poisson ratio = ',poisson
8000    continue

end

c  Subroutine sets up the stiffness matrices, the linear term in the
c  regular displacements, b, and the constant term, C, which come from
c  the periodic boundary conditions, the term linear in the displacments,
c  T, that comes from the thermal strains, and the constant term Y.

subroutine femat(nx,ny,ns,iskip)
real u(546914,2),b(546914,2),T(546914,2)
real dk(100,4,2,4,2),phasemod(100,2),dndx(4),dndy(4)
real g(3,3),econ,ck(3,3),cmu(3,3),cmod(100,3,3)
real es(3,4,2),zcon(2,2,2,2),ss(100,4,2)
```

```
real eigen(100,3),delta(4,2)
integer is(4),iskip
integer*4 ib(546914,9)
integer*2 pix(546914)

common/list3/ib
common/list4/pix
common/list5/dk,b,C,zcon,Y
common/list6/u
common/list8/cmod,T,eigen
common/list10/phasemod,nphase,ss

nss=ns+2
c  Generate dk, zcon, T, and Y on first pass. After that they are
c  constant, since they are independent of the macrostrains.  Only b gets
c  upgraded as the macrostrains change.
c  Line number 1221 is the routine for b.
if(iskip.eq.1) goto 1221

c  initialize stiffness matrices
do 40 m=1,nphase
do 40 l=1,2
do 40 k=1,2
do 40 j=1,4
do 40 i=1,4
dk(m,i,k,j,l)=0.0
40    continue
c  initialize zcon matrix (gives C term for arbitrary macrostrains)
do 42 i=1,2
do 42 j=1,2
do 42 mi=1,2
do 42 mj=1,2
zcon(i,mi,j,mj)=0.0
42    continue
c  (USER) An anisotropic elastic moduli tensor could be input at this point,
c  bypassing the following code, which assumes isotropic elasticity
c  (only two independent numbers making up the elastic moduli
c  tensor, the bulk modulus K and the shear modulus G).

c  Set up elastic moduli matrices for each kind of element
c  ck and cmu are the bulk modulus and shear modulus matrices, which
c  need to multiplied by the actual bulk and shear moduli in each phase.

ck(1,1)=1.0
```

```
ck(1,2)=1.0
ck(1,3)=0.0
ck(2,1)=1.0
ck(2,2)=1.0
ck(2,3)=0.0
ck(3,1)=0.0
ck(3,2)=0.0
ck(3,3)=0.0

cmu(1,1)=1.0
cmu(1,2)=-1.0
cmu(1,3)=0.0
cmu(2,1)=-1.0
cmu(2,2)=1.0
cmu(2,3)=0.0
cmu(3,1)=0.0
cmu(3,2)=0.0
cmu(3,3)=1.0

do 31 k=1,nphase
do 21 j=1,3
do 21 i=1,3
cmod(k,i,j)=phasemod(k,1)*ck(i,j)+phasemod(k,2)*cmu(i,j)
21    continue
31    continue
c  Set up Simpson's integration rule weight vector
do 30 j=1,3
do 30 i=1,3
nm=0
if(i.eq.2) nm=nm+1
if(j.eq.2) nm=nm+1
g(i,j)=4.0**nm
30    continue

c  Loop over the nphase kinds of pixels and
c  Simpson's rule quadrature points in order to compute the stiffness
c  matrices.  Stiffness matrices of bilinear finite elements are quadratic
c  in x and y, so that Simpson's rule quadrature gives exact results.
do 4000 ijk=1,nphase
do 3000 j=1,3
do 3000 i=1,3
x=float(i-1)/2.0
y=float(j-1)/2.0
c  dndx means the negative derivative with respect to x, of the shape
```

```
c  matrix N (see manual, Sec. 2.2), dndy is similar.
dndx(1)=-(1.0-y)
dndx(2)=(1.0-y)
dndx(3)=y
dndx(4)=-y
dndy(1)=-(1.0-x)
dndy(2)=-x
dndy(3)=x
dndy(4)=(1.0-x)
c  now build strain matrix
do 2799 n1=1,3
do 2799 n2=1,4
do 2799 n3=1,2
es(n1,n2,n3)=0.0
2799  continue
do 2797 n=1,4
es(1,n,1)=dndx(n)
es(2,n,2)=dndy(n)
es(3,n,1)=dndy(n)
es(3,n,2)=dndx(n)
2797  continue
c  now do matrix multiply to determine value at (x,y), multiply by
c  proper weight, and sum into dk, the stiffness matrix
do 900 mm=1,2
do 900 nn=1,2
do 900 ii=1,4
do 900 jj=1,4
c  define sum over strain matrices and elastic moduli matrix for
c  stiffness matrix
sum=0.0
do 890 kk=1,3
do 890 ll=1,3
sum=sum+es(kk,ii,mm)*cmod(ijk,kk,ll)*es(ll,jj,nn)
890   continue
dk(ijk,ii,mm,jj,nn)=dk(ijk,ii,mm,jj,nn)+g(i,j)*sum/36.
900   continue
3000  continue
4000  continue

c  Now compute the ss matrices, which give the thermal strain terms
c  for the i'th phase, single pixel.

dndx(1)=-0.5
dndx(2)=0.5
```

```
dndx(3)=0.5
dndx(4)=-0.5
dndy(1)=-0.5
dndy(2)=-0.5
dndy(3)=0.5
dndy(4)=0.5
c  now build average strain matrix
do 3799 n1=1,3
do 3799 n2=1,4
do 3799 n3=1,2
es(n1,n2,n3)=0.0
3799  continue
do 3797 n=1,4
es(1,n,1)=dndx(n)
es(2,n,2)=dndy(n)
es(3,n,1)=dndy(n)
es(3,n,2)=dndx(n)
3797  continue
do 3598 mmm=1,nphase
do 3798 nn=1,2
do 3798 mm=1,4
sum=0.0
do 3698 nm=1,3
do 3698 n=1,3
sum=sum+cmod(mmm,n,nm)*es(n,mm,nn)*eigen(mmm,nm)
3698  continue
ss(mmm,mm,nn)=sum
3798  continue
3598  continue

c  now call subroutine const to generate zcon
c  zcon is a (2,2) x (2,2) matrix
call const(dk,ns,zcon,nx,ny)

c  Now set up linear term, T, for thermal energy. It does not depend
c  on the macrostrains or displacements, so there is no need to update it
c  as the macrostrains change. T is built up out of the ss matrices.

nss=ns+2
do 6066 m2=1,2
do 6066 m=1,nss
T(m,m2)=0.0
6066  continue
```

```
c  For all cases, the correspondence between 1-4 finite element node
c  labels and the 1-9 neighbor labels is (see Table 4 in manual):
c  1:ib(m,9), 2:ib(m,3),3:ib(m,2),4:ib(m,1)
is(1)=9
is(2)=3
is(3)=2
is(4)=1
c  Do all points, but no macrostrain terms
c  note:  factor of 2 on linear thermal term is cancelled
c  by factor of 1/2 out in front of total energy term
do 6601 j=1,ny
do 6601 i=1,nx
m=nx*(j-1)+i
do 6600 mm=1,4
do 6600 nn=1,2
T(ib(m,is(mm)),nn)=T(ib(m,is(mm)),nn)-ss(pix(m),mm,nn)
6600  continue
6601  continue

c  now need to pick up and sum in all terms multiplying macrostrains
do 7788 ipp=1,2
do 7788 jpp=1,2
if(ipp.eq.2.and.jpp.eq.2) goto 7788
exx=0.0
eyy=0.0
exy=0.0
if(ipp.eq.1.and.jpp.eq.1) exx=1.0
if(ipp.eq.1.and.jpp.eq.2) eyy=1.0
if(ipp.eq.2.and.jpp.eq.1) exy=1.0

c  x=nx face
do 6001 i2=1,2
do 6001 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.2.or.i4.eq.3) then
delta(i4,1)=exx*nx
delta(i4,2)=exy*nx
end if
6001  continue

do 6000 j=1,ny-1
m=j*nx
do 6900 nn=1,2
do 6900 mm=1,4
```

```
T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6900  continue
6000  continue

c  y=ny face
do 6011 i2=1,2
do 6011 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.3.or.i4.eq.4) then
delta(i4,1)=exy*ny
delta(i4,2)=eyy*ny
end if
6011  continue
do 6010 i=1,nx-1
m=nx*(ny-1)+i
do 6901 nn=1,2
do 6901 mm=1,4
T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6901  continue
6010  continue

c  x=nx y=ny corner
do 6031 i2=1,2
do 6031 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.2) then
delta(i4,1)=exx*nx
delta(i4,2)=exy*nx
end if
if(i4.eq.4) then
delta(i4,1)=exy*ny
delta(i4,2)=eyy*ny
end if
if(i4.eq.3) then
delta(i4,1)=exy*ny+exx*nx
delta(i4,2)=eyy*ny+exy*nx
end if
6031  continue
m=nx*ny
do 6903 nn=1,2
do 6903 mm=1,4
T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6903  continue
6030  continue
```

```
7788  continue

c  now compute Y, the 0.5(eigen)Cij(eigen) energy, doesn't ever change
c  with macrostrain or displacements
Y=0.0
do 8811 m=1,ns
do 8811 n=1,3
do 8811 nn=1,3
Y=Y+0.5*eigen(pix(m),n)*cmod(pix(m),n,nn)*eigen(pix(m),nn)
8811  continue

c  Following needs to be run after every change in macrostrain
c  when energy is recomputed.

1221  continue
c  Use auxiliary variables (exx, etc.) instead of u() variable, for
c  convenience, and to make the following code easier to read.
exx=u(ns+1,1)
eyy=u(ns+1,2)
exy=u(nss,1)
c  Now set up vector for linear term that comes from periodic boundary
c  conditions.  Notation and conventions same as for T term.
c  This is done using the stiffness matrices, and the periodic terms
c  in the macrostrains.  It is easier to set up b in this way than to
c  analytically write out all the terms involved.

do 5000 m2=1,2
do 5000 m=1,ns
b(m,m2)=0.0
5000  continue
c  For all cases, the correspondence between 1-4 finite element node
c  labels and the 1-9 neighbor labels is (see Table 4 in manual):
c  1:ib(m,9), 2:ib(m,3),3:ib(m,2),4:ib(m,1)
is(1)=9
is(2)=3
is(3)=2
is(4)=1

C=0.0
c  x=nx face
do 2001 i2=1,2
do 2001 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.2.or.i4.eq.3) then
```

```
delta(i4,1)=exx*nx
delta(i4,2)=exy*nx
end if
2001  continue

do 2000 j=1,ny-1
m=j*nx
do 1900 nn=1,2
do 1900 mm=1,4
sum=0.0
do 1899 m2=1,2
do 1899 m4=1,4
sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
1899  continue
b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1900  continue
2000  continue
c  y=ny face
do 2011 i2=1,2
do 2011 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.3.or.i4.eq.4) then
delta(i4,1)=exy*ny
delta(i4,2)=eyy*ny
end if
2011  continue
do 2010 i=1,nx-1
m=nx*(ny-1)+i
do 1901 nn=1,2
do 1901 mm=1,4
sum=0.0
do 2099 m2=1,2
do 2099 m4=1,4
sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2099  continue
b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1901  continue
2010  continue

c  x=nx y=ny corner
do 2031 i2=1,2
do 2031 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.2) then
```

```
delta(i4,1)=exx*nx
delta(i4,2)=exy*nx
end if
if(i4.eq.4) then
delta(i4,1)=exy*ny
delta(i4,2)=eyy*ny
end if
if(i4.eq.3) then
delta(i4,1)=exy*ny+exx*nx
delta(i4,2)=eyy*ny+exy*nx
end if
2031  continue
m=nx*ny
do 1903 nn=1,2
do 1903 mm=1,4
sum=0.0
do 2029 m2=1,2
do 2029 m4=1,4
sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2029  continue
b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1903  continue
2030  continue

return
end

c  Subroutine that computes derivatives of the b-vector with respect
c  to the macrostrains.  Since b is linear in the macrostrains, the
c  derivative with respect to any one of them can be computed simply
c  by letting that macrostrain, within the subroutine, be equal to one,
c  and all the other macrostrains to be zero.
c  Very similar to 1221 loop in femat for b.

subroutine bgrad(nx,ny,ns,exx,eyy,exy)
real b(546914,2)
real dk(100,4,2,4,2),delta(4,2),zcon(2,2,2,2)
integer is(4)
integer*4 ib(546914,9)
integer*2 pix(546914)

common/list3/ib
common/list4/pix
common/list5/dk,b,C,zcon,Y
```

```
c  exx, eyy, and exy are the artificial macrostrains used
c  to get the gradient terms (appropriate combinations of 1's and 0's).

c  Set up vector for linear term

do 5000 m2=1,2
do 5000 m=1,ns
b(m,m2)=0.0
5000  continue
is(1)=9
is(2)=3
is(3)=2
is(4)=1

c  x=nx face
do 2001 i2=1,2
do 2001 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.2.or.i4.eq.3) then
delta(i4,1)=exx*nx
delta(i4,2)=exy*nx
end if
2001  continue

do 2000 j=1,ny-1
m=j*nx
do 1900 nn=1,2
do 1900 mm=1,4
sum=0.0
do 1899 m2=1,2
do 1899 m4=1,4
sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
1899  continue
b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1900  continue
2000  continue
c  y=ny face
do 2011 i2=1,2
do 2011 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.3.or.i4.eq.4) then
delta(i4,1)=exy*ny
delta(i4,2)=eyy*ny
```

```
end if
2011  continue
do 2010 i=1,nx-1
m=nx*(ny-1)+i
do 1901 nn=1,2
do 1901 mm=1,4
sum=0.0
do 2099 m2=1,2
do 2099 m4=1,4
sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2099  continue
b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1901  continue
2010  continue

c  x=nx y=ny corner
do 2031 i2=1,2
do 2031 i4=1,4
delta(i4,i2)=0.0
if(i4.eq.2) then
delta(i4,1)=exx*nx
delta(i4,2)=exy*nx
end if
if(i4.eq.4) then
delta(i4,1)=exy*ny
delta(i4,2)=eyy*ny
end if
if(i4.eq.3) then
delta(i4,1)=exy*ny+exx*nx
delta(i4,2)=eyy*ny+exy*nx
end if
2031  continue
m=nx*ny
do 1903 nn=1,2
do 1903 mm=1,4
sum=0.0
do 2029 m2=1,2
do 2029 m4=1,4
sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2029  continue
b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1903  continue
2030  continue
```

```
return
end


c  Subroutine computes the quadratic term in the macrostrains, that comes
c  from the periodic boundary conditions, and sets it up as a
c  (2,2) x (2,2) matrix that couples to the three macrostrains

      subroutine const(dk,ns,zcon,nx,ny)
      real dk(100,4,2,4,2),zcon(2,2,2,2),delta(4,2)
      real pp(4,4),s(4,4)
      integer*2 pix(546914)
      common/list4/pix

c  routine to set up 4 x 4 matrix for energy term involving macro-strains
c  only, pulled out of femat (really 3 x 3, as ns+2,2 term is not used).
c  Idea is to evaluate the quadratic term in the macrostrains, C,
c  repeatedly for choices of strain like
c  exx=1, exy=1, all others = 0, build up 10 choices, then recombine
c  to get matrix elements by themselves

      nss=ns+2

      do 1111 i=1,4
      do 1111 j=1,4
      s(i,j)=0.0
      pp(i,j)=0.0
1111  continue

      do 5000 ii=1,4
      do 5000 jj=ii,4
      econ=0.0
      exx=0.0
      eyy=0.0
      exy=0.0
      if(ii.eq.1.and.jj.eq.1) exx=1.0
      if(ii.eq.2.and.jj.eq.2) eyy=1.0
      if(ii.eq.3.and.jj.eq.3) exy=1.0
      if(ii.eq.1.and.jj.eq.2) then
      exx=1.0
      eyy=1.0
      end if
      if(ii.eq.1.and.jj.eq.3) then
      exx=1.0
```

```
    exy=1.0
    end if
    if(ii.eq.2.and.jj.eq.3) then
    eyy=1.0
    exy=1.0
    end if
c  x=nx face
    do 2001 i2=1,2
    do 2001 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.2.or.i4.eq.3) then
    delta(i4,1)=exx*nx
    delta(i4,2)=exy*nx
    end if
2001  continue

    do 2000 j=1,ny-1
    m=j*nx
    do 1900 nn=1,2
    do 1900 mm=1,4
    do 1899 m2=1,2
    do 1899 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
1899  continue
1900  continue
2000  continue
c  y=ny face
    do 2011 i2=1,2
    do 2011 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.3.or.i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
2011  continue
    do 2010 i=1,nx-1
    m=nx*(ny-1)+i
    do 1901 nn=1,2
    do 1901 mm=1,4
    do 2099 m2=1,2
    do 2099 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
2099  continue
1901  continue
```

```
2010  continue

c  x=nx y=ny corner
     do 2031 i2=1,2
     do 2031 i4=1,4
     delta(i4,i2)=0.0
     if(i4.eq.2) then
     delta(i4,1)=exx*nx
     delta(i4,2)=exy*nx
     end if
     if(i4.eq.4) then
     delta(i4,1)=exy*ny
     delta(i4,2)=eyy*ny
     end if
     if(i4.eq.3) then
     delta(i4,1)=exy*ny+exx*nx
     delta(i4,2)=eyy*ny+exy*nx
     end if
2031  continue
     m=nx*ny
     do 1903 nn=1,2
     do 1903 mm=1,4
     do 2029 m2=1,2
     do 2029 m4=1,4
     econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
2029  continue
1903  continue
2030  continue
     pp(ii,jj)=econ*2.

5000  continue
     do 6000 i=1,4
     do 6000 j=i,4
     if(i.eq.j) s(i,j)=pp(i,j)
     if(i.ne.j) then
     s(i,j)=pp(i,j)-pp(i,i)-pp(j,j)
     end if
6000  continue
     do 7000 i=1,4
     do 7000 j=1,4
     pp(i,j)=0.5*(s(i,j)+s(j,i))
7000  continue
c  now map pp(i,j) into zcon(2,2,2,2)
     do 7200 i=1,2
```

```
      do 7200 j=1,2
      do 7200 mi=1,2
      do 7200 mj=1,2
      if(i.eq.1) ii=i+mi-1
      if(i.eq.2) ii=i+mi
      if(j.eq.1) jj=j+mj-1
      if(j.eq.2) jj=j+mj
      zcon(i,mi,j,mj)=pp(ii,jj)
      if(i.eq.2.and.mi.eq.2) zcon(i,mi,j,mj)=0.0
      if(j.eq.2.and.mj.eq.2) zcon(i,mi,j,mj)=0.0
7200  continue

      return
      end

c  Subroutine computes the total energy, utot, and the gradient, gb,
c  for the regular displacements as well as for the macrostrains

      subroutine energy(nx,ny,ns,utot)

          real u(546914,2),gb(546914,2)
          real b(546914,2),T(546914,2)
          real cmod(100,3,3),pk(3,4,2),eigen(100,3)
          real dk(100,4,2,4,2),zcon(2,2,2,2),C
          integer*4 ib(546914,9)
          integer*2 pix(546914)

          common/list3/ib
       common/list4/pix
          common/list5/dk,b,C,zcon,Y
          common/list6/u
          common/list7/gb
       common/list8/cmod,T,eigen

       nss=ns+2

       do 2090 m2=1,2
          do 2090 m=1,nss
          gb(m,m2)=0.0
2090    continue
c  Do global matrix multiply via small stiffness matrices, Ah = (A)(h).
c  The long statement below correctly brings in all the terms from
c  the global matrix A using only the small stiffness matrices.
```

```
      do 3000 j=1,2
      do 3000 n=1,2
         do 3000 m=1,ns
    gb(m,j)=gb(m,j)+u(ib(m,1),n)*( dk(pix(ib(m,9)),1,j,4,n)
   &+dk(pix(ib(m,7)),2,j,3,n) )+
   &u(ib(m,2),n)*( dk(pix(ib(m,9)),1,j,3,n) )+
   &u(ib(m,3),n)*( dk(pix(ib(m,9)),1,j,2,n)+dk(pix(ib(m,5)),4,j,3,n))+
   &u(ib(m,4),n)*( dk(pix(ib(m,5)),4,j,2,n) )+
   &u(ib(m,5),n)*( dk(pix(ib(m,6)),3,j,2,n)+dk(pix(ib(m,5)),4,j,1,n))+
   &u(ib(m,6),n)*( dk(pix(ib(m,6)),3,j,1,n) )+
   &u(ib(m,7),n)*( dk(pix(ib(m,6)),3,j,4,n)+dk(pix(ib(m,7)),2,j,1,n))+
   &u(ib(m,8),n)*( dk(pix(ib(m,7)),2,j,4,n) )+
   &u(ib(m,9),n)*( dk(pix(ib(m,9)),1,j,1,n)
   &+dk(pix(ib(m,7)),2,j,2,n)+
   &dk(pix(ib(m,6)),3,j,3,n)+dk(pix(ib(m,5)),4,j,4,n) )
3000    continue

    utot=0.0

    gtot=0.0
    do 3100 m2=1,2
    do 3100 m=1,ns
    utot=utot+0.5*u(m,m2)*gb(m,m2)+b(m,m2)*u(m,m2)
c  this is gradient of energy with respect to normal displacements
    gb(m,m2)=gb(m,m2)+b(m,m2)
3100  continue

c  compute "constant" macrostrain energy term
    C=0.0
    do 7200 i=1,2
    do 7200 j=1,2
    do 7200 mi=1,2
    do 7200 mj=1,2
    C=C+0.5*u(ns+i,mi)*zcon(i,mi,j,mj)*u(ns+j,mj)
7200  continue
    utot=utot+C
c  now add in constant term from thermal energy, Y
    utot=utot+Y
c  now add in linear term in thermal energy
    do 7171 m2=1,2
    do 7171 m=1,nss
    utot=utot+T(m,m2)*u(m,m2)
7171  continue
```

```
c  now compute gradient with respect to macrostrains
c  put in piece from first derivative of zcon quadratic term
     do 7300 i=1,2
     do 7300 mi=1,2
     sum=0.0
     do 7250 j=1,2
     do 7250 mj=1,2
     sum=sum+zcon(i,mi,j,mj)*u(ns+j,mj)
7250  continue
     gb(ns+i,mi)=sum
7300  continue
c  add in piece of gradient, for displacements as well as macrostrains,
c  that come from linear term in thermal energy
     do 3150 m2=1,2
     do 3150 m=1,nss
     gb(m,m2)=gb(m,m2)+T(m,m2)
3150  continue
c  now generate part that comes from b . u term
c  do by calling b generation with appropriate macrostrain set to 1 to
c  get that partial derivative, just use bgrad (taken from femat),
c  skip dk and zcon part
     do 8100 ii=1,3
     exx=0.0
     eyy=0.0
     exy=0.0
     if(ii.eq.1) exx=1.0
     if(ii.eq.2) eyy=1.0
     if(ii.eq.3) exy=1.0
     call bgrad(nx,ny,ns,exx,eyy,exy)
     sum=0.0
      do 8200 m2=1,2
         do 8200 m=1,ns
         sum=sum+u(m,m2)*b(m,m2)
8200    continue
         if(ii.eq.1) gb(ns+1,1)=gb(ns+1,1)+sum
         if(ii.eq.2) gb(ns+1,2)=gb(ns+1,2)+sum
         if(ii.eq.3) gb(ns+2,1)=gb(ns+2,1)+sum
8100    continue
      return
      end

c  Subroutine that carries out the conjugate gradient relaxation process

     subroutine dembx(nx,ny,ns,Lstep,gg,gtest,ldemb,kkk)
```

```
      real u(546914,2),gb(546914,2),b(546914,2)
      real h(546914,2),Ah(546914,2)
      real dk(100,4,2,4,2),zcon(2,2,2,2)
      real lambda,gamma
      integer*4 ib(546914,9)
      integer*2 pix(546914)

       common/list2/h,Ah
          common/list3/ib
       common/list4/pix
          common/list5/dk,b,C,zcon,Y
          common/list6/u
          common/list7/gb

      nss=ns+2
c  Initialize the conjugate direction vector on first call to dembx only.
c  For calls to dembx after the first, we want to continue using the value
c  of h determined in the previous call.  Of course, if npoints is greater
c  than 1, then this initialization step will be run each time a new
c  microstructure is used, as kkk will be reset to 1 every time the
c  counter micro is increased.
      if(kkk.eq.1) then
      do 500 m2=1,2
      do 500 m=1,nss
      h(m,m2)=gb(m,m2)
500   continue
      end if
c  Lstep counts the number of conjugate gradient steps taken
c  in each call to dembx
      Lstep=0

      do 800 ijk=1,ldemb
      Lstep=Lstep+1

      do 290 m2=1,2
      do 290 m=1,nss
      Ah(m,m2)=0.0
290   continue
c  Do global matrix multiply via small stiffness matrices, Ah = (A)(h).
c  The long statement below correctly brings in all the terms from
c  the global matrix A using only the small stiffness matrices.
       do 400 j=1,2
       do 400 n=1,2
```

```
          do 400 m=1,ns
     Ah(m,j)=Ah(m,j)+h(ib(m,1),n)*( dk(pix(ib(m,9)),1,j,4,n)
    &+dk(pix(ib(m,7)),2,j,3,n) )+
    &h(ib(m,2),n)*( dk(pix(ib(m,9)),1,j,3,n) )+
    &h(ib(m,3),n)*( dk(pix(ib(m,9)),1,j,2,n)+dk(pix(ib(m,5)),4,j,3,n))+
    &h(ib(m,4),n)*( dk(pix(ib(m,5)),4,j,2,n) )+
    &h(ib(m,5),n)*( dk(pix(ib(m,6)),3,j,2,n)+dk(pix(ib(m,5)),4,j,1,n))+
    &h(ib(m,6),n)*( dk(pix(ib(m,6)),3,j,1,n) )+
    &h(ib(m,7),n)*( dk(pix(ib(m,6)),3,j,4,n)+dk(pix(ib(m,7)),2,j,1,n))+
    &h(ib(m,8),n)*( dk(pix(ib(m,7)),2,j,4,n) )+
    &h(ib(m,9),n)*( dk(pix(ib(m,9)),1,j,1,n)
    &+dk(pix(ib(m,7)),2,j,2,n)+
    &dk(pix(ib(m,6)),3,j,3,n)+dk(pix(ib(m,5)),4,j,4,n) )
400    continue

c  The above accurately gives the second derivative matrix with respect
c  to nodal displacements, but fails to give the 2nd derivative terms that
c  include the macrostrains [ du d(strain) and d(strain)d(strain) ].
c  Use repeated calls to bgrad to generate mixed 2nd derivatives terms,
c  plus use zcon in order to correct the matrix multiply and correctly bring
c  in macrostrain terms (see manual, Sec. 2.4).
     do 8100 ii=1,3
     e11=0.0
     e22=0.0
     e12=0.0
     if(ii.eq.1) e11=1.0
     if(ii.eq.2) e22=1.0
     if(ii.eq.3) e12=1.0
     call bgrad(nx,ny,ns,e11,e22,e12)
c  now fill in terms from matrix multiply
c  right hand sides, 1 to ns
     do 3333 m=1,ns
     do 3333 m1=1,2
     if(ii.eq.1) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(ns+1,1)
     if(ii.eq.2) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(ns+1,2)
     if(ii.eq.3) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(nss,1)
3333  continue
c  now do across bottom, 1 to ns
     do 3334 m=1,ns
     if(ii.eq.1) Ah(ns+1,1)=Ah(ns+1,1)+b(m,1)*h(m,1)+
    +b(m,2)*h(m,2)
     if(ii.eq.2) Ah(ns+1,2)=Ah(ns+1,2)+b(m,1)*h(m,1)+
    +b(m,2)*h(m,2)
     if(ii.eq.3) Ah(nss,1)=Ah(nss,1)+b(m,1)*h(m,1)+
```

```
   +b(m,2)*h(m,2)
3334  continue
c  now do righthand corner terms, ns+1 to nss
    do 3335 m=1,2
    do 3335 m1=1,2
    if(ii.eq.1) Ah(ns+1,1)=Ah(ns+1,1)+zcon(1,1,m,m1)*h(ns+m,m1)
    if(ii.eq.2) Ah(ns+1,2)=Ah(ns+1,2)+zcon(1,2,m,m1)*h(ns+m,m1)
    if(ii.eq.3) Ah(nss,1)=Ah(nss,1)+zcon(2,1,m,m1)*h(ns+m,m1)
3335  continue

8100    continue

    hAh=0.0
    do 530 m2=1,2
    do 530 m=1,nss
    hAh=hAh+h(m,m2)*Ah(m,m2)
530   continue

    lambda=gg/hAh
    do 540 m2=1,2
    do 540 m=1,nss
    u(m,m2)=u(m,m2)-lambda*h(m,m2)
    gb(m,m2)=gb(m,m2)-lambda*Ah(m,m2)
540   continue

    gglast=gg
    gg=0.0
    do 550 m2=1,2
    do 550 m=1,nss
    gg=gg+gb(m,m2)*gb(m,m2)
550   continue
    if(gg.lt.gtest) goto 1000

    gamma=gg/gglast
    do 570 m2=1,2
    do 570 m=1,nss
    h(m,m2)=gb(m,m2)+gamma*h(m,m2)
570   continue

800   continue

1000  continue
    return
    end
```

```
c  Subroutine that computes the three average stresses
c  and three average strains.

      subroutine stress(nx,ny,ns,iswitch)

      real u(546914,2),uu(4,2),pmax(546914,2)
      real T(546914,2),eigen(100,3)
      real dndx(4),dndy(4),es(3,4,2),cmod(100,3,3)
      integer*4 ib(546914,9)
      integer*2 pix(546914)

      common/list1/strxx,stryy,strxy
      common/list3/ib
      common/list4/pix
      common/list6/u
      common/list8/cmod,T,eigen
      common/list11/sxx,syy,sxy
      common/list12/pmax

      nss=ns+2
      exx=u(ns+1,1)
      eyy=u(ns+1,2)
      exy=u(nss,1)
      if(iswitch.eq.1) then
c       open(unit=12,file='101bw-stress-strain-allmat.txt')
c       write(12,1134) nx,ny
c1134  format(2i5)
      end if

c set up single pixel strain matrix

      dndx(1)=-0.5
      dndx(2)=0.5
      dndx(3)=0.5
      dndx(4)=-0.5
      dndy(1)=-0.5
      dndy(2)=-0.5
      dndy(3)=0.5
      dndy(4)=0.5

c initialize pmax for iswitch=1
      if(iswitch.eq.1) then
      do 5555 m=1,ns
```

```
      pmax(m,1)=0.0
      pmax(m,2)=m
5555  continue
      end if
c  Build average strain matrix, follows code in femat, but for average
c  strain over a pixel, not the strain at a point
      do 2799 n1=1,3
      do 2799 n2=1,4
      do 2799 n3=1,2
      es(n1,n2,n3)=0.0
2799  continue
      do 2797 n=1,4
      es(1,n,1)=dndx(n)
      es(2,n,2)=dndy(n)
      es(3,n,1)=dndy(n)
      es(3,n,2)=dndx(n)
2797  continue
c  now compute average stresses and strains in each pixel
      sxx=0.0
      syy=0.0
      sxy=0.0
      strxx=0.0
      stryy=0.0
      strxy=0.0
      do 470 j=1,ny
      do 470 i=1,nx
      m=(j-1)*nx+i
      if(iswitch.eq.1) then
c       write(14,*) ' i j in stress m ',i,j,m
c       call flush(14)
      end if
c  load in elements of 4-vector using pd. bd. conds.
      do 9898 mm=1,2
      uu(1,mm)=u(m,mm)
      uu(2,mm)=u(ib(m,3),mm)
      uu(3,mm)=u(ib(m,2),mm)
      uu(4,mm)=u(ib(m,1),mm)
9898  continue
c  Correct for periodic boundary conditions, some displacements are wrong
c  for a pixel on a periodic boundary.  Since they come from an opposite
c  face, need to put in applied strain to correct them.
      if(i.eq.nx) then
      uu(2,1)=uu(2,1)+exx*nx
      uu(2,2)=uu(2,2)+exy*nx
```

```
      uu(3,1)=uu(3,1)+exx*nx
      uu(3,2)=uu(3,2)+exy*nx
      end if
      if(j.eq.ny) then
      uu(3,1)=uu(3,1)+exy*ny
      uu(3,2)=uu(3,2)+eyy*ny
      uu(4,1)=uu(4,1)+exy*ny
      uu(4,2)=uu(4,2)+eyy*ny
      end if
      str11=0.0
      str22=0.0
      str12=0.0
      s11=0.0
      s22=0.0
      s12=0.0
c********compute average stress and strain tensor in each pixel*************
c  First put thermal strain-induced stresses into stress tensor
      do 465 n=1,3
      str11=str11-cmod(pix(m),1,n)*eigen(pix(m),n)
      str22=str22-cmod(pix(m),2,n)*eigen(pix(m),n)
      str12=str12-cmod(pix(m),3,n)*eigen(pix(m),n)
465   continue
      do 466 n2=1,2
      do 466 n4=1,4
c  compute non-thermal strains in each pixel
      s11=s11+es(1,n4,n2)*uu(n4,n2)
      s22=s22+es(2,n4,n2)*uu(n4,n2)
      s12=s12+es(3,n4,n2)*uu(n4,n2)
      do 466 n=1,3
c  compute stresses in each pixel that include both non-thermal
c  and thermal strains
      str11=str11+cmod(pix(m),1,n)*es(n,n4,n2)*uu(n4,n2)
      str22=str22+cmod(pix(m),2,n)*es(n,n4,n2)*uu(n4,n2)
      str12=str12+cmod(pix(m),3,n)*es(n,n4,n2)*uu(n4,n2)
466   continue
c  Sum local stresses and strains into global stresses and strains
      strxx=strxx+str11
      stryy=stryy+str22
      strxy=strxy+str12
      sxx=sxx+s11
      syy=syy+s22
      sxy=sxy+s12
      if(iswitch.eq.1) then
      sa=0.5*((str11+str22)+sqrt((str11-str22)**2+4.*str12*str12))
```

```
      sb=0.5*((str11+str22)-sqrt((str11-str22)**2+4.*str12*str12))
      ea=0.5*((s11+s22)+sqrt((s11-s22)**2+4.*s12*s12))
      eb=0.5*((s11+s22)-sqrt((s11-s22)**2+4.*s12*s12))
c     if(ea.gt.eb) pmax(m,1)=ea
c     if(eb.gt.ea) pmax(m,1)=eb
      if(sa.gt.sb) pmax(m,1)=sa
      if(sb.gt.sa) pmax(m,1)=sb
      pmax(m,2)=m
c     write(14,*) m,pmax(m,1),pmax(m,2)
c     call flush(14)
c     write(12,1137) i,j,str11,str22,str12,s11,s22,s12
c     write(12,1138) i,j,sa,sb,ea,eb
1137  format(2i4,6f15.9)
1138  format(2i4,4f15.9)
      end if
470   continue

c  Area average global stresses and strains

            strxx=strxx/float(ns)
            stryy=stryy/float(ns)
            strxy=strxy/float(ns)
            sxx=sxx/float(ns)
            syy=syy/float(ns)
            sxy=sxy/float(ns)

      return
      end

c  Subroutine to count volume fractions of various phases

      subroutine assig(ns,nphase,prob)
      integer*2 pix(546914)
      real prob(100)
      common/list4/pix

            do 999 i=1,nphase
            prob(i)=0.0
999         continue

      do 1000 m=1,ns
      do 1000 i=1,nphase
        if(pix(m).eq.i) then
            prob(i)=prob(i)+1
```

```
          end if
1000    continue

          do 998 i=1,nphase
          prob(i)=prob(i)/float(ns)
998       continue

        return
        end

c  Subroutine to set up image of microstructure
c  adds border all around of mortar (phase 1)

      subroutine ppixel(nx,ny,ns,nphase)
      integer*2 pix(546914)
      common/list4/pix

      call srand(-147)
c  (USER)  If you want to set up a test image inside the program, instead
c  of reading it in from a file, this should be done inside this subroutine.

      do 200 j=1,ny
      do 200 i=1,nx
      m=nx*(j-1)+i
      read(9,*) pix(m)
c turn white border into black matrix
      if(pix(m).eq.4) pix(m)=1
200   continue
      close(9)

c make phases 5-14 matrix phases but with slightly different moduli

      do 300 j=1,ny
      do 300 i=1,nx
      m=nx*(j-1)+i
      if(pix(m).eq.1) then
      mm=5+10*rand(0)
      if(mm.eq.15) mm=14
      pix(m)=mm
      end if
300   continue
      close(19)

c  Check for wrong phase labels--less than 1 or greater than nphase
```

```
      do 500 m=1,ns
      if(pix(m).lt.1) then
       write(7,*) 'Phase label in pix < 1--error at ',m,pix(m)
      end if
      if(pix(m).gt.nphase) then
       write(7,*) 'Phase label in pix > nphase--error at ',m,pix(m)
      end if
500    continue

      return
      end
      subroutine crack(ns,nx,ny,strcrit,micro,ncc)
      real pmax(546914,2)
      real paux(546914,2)
      integer*2 pix(546914)
      common/list4/pix
      common/list12/pmax
      common/list13/ncrack

c Find where max principal strain exceeds strcrit.
c Then change the phase of these pixels to crack = 4

C First find the stresses or strains that exceed strcrit


      ncracks=0
      do 100 j=1,ny
      do 90 i=1,nx
      m=(j-1)*nx+i
      if(pix(m).eq.2.or.pix(m).eq.4)goto 90
      if(pmax(m,1).gt.strcrit)then
      ncracks=ncracks+1
      paux(ncracks,1)=pmax(m,1)
      paux(ncracks,2)=pmax(m,2)
      end if
90    continue
100   continue

      write(6,*)"ncracks=",ncracks

C Now sort those cracks in paux from max to min

      do 101 m=1,ncracks
      do 91 m1=m+1,ncracks
```

```
      if(paux(m1,1).gt.paux(m,1)) then
      xxx=paux(m,1)
      yyy=paux(m,2)
      paux(m,1)=paux(m1,1)
      paux(m,2)=paux(m1,2)
      paux(m1,1)=xxx
      paux(m1,2)=yyy
      end if
91    continue
101   continue

C Now switch phase to a crack for maxcracks or less  per cycle.

      maxcracks=400
      ncc=maxcracks

      if(ncracks.le.maxcracks)ncc=ncracks
      if(ncracks.gt.maxcracks)ncracks=maxcracks

      do 120 m=1,ncc
      m1=paux(m,2)
      pix(m1)=4
120   continue
      write(7,*) ncc,' pixels cracked and eliminated at ',micro,' step'

      return
      end

      subroutine order(ns)
      real pmax(546914,2)
      integer*2 pix(546914)
      common/list4/pix
      common/list12/pmax
      common/list13/ncrack

c order the first ten stresses to get the highest 10, remember
c where they occurred. Then change the phase of these 10 pixels
c to crack (4).

      do 100 m=1,ncrack
      do 90 m1=m+1,ns
      if(pmax(m1,1).gt.pmax(m,1)) then
      xxx=pmax(m,1)
      yyy=pmax(m,2)
```

```
      pmax(m,1)=pmax(m1,1)
      pmax(m,2)=pmax(m1,2)
      pmax(m1,1)=xxx
      pmax(m1,2)=yyy
      end if
90    continue
100   continue

c now zero out top ncrack tensile stress pixels
      do 200 m=1,ncrack
      mm=pmax(m,2)
      write(7,*) 'order ',m,mm
      pix(mm)=4
200   continue
      return
      end
      subroutine image(ns,micro,nx,ny)
      integer*2 pix(546914)
      common/list4/pix
      character*6 namef
      character*9 namefff
      character*31 nameff

      namef(1:2)='00'
      namef(3:6)='.pgm'
      if(micro.lt.10) write(namef(2:2),'(I1)') micro
      if(micro.ge.10) write(namef(1:2),'(I2)') micro
      open(unit=14,file=namef)
      write(14,133)
      write(14,134) nx,ny
      write(14,157)
133   format('P2')
134   format(i4,1x,i4)
157   format('4')

      do 741 j=ny,1,-1
      do 741 i=1,nx
      m=nx*(j-1)+i
      mm=pix(m)
c make irregular matrix all look the same in the image
      if(mm.ge.5.and.mm.le.14) mm=1
      write(14,22) mm
22    format(i1)
741   continue
```

```
      close(14)

c convert pgm file to gif

      nameff(1:8)='convert '
      nameff(9:14)=namef
      nameff(15:24)=' -depth 8 '
      nameff(25:30)=namef
      nameff(28:30)='gif'
      nameff(31:31)=' '
      call system(nameff)
c rm pgm files
      namefff(1:3)='rm '
      namefff(4:9)=namef
      call system(namefff)

      return
      end
      subroutine burn(nx,ny,ns)
      integer*2 pixc(nx,ny),pix(546914)
      integer*2 in(4),jn(4),old(10000000,2),new(10000000,2)
      integer*2 save(10000000,2)
      common/list4/pix

c transfer pix to pixc
      do 333 j=1,ny
      do 333 i=1,nx
      m=nx*(j-1)+i
      pixc(i,j)=pix(m)
c set random 5-14 matrix to plain mortar=1
      if(pix(m).gt.4) pixc(i,j)=1
333   continue

c then all particles stored in the particle image will be processed.
      icutoff=15

c  Direction labels to check for burning path (nearest neighbor information
c  in 3-D digital system).
      in(1)=-1
      in(2)=1
      in(3)=0
      in(4)=0
      jn(1)=0
      jn(2)=0
```

```
      jn(3)=-1
      jn(4)=1

c This next part of the program does the actual burning = cluster
c identification.

c can start with boundaries since we only want existing clusters
      do 1000 j=1,ny
      do 1000 i=1,nx

c don't choose any pore pixel or previously burned and identified particle
c pixel, which is still equal to the value of burned. or background pixel
c This assumes that matrix = 1 and aggregate = 3. Expansive = 2
c and negative means already burned. Cracks = 4.
      if(pixc(i,j).lt.0.or.pixc(i,j).eq.2.or.pixc(i,j).eq.4) goto 1000
      ipart=pixc(i,j)

      iold=0
      isave=0
c  find a first particle pixel
      if(pixc(i,j).eq.ipart) then
      pixc(i,j)=-pixc(i,j)
      iold=iold+1
      isave=isave+1
      old(iold,1)=i
      old(iold,2)=j
c save particle pixel coordinates
      save(iold,1)=i
      save(iold,2)=j
      end if

c  Now start building up new burned pixels from old set of burned pixels,
c  thus propagating the fire and identifying new particle pixels connnected
c  to the ones already burned.

60    inew=0
      do 100 ijk=1,iold
      ii=old(ijk,1)
      jj=old(ijk,2)
c  check all four nearest neighbors of previously burned pixel
      do 90 n=1,4
      i1=ii+in(n)
      j1=jj+jn(n)
```

```
c  Hard boundary conditions (can't pass beyond boundary)

      if(j1.lt.1.or.j1.gt.ny) then
      goto 90
      end if
      if(i1.lt.1.or.i1.gt.nx) then
      goto 90
      end if

c  Store (i,j) labels of newly burned pixels in array new().
      if(pixc(i1,j1).eq.ipart) then
      pixc(i1,j1)=-pixc(i1,j1)
      inew=inew+1
      new(inew,1)=i1
      new(inew,2)=j1
      end if
90    continue
100   continue

c  If new pixels were burned, then transfer labels to old() array, start
c  burning process over again.
      if(inew.gt.0) then
      iold=inew
      do 150 ijk=1,inew
      old(ijk,1)=new(ijk,1)
      old(ijk,2)=new(ijk,2)
      save(ijk+isave,1)=new(ijk,1)
      save(ijk+isave,2)=new(ijk,2)
150   continue
      isave=isave+inew
      goto 60
      end if

c  CHECK 1 - is particle too small?

c  If particle is not big enough, go to restore, wipe it out,
c  and try again.  icutoff is the minimum
c  number of pixels that a particle must have to be kept.
      if(isave.lt.icutoff) then
c then turn iold pixels into crack = 4. If greater than
c icutoff then leave negative and go find another cluster
c when all clusters are found, turn all negatives back into positives
       write(8,*) ' found small cluster to remove = ',isave
      call flush(8)
```

```
      do 225 m=1,isave
      iii=save(m,1)
      jjj=save(m,2)
      pixc(iii,jjj)=4
225   continue
      end if

1000  continue

c transfer pixc back to pix and set neg to pos
c preserve random matrix in pix(m)
      do 600 j=1,ny
      do 600 i=1,nx
      if(pixc(i,j).lt.0) pixc(i,j)=-pixc(i,j)
      m=nx*(j-1)+i
      if(pix(m).gt.4) then
        if(pixc(i,j).eq.4) then
        pix(m)=4
        goto 600
        end if
      goto 600
      end if
      pix(m)=pixc(i,j)
600   continue
      return
      end
```

B. To restart the code: For gifs from 99 to 198

- 'thermal2d-stress-cracking_restart.f' (the FEM program for restarting the crack simulation)

```
c                Copyright Notice
c
c This software was developed at the National Institute of Standards
c and Technology (NIST) by employees of the Federal Government in the
c course of their official duties. Pursuant to title 17 Section 105
c of the United States Code this software is not subject to copyright
c protection and is in the public domain.
c
c This is an experimental system.  NIST assumes no responsibility
c whatsoever for its use by other parties, and makes no guarantees,
c expressed or implied, about its quality, reliability, or any other
c characteristic.
c
c We would appreciate acknowledgment if the software is used.
c
c
c
c Contact: Osamah H. Dehwah  osamah.dehwah@nist.gov
c        NIST
c        Engineering Laboratory
c        Materials and Structural Systems Division
c        Infrastructure Materials Group

c This restart file is similar to "thermal2d-critical-stress-cracking.f" file
c however, the input phases parameters are taken from "phasemod" file,
c and no need to define it here again.
c This version scans each time and deletes pixels whose
c maximum principal strain exceeds a critical value, strcrit.
c After each round, program eliminates small isolated aggregate and mortar clusters
c Does not check expansive phase as this never cracks
c***** pmax(m,1)=max principal STRAIN, pmax(m,2)=m

c This version also puts some randomness into matrix, to see if that
c makes the cracks have more of a random direction rather than following
c the i-j directions.

c PERIODIC BOUNDARY CONDITIONS - no displacements held at constant
c  ************************  thermal2d.f  ************************
c  BACKGROUND
```

```
c  Program adjusts dimensions of unit cell,
c  [(1 + macrostrain) times dimension],
c  in response to phases that have a non-zero eigenstrain and
c  arbitrary elastic moduli tensors.
c  All three macrostrains can adjust their values (2-d program), and are
c  stored in the last two positions in the displacement vector u,
c  as listed below. Periodic boundaries are maintained.
c  In the comments below, (USER) means that this is a section of code
c  that the user might have to change for  his particular problem.
c  Therefore the user is encouraged to search for this string.

c  PROBLEM AND VARIABLE DEFINITION

c  The problem being solved is the minimization of the elastic energy
c  1/2 uAu + bu + C + Tu + Y, where b and C are also functions of the
c  macrostrains.
c  The small array zcon computes the thermal strain energy associated
c  with macrostrains (C term), T is the thermal energy term linear in the
c  displacements (built from ss), b is the regular energy term linear in the
c  displacements, u is the displacements including the macrostrains, gb
c  is the energy gradient vector, h,Ah are auxiliary vectors,
c  dk is the single pixel stiffness matrix, pix is the phase
c  identification vector (pix=1 for phase 1, etc.), and ib is the
c  integer matrix for mapping labels from the 1-9 nearest neighbor
c  labelling to the 1-d system labelling.
c  The array prob(i) contains the volume fractions of the i'th phase,
c  strxx, etc. are the three independent area averaged stresses
c  (Voigt notation), sxx, etc. are the three independent area
c  averaged strains (Voigt notation, not counting the thermal strains),
c  cmod(i,3,3) gives the elastic modulus tensor
c  of the i'th phase, and dk(i,4,2,4,2) is the stiffness matrix of the i'th
c  phase.  The parameter nphase gives the number of phases being considered
c  in the problem, and is set by the user.

c  DIMENSIONS

c  The main arrays of the problem, u, gb, h, Ah, b, and T, are dimensioned
c  as (nx*ny)+2, which is the number of nodal displacements plus two for
c  the macrostrains.
c  The program currently assumes that the number of different phases is
c  100, since phasemod, eigen, and ss (the moduli, eigenstrains, and
c  auxiliary eigenstrain variable for each phase)
c  and dk are dimensioned to have at most 100 different kinds.  This
c  is easily changed.  The parameter nphase gives the number of phases
```

```
c  expected in the problem, and is user-specified.  All major arrays
c  are passed to subroutines via simple common statements.

c  NOTE ON USE OF PROGRAM:  Program is set up to allow the macrostrains,
c  which control the overall size of the system, to be dynamic
c  variables, which are adjusted in order to minimize the overall
c  energy.  That means that if there are no eigenstrains specified
c  for any of the phases, the overall strain will always relax to
c  zero.  If it is desired to simply apply a fixed strain, with no
c  eigenstrains, then in subroutines Energy and Dembx, one must
c  zero out the elements of gb (in energy and in dembx) that
c  correspond to the macrostrains.  This is easily done.
c  This will fix the gradients of the macrostrains to always to be
c  zero, so that they will not change, so the applied strain (initial
c  values of the macrostrains) will remain fixed.

c  STRONGLY SUGGESTED:  READ MANUAL BEFORE USING PROGRAM!!!

c  (USER)  Change these dimensions and in other subroutines at same time.
c  For example, search and replace all occurrences throughout the
c  program of "(402" by "(1602", to go from a 20 x 20 system to
c  a 40 x 40 system.
c The u and similar arrays are dimensioned (nx times ny) + 2.

      real u(546914,2),gb(546914,2),b(546914,2),pmax(546914,2)
      real h(546914,2),Ah(546914,2),T(546914,2)
    real C,dk(100,4,2,4,2)
      real cmod(100,3,3),ss(100,4,2),eigen(100,3)
    real zcon(2,2,2,2),pk(3,4,2)
    real phasemod(100,2),prob(100)
    integer in(9),jn(9),kn(9)
    integer*4 ib(546914,9)
    integer*2 pix(546914)

      common/list1/strxx,stryy,strxy
      common/list2/h,Ah
    common/list3/ib
      common/list4/pix
    common/list5/dk,b,C,zcon,Y
    common/list6/u
    common/list7/gb
      common/list8/cmod,T,eigen
      common/list10/phasemod,nphase,ss
      common/list11/sxx,syy,sxy
```

```
      common/list12/pmax
      common/list13/ncrack

c  (USER)  Unit 9 is the microstructure input file, unit 7
c  is the results output file.
c Unit 8 collects the macrostrains as a function of relaxation steps
c in case this is of interest
c   open (9,file='ab4.mic')
    open (7,file='ab4-crack-data-critical-stress-006.out')

c (USER) nx and ny are the size of the lattice
c note that the dimension of the arrays is (nx times ny) +2
      nx=844
      ny=648
c      ns=total number of sites = nx * ny
      ns=nx*ny
      write(7,9010) nx,ny,ns
9010  format(' nx= ',i4,'  ny= ',i4,' ns = ',i8)
      call srand(-189)

c  Add two more entries in the displacement vector for the 3 macrostrains,
c  u(ns+1,1) = exx, u(ns+1,2) = eyy, u(ns+2,1) = exy, u(ns+2,2) = never used
      nss=ns+2

c  (USER) nphase is the number of phases being considered in the problem.
c  The values of pix(m) will run from 1 to nphase.
c  add extra zero phase for cracks = 4.
    nphase=3+1+10

c (USER) value of critical strain (in this case stress) that determines if pixel is cracked
c       strcrit=0.009
      strcrit=0.05
      write(7,*) ' value of critical strain = ',strcrit
      write(7,*)
      call flush(7)
c  (USER) gtest is the stopping criterion, compared to gg=gb*gb.
c  If gtest=abc*ns, when gg < gtest, the rms value per pixel
c  of gb is less than sqrt(abc).
c  The value of gtest must be adjusted for each microstructure to
c  enable valid results.
      gtest=1.e-12*(nx*ny)
      write(7,*) 'relaxation criterion gtest = ',gtest

c  (USER)
```

```
c  The parameter phasemod(i,j) is the bulk (i,1)and shear (i,2) moduli of
c  the i'th phase. These can be
c  input in terms of Young's modulus E (i,1) and Poisson's ratio nu (i,2).
c  The program, in do 1144 loop, then changes them to bulk and shear
c  moduli, using relations for isotropic elastic moduli.
c  For anisotropic moduli tensors, one can directly input the whole tensor
c  cmod in subroutine femat, and skip this part.
c  If you wish to input in terms of bulk (i,1) and shear (i,2) moduli,
c  then simply comment out do 1144 loop.

Cc mortar (moduli are in GPa). Poisson's ratio is unitless.
C   phasemod(1,1)=50.0
C   phasemod(1,2)=0.25
Cc chert (expansive aggregate)
C   phasemod(2,1)=80.
C   phasemod(2,2)=0.2
Cc regular aggregate
C   phasemod(3,1)=80.
C       phasemod(3,2)=0.2
Cc cracked material
C   phasemod(4,1)=0.0
C       phasemod(4,2)=0.2
C       do 178 i=5,14
C   phasemod(i,1)=50.0+(rand(0)-0.5)*16.0
C   phasemod(i,2)=0.25
C178     continue

Cc convert E and nu to K and G
C       do 1144 i=1,nphase
C       save=phasemod(i,1)
C       phasemod(i,1)=phasemod(i,1)/2./(1.-phasemod(i,2))
C       phasemod(i,2)=save/2./(1.+phasemod(i,2))
C1144    continue

C     open(777,file="phasemod_values")
C       do i=1,nphase
C       write(777,*)phasemod(i,1),phasemod(i,2)
C       end do
C     close(777)

      open(777,file="phasemod_values")
       do i=1,nphase
       read(777,*)phasemod(i,1),phasemod(i,2)
       end do
```

```
      close(777)

c  (USER) input eigen (thermal) strains for each phase.
c  (1=xx, 2=yy, 3=xy).
c   A positive value means that the phase wants to expand, a negative
c   value means the phase wants to shrink.
        eigen(1,1)=0.0
        eigen(1,2)=0.0
        eigen(1,3)=0.0
        eigen(2,1)=0.01
        eigen(2,2)=0.01
        eigen(2,3)=0.0
        eigen(3,1)=0.0
        eigen(3,2)=0.0
        eigen(3,3)=0.0
        eigen(4,1)=0.0
        eigen(4,2)=0.0
        eigen(4,3)=0.0
        do 47 i=5,14
        eigen(i,1)=eigen(1,1)
        eigen(i,2)=eigen(1,2)
        eigen(i,3)=eigen(1,3)
47      continue

c  Construct the 9 neighbor table, ib(m,n)

c  First construct the 9 neighbor table in terms of delta i and delta j
c  information (see Table 3 in manual)
      in(1)=0
      in(2)=1
      in(3)=1
      in(4)=1
      in(5)=0
      in(6)=-1
      in(7)=-1
      in(8)=-1
      in(9)=0

      jn(1)=1
      jn(2)=1
      jn(3)=0
      jn(4)=-1
      jn(5)=-1
      jn(6)=-1
```

```
      jn(7)=0
      jn(8)=1
      jn(9)=0

c  Now construct neighbor table according to 1-d labels
c  Matrix ib(m,n) gives the 1-d label of the n'th neighbor (n=1,9) of
c  the node labelled m.
      do 1020 j=1,ny
      do 1020 i=1,nx
      m=nx*(j-1)+i
      do 1004 n=1,9
      i1=i+in(n)
      j1=j+jn(n)
      if(i1.lt.1) i1=i1+nx
      if(i1.gt.nx) i1=i1-nx
      if(j1.lt.1) j1=j1+ny
      if(j1.gt.ny) j1=j1-ny
      m1=nx*(j1-1)+i1
      ib(m,n)=m1
1004  continue
1020  continue

      ioutnum=1000
c initialize pmax
      do 5555 m=1,ns
      pmax(m,1)=0.0
      pmax(m,2)=m
5555  continue
c  Compute the average stress and strain, as well as the macrostrains (overall
c  system size and shape) in each microstructure.
c  (USER) npoints is the number of microstructures to use.
       npoints=99

c  Read in an initial microstructure in subroutine ppixel, and set up pix(m)
c  with the appropriate phase assignments.
       call ppixel(nx,ny,ns,nphase)
       call assig(ns,nphase,prob)
     do 8055 i=1,nphase
     write(7,9065) i,prob(i)
8055
     continue
c  output elastic moduli (bulk and shear) for each phase

c make image (00.pgm) of uncracked microstructure
```

```
      call image(ns,0,nx,ny)
c ncc is the number of cracked pixels generated on one iteration of micro
c ncctot is the running total number of cracked pixels
      ncc=0
      ncctot=0
      do 8000 micro=1,npoints
c this statement stops the fracture process if no new cracks have
c been identified on a given iteration
c do loop just iterates until micro reaches npoints then
c program finishes
      if(ncc.eq.0.and.micro.gt.1) goto 8000
c  Count and output the area fractions of the different phases
      call assig(ns,nphase,prob)
    do 8050 i=1,nphase
    write(7,9065) i,prob(i)
9065
    format(' Area fraction of phase ',i3,'  is ',f10.8)
8050
    continue
c  output elastic moduli (bulk and shear) for each phase
      write(7,*) '  Phase Moduli'
      do 111 i=1,nphase
      write(7,9020) i,phasemod(i,1),phasemod(i,2)
9020
    format(' Phase ',i3,' bulk = ',f12.6,' shear = ',f12.6)
111
    continue
c  output thermal strains for each phase
      write(7,*) '  Thermal Strains'
      do 119 i=1,nphase
      write(7,9029) i,eigen(i,1),eigen(i,2),eigen(i,3)
9029    format('Phase ',i3,'  ',3f6.2)
119
    continue
      call flush(7)

c  (USER) Set inital macrostrains of computational cell
     u(ns+1,1)=0.0
     u(ns+1,2)=0.0
     u(nss,1)=0.0
     u(nss,2)=0.0
c Apply homogeneous macroscopic strain as the initial condition
c to displacement variables
    do 1050 j=1,ny
```

```
      do 1050 i=1,nx

    m=nx*(j-1)+i

    x=float(i-1)

    y=float(j-1)
            u(m,1)=x*u(ns+1,1)+y*u(nss,1)
            u(m,2)=x*u(nss,1)+y*u(ns+1,2)
1050
    continue

c  Set up the finite element stiffness matrices,the constant, C,
c  the vector, b, required for the energy. b and C depend on the macrostrains.
c  When they are updated, the values of b and C are updated too via
c  calling subroutine femat.
c  Only compute the thermal strain terms the first time femat is called,
c  (iskip=0) as they are unaffected by later changes (iskip=1) in
c  displacements and macrostrains.
c  Compute initial value of gradient gb and gg=gb*gb.
      iskip=0
    call femat(nx,ny,ns,iskip)
      call energy(nx,ny,ns,utot)
    gg=0.0
      do 100 m2=1,2
      do 100 m=1,nss
      gg=gg+gb(m,m2)*gb(m,m2)
100    continue
    write(7,9042) utot,gg
9042
    format(' energy = ',e15.8,'  gg=  ',e15.8)
      call flush(7)

c  Relaxation loop
c  (USER) kmax is the maximum number of times that dembx will be called,
c  with ldemb conjugate gradient steps performed during each call.
c  The total number of conjugate gradient steps allowed for a given elastic
c  computation is kmax*ldemb.
      kmax=400
      ldemb=1000
      ltot=0
      ncrack=200
C       ncrack=10
```

```
      do 5000 kkk=1,kmax

c  Call dembx to implement conjugate gradient routine
      write(7,*) 'Going into dembx, call no. ',kkk
      call dembx(nx,ny,ns,Lstep,gg,gtest,ldemb,kkk)
      ltot=ltot+Lstep
c  Call energy to compute energy after dembx call. If gg < gtest, this
c  will be the final energy.  If gg is still larger than gtest, then this
c  will give an intermediate energy with which to check how the
c  relaxation process is coming along. The call to energy does not
c  change the gradient or the value of gg.
c  Need to first call femat to update the vector b, as the value of the
c  components of b depend on the macrostrains.
      iskip=1
    call femat(nx,ny,ns,iskip)
      call energy(nx,ny,ns,utot)
    write(7,9043) utot,gg,ltot
9043    format(' energy = ',e15.8,' gg=  ',e15.8,' ltot = ',i6)
      call flush(7)

c  If relaxation process is finished, jump out of loop
      if(gg.lt.gtest) goto 444
c  Output stresses, strains, and macrostrains as an additional aid in judging
c  how well the relaxation process is proceeding.
     iswitch=0
     call stress(nx,ny,ns,iswitch)
     write(7,*) ' stresses:  xx,yy,xy'
     write(7,*) strxx,stryy,strxy
     write(7,*) ' strains:  xx,yy,xy'
     write(7,*) sxx,syy,sxy

     write(7,*) ' macrostrains in same order'
2745  format(2i8,3f15.8)
     call flush(7)
     call flush(8)
     write(7,*) 'avg = ',(u(ns+1,1)+u(ns+1,2))/2.

5000    continue

444   iswitch=1
     write(7,*) ' going into stress iswitch=1'
     call flush(7)
     write(7,*) u(ns+1,1),u(ns+1,2),u(ns+2,1)
     call stress(nx,ny,ns,iswitch)
```

```
      write(7,*) ' out of stress iswitch=1'
      call flush(7)
      write(7,*) ' stresses:  xx,yy,xy'
      write(7,*) strxx,stryy,strxy
      write(7,*) ' strains:  xx,yy,xy'
      write(7,*) sxx,syy,sxy
      call flush(7)

      write(7,*) ' macrostrains in same order'
      write(7,*) u(ns+1,1),u(ns+1,2),u(ns+2,1)
      write(7,*) 'avg = ',(u(ns+1,1)+u(ns+1,2))/2.
      call flush(7)
c     call order(ns)
c subroutine crack identifies pixels that exceed the
c critical strain = strcrit
      call crack(ns,nx,ny,strcrit,micro,ncc)
      ncctot=ncctot+ncc
      write(7,*) ' running total # of cracked pixels = ',ncctot
c     write(7,*) ' done with order'
      write(7,*) ' done with crack'
      call flush(7)
      call burn(nx,ny,ns)
      write(7,*) ' done with burn'
      call flush(7)

      ioutnum=ioutnum+1
      do 9000 j=1,ny
      do 9900 i=1,nx
        m=(j-1)*nx+i
        write(ioutnum,*)pix(m)
9900  end do
9000  end do
      close(ioutnum)

c  make image (00+.pgm) of cracked microstructure
      call image(ns,micro,nx,ny)
      write(7,*) ' done with image',micro
      call flush(7)

        bulk=(strxx+stryy)/(sxx+syy)/2.
        shear=strxy/sxy
        young=4.*bulk*shear/(bulk+shear)
        poisson=(bulk-shear)/(bulk+shear)
        write(7,*) ' bulk modulus = ',bulk
```

```
      write(7,*) ' shear modulus = ',shear
      write(7,*) ' Young modulus = ',young
      write(7,*) ' Poisson ratio = ',poisson
8000    continue

      end

c  Subroutine sets up the stiffness matrices, the linear term in the
c  regular displacements, b, and the constant term, C, which come from
c  the periodic boundary conditions, the term linear in the displacments,
c  T, that comes from the thermal strains, and the constant term Y.

     subroutine femat(nx,ny,ns,iskip)
     real u(546914,2),b(546914,2),T(546914,2)
     real dk(100,4,2,4,2),phasemod(100,2),dndx(4),dndy(4)
     real g(3,3),econ,ck(3,3),cmu(3,3),cmod(100,3,3)
     real es(3,4,2),zcon(2,2,2,2),ss(100,4,2)
     real eigen(100,3),delta(4,2)
     integer is(4),iskip
     integer*4 ib(546914,9)
     integer*2 pix(546914)

    common/list3/ib
      common/list4/pix
    common/list5/dk,b,C,zcon,Y
    common/list6/u
      common/list8/cmod,T,eigen
      common/list10/phasemod,nphase,ss

     nss=ns+2
c  Generate dk, zcon, T, and Y on first pass. After that they are
c  constant, since they are independent of the macrostrains.  Only b gets
c  upgraded as the macrostrains change.
c  Line number 1221 is the routine for b.
     if(iskip.eq.1) goto 1221

c  initialize stiffness matrices
     do 40 m=1,nphase
     do 40 l=1,2
     do 40 k=1,2
     do 40 j=1,4
     do 40 i=1,4
     dk(m,i,k,j,l)=0.0
40    continue
```

```
c  initialize zcon matrix (gives C term for arbitrary macrostrains)
      do 42 i=1,2
      do 42 j=1,2
      do 42 mi=1,2
      do 42 mj=1,2
      zcon(i,mi,j,mj)=0.0
42    continue
c  (USER) An anisotropic elastic moduli tensor could be input at this point,
c  bypassing the following code, which assumes isotropic elasticity
c  (only two independent numbers making up the elastic moduli
c  tensor, the bulk modulus K and the shear modulus G).

c  Set up elastic moduli matrices for each kind of element
c  ck and cmu are the bulk modulus and shear modulus matrices, which
c  need to multiplied by the actual bulk and shear moduli in each phase.

      ck(1,1)=1.0
      ck(1,2)=1.0
      ck(1,3)=0.0
      ck(2,1)=1.0
      ck(2,2)=1.0
      ck(2,3)=0.0
      ck(3,1)=0.0
      ck(3,2)=0.0
      ck(3,3)=0.0

      cmu(1,1)=1.0
      cmu(1,2)=-1.0
      cmu(1,3)=0.0
      cmu(2,1)=-1.0
      cmu(2,2)=1.0
      cmu(2,3)=0.0
      cmu(3,1)=0.0
      cmu(3,2)=0.0
      cmu(3,3)=1.0

      do 31 k=1,nphase
      do 21 j=1,3
      do 21 i=1,3
      cmod(k,i,j)=phasemod(k,1)*ck(i,j)+phasemod(k,2)*cmu(i,j)
21    continue
31    continue
c  Set up Simpson's integration rule weight vector
      do 30 j=1,3
```

```
      do 30 i=1,3
      nm=0
      if(i.eq.2) nm=nm+1
      if(j.eq.2) nm=nm+1
      g(i,j)=4.0**nm
30    continue

c  Loop over the nphase kinds of pixels and
c  Simpson's rule quadrature points in order to compute the stiffness
c  matrices.  Stiffness matrices of bilinear finite elements are quadratic
c  in x and y, so that Simpson's rule quadrature gives exact results.
      do 4000 ijk=1,nphase
      do 3000 j=1,3
      do 3000 i=1,3
      x=float(i-1)/2.0
      y=float(j-1)/2.0
c  dndx means the negative derivative with respect to x, of the shape
c  matrix N (see manual, Sec. 2.2), dndy is similar.
      dndx(1)=-(1.0-y)
      dndx(2)=(1.0-y)
      dndx(3)=y
      dndx(4)=-y
      dndy(1)=-(1.0-x)
      dndy(2)=-x
      dndy(3)=x
      dndy(4)=(1.0-x)
c  now build strain matrix
      do 2799 n1=1,3
      do 2799 n2=1,4
      do 2799 n3=1,2
      es(n1,n2,n3)=0.0
2799  continue
      do 2797 n=1,4
      es(1,n,1)=dndx(n)
      es(2,n,2)=dndy(n)
      es(3,n,1)=dndy(n)
      es(3,n,2)=dndx(n)
2797  continue
c  now do matrix multiply to determine value at (x,y), multiply by
c  proper weight, and sum into dk, the stiffness matrix
      do 900 mm=1,2
      do 900 nn=1,2
      do 900 ii=1,4
      do 900 jj=1,4
```

```
c  define sum over strain matrices and elastic moduli matrix for
c  stiffness matrix
     sum=0.0
     do 890 kk=1,3
     do 890 ll=1,3
     sum=sum+es(kk,ii,mm)*cmod(ijk,kk,ll)*es(ll,jj,nn)
890   continue
     dk(ijk,ii,mm,jj,nn)=dk(ijk,ii,mm,jj,nn)+g(i,j)*sum/36.
900   continue
3000  continue
4000  continue

c  Now compute the ss matrices, which give the thermal strain terms
c  for the i'th phase, single pixel.

     dndx(1)=-0.5
     dndx(2)=0.5
     dndx(3)=0.5
     dndx(4)=-0.5
     dndy(1)=-0.5
     dndy(2)=-0.5
     dndy(3)=0.5
     dndy(4)=0.5
c  now build average strain matrix
     do 3799 n1=1,3
     do 3799 n2=1,4
     do 3799 n3=1,2
     es(n1,n2,n3)=0.0
3799  continue
     do 3797 n=1,4
     es(1,n,1)=dndx(n)
     es(2,n,2)=dndy(n)
     es(3,n,1)=dndy(n)
     es(3,n,2)=dndx(n)
3797  continue
     do 3598 mmm=1,nphase
     do 3798 nn=1,2
     do 3798 mm=1,4
     sum=0.0
     do 3698 nm=1,3
     do 3698 n=1,3
     sum=sum+cmod(mmm,n,nm)*es(n,mm,nn)*eigen(mmm,nm)
3698  continue
     ss(mmm,mm,nn)=sum
```

```
3798  continue
3598  continue

c  now call subroutine const to generate zcon
c  zcon is a (2,2) x (2,2) matrix
     call const(dk,ns,zcon,nx,ny)

c  Now set up linear term, T, for thermal energy. It does not depend
c  on the macrostrains or displacements, so there is no need to update it
c  as the macrostrains change. T is built up out of the ss matrices.

     nss=ns+2
     do 6066 m2=1,2
     do 6066 m=1,nss
     T(m,m2)=0.0
6066  continue

c  For all cases, the correspondence between 1-4 finite element node
c  labels and the 1-9 neighbor labels is (see Table 4 in manual):
c  1:ib(m,9), 2:ib(m,3),3:ib(m,2),4:ib(m,1)
     is(1)=9
     is(2)=3
     is(3)=2
     is(4)=1
c  Do all points, but no macrostrain terms
c  note:  factor of 2 on linear thermal term is cancelled
c  by factor of 1/2 out in front of total energy term
     do 6601 j=1,ny
     do 6601 i=1,nx
     m=nx*(j-1)+i
     do 6600 mm=1,4
     do 6600 nn=1,2
     T(ib(m,is(mm)),nn)=T(ib(m,is(mm)),nn)-ss(pix(m),mm,nn)
6600  continue
6601  continue

c  now need to pick up and sum in all terms multiplying macrostrains
     do 7788 ipp=1,2
     do 7788 jpp=1,2
     if(ipp.eq.2.and.jpp.eq.2) goto 7788
     exx=0.0
     eyy=0.0
     exy=0.0
     if(ipp.eq.1.and.jpp.eq.1) exx=1.0
```

```
      if(ipp.eq.1.and.jpp.eq.2) eyy=1.0
      if(ipp.eq.2.and.jpp.eq.1) exy=1.0

c  x=nx face
      do 6001 i2=1,2
      do 6001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
6001  continue

      do 6000 j=1,ny-1
      m=j*nx
      do 6900 nn=1,2
      do 6900 mm=1,4
      T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6900  continue
6000  continue

c  y=ny face
      do 6011 i2=1,2
      do 6011 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.3.or.i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
6011  continue
      do 6010 i=1,nx-1
      m=nx*(ny-1)+i
      do 6901 nn=1,2
      do 6901 mm=1,4
      T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6901  continue
6010  continue

c  x=nx y=ny corner
      do 6031 i2=1,2
      do 6031 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2) then
      delta(i4,1)=exx*nx
```

```
      delta(i4,2)=exy*nx
      end if
      if(i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
      if(i4.eq.3) then
      delta(i4,1)=exy*ny+exx*nx
      delta(i4,2)=eyy*ny+exy*nx
      end if
6031  continue
      m=nx*ny
      do 6903 nn=1,2
      do 6903 mm=1,4
      T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6903  continue
6030  continue
7788  continue

c  now compute Y, the 0.5(eigen)Cij(eigen) energy, doesn't ever change
c  with macrostrain or displacements
      Y=0.0
      do 8811 m=1,ns
      do 8811 n=1,3
      do 8811 nn=1,3
      Y=Y+0.5*eigen(pix(m),n)*cmod(pix(m),n,nn)*eigen(pix(m),nn)
8811  continue

c  Following needs to be run after every change in macrostrain
c  when energy is recomputed.

1221  continue
c  Use auxiliary variables (exx, etc.) instead of u() variable, for
c  convenience, and to make the following code easier to read.
      exx=u(ns+1,1)
      eyy=u(ns+1,2)
      exy=u(nss,1)
c  Now set up vector for linear term that comes from periodic boundary
c  conditions.  Notation and conventions same as for T term.
c  This is done using the stiffness matrices, and the periodic terms
c  in the macrostrains.  It is easier to set up b in this way than to
c  analytically write out all the terms involved.

      do 5000 m2=1,2
```

```
      do 5000 m=1,ns
      b(m,m2)=0.0
5000  continue
c  For all cases, the correspondence between 1-4 finite element node
c  labels and the 1-9 neighbor labels is (see Table 4 in manual):
c  1:ib(m,9), 2:ib(m,3),3:ib(m,2),4:ib(m,1)
      is(1)=9
      is(2)=3
      is(3)=2
      is(4)=1

      C=0.0
c  x=nx face
      do 2001 i2=1,2
      do 2001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
2001  continue

      do 2000 j=1,ny-1
      m=j*nx
      do 1900 nn=1,2
      do 1900 mm=1,4
      sum=0.0
      do 1899 m2=1,2
      do 1899 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
1899  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1900  continue
2000  continue
c  y=ny face
      do 2011 i2=1,2
      do 2011 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.3.or.i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
2011  continue
      do 2010 i=1,nx-1
```

```
      m=nx*(ny-1)+i
      do 1901 nn=1,2
      do 1901 mm=1,4
      sum=0.0
      do 2099 m2=1,2
      do 2099 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2099  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1901  continue
2010  continue

c  x=nx y=ny corner
      do 2031 i2=1,2
      do 2031 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
      if(i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
      if(i4.eq.3) then
      delta(i4,1)=exy*ny+exx*nx
      delta(i4,2)=eyy*ny+exy*nx
      end if
2031  continue
      m=nx*ny
      do 1903 nn=1,2
      do 1903 mm=1,4
      sum=0.0
      do 2029 m2=1,2
      do 2029 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2029  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1903  continue
2030  continue

      return
      end
```

```
c  Subroutine that computes derivatives of the b-vector with respect
c  to the macrostrains.  Since b is linear in the macrostrains, the
c  derivative with respect to any one of them can be computed simply
c  by letting that macrostrain, within the subroutine, be equal to one,
c  and all the other macrostrains to be zero.
c  Very similar to 1221 loop in femat for b.

      subroutine bgrad(nx,ny,ns,exx,eyy,exy)
      real b(546914,2)
      real dk(100,4,2,4,2),delta(4,2),zcon(2,2,2,2)
      integer is(4)
      integer*4 ib(546914,9)
      integer*2 pix(546914)

      common/list3/ib
      common/list4/pix
      common/list5/dk,b,C,zcon,Y

c  exx, eyy, and exy are the artificial macrostrains used
c  to get the gradient terms (appropriate combinations of 1's and 0's).

c  Set up vector for linear term

      do 5000 m2=1,2
      do 5000 m=1,ns
      b(m,m2)=0.0
5000  continue
      is(1)=9
      is(2)=3
      is(3)=2
      is(4)=1

c  x=nx face
      do 2001 i2=1,2
      do 2001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
2001  continue

      do 2000 j=1,ny-1
      m=j*nx
```

```
      do 1900 nn=1,2
      do 1900 mm=1,4
      sum=0.0
      do 1899 m2=1,2
      do 1899 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
1899  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1900  continue
2000  continue
c  y=ny face
      do 2011 i2=1,2
      do 2011 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.3.or.i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
2011  continue
      do 2010 i=1,nx-1
      m=nx*(ny-1)+i
      do 1901 nn=1,2
      do 1901 mm=1,4
      sum=0.0
      do 2099 m2=1,2
      do 2099 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2099  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1901  continue
2010  continue

c  x=nx y=ny corner
      do 2031 i2=1,2
      do 2031 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
      if(i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
```

```
      if(i4.eq.3) then
      delta(i4,1)=exy*ny+exx*nx
      delta(i4,2)=eyy*ny+exy*nx
      end if
2031  continue
      m=nx*ny
      do 1903 nn=1,2
      do 1903 mm=1,4
      sum=0.0
      do 2029 m2=1,2
      do 2029 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2029  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1903  continue
2030  continue

      return
      end

c  Subroutine computes the quadratic term in the macrostrains, that comes
c  from the periodic boundary conditions, and sets it up as a
c  (2,2) x (2,2) matrix that couples to the three macrostrains

      subroutine const(dk,ns,zcon,nx,ny)
      real dk(100,4,2,4,2),zcon(2,2,2,2),delta(4,2)
      real pp(4,4),s(4,4)
      integer*2 pix(546914)
      common/list4/pix

c  routine to set up 4 x 4 matrix for energy term involving macro-strains
c  only, pulled out of femat (really 3 x 3, as ns+2,2 term is not used).
c  Idea is to evaluate the quadratic term in the macrostrains, C,
c  repeatedly for choices of strain like
c  exx=1, exy=1, all others = 0, build up 10 choices, then recombine
c  to get matrix elements by themselves

      nss=ns+2

      do 1111 i=1,4
      do 1111 j=1,4
      s(i,j)=0.0
      pp(i,j)=0.0
1111  continue
```

```
      do 5000 ii=1,4
      do 5000 jj=ii,4
      econ=0.0
      exx=0.0
      eyy=0.0
      exy=0.0
      if(ii.eq.1.and.jj.eq.1) exx=1.0
      if(ii.eq.2.and.jj.eq.2) eyy=1.0
      if(ii.eq.3.and.jj.eq.3) exy=1.0
      if(ii.eq.1.and.jj.eq.2) then
      exx=1.0
      eyy=1.0
      end if
      if(ii.eq.1.and.jj.eq.3) then
      exx=1.0
      exy=1.0
      end if
      if(ii.eq.2.and.jj.eq.3) then
      eyy=1.0
      exy=1.0
      end if
c  x=nx face
      do 2001 i2=1,2
      do 2001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
2001  continue

     do 2000 j=1,ny-1
      m=j*nx
      do 1900 nn=1,2
      do 1900 mm=1,4
      do 1899 m2=1,2
      do 1899 m4=1,4
      econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
1899  continue
1900  continue
2000  continue
c  y=ny face
      do 2011 i2=1,2
```

```
    do 2011 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.3.or.i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
2011  continue
    do 2010 i=1,nx-1
    m=nx*(ny-1)+i
    do 1901 nn=1,2
    do 1901 mm=1,4
    do 2099 m2=1,2
    do 2099 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
2099  continue
1901  continue
2010  continue

c  x=nx y=ny corner
    do 2031 i2=1,2
    do 2031 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.2) then
    delta(i4,1)=exx*nx
    delta(i4,2)=exy*nx
    end if
    if(i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
    if(i4.eq.3) then
    delta(i4,1)=exy*ny+exx*nx
    delta(i4,2)=eyy*ny+exy*nx
    end if
2031  continue
    m=nx*ny
    do 1903 nn=1,2
    do 1903 mm=1,4
    do 2029 m2=1,2
    do 2029 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
2029  continue
1903  continue
2030  continue
```

```
      pp(ii,jj)=econ*2.

5000  continue
      do 6000 i=1,4
      do 6000 j=i,4
      if(i.eq.j) s(i,j)=pp(i,j)
      if(i.ne.j) then
      s(i,j)=pp(i,j)-pp(i,i)-pp(j,j)
      end if
6000  continue
      do 7000 i=1,4
      do 7000 j=1,4
      pp(i,j)=0.5*(s(i,j)+s(j,i))
7000  continue
c  now map pp(i,j) into zcon(2,2,2,2)
      do 7200 i=1,2
      do 7200 j=1,2
      do 7200 mi=1,2
      do 7200 mj=1,2
      if(i.eq.1) ii=i+mi-1
      if(i.eq.2) ii=i+mi
      if(j.eq.1) jj=j+mj-1
      if(j.eq.2) jj=j+mj
      zcon(i,mi,j,mj)=pp(ii,jj)
      if(i.eq.2.and.mi.eq.2) zcon(i,mi,j,mj)=0.0
      if(j.eq.2.and.mj.eq.2) zcon(i,mi,j,mj)=0.0
7200  continue

      return
      end

c  Subroutine computes the total energy, utot, and the gradient, gb,
c  for the regular displacements as well as for the macrostrains

      subroutine energy(nx,ny,ns,utot)

            real u(546914,2),gb(546914,2)
            real b(546914,2),T(546914,2)
            real cmod(100,3,3),pk(3,4,2),eigen(100,3)
            real dk(100,4,2,4,2),zcon(2,2,2,2),C
            integer*4 ib(546914,9)
            integer*2 pix(546914)

            common/list3/ib
```

```
      common/list4/pix
         common/list5/dk,b,C,zcon,Y
         common/list6/u
         common/list7/gb
      common/list8/cmod,T,eigen

      nss=ns+2

      do 2090 m2=1,2
         do 2090 m=1,nss
         gb(m,m2)=0.0
2090    continue
c  Do global matrix multiply via small stiffness matrices, Ah = (A)(h).
c  The long statement below correctly brings in all the terms from
c  the global matrix A using only the small stiffness matrices.

      do 3000 j=1,2
      do 3000 n=1,2
         do 3000 m=1,ns
     gb(m,j)=gb(m,j)+u(ib(m,1),n)*( dk(pix(ib(m,9)),1,j,4,n)
    &+dk(pix(ib(m,7)),2,j,3,n) )+
    &u(ib(m,2),n)*( dk(pix(ib(m,9)),1,j,3,n) )+
    &u(ib(m,3),n)*( dk(pix(ib(m,9)),1,j,2,n)+dk(pix(ib(m,5)),4,j,3,n))+
    &u(ib(m,4),n)*( dk(pix(ib(m,5)),4,j,2,n) )+
    &u(ib(m,5),n)*( dk(pix(ib(m,6)),3,j,2,n)+dk(pix(ib(m,5)),4,j,1,n))+
    &u(ib(m,6),n)*( dk(pix(ib(m,6)),3,j,1,n) )+
    &u(ib(m,7),n)*( dk(pix(ib(m,6)),3,j,4,n)+dk(pix(ib(m,7)),2,j,1,n))+
    &u(ib(m,8),n)*( dk(pix(ib(m,7)),2,j,4,n) )+
    &u(ib(m,9),n)*( dk(pix(ib(m,9)),1,j,1,n)
    &+dk(pix(ib(m,7)),2,j,2,n)+
    &dk(pix(ib(m,6)),3,j,3,n)+dk(pix(ib(m,5)),4,j,4,n) )
3000    continue

     utot=0.0

     gtot=0.0
     do 3100 m2=1,2
     do 3100 m=1,ns
     utot=utot+0.5*u(m,m2)*gb(m,m2)+b(m,m2)*u(m,m2)
c  this is gradient of energy with respect to normal displacements
     gb(m,m2)=gb(m,m2)+b(m,m2)
3100  continue

c  compute "constant" macrostrain energy term
```

```
      C=0.0
      do 7200 i=1,2
      do 7200 j=1,2
      do 7200 mi=1,2
      do 7200 mj=1,2
      C=C+0.5*u(ns+i,mi)*zcon(i,mi,j,mj)*u(ns+j,mj)
7200  continue
      utot=utot+C
c  now add in constant term from thermal energy, Y
      utot=utot+Y
c  now add in linear term in thermal energy
      do 7171 m2=1,2
      do 7171 m=1,nss
      utot=utot+T(m,m2)*u(m,m2)
7171  continue

c  now compute gradient with respect to macrostrains
c  put in piece from first derivative of zcon quadratic term
      do 7300 i=1,2
      do 7300 mi=1,2
      sum=0.0
      do 7250 j=1,2
      do 7250 mj=1,2
      sum=sum+zcon(i,mi,j,mj)*u(ns+j,mj)
7250  continue
      gb(ns+i,mi)=sum
7300  continue
c  add in piece of gradient, for displacements as well as macrostrains,
c  that come from linear term in thermal energy
      do 3150 m2=1,2
      do 3150 m=1,nss
      gb(m,m2)=gb(m,m2)+T(m,m2)
3150  continue
c  now generate part that comes from b . u term
c  do by calling b generation with appropriate macrostrain set to 1 to
c  get that partial derivative, just use bgrad (taken from femat),
c  skip dk and zcon part
      do 8100 ii=1,3
      exx=0.0
      eyy=0.0
      exy=0.0
      if(ii.eq.1) exx=1.0
      if(ii.eq.2) eyy=1.0
      if(ii.eq.3) exy=1.0
```

```
      call bgrad(nx,ny,ns,exx,eyy,exy)
      sum=0.0
       do 8200 m2=1,2
          do 8200 m=1,ns
          sum=sum+u(m,m2)*b(m,m2)
8200    continue
          if(ii.eq.1) gb(ns+1,1)=gb(ns+1,1)+sum
          if(ii.eq.2) gb(ns+1,2)=gb(ns+1,2)+sum
          if(ii.eq.3) gb(ns+2,1)=gb(ns+2,1)+sum
8100    continue
       return
       end

c  Subroutine that carries out the conjugate gradient relaxation process

      subroutine dembx(nx,ny,ns,Lstep,gg,gtest,ldemb,kkk)

      real u(546914,2),gb(546914,2),b(546914,2)
      real h(546914,2),Ah(546914,2)
      real dk(100,4,2,4,2),zcon(2,2,2,2)
      real lambda,gamma
      integer*4 ib(546914,9)
      integer*2 pix(546914)

       common/list2/h,Ah
          common/list3/ib
       common/list4/pix
          common/list5/dk,b,C,zcon,Y
          common/list6/u
          common/list7/gb

      nss=ns+2
c  Initialize the conjugate direction vector on first call to dembx only.
c  For calls to dembx after the first, we want to continue using the value
c  of h determined in the previous call.  Of course, if npoints is greater
c  than 1, then this initialization step will be run each time a new
c  microstructure is used, as kkk will be reset to 1 every time the
c  counter micro is increased.
      if(kkk.eq.1) then
      do 500 m2=1,2
      do 500 m=1,nss
      h(m,m2)=gb(m,m2)
500   continue
      end if
```

```
c  Lstep counts the number of conjugate gradient steps taken
c  in each call to dembx
     Lstep=0

     do 800 ijk=1,ldemb
     Lstep=Lstep+1

     do 290 m2=1,2
     do 290 m=1,nss
     Ah(m,m2)=0.0
290   continue
c  Do global matrix multiply via small stiffness matrices, Ah = (A)(h).
c  The long statement below correctly brings in all the terms from
c  the global matrix A using only the small stiffness matrices.
       do 400 j=1,2
       do 400 n=1,2
          do 400 m=1,ns
     Ah(m,j)=Ah(m,j)+h(ib(m,1),n)*( dk(pix(ib(m,9)),1,j,4,n)
    &+dk(pix(ib(m,7)),2,j,3,n) )+
    &h(ib(m,2),n)*( dk(pix(ib(m,9)),1,j,3,n) )+
    &h(ib(m,3),n)*( dk(pix(ib(m,9)),1,j,2,n)+dk(pix(ib(m,5)),4,j,3,n))+
    &h(ib(m,4),n)*( dk(pix(ib(m,5)),4,j,2,n) )+
    &h(ib(m,5),n)*( dk(pix(ib(m,6)),3,j,2,n)+dk(pix(ib(m,5)),4,j,1,n))+
    &h(ib(m,6),n)*( dk(pix(ib(m,6)),3,j,1,n) )+
    &h(ib(m,7),n)*( dk(pix(ib(m,6)),3,j,4,n)+dk(pix(ib(m,7)),2,j,1,n))+
    &h(ib(m,8),n)*( dk(pix(ib(m,7)),2,j,4,n) )+
    &h(ib(m,9),n)*( dk(pix(ib(m,9)),1,j,1,n)
    &+dk(pix(ib(m,7)),2,j,2,n)+
    &dk(pix(ib(m,6)),3,j,3,n)+dk(pix(ib(m,5)),4,j,4,n) )
400    continue

c  The above accurately gives the second derivative matrix with respect
c  to nodal displacements, but fails to give the 2nd derivative terms that
c  include the macrostrains [ du d(strain) and d(strain)d(strain) ].
c  Use repeated calls to bgrad to generate mixed 2nd derivatives terms,
c  plus use zcon in order to correct the matrix multiply and correctly bring
c  in macrostrain terms (see manual, Sec. 2.4).
     do 8100 ii=1,3
     e11=0.0
     e22=0.0
     e12=0.0
     if(ii.eq.1) e11=1.0
     if(ii.eq.2) e22=1.0
     if(ii.eq.3) e12=1.0
```

```
      call bgrad(nx,ny,ns,e11,e22,e12)
c  now fill in terms from matrix multiply
c  right hand sides, 1 to ns
      do 3333 m=1,ns
      do 3333 m1=1,2
      if(ii.eq.1) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(ns+1,1)
      if(ii.eq.2) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(ns+1,2)
      if(ii.eq.3) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(nss,1)
3333  continue
c  now do across bottom, 1 to ns
      do 3334 m=1,ns
      if(ii.eq.1) Ah(ns+1,1)=Ah(ns+1,1)+b(m,1)*h(m,1)+
     +b(m,2)*h(m,2)
      if(ii.eq.2) Ah(ns+1,2)=Ah(ns+1,2)+b(m,1)*h(m,1)+
     +b(m,2)*h(m,2)
      if(ii.eq.3) Ah(nss,1)=Ah(nss,1)+b(m,1)*h(m,1)+
     +b(m,2)*h(m,2)
3334  continue
c  now do righthand corner terms, ns+1 to nss
      do 3335 m=1,2
      do 3335 m1=1,2
      if(ii.eq.1) Ah(ns+1,1)=Ah(ns+1,1)+zcon(1,1,m,m1)*h(ns+m,m1)
      if(ii.eq.2) Ah(ns+1,2)=Ah(ns+1,2)+zcon(1,2,m,m1)*h(ns+m,m1)
      if(ii.eq.3) Ah(nss,1)=Ah(nss,1)+zcon(2,1,m,m1)*h(ns+m,m1)
3335  continue

8100    continue

      hAh=0.0
      do 530 m2=1,2
      do 530 m=1,nss
      hAh=hAh+h(m,m2)*Ah(m,m2)
530   continue

      lambda=gg/hAh
      do 540 m2=1,2
      do 540 m=1,nss
      u(m,m2)=u(m,m2)-lambda*h(m,m2)
      gb(m,m2)=gb(m,m2)-lambda*Ah(m,m2)
540   continue

      gglast=gg
      gg=0.0
      do 550 m2=1,2
```

```
      do 550 m=1,nss
      gg=gg+gb(m,m2)*gb(m,m2)
550   continue
      if(gg.lt.gtest) goto 1000

      gamma=gg/gglast
      do 570 m2=1,2
      do 570 m=1,nss
      h(m,m2)=gb(m,m2)+gamma*h(m,m2)
570   continue

800   continue

1000  continue
      return
      end

c  Subroutine that computes the three average stresses
c  and three average strains.

      subroutine stress(nx,ny,ns,iswitch)

      real u(546914,2),uu(4,2),pmax(546914,2)
      real T(546914,2),eigen(100,3)
      real dndx(4),dndy(4),es(3,4,2),cmod(100,3,3)
      integer*4 ib(546914,9)
      integer*2 pix(546914)

      common/list1/strxx,stryy,strxy
      common/list3/ib
      common/list4/pix
      common/list6/u
      common/list8/cmod,T,eigen
      common/list11/sxx,syy,sxy
      common/list12/pmax

      nss=ns+2
      exx=u(ns+1,1)
      eyy=u(ns+1,2)
      exy=u(nss,1)
      if(iswitch.eq.1) then
c     open(unit=12,file='101bw-stress-strain-allmat.txt')
c     write(12,1134) nx,ny
c1134  format(2i5)
```

```
      end if

c set up single pixel strain matrix

      dndx(1)=-0.5
      dndx(2)=0.5
      dndx(3)=0.5
      dndx(4)=-0.5
      dndy(1)=-0.5
      dndy(2)=-0.5
      dndy(3)=0.5
      dndy(4)=0.5

c initialize pmax for iswitch=1
      if(iswitch.eq.1) then
      do 5555 m=1,ns
      pmax(m,1)=0.0
      pmax(m,2)=m
5555  continue
      end if
c  Build average strain matrix, follows code in femat, but for average
c  strain over a pixel, not the strain at a point
      do 2799 n1=1,3
      do 2799 n2=1,4
      do 2799 n3=1,2
      es(n1,n2,n3)=0.0
2799  continue
      do 2797 n=1,4
      es(1,n,1)=dndx(n)
      es(2,n,2)=dndy(n)
      es(3,n,1)=dndy(n)
      es(3,n,2)=dndx(n)
2797  continue
c  now compute average stresses and strains in each pixel
      sxx=0.0
      syy=0.0
      sxy=0.0
      strxx=0.0
      stryy=0.0
      strxy=0.0
      do 470 j=1,ny
      do 470 i=1,nx
      m=(j-1)*nx+i
      if(iswitch.eq.1) then
```

```
c      write(14,*) ' i j in stress m ',i,j,m
c      call flush(14)
       end if
c  load in elements of 4-vector using pd. bd. conds.
       do 9898 mm=1,2
       uu(1,mm)=u(m,mm)
       uu(2,mm)=u(ib(m,3),mm)
       uu(3,mm)=u(ib(m,2),mm)
       uu(4,mm)=u(ib(m,1),mm)
9898   continue
c  Correct for periodic boundary conditions, some displacements are wrong
c  for a pixel on a periodic boundary.  Since they come from an opposite
c  face, need to put in applied strain to correct them.
       if(i.eq.nx) then
       uu(2,1)=uu(2,1)+exx*nx
       uu(2,2)=uu(2,2)+exy*nx
       uu(3,1)=uu(3,1)+exx*nx
       uu(3,2)=uu(3,2)+exy*nx
       end if
       if(j.eq.ny) then
       uu(3,1)=uu(3,1)+exy*ny
       uu(3,2)=uu(3,2)+eyy*ny
       uu(4,1)=uu(4,1)+exy*ny
       uu(4,2)=uu(4,2)+eyy*ny
       end if
       str11=0.0
       str22=0.0
       str12=0.0
       s11=0.0
       s22=0.0
       s12=0.0
c********compute average stress and strain tensor in each pixel**********
c  First put thermal strain-induced stresses into stress tensor
       do 465 n=1,3
       str11=str11-cmod(pix(m),1,n)*eigen(pix(m),n)
       str22=str22-cmod(pix(m),2,n)*eigen(pix(m),n)
       str12=str12-cmod(pix(m),3,n)*eigen(pix(m),n)
465    continue
       do 466 n2=1,2
       do 466 n4=1,4
c  compute non-thermal strains in each pixel
       s11=s11+es(1,n4,n2)*uu(n4,n2)
       s22=s22+es(2,n4,n2)*uu(n4,n2)
       s12=s12+es(3,n4,n2)*uu(n4,n2)
```

```
      do 466 n=1,3
c  compute stresses in each pixel that include both non-thermal
c  and thermal strains
      str11=str11+cmod(pix(m),1,n)*es(n,n4,n2)*uu(n4,n2)
      str22=str22+cmod(pix(m),2,n)*es(n,n4,n2)*uu(n4,n2)
      str12=str12+cmod(pix(m),3,n)*es(n,n4,n2)*uu(n4,n2)
466   continue
c  Sum local stresses and strains into global stresses and strains
      strxx=strxx+str11
      stryy=stryy+str22
      strxy=strxy+str12
      sxx=sxx+s11
      syy=syy+s22
      sxy=sxy+s12
      if(iswitch.eq.1) then
      sa=0.5*((str11+str22)+sqrt((str11-str22)**2+4.*str12*str12))
      sb=0.5*((str11+str22)-sqrt((str11-str22)**2+4.*str12*str12))
      ea=0.5*((s11+s22)+sqrt((s11-s22)**2+4.*s12*s12))
      eb=0.5*((s11+s22)-sqrt((s11-s22)**2+4.*s12*s12))
c       if(ea.gt.eb) pmax(m,1)=ea
c       if(eb.gt.ea) pmax(m,1)=eb
      if(sa.gt.sb) pmax(m,1)=sa
      if(sb.gt.sa) pmax(m,1)=sb
      pmax(m,2)=m
c       write(14,*) m,pmax(m,1),pmax(m,2)
c       call flush(14)
c       write(12,1137) i,j,str11,str22,str12,s11,s22,s12
c       write(12,1138) i,j,sa,sb,ea,eb
1137  format(2i4,6f15.9)
1138  format(2i4,4f15.9)
      end if
470   continue

c  Area average global stresses and strains

            strxx=strxx/float(ns)
            stryy=stryy/float(ns)
            strxy=strxy/float(ns)
            sxx=sxx/float(ns)
            syy=syy/float(ns)
            sxy=sxy/float(ns)

      return
      end
```

```
c  Subroutine to count volume fractions of various phases

      subroutine assig(ns,nphase,prob)
      integer*2 pix(546914)
      real prob(100)
      common/list4/pix

           do 999 i=1,nphase
           prob(i)=0.0
999        continue

      do 1000 m=1,ns
      do 1000 i=1,nphase
        if(pix(m).eq.i) then
           prob(i)=prob(i)+1
           end if
1000     continue

           do 998 i=1,nphase
           prob(i)=prob(i)/float(ns)
998        continue

         return
         end

c  Subroutine to set up image of microstructure
c  adds border all around of mortar (phase 1)

      subroutine ppixel(nx,ny,ns,nphase)
      integer*2 pix(546914)
      common/list4/pix

      call srand(-147)
c  (USER)  If you want to set up a test image inside the program, instead
c  of reading it in from a file, this should be done inside this subroutine.

      open (9,file='micro.in')

      do 200 j=1,ny
      do 200 i=1,nx
      m=nx*(j-1)+i
      read(9,*) pix(m)
c turn white border into black matrix
```

```
C      if(pix(m).eq.4) pix(m)=1
200   continue
      close(9)

c make phases 5-14 matrix phases but with slightly different moduli

C      do 300 j=1,ny
C      do 300 i=1,nx
C      m=nx*(j-1)+i
C      if(pix(m).eq.1) then
C      mm=5+10*rand(0)
C      if(mm.eq.15) mm=14
C      pix(m)=mm
C      end if
C300   continue
C      close(19)

c  Check for wrong phase labels--less than 1 or greater than nphase
       do 500 m=1,ns
       if(pix(m).lt.1) then
        write(7,*) 'Phase label in pix < 1--error at ',m,pix(m)
       end if
       if(pix(m).gt.nphase) then
        write(7,*) 'Phase label in pix > nphase--error at ',m,pix(m)
       end if
500    continue

      return
      end
      subroutine crack(ns,nx,ny,strcrit,micro,ncc)
      real pmax(546914,2)
      real paux(546914,2)
      integer*2 pix(546914)
      common/list4/pix
      common/list12/pmax
      common/list13/ncrack

c Find where max principal strain exceeds strcrit.
c Then change the phase of these pixels to crack = 4

C First find the stresses or strains that exceed strcrit


      ncracks=0
```

```
    do 100 j=1,ny
    do 90 i=1,nx
    m=(j-1)*nx+i
    if(pix(m).eq.2.or.pix(m).eq.4)goto 90
    if(pmax(m,1).gt.strcrit)then
    ncracks=ncracks+1
    paux(ncracks,1)=pmax(m,1)
    paux(ncracks,2)=pmax(m,2)
    end if
90   continue
100  continue

    write(6,*)"ncracks=",ncracks

C Now sort those cracks in paux from max to min

    do 101 m=1,ncracks
    do 91 m1=m+1,ncracks
    if(paux(m1,1).gt.paux(m,1)) then
    xxx=paux(m,1)
    yyy=paux(m,2)
    paux(m,1)=paux(m1,1)
    paux(m,2)=paux(m1,2)
    paux(m1,1)=xxx
    paux(m1,2)=yyy
    end if
91   continue
101  continue

C Now switch phase to a crack for maxcracks or less  per cycle.

    maxcracks=1
    ncc=maxcracks

    if(ncracks.le.maxcracks)ncc=ncracks
    if(ncracks.gt.maxcracks)ncracks=maxcracks

    do 120 m=1,ncc
    m1=paux(m,2)
    pix(m1)=4
120  continue
    write(7,*) ncc,' pixels cracked and eliminated at ',micro,' step'

    return
```

```
      end

      subroutine order(ns)
      real pmax(546914,2)
      integer*2 pix(546914)
      common/list4/pix
      common/list12/pmax
      common/list13/ncrack

c order the first ten stresses to get the highest 10, remember
c where they occurred. Then change the phase of these 10 pixels
c to crack (4).

      do 100 m=1,ncrack
      do 90 m1=m+1,ns
      if(pmax(m1,1).gt.pmax(m,1)) then
      xxx=pmax(m,1)
      yyy=pmax(m,2)
      pmax(m,1)=pmax(m1,1)
      pmax(m,2)=pmax(m1,2)
      pmax(m1,1)=xxx
      pmax(m1,2)=yyy
      end if
90    continue
100   continue

c now zero out top ncrack tensile stress pixels
      do 200 m=1,ncrack
      mm=pmax(m,2)
      write(7,*) 'order ',m,mm
      pix(mm)=4
200   continue
      return
      end
      subroutine image(ns,micro,nx,ny)
      integer*2 pix(546914)
      common/list4/pix
      character*6 namef
      character*9 namefff
      character*31 nameff

      namef(1:2)='00'
      namef(3:6)='.pgm'
      if(micro.lt.10) write(namef(2:2),'(I1)') micro
```

```
      if(micro.ge.10) write(namef(1:2),'(I2)') micro
      open(unit=14,file=namef)
      write(14,133)
      write(14,134) nx,ny
      write(14,157)
133   format('P2')
134   format(i4,1x,i4)
157   format('4')

      do 741 j=ny,1,-1
      do 741 i=1,nx
      m=nx*(j-1)+i
      mm=pix(m)
c make irregular matrix all look the same in the image
      if(mm.ge.5.and.mm.le.14) mm=1
      write(14,22) mm
22    format(i1)
741   continue
      close(14)

c convert pgm file to gif

      nameff(1:8)='convert '
      nameff(9:14)=namef
      nameff(15:24)=' -depth 8 '
      nameff(25:30)=namef
      nameff(28:30)='gif'
      nameff(31:31)=' '
      call system(nameff)
c rm pgm files
      namefff(1:3)='rm '
      namefff(4:9)=namef
      call system(namefff)

      return
      end
      subroutine burn(nx,ny,ns)
      integer*2 pixc(nx,ny),pix(546914)
      integer*2 in(4),jn(4),old(10000000,2),new(10000000,2)
      integer*2 save(10000000,2)
      common/list4/pix

c transfer pix to pixc
      do 333 j=1,ny
```

```
      do 333 i=1,nx
      m=nx*(j-1)+i
      pixc(i,j)=pix(m)
c set random 5-14 matrix to plain mortar=1
      if(pix(m).gt.4) pixc(i,j)=1
333   continue

c then all particles stored in the particle image will be processed.
      icutoff=15

c  Direction labels to check for burning path (nearest neighbor information
c  in 3-D digital system).
      in(1)=-1
      in(2)=1
      in(3)=0
      in(4)=0
      jn(1)=0
      jn(2)=0
      jn(3)=-1
      jn(4)=1

c This next part of the program does the actual burning = cluster
c identification.

c can start with boundaries since we only want existing clusters
      do 1000 j=1,ny
      do 1000 i=1,nx

c don't choose any pore pixel or previously burned and identified particle
c pixel, which is still equal to the value of burned. or background pixel
c This assumes that matrix = 1 and aggregate = 3. Expansive = 2
c and negative means already burned. Cracks = 4.
      if(pixc(i,j).lt.0.or.pixc(i,j).eq.2.or.pixc(i,j).eq.4) goto 1000
      ipart=pixc(i,j)

      iold=0
      isave=0
c  find a first particle pixel
      if(pixc(i,j).eq.ipart) then
      pixc(i,j)=-pixc(i,j)
      iold=iold+1
      isave=isave+1
      old(iold,1)=i
      old(iold,2)=j
```

```
c save particle pixel coordinates
      save(iold,1)=i
      save(iold,2)=j
      end if

c  Now start building up new burned pixels from old set of burned pixels,
c  thus propagating the fire and identifying new particle pixels connnected
c  to the ones already burned.

60    inew=0
      do 100 ijk=1,iold
      ii=old(ijk,1)
      jj=old(ijk,2)
c  check all four nearest neighbors of previously burned pixel
      do 90 n=1,4
      i1=ii+in(n)
      j1=jj+jn(n)

c  Hard boundary conditions (can't pass beyond boundary)

      if(j1.lt.1.or.j1.gt.ny) then
      goto 90
      end if
      if(i1.lt.1.or.i1.gt.nx) then
      goto 90
      end if

c  Store (i,j) labels of newly burned pixels in array new().
      if(pixc(i1,j1).eq.ipart) then
      pixc(i1,j1)=-pixc(i1,j1)
      inew=inew+1
      new(inew,1)=i1
      new(inew,2)=j1
      end if
90    continue
100   continue

c  If new pixels were burned, then transfer labels to old() array, start
c  burning process over again.
      if(inew.gt.0) then
      iold=inew
      do 150 ijk=1,inew
      old(ijk,1)=new(ijk,1)
      old(ijk,2)=new(ijk,2)
```

```
      save(ijk+isave,1)=new(ijk,1)
      save(ijk+isave,2)=new(ijk,2)
150   continue
      isave=isave+inew
      goto 60
      end if

c  CHECK 1 - is particle too small?

c  If particle is not big enough, go to restore, wipe it out,
c  and try again.  icutoff is the minimum
c  number of pixels that a particle must have to be kept.
      if(isave.lt.icutoff) then
c then turn iold pixels into crack = 4. If greater than
c icutoff then leave negative and go find another cluster
c when all clusters are found, turn all negatives back into positives
       write(8,*) ' found small cluster to remove = ',isave
      call flush(8)
      do 225 m=1,isave
      iii=save(m,1)
      jjj=save(m,2)
      pixc(iii,jjj)=4
225   continue
      end if

1000  continue

c transfer pixc back to pix and set neg to pos
c preserve random matrix in pix(m)
      do 600 j=1,ny
      do 600 i=1,nx
      if(pixc(i,j).lt.0) pixc(i,j)=-pixc(i,j)
      m=nx*(j-1)+i
      if(pix(m).gt.4) then
        if(pixc(i,j).eq.4) then
        pix(m)=4
        goto 600
        end if
      goto 600
      end if
      pix(m)=pixc(i,j)
600   continue
      return
      end
```

C. **To obtain physical/mechanical properties**

- ‘thermal2d-moduli.f’

```
c                Copyright Notice
c
c This software was developed at the National Institute of Standards
c and Technology (NIST) by employees of the Federal Government in the
c course of their official duties. Pursuant to title 17 Section 105
c of the United States Code this software is not subject to copyright
c protection and is in the public domain.
c
c This is an experimental system.  NIST assumes no responsibility
c whatsoever for its use by other parties, and makes no guarantees,
c expressed or implied, about its quality, reliability, or any other
c characteristic.
c
c We would appreciate acknowledgment if the software is used.
c
c
c
c Contact: Osamah H. Dehwah  osamah.dehwah@nist.gov
c        NIST
c        Engineering Laboratory
c        Materials and Structural Systems Division
c        Infrastructure Materials Group

c This version does not allow macrostrains to change
c macrostrains are set at beginning and used to compute
c the elastic moduli, bulk and shear at the same time
c all eigenstrains = 0

c This version also puts some randomness into matrix, to see if that
c makes the cracks have more of a random direction rather than following
c the i-j directions.

c PERIODIC BOUNDARY CONDITIONS - no displacements held at constant
c  ************************  thermal2d.f  ************************
c  BACKGROUND

c  Program adjusts dimensions of unit cell,
c  [(1 + macrostrain) times dimension],
c  in response to phases that have a non-zero eigenstrain and
c  arbitrary elastic moduli tensors.
c  All three macrostrains can adjust their values (2-d program), and are
```

```
c  stored in the last two positions in the displacement vector u,
c  as listed below. Periodic boundaries are maintained.
c  In the comments below, (USER) means that this is a section of code
c  that the user might have to change for  his particular problem.
c  Therefore the user is encouraged to search for this string.

c  PROBLEM AND VARIABLE DEFINITION

c  The problem being solved is the minimization of the elastic energy
c  1/2 uAu + bu + C + Tu + Y, where b and C are also functions of the
c  macrostrains.
c  The small array zcon computes the thermal strain energy associated
c  with macrostrains (C term), T is the thermal energy term linear in the
c  displacements (built from ss), b is the regular energy term linear in the
c  displacements, u is the displacements including the macrostrains, gb
c  is the energy gradient vector, h,Ah are auxiliary vectors,
c  dk is the single pixel stiffness matrix, pix is the phase
c  identification vector (pix=1 for phase 1, etc.), and ib is the
c  integer matrix for mapping labels from the 1-9 nearest neighbor
c  labelling to the 1-d system labelling.
c  The array prob(i) contains the volume fractions of the i'th phase,
c  strxx, etc. are the three independent area averaged stresses
c  (Voigt notation), sxx, etc. are the three independent area
c  averaged strains (Voigt notation, not counting the thermal strains),
c  cmod(i,3,3) gives the elastic modulus tensor
c  of the i'th phase, and dk(i,4,2,4,2) is the stiffness matrix of the i'th
c  phase.  The parameter nphase gives the number of phases being considered
c  in the problem, and is set by the user.

c  DIMENSIONS

c  The main arrays of the problem, u, gb, h, Ah, b, and T, are dimensioned
c  as (nx*ny)+2, which is the number of nodal displacements plus two for
c  the macrostrains.
c  The program currently assumes that the number of different phases is
c  100, since phasemod, eigen, and ss (the moduli, eigenstrains, and
c  auxiliary eigenstrain variable for each phase)
c  and dk are dimensioned to have at most 100 different kinds.  This
c  is easily changed.  The parameter nphase gives the number of phases
c  expected in the problem, and is user-specified.  All major arrays
c  are passed to subroutines via simple common statements.

c  NOTE ON USE OF PROGRAM:  Program is set up to allow the macrostrains,
c  which control the overall size of the system, to be dynamic
```

```
c  variables, which are adjusted in order to minimize the overall
c  energy.  That means that if there are no eigenstrains specified
c  for any of the phases, the overall strain will always relax to
c  zero.  If it is desired to simply apply a fixed strain, with no
c  eigenstrains, then in subroutines Energy and Dembx, one must
c  zero out the elements of gb (in energy and in dembx) that
c  correspond to the macrostrains.  This is easily done.
c  This will fix the gradients of the macrostrains to always to be
c  zero, so that they will not change, so the applied strain (initial
c  values of the macrostrains) will remain fixed.

c  STRONGLY SUGGESTED:  READ MANUAL BEFORE USING PROGRAM!!!

c  (USER)  Change these dimensions and in other subroutines at same time.
c  For example, search and replace all occurrences throughout the
c  program of "(402" by "(1602", to go from a 20 x 20 system to
c  a 40 x 40 system.
c The u and similar arrays are dimensioned (nx times ny) + 2.

      real u(546914,2),gb(546914,2),b(546914,2)
      real h(546914,2),Ah(546914,2),T(546914,2)
         real C,dk(100,4,2,4,2)
      real cmod(100,3,3),ss(100,4,2),eigen(100,3)
         real zcon(2,2,2,2),pk(3,4,2)
         real phasemod(100,2),prob(100)
         integer in(9),jn(9),kn(9)
         integer*4 ib(546914,9)
         integer*2 pix(546914)

      common/list1/strxx,stryy,strxy
      common/list2/h,Ah
         common/list3/ib
      common/list4/pix
         common/list5/dk,b,C,zcon,Y
         common/list6/u
         common/list7/gb
      common/list8/cmod,T,eigen
      common/list10/phasemod,nphase,ss
      common/list11/sxx,syy,sxy
      common/list12/pmax
      common/list13/ncrack

c  (USER)  Unit 9 is the microstructure input file, unit 7
c  is the results output file.
```

```
c          open (9,file='micro.in')
           open (7,file='ab4-elas-mod.out')

c (USER) nx and ny are the size of the lattice
c note that the dimension of the arrays is (nx times ny) +2
      nx=844
      ny=648
c      ns=total number of sites
      ns=nx*ny
      write(7,9010) nx,ny,ns
9010  format(' nx= ',i4,'  ny= ',i4,' ns = ',i8)
      call srand(-189)

c  Add two more entries in the displacement vector for the 3 macrostrains,
c  u(ns+1,1) = exx, u(ns+1,2) = eyy, u(ns+2,1) = exy, u(ns+2,2) = never used
      nss=ns+2

c  (USER) nphase is the number of phases being considered in the problem.
c  The values of pix(m) will run from 1 to nphase.
c  add extra zero phase for cracks = 4.
           nphase=3+1+10

      call flush(7)
c  (USER) gtest is the stopping criterion, compared to gg=gb*gb.
c  If gtest=abc*ns, when gg < gtest, the rms value per pixel
c  of gb is less than sqrt(abc).
c  The value of gtest must be adjusted for each microstructure to
c  enable valid results.
      gtest=1.e-12*(nx*ny)
      write(7,*) 'relaxation criterion gtest = ',gtest

c  (USER)
c  The parameter phasemod(i,j) is the bulk (i,1)and shear (i,2) moduli of
c  the i'th phase. These can be
c  input in terms of Young's modulus E (i,1) and Poisson's ratio nu (i,2).
c  The program, in do 1144 loop, then changes them to bulk and shear
c  moduli, using relations for isotropic elastic moduli.
c  For anisotropic moduli tensors, one can directly input the whole tensor
c  cmod in subroutine femat, and skip this part.
c  If you wish to input in terms of bulk (i,1) and shear (i,2) moduli,
c  then simply comment out do 1144 loop.

cc mortar (moduli are in GPa). Poisson's ratio is unitless.
c          phasemod(1,1)=50.0
```

```
c          phasemod(1,2)=0.25
cc chert (expansive aggregate)
c          phasemod(2,1)=80.
c          phasemod(2,2)=0.2
cc regular aggregate
c          phasemod(3,1)=80.
c       phasemod(3,2)=0.2
cc cracked material
c          phasemod(4,1)=0.0
c       phasemod(4,2)=0.2
c       do 178 i=5,14
c          phasemod(i,1)=50.0+(rand(0)-0.5)*16.0
c          phasemod(i,2)=0.25
c178     continue

cc convert E and nu to K and G
c       do 1144 i=1,nphase
c       save=phasemod(i,1)
c       phasemod(i,1)=phasemod(i,1)/2./(1.-phasemod(i,2))
c       phasemod(i,2)=save/2./(1.+phasemod(i,2))
c1144    continue

     open(777,file="phasemod_values")
      do i=1,nphase
      read(777,*)phasemod(i,1),phasemod(i,2)
      end do
     close(777)

c  (USER) input eigen (thermal) strains for each phase.
c  (1=xx, 2=yy, 3=xy).
c   A positive value means that the phase wants to expand, a negative
c   value means the phase wants to shrink.
       eigen(1,1)=0.0
       eigen(1,2)=0.0
       eigen(1,3)=0.0
       eigen(2,1)=0.0
       eigen(2,2)=0.0
       eigen(2,3)=0.0
       eigen(3,1)=0.0
       eigen(3,2)=0.0
       eigen(3,3)=0.0
       eigen(4,1)=0.0
       eigen(4,2)=0.0
       eigen(4,3)=0.0
```

```
      do 47 i=5,14
      eigen(i,1)=eigen(1,1)
      eigen(i,2)=eigen(1,2)
      eigen(i,3)=eigen(1,3)
47     continue

c  Construct the 9 neighbor table, ib(m,n)

c  First construct the 9 neighbor table in terms of delta i and delta j
c  information (see Table 3 in manual)
     in(1)=0
     in(2)=1
     in(3)=1
     in(4)=1
     in(5)=0
     in(6)=-1
     in(7)=-1
     in(8)=-1
     in(9)=0

     jn(1)=1
     jn(2)=1
     jn(3)=0
     jn(4)=-1
     jn(5)=-1
     jn(6)=-1
     jn(7)=0
     jn(8)=1
     jn(9)=0

c  Now construct neighbor table according to 1-d labels
c  Matrix ib(m,n) gives the 1-d label of the n'th neighbor (n=1,9) of
c  the node labelled m.
     do 1020 j=1,ny
     do 1020 i=1,nx
     m=nx*(j-1)+i
     do 1004 n=1,9
     i1=i+in(n)
     j1=j+jn(n)
     if(i1.lt.1) i1=i1+nx
     if(i1.gt.nx) i1=i1-nx
     if(j1.lt.1) j1=j1+ny
     if(j1.gt.ny) j1=j1-ny
     m1=nx*(j1-1)+i1
```

```
     ib(m,n)=m1
1004  continue
1020  continue

c  Compute the average stress and strain, as well as the macrostrains (overall
c  system size and shape) in each microstructure.
c  (USER) npoints is the number of microstructures to use.
       npoints=1

c  Read in an initial microstructure in subroutine ppixel, and set up pix(m)
c  with the appropriate phase assignments.
       call ppixel(nx,ny,ns,nphase)
       call assig(ns,nphase,prob)
          do 8055 i=1,nphase
          write(7,9065) i,prob(i)
8055    continue
c  output elastic moduli (bulk and shear) for each phase

c make image (00.pgm) of uncracked microstructure
       call image(ns,0,nx,ny)
       do 8000 micro=1,npoints
c  Count and output the area fractions of the different phases
       call assig(ns,nphase,prob)
          do 8050 i=1,nphase
          write(7,9065) i,prob(i)
9065    format(' Area fraction of phase ',i3,'  is ',f10.8)
8050    continue
c  output elastic moduli (bulk and shear) for each phase
       write(7,*) '  Phase Moduli'
       do 111 i=1,nphase
       write(7,9020) i,phasemod(i,1),phasemod(i,2)
9020    format(' Phase ',i3,' bulk = ',f12.6,' shear = ',f12.6)
111     continue
c  output thermal strains for each phase
       write(7,*) '  Thermal Strains'
       do 119 i=1,nphase
       write(7,9029) i,eigen(i,1),eigen(i,2),eigen(i,3)
9029    format('Phase ',i3,'  ',3f6.2)
119     continue
       call flush(7)

c  (USER) Set inital macrostrains of computational cell
     u(ns+1,1)=0.01
     u(ns+1,2)=0.01
```

```
      u(nss,1)=0.01
      u(nss,2)=0.0
c Apply homogeneous macroscopic strain as the initial condition
c to displacement variables
            do 1050 j=1,ny
         do 1050 i=1,nx
                     m=nx*(j-1)+i
                     x=float(i-1)
                     y=float(j-1)
              u(m,1)=x*u(ns+1,1)+y*u(nss,1)
              u(m,2)=x*u(nss,1)+y*u(ns+1,2)
1050     continue

c  Set up the finite element stiffness matrices,the constant, C,
c  the vector, b, required for the energy. b and C depend on the macrostrains.
c  When they are updated, the values of b and C are updated too via
c  calling subroutine femat.
c  Only compute the thermal strain terms the first time femat is called,
c  (iskip=0) as they are unaffected by later changes (iskip=1) in
c  displacements and macrostrains.
c  Compute initial value of gradient gb and gg=gb*gb.
         iskip=0
            call femat(nx,ny,ns,iskip)
         call energy(nx,ny,ns,utot)
            gg=0.0
         do 100 m2=1,2
         do 100 m=1,nss
         gg=gg+gb(m,m2)*gb(m,m2)
100      continue
            write(7,9042) utot,gg
9042     format(' energy = ',e15.8,'  gg=  ',e15.8)
         call flush(7)

c  Relaxation loop
c  (USER) kmax is the maximum number of times that dembx will be called,
c  with ldemb conjugate gradient steps performed during each call.
c  The total number of conjugate gradient steps allowed for a given elastic
c  computation is kmax*ldemb.
         kmax=400
         ldemb=1000
         ltot=0
         ncrack=200

         do 5000 kkk=1,kmax
```

```
c  Call dembx to implement conjugate gradient routine
      write(7,*) 'Going into dembx, call no. ',kkk
      call dembx(nx,ny,ns,Lstep,gg,gtest,ldemb,kkk)
      ltot=ltot+Lstep
c  Call energy to compute energy after dembx call. If gg < gtest, this
c  will be the final energy.  If gg is still larger than gtest, then this
c  will give an intermediate energy with which to check how the
c  relaxation process is coming along. The call to energy does not
c  change the gradient or the value of gg.
c  Need to first call femat to update the vector b, as the value of the
c  components of b depend on the macrostrains.
      iskip=1
          call femat(nx,ny,ns,iskip)
      call energy(nx,ny,ns,utot)
          write(7,9043) utot,gg,ltot
9043    format(' energy = ',e15.8,' gg=  ',e15.8,' ltot = ',i6)
      call flush(7)

c  If relaxation process is finished, jump out of loop
      if(gg.lt.gtest) goto 444
c  Output stresses, strains, and macrostrains as an additional aid in judging
c  how well the relaxation process is proceeding.
     iswitch=0
     call stress(nx,ny,ns,iswitch)
     write(7,*) ' stresses:  xx,yy,xy'
     write(7,*) strxx,stryy,strxy
     write(7,*) ' strains:  xx,yy,xy'
     write(7,*) sxx,syy,sxy

     write(7,*) ' macrostrains in same order'
2745  format(2i8,3f15.8)
     call flush(7)
     call flush(8)
     write(7,*) 'avg = ',(u(ns+1,1)+u(ns+1,2))/2.

5000    continue

444   iswitch=1
     write(7,*) ' going into stress iswitch=1'
     call flush(7)
     write(7,*) u(ns+1,1),u(ns+1,2),u(ns+2,1)
     call stress(nx,ny,ns,iswitch)
     write(7,*) ' out of stress iswitch=1'
```

```
      call flush(7)
      write(7,*) ' stresses:  xx,yy,xy'
      write(7,*) strxx,stryy,strxy
      write(7,*) ' strains:  xx,yy,xy'
      write(7,*) sxx,syy,sxy
      call flush(7)

      write(7,*) ' macrostrains in same order'
      write(7,*) u(ns+1,1),u(ns+1,2),u(ns+2,1)
      write(7,*) 'avg = ',(u(ns+1,1)+u(ns+1,2))/2.
      call flush(7)

       bulk=(strxx+stryy)/(sxx+syy)/2.
       shear=strxy/sxy
       young=4.*bulk*shear/(bulk+shear)
       poisson=(bulk-shear)/(bulk+shear)
       write(7,*) ' bulk modulus = ',bulk
       write(7,*) ' shear modulus = ',shear
       write(7,*) ' Young modulus = ',young
       write(7,*) ' Poisson ratio = ',poisson
8000    continue

       end

c  Subroutine sets up the stiffness matrices, the linear term in the
c  regular displacements, b, and the constant term, C, which come from
c  the periodic boundary conditions, the term linear in the displacments,
c  T, that comes from the thermal strains, and the constant term Y.

      subroutine femat(nx,ny,ns,iskip)
      real u(546914,2),b(546914,2),T(546914,2)
      real dk(100,4,2,4,2),phasemod(100,2),dndx(4),dndy(4)
      real g(3,3),econ,ck(3,3),cmu(3,3),cmod(100,3,3)
      real es(3,4,2),zcon(2,2,2,2),ss(100,4,2)
      real eigen(100,3),delta(4,2)
      integer is(4),iskip
      integer*4 ib(546914,9)
      integer*2 pix(546914)

          common/list3/ib
        common/list4/pix
          common/list5/dk,b,C,zcon,Y
          common/list6/u
        common/list8/cmod,T,eigen
```

```
      common/list10/phasemod,nphase,ss

     nss=ns+2
c  Generate dk, zcon, T, and Y on first pass. After that they are
c  constant, since they are independent of the macrostrains.  Only b gets
c  upgraded as the macrostrains change.
c  Line number 1221 is the routine for b.
     if(iskip.eq.1) goto 1221

c  initialize stiffness matrices
     do 40 m=1,nphase
     do 40 l=1,2
     do 40 k=1,2
     do 40 j=1,4
     do 40 i=1,4
     dk(m,i,k,j,l)=0.0
40    continue
c  initialize zcon matrix (gives C term for arbitrary macrostrains)
     do 42 i=1,2
     do 42 j=1,2
     do 42 mi=1,2
     do 42 mj=1,2
     zcon(i,mi,j,mj)=0.0
42    continue
c  (USER) An anisotropic elastic moduli tensor could be input at this point,
c  bypassing the following code, which assumes isotropic elasticity
c  (only two independent numbers making up the elastic moduli
c  tensor, the bulk modulus K and the shear modulus G).

c  Set up elastic moduli matrices for each kind of element
c  ck and cmu are the bulk modulus and shear modulus matrices, which
c  need to multiplied by the actual bulk and shear moduli in each phase.

     ck(1,1)=1.0
     ck(1,2)=1.0
     ck(1,3)=0.0
     ck(2,1)=1.0
     ck(2,2)=1.0
     ck(2,3)=0.0
     ck(3,1)=0.0
     ck(3,2)=0.0
     ck(3,3)=0.0

     cmu(1,1)=1.0
```

```
      cmu(1,2)=-1.0
      cmu(1,3)=0.0
      cmu(2,1)=-1.0
      cmu(2,2)=1.0
      cmu(2,3)=0.0
      cmu(3,1)=0.0
      cmu(3,2)=0.0
      cmu(3,3)=1.0

      do 31 k=1,nphase
      do 21 j=1,3
      do 21 i=1,3
      cmod(k,i,j)=phasemod(k,1)*ck(i,j)+phasemod(k,2)*cmu(i,j)
21    continue
31    continue
c  Set up Simpson's integration rule weight vector
      do 30 j=1,3
      do 30 i=1,3
      nm=0
      if(i.eq.2) nm=nm+1
      if(j.eq.2) nm=nm+1
      g(i,j)=4.0**nm
30    continue

c  Loop over the nphase kinds of pixels and
c  Simpson's rule quadrature points in order to compute the stiffness
c  matrices.  Stiffness matrices of bilinear finite elements are quadratic
c  in x and y, so that Simpson's rule quadrature gives exact results.
      do 4000 ijk=1,nphase
      do 3000 j=1,3
      do 3000 i=1,3
      x=float(i-1)/2.0
      y=float(j-1)/2.0
c  dndx means the negative derivative with respect to x, of the shape
c  matrix N (see manual, Sec. 2.2), dndy is similar.
      dndx(1)=-(1.0-y)
      dndx(2)=(1.0-y)
      dndx(3)=y
      dndx(4)=-y
      dndy(1)=-(1.0-x)
      dndy(2)=-x
      dndy(3)=x
      dndy(4)=(1.0-x)
c  now build strain matrix
```

```
      do 2799 n1=1,3
      do 2799 n2=1,4
      do 2799 n3=1,2
      es(n1,n2,n3)=0.0
2799  continue
      do 2797 n=1,4
      es(1,n,1)=dndx(n)
      es(2,n,2)=dndy(n)
      es(3,n,1)=dndy(n)
      es(3,n,2)=dndx(n)
2797  continue
c  now do matrix multiply to determine value at (x,y), multiply by
c  proper weight, and sum into dk, the stiffness matrix
      do 900 mm=1,2
      do 900 nn=1,2
      do 900 ii=1,4
      do 900 jj=1,4
c  define sum over strain matrices and elastic moduli matrix for
c  stiffness matrix
      sum=0.0
      do 890 kk=1,3
      do 890 ll=1,3
      sum=sum+es(kk,ii,mm)*cmod(ijk,kk,ll)*es(ll,jj,nn)
890   continue
      dk(ijk,ii,mm,jj,nn)=dk(ijk,ii,mm,jj,nn)+g(i,j)*sum/36.
900   continue
3000  continue
4000  continue

c  Now compute the ss matrices, which give the thermal strain terms
c  for the i'th phase, single pixel.

      dndx(1)=-0.5
      dndx(2)=0.5
      dndx(3)=0.5
      dndx(4)=-0.5
      dndy(1)=-0.5
      dndy(2)=-0.5
      dndy(3)=0.5
      dndy(4)=0.5
c  now build average strain matrix
      do 3799 n1=1,3
      do 3799 n2=1,4
      do 3799 n3=1,2
```

```
      es(n1,n2,n3)=0.0
3799  continue
      do 3797 n=1,4
      es(1,n,1)=dndx(n)
      es(2,n,2)=dndy(n)
      es(3,n,1)=dndy(n)
      es(3,n,2)=dndx(n)
3797  continue
      do 3598 mmm=1,nphase
      do 3798 nn=1,2
      do 3798 mm=1,4
      sum=0.0
      do 3698 nm=1,3
      do 3698 n=1,3
      sum=sum+cmod(mmm,n,nm)*es(n,mm,nn)*eigen(mmm,nm)
3698  continue
      ss(mmm,mm,nn)=sum
3798  continue
3598  continue

c  now call subroutine const to generate zcon
c  zcon is a (2,2) x (2,2) matrix
      call const(dk,ns,zcon,nx,ny)

c  Now set up linear term, T, for thermal energy. It does not depend
c  on the macrostrains or displacements, so there is no need to update it
c  as the macrostrains change. T is built up out of the ss matrices.

      nss=ns+2
      do 6066 m2=1,2
      do 6066 m=1,nss
      T(m,m2)=0.0
6066  continue

c  For all cases, the correspondence between 1-4 finite element node
c  labels and the 1-9 neighbor labels is (see Table 4 in manual):
c  1:ib(m,9), 2:ib(m,3),3:ib(m,2),4:ib(m,1)
      is(1)=9
      is(2)=3
      is(3)=2
      is(4)=1
c  Do all points, but no macrostrain terms
c  note:  factor of 2 on linear thermal term is cancelled
c  by factor of 1/2 out in front of total energy term
```

```
      do 6601 j=1,ny
      do 6601 i=1,nx
      m=nx*(j-1)+i
      do 6600 mm=1,4
      do 6600 nn=1,2
      T(ib(m,is(mm)),nn)=T(ib(m,is(mm)),nn)-ss(pix(m),mm,nn)
6600  continue
6601  continue

c  now need to pick up and sum in all terms multiplying macrostrains
      do 7788 ipp=1,2
      do 7788 jpp=1,2
      if(ipp.eq.2.and.jpp.eq.2) goto 7788
      exx=0.0
      eyy=0.0
      exy=0.0
      if(ipp.eq.1.and.jpp.eq.1) exx=1.0
      if(ipp.eq.1.and.jpp.eq.2) eyy=1.0
      if(ipp.eq.2.and.jpp.eq.1) exy=1.0

c  x=nx face
      do 6001 i2=1,2
      do 6001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
6001  continue

      do 6000 j=1,ny-1
      m=j*nx
      do 6900 nn=1,2
      do 6900 mm=1,4
      T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6900  continue
6000  continue

c  y=ny face
      do 6011 i2=1,2
      do 6011 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.3.or.i4.eq.4) then
      delta(i4,1)=exy*ny
```

```
    delta(i4,2)=eyy*ny
    end if
6011  continue
    do 6010 i=1,nx-1
    m=nx*(ny-1)+i
    do 6901 nn=1,2
    do 6901 mm=1,4
    T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6901  continue
6010  continue

c  x=nx y=ny corner
    do 6031 i2=1,2
    do 6031 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.2) then
    delta(i4,1)=exx*nx
    delta(i4,2)=exy*nx
    end if
    if(i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
    if(i4.eq.3) then
    delta(i4,1)=exy*ny+exx*nx
    delta(i4,2)=eyy*ny+exy*nx
    end if
6031  continue
    m=nx*ny
    do 6903 nn=1,2
    do 6903 mm=1,4
    T(ns+ipp,jpp)=T(ns+ipp,jpp)-ss(pix(m),mm,nn)*delta(mm,nn)
6903  continue
6030  continue
7788  continue

c  now compute Y, the 0.5(eigen)Cij(eigen) energy, doesn't ever change
c  with macrostrain or displacements
    Y=0.0
    do 8811 m=1,ns
    do 8811 n=1,3
    do 8811 nn=1,3
    Y=Y+0.5*eigen(pix(m),n)*cmod(pix(m),n,nn)*eigen(pix(m),nn)
8811  continue
```

```
c  Following needs to be run after every change in macrostrain
c  when energy is recomputed.

1221  continue
c  Use auxiliary variables (exx, etc.) instead of u() variable, for
c  convenience, and to make the following code easier to read.
     exx=u(ns+1,1)
     eyy=u(ns+1,2)
     exy=u(nss,1)
c  Now set up vector for linear term that comes from periodic boundary
c  conditions.  Notation and conventions same as for T term.
c  This is done using the stiffness matrices, and the periodic terms
c  in the macrostrains.  It is easier to set up b in this way than to
c  analytically write out all the terms involved.

     do 5000 m2=1,2
     do 5000 m=1,ns
     b(m,m2)=0.0
5000  continue
c  For all cases, the correspondence between 1-4 finite element node
c  labels and the 1-9 neighbor labels is (see Table 4 in manual):
c  1:ib(m,9), 2:ib(m,3),3:ib(m,2),4:ib(m,1)
     is(1)=9
     is(2)=3
     is(3)=2
     is(4)=1

     C=0.0
c  x=nx face
     do 2001 i2=1,2
     do 2001 i4=1,4
     delta(i4,i2)=0.0
     if(i4.eq.2.or.i4.eq.3) then
     delta(i4,1)=exx*nx
     delta(i4,2)=exy*nx
     end if
2001  continue

     do 2000 j=1,ny-1
     m=j*nx
     do 1900 nn=1,2
     do 1900 mm=1,4
     sum=0.0
```

```
    do 1899 m2=1,2
    do 1899 m4=1,4
    sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
1899  continue
    b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1900  continue
2000  continue
c  y=ny face
    do 2011 i2=1,2
    do 2011 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.3.or.i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
2011  continue
    do 2010 i=1,nx-1
    m=nx*(ny-1)+i
    do 1901 nn=1,2
    do 1901 mm=1,4
    sum=0.0
    do 2099 m2=1,2
    do 2099 m4=1,4
    sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2099  continue
    b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1901  continue
2010  continue

c  x=nx y=ny corner
    do 2031 i2=1,2
    do 2031 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.2) then
    delta(i4,1)=exx*nx
    delta(i4,2)=exy*nx
    end if
    if(i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
    if(i4.eq.3) then
    delta(i4,1)=exy*ny+exx*nx
    delta(i4,2)=eyy*ny+exy*nx
```

```
      end if
2031  continue
      m=nx*ny
      do 1903 nn=1,2
      do 1903 mm=1,4
      sum=0.0
      do 2029 m2=1,2
      do 2029 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2029  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1903  continue
2030  continue

      return
      end

c  Subroutine that computes derivatives of the b-vector with respect
c  to the macrostrains.  Since b is linear in the macrostrains, the
c  derivative with respect to any one of them can be computed simply
c  by letting that macrostrain, within the subroutine, be equal to one,
c  and all the other macrostrains to be zero.
c  Very similar to 1221 loop in femat for b.

      subroutine bgrad(nx,ny,ns,exx,eyy,exy)
      real b(546914,2)
      real dk(100,4,2,4,2),delta(4,2),zcon(2,2,2,2)
      integer is(4)
      integer*4 ib(546914,9)
      integer*2 pix(546914)

      common/list3/ib
      common/list4/pix
      common/list5/dk,b,C,zcon,Y

c  exx, eyy, and exy are the artificial macrostrains used
c  to get the gradient terms (appropriate combinations of 1's and 0's).

c  Set up vector for linear term

      do 5000 m2=1,2
      do 5000 m=1,ns
      b(m,m2)=0.0
5000  continue
```

```
      is(1)=9
      is(2)=3
      is(3)=2
      is(4)=1

c  x=nx face
      do 2001 i2=1,2
      do 2001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
2001  continue

      do 2000 j=1,ny-1
      m=j*nx
      do 1900 nn=1,2
      do 1900 mm=1,4
      sum=0.0
      do 1899 m2=1,2
      do 1899 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
1899  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1900  continue
2000  continue
c  y=ny face
      do 2011 i2=1,2
      do 2011 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.3.or.i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
2011  continue
      do 2010 i=1,nx-1
      m=nx*(ny-1)+i
      do 1901 nn=1,2
      do 1901 mm=1,4
      sum=0.0
      do 2099 m2=1,2
      do 2099 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
```

```
2099  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1901  continue
2010  continue

c  x=nx y=ny corner
      do 2031 i2=1,2
      do 2031 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2) then
      delta(i4,1)=exx*nx
      delta(i4,2)=exy*nx
      end if
      if(i4.eq.4) then
      delta(i4,1)=exy*ny
      delta(i4,2)=eyy*ny
      end if
      if(i4.eq.3) then
      delta(i4,1)=exy*ny+exx*nx
      delta(i4,2)=eyy*ny+exy*nx
      end if
2031  continue
      m=nx*ny
      do 1903 nn=1,2
      do 1903 mm=1,4
      sum=0.0
      do 2029 m2=1,2
      do 2029 m4=1,4
      sum=sum+delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)
2029  continue
      b(ib(m,is(mm)),nn)=b(ib(m,is(mm)),nn)+sum
1903  continue
2030  continue

      return
      end

c  Subroutine computes the quadratic term in the macrostrains, that comes
c  from the periodic boundary conditions, and sets it up as a
c  (2,2) x (2,2) matrix that couples to the three macrostrains

      subroutine const(dk,ns,zcon,nx,ny)
      real dk(100,4,2,4,2),zcon(2,2,2,2),delta(4,2)
      real pp(4,4),s(4,4)
```

```
      integer*2 pix(546914)
      common/list4/pix

c  routine to set up 4 x 4 matrix for energy term involving macro-strains
c  only, pulled out of femat (really 3 x 3, as ns+2,2 term is not used).
c  Idea is to evaluate the quadratic term in the macrostrains, C,
c  repeatedly for choices of strain like
c  exx=1, exy=1, all others = 0, build up 10 choices, then recombine
c  to get matrix elements by themselves

      nss=ns+2

      do 1111 i=1,4
      do 1111 j=1,4
      s(i,j)=0.0
      pp(i,j)=0.0
1111  continue

      do 5000 ii=1,4
      do 5000 jj=ii,4
      econ=0.0
      exx=0.0
      eyy=0.0
      exy=0.0
      if(ii.eq.1.and.jj.eq.1) exx=1.0
      if(ii.eq.2.and.jj.eq.2) eyy=1.0
      if(ii.eq.3.and.jj.eq.3) exy=1.0
      if(ii.eq.1.and.jj.eq.2) then
      exx=1.0
      eyy=1.0
      end if
      if(ii.eq.1.and.jj.eq.3) then
      exx=1.0
      exy=1.0
      end if
      if(ii.eq.2.and.jj.eq.3) then
      eyy=1.0
      exy=1.0
      end if
c  x=nx face
      do 2001 i2=1,2
      do 2001 i4=1,4
      delta(i4,i2)=0.0
      if(i4.eq.2.or.i4.eq.3) then
```

```
    delta(i4,1)=exx*nx
    delta(i4,2)=exy*nx
    end if
2001  continue

    do 2000 j=1,ny-1
    m=j*nx
    do 1900 nn=1,2
    do 1900 mm=1,4
    do 1899 m2=1,2
    do 1899 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
1899  continue
1900  continue
2000  continue
c  y=ny face
    do 2011 i2=1,2
    do 2011 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.3.or.i4.eq.4) then
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
2011  continue
    do 2010 i=1,nx-1
    m=nx*(ny-1)+i
    do 1901 nn=1,2
    do 1901 mm=1,4
    do 2099 m2=1,2
    do 2099 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
2099  continue
1901  continue
2010  continue

c  x=nx y=ny corner
    do 2031 i2=1,2
    do 2031 i4=1,4
    delta(i4,i2)=0.0
    if(i4.eq.2) then
    delta(i4,1)=exx*nx
    delta(i4,2)=exy*nx
    end if
    if(i4.eq.4) then
```

```
    delta(i4,1)=exy*ny
    delta(i4,2)=eyy*ny
    end if
    if(i4.eq.3) then
    delta(i4,1)=exy*ny+exx*nx
    delta(i4,2)=eyy*ny+exy*nx
    end if
2031  continue
    m=nx*ny
    do 1903 nn=1,2
    do 1903 mm=1,4
    do 2029 m2=1,2
    do 2029 m4=1,4
    econ=econ+0.5*delta(m4,m2)*dk(pix(m),m4,m2,mm,nn)*delta(mm,nn)
2029  continue
1903  continue
2030  continue
    pp(ii,jj)=econ*2.

5000  continue
    do 6000 i=1,4
    do 6000 j=i,4
    if(i.eq.j) s(i,j)=pp(i,j)
    if(i.ne.j) then
    s(i,j)=pp(i,j)-pp(i,i)-pp(j,j)
    end if
6000  continue
    do 7000 i=1,4
    do 7000 j=1,4
    pp(i,j)=0.5*(s(i,j)+s(j,i))
7000  continue
c  now map pp(i,j) into zcon(2,2,2,2)
    do 7200 i=1,2
    do 7200 j=1,2
    do 7200 mi=1,2
    do 7200 mj=1,2
    if(i.eq.1) ii=i+mi-1
    if(i.eq.2) ii=i+mi
    if(j.eq.1) jj=j+mj-1
    if(j.eq.2) jj=j+mj
    zcon(i,mi,j,mj)=pp(ii,jj)
    if(i.eq.2.and.mi.eq.2) zcon(i,mi,j,mj)=0.0
    if(j.eq.2.and.mj.eq.2) zcon(i,mi,j,mj)=0.0
7200  continue
```

```
      return
      end

c  Subroutine computes the total energy, utot, and the gradient, gb,
c  for the regular displacements as well as for the macrostrains

      subroutine energy(nx,ny,ns,utot)

          real u(546914,2),gb(546914,2)
          real b(546914,2),T(546914,2)
          real cmod(100,3,3),pk(3,4,2),eigen(100,3)
          real dk(100,4,2,4,2),zcon(2,2,2,2),C
          integer*4 ib(546914,9)
          integer*2 pix(546914)

          common/list3/ib
        common/list4/pix
          common/list5/dk,b,C,zcon,Y
          common/list6/u
          common/list7/gb
        common/list8/cmod,T,eigen

        nss=ns+2

        do 2090 m2=1,2
          do 2090 m=1,nss
          gb(m,m2)=0.0
2090    continue
c  Do global matrix multiply via small stiffness matrices, Ah = (A)(h).
c  The long statement below correctly brings in all the terms from
c  the global matrix A using only the small stiffness matrices.

        do 3000 j=1,2
        do 3000 n=1,2
          do 3000 m=1,ns
     gb(m,j)=gb(m,j)+u(ib(m,1),n)*( dk(pix(ib(m,9)),1,j,4,n)
    &+dk(pix(ib(m,7)),2,j,3,n) )+
    &u(ib(m,2),n)*( dk(pix(ib(m,9)),1,j,3,n) )+
    &u(ib(m,3),n)*( dk(pix(ib(m,9)),1,j,2,n)+dk(pix(ib(m,5)),4,j,3,n))+
    &u(ib(m,4),n)*( dk(pix(ib(m,5)),4,j,2,n) )+
    &u(ib(m,5),n)*( dk(pix(ib(m,6)),3,j,2,n)+dk(pix(ib(m,5)),4,j,1,n))+
    &u(ib(m,6),n)*( dk(pix(ib(m,6)),3,j,1,n) )+
    &u(ib(m,7),n)*( dk(pix(ib(m,6)),3,j,4,n)+dk(pix(ib(m,7)),2,j,1,n))+
```

```
     &u(ib(m,8),n)*( dk(pix(ib(m,7)),2,j,4,n) )+
     &u(ib(m,9),n)*( dk(pix(ib(m,9)),1,j,1,n)
     &+dk(pix(ib(m,7)),2,j,2,n)+
     &dk(pix(ib(m,6)),3,j,3,n)+dk(pix(ib(m,5)),4,j,4,n) )
3000    continue

      utot=0.0

      gtot=0.0
      do 3100 m2=1,2
      do 3100 m=1,ns
      utot=utot+0.5*u(m,m2)*gb(m,m2)+b(m,m2)*u(m,m2)
c  this is gradient of energy with respect to normal displacements
      gb(m,m2)=gb(m,m2)+b(m,m2)
3100  continue

c  compute "constant" macrostrain energy term
      C=0.0
      do 7200 i=1,2
      do 7200 j=1,2
      do 7200 mi=1,2
      do 7200 mj=1,2
      C=C+0.5*u(ns+i,mi)*zcon(i,mi,j,mj)*u(ns+j,mj)
7200  continue
      utot=utot+C
c  now add in constant term from thermal energy, Y
      utot=utot+Y
c  now add in linear term in thermal energy
      do 7171 m2=1,2
      do 7171 m=1,nss
      utot=utot+T(m,m2)*u(m,m2)
7171  continue

c  now compute gradient with respect to macrostrains
c  put in piece from first derivative of zcon quadratic term
      do 7300 i=1,2
      do 7300 mi=1,2
      sum=0.0
      do 7250 j=1,2
      do 7250 mj=1,2
      sum=sum+zcon(i,mi,j,mj)*u(ns+j,mj)
7250  continue
      gb(ns+i,mi)=sum
7300  continue
```

```
c  add in piece of gradient, for displacements as well as macrostrains,
c  that come from linear term in thermal energy
     do 3150 m2=1,2
     do 3150 m=1,nss
     gb(m,m2)=gb(m,m2)+T(m,m2)
3150  continue
c  now generate part that comes from b . u term
c  do by calling b generation with appropriate macrostrain set to 1 to
c  get that partial derivative, just use bgrad (taken from femat),
c  skip dk and zcon part
     do 8100 ii=1,3
     exx=0.0
     eyy=0.0
     exy=0.0
     if(ii.eq.1) exx=1.0
     if(ii.eq.2) eyy=1.0
     if(ii.eq.3) exy=1.0
     call bgrad(nx,ny,ns,exx,eyy,exy)
     sum=0.0
      do 8200 m2=1,2
          do 8200 m=1,ns
          sum=sum+u(m,m2)*b(m,m2)
8200    continue
          if(ii.eq.1) gb(ns+1,1)=gb(ns+1,1)+sum
          if(ii.eq.2) gb(ns+1,2)=gb(ns+1,2)+sum
          if(ii.eq.3) gb(ns+2,1)=gb(ns+2,1)+sum
8100    continue
c zero out macrostrain part of gradient
       gb(ns+1,1)=0.0
       gb(ns+1,2)=0.0
       gb(ns+2,1)=0.0
       gb(ns+2,2)=0.0
       return
       end

c  Subroutine that carries out the conjugate gradient relaxation process

     subroutine dembx(nx,ny,ns,Lstep,gg,gtest,ldemb,kkk)

     real u(546914,2),gb(546914,2),b(546914,2)
     real h(546914,2),Ah(546914,2)
     real dk(100,4,2,4,2),zcon(2,2,2,2)
     real lambda,gamma
     integer*4 ib(546914,9)
```

```
      integer*2 pix(546914)

       common/list2/h,Ah
          common/list3/ib
       common/list4/pix
          common/list5/dk,b,C,zcon,Y
          common/list6/u
          common/list7/gb

      nss=ns+2
c  Initialize the conjugate direction vector on first call to dembx only.
c  For calls to dembx after the first, we want to continue using the value
c  of h determined in the previous call.  Of course, if npoints is greater
c  than 1, then this initialization step will be run each time a new
c  microstructure is used, as kkk will be reset to 1 every time the
c  counter micro is increased.
      if(kkk.eq.1) then
      do 500 m2=1,2
      do 500 m=1,nss
      h(m,m2)=gb(m,m2)
500   continue
      end if
c  Lstep counts the number of conjugate gradient steps taken
c  in each call to dembx
      Lstep=0

      do 800 ijk=1,ldemb
      Lstep=Lstep+1

      do 290 m2=1,2
      do 290 m=1,nss
      Ah(m,m2)=0.0
290   continue
c  Do global matrix multiply via small stiffness matrices, Ah = (A)(h).
c  The long statement below correctly brings in all the terms from
c  the global matrix A using only the small stiffness matrices.
       do 400 j=1,2
       do 400 n=1,2
          do 400 m=1,ns
      Ah(m,j)=Ah(m,j)+h(ib(m,1),n)*( dk(pix(ib(m,9)),1,j,4,n)
     &+dk(pix(ib(m,7)),2,j,3,n) )+
     &h(ib(m,2),n)*( dk(pix(ib(m,9)),1,j,3,n) )+
     &h(ib(m,3),n)*( dk(pix(ib(m,9)),1,j,2,n)+dk(pix(ib(m,5)),4,j,3,n))+
     &h(ib(m,4),n)*( dk(pix(ib(m,5)),4,j,2,n) )+
```

```
     &h(ib(m,5),n)*( dk(pix(ib(m,6)),3,j,2,n)+dk(pix(ib(m,5)),4,j,1,n))+
     &h(ib(m,6),n)*( dk(pix(ib(m,6)),3,j,1,n) )+
     &h(ib(m,7),n)*( dk(pix(ib(m,6)),3,j,4,n)+dk(pix(ib(m,7)),2,j,1,n))+
     &h(ib(m,8),n)*( dk(pix(ib(m,7)),2,j,4,n) )+
     &h(ib(m,9),n)*( dk(pix(ib(m,9)),1,j,1,n)
     &+dk(pix(ib(m,7)),2,j,2,n)+
     &dk(pix(ib(m,6)),3,j,3,n)+dk(pix(ib(m,5)),4,j,4,n) )
400    continue

c  The above accurately gives the second derivative matrix with respect
c  to nodal displacements, but fails to give the 2nd derivative terms that
c  include the macrostrains [ du d(strain) and d(strain)d(strain) ].
c  Use repeated calls to bgrad to generate mixed 2nd derivatives terms,
c  plus use zcon in order to correct the matrix multiply and correctly bring
c  in macrostrain terms (see manual, Sec. 2.4).
      do 8100 ii=1,3
      e11=0.0
      e22=0.0
      e12=0.0
      if(ii.eq.1) e11=1.0
      if(ii.eq.2) e22=1.0
      if(ii.eq.3) e12=1.0
      call bgrad(nx,ny,ns,e11,e22,e12)
c  now fill in terms from matrix multiply
c  right hand sides, 1 to ns
      do 3333 m=1,ns
      do 3333 m1=1,2
      if(ii.eq.1) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(ns+1,1)
      if(ii.eq.2) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(ns+1,2)
      if(ii.eq.3) Ah(m,m1)=Ah(m,m1)+b(m,m1)*h(nss,1)
3333  continue
c  now do across bottom, 1 to ns
      do 3334 m=1,ns
      if(ii.eq.1) Ah(ns+1,1)=Ah(ns+1,1)+b(m,1)*h(m,1)+
     +b(m,2)*h(m,2)
      if(ii.eq.2) Ah(ns+1,2)=Ah(ns+1,2)+b(m,1)*h(m,1)+
     +b(m,2)*h(m,2)
      if(ii.eq.3) Ah(nss,1)=Ah(nss,1)+b(m,1)*h(m,1)+
     +b(m,2)*h(m,2)
3334  continue
c  now do righthand corner terms, ns+1 to nss
      do 3335 m=1,2
      do 3335 m1=1,2
      if(ii.eq.1) Ah(ns+1,1)=Ah(ns+1,1)+zcon(1,1,m,m1)*h(ns+m,m1)
```

```
    if(ii.eq.2) Ah(ns+1,2)=Ah(ns+1,2)+zcon(1,2,m,m1)*h(ns+m,m1)
    if(ii.eq.3) Ah(nss,1)=Ah(nss,1)+zcon(2,1,m,m1)*h(ns+m,m1)
3335  continue

8100    continue

c zero out macrostrain part of Ah
    Ah(ns+1,1)=0.0
    Ah(ns+1,2)=0.0
    Ah(ns+2,1)=0.0
    Ah(ns+2,2)=0.0

    hAh=0.0
    do 530 m2=1,2
    do 530 m=1,nss
    hAh=hAh+h(m,m2)*Ah(m,m2)
530   continue

    lambda=gg/hAh
    do 540 m2=1,2
    do 540 m=1,nss
    u(m,m2)=u(m,m2)-lambda*h(m,m2)
    gb(m,m2)=gb(m,m2)-lambda*Ah(m,m2)
540   continue

    gglast=gg
    gg=0.0
    do 550 m2=1,2
    do 550 m=1,nss
    gg=gg+gb(m,m2)*gb(m,m2)
550   continue
    if(gg.lt.gtest) goto 1000

    gamma=gg/gglast
    do 570 m2=1,2
    do 570 m=1,nss
    h(m,m2)=gb(m,m2)+gamma*h(m,m2)
570   continue

800   continue

1000  continue
    return
    end
```

```
c  Subroutine that computes the three average stresses
c  and three average strains.

      subroutine stress(nx,ny,ns,iswitch)

      real u(546914,2),uu(4,2),pmax(546914,2)
      real T(546914,2),eigen(100,3)
      real dndx(4),dndy(4),es(3,4,2),cmod(100,3,3)
      integer*4 ib(546914,9)
      integer*2 pix(546914)

      common/list1/strxx,stryy,strxy
      common/list3/ib
      common/list4/pix
      common/list6/u
      common/list8/cmod,T,eigen
      common/list11/sxx,syy,sxy
      common/list12/pmax

      nss=ns+2
      exx=u(ns+1,1)
      eyy=u(ns+1,2)
      exy=u(nss,1)
      if(iswitch.eq.1) then
c       open(unit=12,file='101bw-stress-strain-allmat.txt')
c       write(12,1134) nx,ny
c1134  format(2i5)
      end if

c set up single pixel strain matrix

      dndx(1)=-0.5
      dndx(2)=0.5
      dndx(3)=0.5
      dndx(4)=-0.5
      dndy(1)=-0.5
      dndy(2)=-0.5
      dndy(3)=0.5
      dndy(4)=0.5

c initialize pmax for iswitch=1
      if(iswitch.eq.1) then
      do 5555 m=1,ns
```

```
      pmax(m,1)=0.0
      pmax(m,2)=m
5555  continue
      end if
c  Build average strain matrix, follows code in femat, but for average
c  strain over a pixel, not the strain at a point
      do 2799 n1=1,3
      do 2799 n2=1,4
      do 2799 n3=1,2
      es(n1,n2,n3)=0.0
2799  continue
      do 2797 n=1,4
      es(1,n,1)=dndx(n)
      es(2,n,2)=dndy(n)
      es(3,n,1)=dndy(n)
      es(3,n,2)=dndx(n)
2797  continue
c  now compute average stresses and strains in each pixel
      sxx=0.0
      syy=0.0
      sxy=0.0
      strxx=0.0
      stryy=0.0
      strxy=0.0
      do 470 j=1,ny
      do 470 i=1,nx
      m=(j-1)*nx+i
      if(iswitch.eq.1) then
c       write(14,*) ' i j in stress m ',i,j,m
c       call flush(14)
      end if
c  load in elements of 4-vector using pd. bd. conds.
      do 9898 mm=1,2
      uu(1,mm)=u(m,mm)
      uu(2,mm)=u(ib(m,3),mm)
      uu(3,mm)=u(ib(m,2),mm)
      uu(4,mm)=u(ib(m,1),mm)
9898  continue
c  Correct for periodic boundary conditions, some displacements are wrong
c  for a pixel on a periodic boundary.  Since they come from an opposite
c  face, need to put in applied strain to correct them.
      if(i.eq.nx) then
      uu(2,1)=uu(2,1)+exx*nx
      uu(2,2)=uu(2,2)+exy*nx
```

```
      uu(3,1)=uu(3,1)+exx*nx
      uu(3,2)=uu(3,2)+exy*nx
      end if
      if(j.eq.ny) then
      uu(3,1)=uu(3,1)+exy*ny
      uu(3,2)=uu(3,2)+eyy*ny
      uu(4,1)=uu(4,1)+exy*ny
      uu(4,2)=uu(4,2)+eyy*ny
      end if
      str11=0.0
      str22=0.0
      str12=0.0
      s11=0.0
      s22=0.0
      s12=0.0
c********compute average stress and strain tensor in each pixel*************
c  First put thermal strain-induced stresses into stress tensor
      do 465 n=1,3
      str11=str11-cmod(pix(m),1,n)*eigen(pix(m),n)
      str22=str22-cmod(pix(m),2,n)*eigen(pix(m),n)
      str12=str12-cmod(pix(m),3,n)*eigen(pix(m),n)
465   continue
      do 466 n2=1,2
      do 466 n4=1,4
c  compute non-thermal strains in each pixel
      s11=s11+es(1,n4,n2)*uu(n4,n2)
      s22=s22+es(2,n4,n2)*uu(n4,n2)
      s12=s12+es(3,n4,n2)*uu(n4,n2)
      do 466 n=1,3
c  compute stresses in each pixel that include both non-thermal
c  and thermal strains
      str11=str11+cmod(pix(m),1,n)*es(n,n4,n2)*uu(n4,n2)
      str22=str22+cmod(pix(m),2,n)*es(n,n4,n2)*uu(n4,n2)
      str12=str12+cmod(pix(m),3,n)*es(n,n4,n2)*uu(n4,n2)
466   continue
c  Sum local stresses and strains into global stresses and strains
      strxx=strxx+str11
      stryy=stryy+str22
      strxy=strxy+str12
      sxx=sxx+s11
      syy=syy+s22
      sxy=sxy+s12
      if(iswitch.eq.1) then
      sa=0.5*((str11+str22)+sqrt((str11-str22)**2+4.*str12*str12))
```

```
      sb=0.5*((str11+str22)-sqrt((str11-str22)**2+4.*str12*str12))
      ea=0.5*((s11+s22)+sqrt((s11-s22)**2+4.*s12*s12))
      eb=0.5*((s11+s22)-sqrt((s11-s22)**2+4.*s12*s12))
      if(ea.gt.eb) pmax(m,1)=ea
      if(eb.gt.ea) pmax(m,2)=eb
      pmax(m,2)=m
c     write(14,*) m,pmax(m,1),pmax(m,2)
c     call flush(14)
c     write(12,1137) i,j,str11,str22,str12,s11,s22,s12
c     write(12,1138) i,j,sa,sb,ea,eb
1137  format(2i4,6f15.9)
1138  format(2i4,4f15.9)
      end if
470   continue

c  Area average global stresses and strains

            strxx=strxx/float(ns)
            stryy=stryy/float(ns)
            strxy=strxy/float(ns)
            sxx=sxx/float(ns)
            syy=syy/float(ns)
            sxy=sxy/float(ns)

      return
      end

c  Subroutine to count volume fractions of various phases

      subroutine assig(ns,nphase,prob)
      integer*2 pix(546914)
      real prob(100)
      common/list4/pix

            do 999 i=1,nphase
            prob(i)=0.0
999         continue

      do 1000 m=1,ns
      do 1000 i=1,nphase
        if(pix(m).eq.i) then
            prob(i)=prob(i)+1
            end if
1000    continue
```

```
          do 998 i=1,nphase
          prob(i)=prob(i)/float(ns)
998       continue

        return
        end

c  Subroutine to set up image of microstructure
c  adds border all around of mortar (phase 1)

      subroutine ppixel(nx,ny,ns,nphase)
      integer*2 pix(546914)
      common/list4/pix

      call srand(-147)
c  (USER)  If you want to set up a test image inside the program, instead
c  of reading it in from a file, this should be done inside this subroutine.

      open (9,file='micro.in2')

      do 200 j=1,ny
      do 200 i=1,nx
      m=nx*(j-1)+i
      read(9,*) pix(m)
c turn white border into black matrix
!     if(pix(m).eq.4) pix(m)=1
200   continue
      close(9)

c make phases 5-14 matrix phases but with slightly different moduli

c     do 300 j=1,ny
c     do 300 i=1,nx
c     m=nx*(j-1)+i
c     if(pix(m).eq.1) then
c     mm=5+10*rand(0)
c     if(mm.eq.15) mm=14
c     pix(m)=mm
c     end if
c00   continue
c     close(19)

c   Check for wrong phase labels--less than 1 or greater than nphase
```

```
      do 500 m=1,ns
     if(pix(m).lt.1) then
      write(7,*) 'Phase label in pix < 1--error at ',m,pix(m)
     end if
      if(pix(m).gt.nphase) then
       write(7,*) 'Phase label in pix > nphase--error at ',m,pix(m)
      end if
500    continue

     return
     end
     subroutine image(ns,micro,nx,ny)
     integer*2 pix(546914)
     common/list4/pix
     character*6 namef
     character*9 namefff
     character*31 nameff

     namef(1:2)='00'
     namef(3:6)='.pgm'
     if(micro.lt.10) write(namef(2:2),'(I1)') micro
     if(micro.ge.10) write(namef(1:2),'(I2)') micro
     open(unit=14,file=namef)
     write(14,133)
     write(14,134) nx,ny
     write(14,157)
133   format('P2')
134   format(i4,1x,i4)
157   format('4')

     do 741 j=ny,1,-1
     do 741 i=1,nx
     m=nx*(j-1)+i
     mm=pix(m)
c make irregular matrix all look the same in the image
     if(mm.ge.5.and.mm.le.14) mm=1
     write(14,22) mm
22    format(i1)
741   continue
     close(14)

c convert pgm file to gif

     nameff(1:8)='convert '
```

```
      nameff(9:14)=namef
      nameff(15:24)=' -depth 8 '
      nameff(25:30)=namef
      nameff(28:30)='gif'
      nameff(31:31)=' '
      call system(nameff)
c rm pgm files
      namefff(1:3)='rm '
      namefff(4:9)=namef
      call system(namefff)

      return
      end
```