Percolation in Two-Dimensional Grain Boundary Structures and Polycrystal Property Closures

David T. Fullwood

Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation

https://scholarsarchive.byu.edu/etd/676

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
PERCOLATION IN TWO-DIMENSIONAL GRAIN BOUNDARY STRUCTURES,
AND POLYCRYSTAL PROPERTY CLOSURES

by

David T. Fullwood

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Mechanical Engineering Department

Brigham Young University

December 2005
of a thesis submitted by

David T. Fullwood

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

________________________________________________________________________

Date  Brent L. Adams, Chair

________________________________________________________________________

Date  Michael Dorff

________________________________________________________________________

Date  Brian D. Jensen
As chair of the candidate’s graduate committee, I have read the dissertation of David T. Fullwood in its final form and have found that (1) its format, citations and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date  Brent L. Adams
      Chair, Graduate Committee

Accepted for the Department

Matthew R. Jones
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A Fulton College of Engineering and Technology
ABSTRACT

PERCOLATION IN TWO-DIMENSIONAL GRAIN BOUNDARY STRUCTURES, AND POLYCRYSTAL PROPERTY CLOSURES

David T. Fullwood
Mechanical Engineering Department
Master of Science

The thesis addresses two topics in the study of material properties as determined by the microstructure. The first topic involves percolation as a tool in relating the grain boundary structure to global properties such as fracture and corrosion resistance. The second investigates optimization techniques in order to find the space of values that properties of a material can take, from consideration of the microstructure.

In part I, the applicability of standard lattice percolation models to a random 2-D grain structure is explored. A random network based on the triangle lattice is proposed as a more appropriate model, and results in a higher percolation threshold (0.711 compared with 0.653 for the standard hexagonal lattice). The triple junction constraint inherent in grain boundary structures is subsequently applied to the new network. This results in a lowering of the percolation threshold to 0.686, which turns out to be very close to the value obtained from the hexagonal lattice under the same constraint.

In Part II, an efficient method for finding the closure of a bi-objective optimization problem involving two material properties is formulated. The method is based upon two algorithms developed to find the Pareto front in multi-objective problems – the weighted sum, and the normal boundary intersection methods. The resultant procedure uses quadratic programming (QP) to find as many points on the closure as possible,
changing to the less efficient sequential quadratic programming (SQP) only where necessary to find points on concave, or almost concave, regions.

Improvements on the method are demonstrated using extra linear constraints in the generalized weighted sum (GWS) algorithm, and multiple GWS trials at each stage.

Optimization using only the linear part of the problem is shown to give excellent results for the particular example used, and may help to achieve an approximate closure more quickly for larger problems.

An adaptation of a common Pareto front method from genetic algorithms – the maxmin algorithm – is also demonstrated. The efficiency of the method is found to be reasonable for finding closures in the test case.

Other general optimization techniques for the form of the problem in-hand are explored for completeness of the study. These include a survey of unconstrained techniques that might be useful in large-scale problems, and an in-depth application of QP for the constrained case.
ACKNOWLEDGEMENTS

The main thanks for this work goes to Professor Brent Adams for providing the means, and the encouragement, to come to BYU. His insights and motivation over the past year have been invaluable.

In Part I of the thesis I am indebted to Jay Basinger for providing the percolation code used as a basis for much of the analysis.

In Part II Massimiliano Binci (Drexel) and Carl Gao (BYU) provided the necessary example problems, and Dr Surya Kalidindi (Drexel) acted as an extremely useful sounding board.

And, most importantly, my wife, Joan, and children, Emily, Amy, Ashleigh and Ethan, have been extremely supportive despite the upheaval from a previously stable lifestyle.
Table of Contents

Part I: Percolation in Two-Dimensional Grain Boundary Structures ..............................................................1
  1.1 Background ..................................................................................................................................1
  1.2 Introduction .................................................................................................................................1
  1.3 Description of the new structure .................................................................................................4
  1.4 Discussion of expected results .....................................................................................................7
  1.5 Percolation threshold results for the pseudo lattice ....................................................................10
  1.6 Application of triple junction constraints ...................................................................................11
  1.7 Summary ..................................................................................................................................15

Part II: Property Closures in Polycrystals .................................................................................................17
  2.1 Formulation of the problem .........................................................................................................17
  2.2 Bi-objective optimization ..........................................................................................................22
    2.2.1 Generalised weighted sum and adaptive NBI methods .....................................................22
    2.2.2 Improvements to the method ............................................................................................29
    2.2.3 Accuracy of the purely linear solution ..............................................................................30
    2.2.4 The genetic algorithm approach .......................................................................................32
    2.2.5 Conclusions for Bi-objective optimization .......................................................................34
  2.3 General optimization techniques - the shape of the original problem ........................................36
  2.4 Examples on quadratic forms .....................................................................................................38
  2.5 Optimization algorithms ............................................................................................................44
  2.6 The unconstrained problem .........................................................................................................45
    2.6.1 Options for the unconstrained problem .............................................................................46
    2.6.2 Steepest descent .................................................................................................................47
    2.6.3 Quasi-Newton method ......................................................................................................48
    2.6.4 Conjugate gradient method ...............................................................................................50
    2.6.5 Minimal residual method ..................................................................................................52
    2.6.6 Conjugate gradient with preconditioning .........................................................................53
  2.7 Constrained problem ....................................................................................................................54
    2.7.1 Incorporation of equality constraints ................................................................................54
    2.7.2 Quadratic programming ....................................................................................................55
  2.8 Summary of optimisation techniques ............................................................................................58

References .....................................................................................................................................................59
Appendix .......................................................................................................................................................63
Appendix 1: Percolation Codes and Software Architecture ..........................................................................65
Appendix 2: Optimization Codes .............................................................................................................93
Table of Figures

Figure 1: Grain boundary network for 304 stainless steel studied in [17]; the heavier boundaries are HAR, while the light ones are CSL or LAR. .......................................................... 3
Figure 2: The basis triangle lattice with 3 bonds per vertex ................................................................. 5
Figure 3: A random pseudo lattice with P(HAR)=0.65; the heavier lines represent the HAR boundaries. ... 6
Figure 4: Comparison of number of edges per grain for a real grain structure (black) and the pseudo lattice (grey) introduced in this paper ................................................................. 7
Figure 5: Obtain a square lattice by deleting the dotted lines from the triangle lattice and translating every other horizontal line ............................................................................................. 9
Figure 6: Obtain a hexagonal lattice by deleting the dotted bonds in the triangle lattice, and expanding the gap between the required vertices to arrive at hexagons ........................................................................... 9
Figure 7: J_i for the unconstrained and constrained cases on a pseudo lattice. The graphs are for: J_0 – dotted; J_1 – dot-dash; J_2 - dashed; J_3 – solid .................................................................................. 13
Figure 8: P(HAR)=0.4 for constrained and unconstrained cases respectively. The constraint results in larger HAR clusters / more closed grains ........................................................................ 15
Figure 9: Tensile specimens with two isotropic materials – the black material being stiff, the white material being soft .................................................................................................................. 19
Figure 10: Point (asterisk) found on the boundary of the property closure for an arbitrary n .................. 23
Figure 11: Points on the boundary of the property closure for C_{11} and C_{12} found using GWS approach ......................................................................................................................... 24
Figure 12: Times to solve 100 identical problems using QP and SQP algorithms in MATLAB. Points above the dotted line indicate that the SQP algorithm took longer ........................................................................ 26
Figure 13: Traditional NBI approach. A chord is drawn between the minimum points for P and Q, and normal lines are used to constrain the search for boundary points ........................................................................ 26
Figure 14: Schematic for finding a point on a concave region of the property closure between points p_k and p_{k+1} that have already been found .................................................................................. 27
Figure 15: Points found using the GWS and ANBI methods .................................................................. 28
Figure 16: Points on the closure found using only ANBI (after the original 7 points were found using GWS) ......................................................................................................................... 29
Figure 17: Points (circles) determined from linear optimization, plotted on the full closure .................. 31
Figure 18: Genetic algorithm results (asterisks) vs. ANBI method (line) .................................................. 34
Figure 19: Ellipse  \( P = x_1^2 + 2x_2^2 + x_3^2 \) ........................................................................................ 40
Figure 20: Each slice in the \( x_1-x_2 \) plane is a circle; the maximum in this case is at \( x_1=0 \), and either \( x_1=3 \) or \( x_2=3 \) on their respective axes .......................................................................................................... 42
Figure 21: Example demonstrating that the optimum point for a quadratic surface is not necessarily determined by the extreme eigenvalues; the dotted line represents the feasible region; the asterisk is the position of the optimum point ........................................................................................................ 43
Figure 22: Plotting the error (left) and the first two variable (right) for the steepest descent method .......... 48
Figure 23: Plot of error (left) and the first two variables (right) ................................................................. 49
Figure 24: Plot of error (left) and the first two variables (right) for the conjugate gradient method ......... 51
Figure 25: Error plot for Minimal Residual method ................................................................................ 52
Figure 26: Error plot for the conjugate gradient method with a poorly conditioned matrix .................. 53
Figure 27: Example of optimisation of 2-variable quadratic using the dual method ................................. 56
Figure 28: Example of 10-variable QP problem; variables 1(=x_1) and 3(=x_3) are plotted. The starting point is the unconstrained optimum close to (0,0). The path to the constrained optimum (marked with a ‘*’) is shown. Note that the feasible region is only defined as in the drawing when the optimum point is reached; at other points the effect of the other variables will change its shape .......................................................................................................................... 57
List of Tables

Table 1: Percolation thresholds for standard lattices ([11]); note that the number of bonds per vertex is $\frac{1}{2}$ the number of bonds meeting at each vertex. ................................................................. 8

Table 2: Typical data from percolation searches on a single pseudo lattice with varying P(HAR). For this case the calculated threshold is 0.7114. A ‘Percolation’ value of 0 / 1 indicates that percolation has not / has occurred (respectively) for this value of P(HAR)........................................................................................................ 11

Table 3: Percolation thresholds ($P_c$) and approximate values of $(J_0+J_1)$ for various cases. ......................... 14
Part I: Percolation in Two-Dimensional Grain Boundary Structures

1.1 Background

Percolation theory is the study of connected paths [1],[2]; and in particular, the study of connected paths that completely span a space. It constitutes a branch of graph theory, and is based upon graph structures that are made up of ‘open’ and ‘closed’ bonds. If a bond is open, it can form part of a path; if it is closed, then it cannot.

Connected open bonds are termed clusters. Hence, the higher the fraction of open bonds in the graph, the larger the clusters (on average) will be. When a cluster spans the whole space it is termed a spanning cluster; hence it will contain a path that spans the whole space.

A spanning path may be related to some critical physical phenomenon, such as a fracture across the whole sample, or full penetration from corrosion.

In typical percolation theory, the bonds are laid out on a regular lattice. The most popular lattices are the triangular, square and hexagonal lattices. The percolation threshold for a given lattice is the minimum fraction of open bonds that must be present for the probability of an infinite spanning path to be one.

1.2 Introduction

Percolation theory is a natural mathematical tool for studying certain properties of grain boundary networks. In particular, the distinct properties of high angle random boundaries (HAR) when compared against other grain boundaries (low angle random – LAR – and / or coincident site lattice – CSL) creates a
natural definition of open and closed bonds that may be used in the percolation model. This in turn may be used to predict path-based phenomena such as fracture ([3],[4],[5]) or corrosion ([6],[7]).

In the case of fracture, grain boundaries consist of regions where atomic interfaces are less ordered, and result in a higher potential energy. If a fracture surface develops along the grain boundary, this potential energy will reduce the amount of energy required to form the new surface, and may result in intergranular fracture becoming the dominant mode of failure.

In a brittle material, the crack driving force, G, across a grain is $2\gamma_s$, as usual. The value of G along a grain boundary will be approximately $2(1.2\gamma_s)\gamma_b$, where the value of 1.2 represents a higher surface-generation energy for new surface along a grain boundary [8], and $\gamma_b$ is the extra potential energy at the grain boundary due to disorder. The value of $\gamma_b$ is often approximated to be zero for LAR and CSL boundaries, and assigned a single value for all HAR boundaries (e.g. [9]). More exact values can be calculated [10], but the two-value approximation is often sufficient.

Various methods have been used to investigate intergranular fracture using the two value estimation for $\gamma_b$, including semi-empirical [9], minimal surface [11-12] and fuse-lattice [13]. However, the division of boundaries into high or low energies (open or closed bonds) allows percolation theory to be used as yet another tool in this area of investigation.

Traditionally percolation theory as applied to grain boundaries has utilised the hexagonal (or ‘honeycomb’) lattice as the basis for the theory (e.g., [14],[15],[16]). This lattice naturally reflects the triple junction network found in a grain structure.

A recent paper [17] reported results from studying percolation effects across real grain boundaries in a 2-D section of 304 stainless steel, and compared them with the traditional models. The data was taken from a detailed OIM scan with a 6µm step size over a 4mm square sample of the material. HAR grain boundaries were defined to be the ‘open bonds’. The HAR boundaries were those with misorientation
above 15° that did not fall under the criterion for a CSL boundary. This criterion was varied until a cluster of HAR boundaries spanned the sample. A figure of the percolating network is shown in Figure 1.

![Percolating Network](image)

**Figure 1:** Grain boundary network for 304 stainless steel studied in [17]; the heavier boundaries are HAR, while the light ones are CSL or LAR.

Non-contributing loops were removed from the analysis, and a bond was defined as a segment of grain boundary between two triple junctions.

Percolation was found to occur at $P(\text{HAR})=0.46$ (i.e. the probability that a given boundary is HAR was 0.46). This compares with a theoretical value of 0.653 for the standard hexagonal lattice [18], or 0.659 for the standard hexagonal lattice with constraints at the triple junctions [19] (using the value obtained for ‘twinned’ material).

As discussed in [17], the main reason for the discrepancy appears to be the presence of two distinct networks within the grain structure – a ‘contributing’ network of HAR and non-HAR grain boundaries that determine the main characteristics of the HAR clusters; and within this network, a ‘non-contributing’ network of CSL and LAR boundaries that lie within the outer network, and bias the ratio of
non-HAR boundaries. These belong to twins and sub-grains, and constitute up to around 66% of the total grain boundary length, or 45% of the total number of ‘bonds’ (segments of grain boundaries between triple junctions).

However, while this explanation gives a plausible reason for the discrepancy between the measured and the theoretical percolation threshold, the question remained open as to the applicability of the standard lattice model to random grain structures.

If we define an edge in the grain boundary network to be a segment connecting two adjacent triple junctions, then it may be readily seen from Figure 1 that the number of edges per grain is somewhat random, and is certainly not accurately described by the hexagonal lattice, which has a constant six edges per grain.

This paper studies the effect of such differences when it comes to determining the percolation threshold. A more suitable model derived from lattice structures is proposed and compared with the random grain network. The percolation threshold for the new structure is determined. The affect of applying a triple junction constraint, to better reflect actual grain properties, is also investigated.

In a preprint brought to the authors’ attention during peer review of this paper, a limited study of an ‘irregular lattice’ is undertaken in the context of scaling laws [20]. This structure is obtained starting from the traditional hexagonal lattice and switching boundaries to vary the number of edges for random grains. While the methodology and scope of the preprint is significantly different from this paper, it is notable that irregular structures have recently been of interest to major authors in this field.

### 1.3 Description of the new structure

The basis of the following approach is to assume that a 2-D graph of grain boundaries has only triple junctions. This is consistent with the standard approaches to grain boundary networks [14]-[16].
Secondly, lengths of the grain boundary sections are ignored, and each section between grain boundary junctions is considered as a single bond (again consistent with the standard percolation approach). Thus grain boundary networks will be compared with lattice based structures in a topological manner.

Based upon these assumptions, the starting point is to define a random ‘pseudo lattice’ that has similar structure to a 2-D grain boundary network. We start from a standard triangular lattice as shown in Figure 2. Each vertex is assigned 3 bonds (as shown) to form the standard unit cell for the lattice. Thus there are 3 bonds per vertex (in this paper the number of bonds per vertex means the number of bonds divided by the number of vertices – not the number of bonds meeting at each vertex). Since we require a triple junction at each vertex we must delete an average of 1.5 bonds per vertex.

Figure 2: The basis triangle lattice with 3 bonds per vertex

Unit cells are populated from left to right, and from bottom to top of the lattice. For the current unit cell, if the vertex is already attached to a total of $b$ bonds (from unit cells to the left and below), then $3-b$ bonds are randomly chosen from the unit cell to complete the triple junction. For the left and bottom edges of the lattice, the value of $b$ is chosen randomly. A sample resultant pseudo lattice is shown in Figure 3.
In order to determine whether this is a reasonable representation of a 2-D grain structure, we compare it to a real grain sample. Clearly the number of bonds per vertex will be correct if the real grain structure is made up purely of triple junctions. The other parameter that may be used to compare the topological equivalence of the networks is the distribution of the number of bonds (or edges) per grain. In the sample (Figure 1, and [17]) the average number of edges per grain is 4.8 over 413 grains; in our example pseudo lattice, the number is 5.6 over 183 grains (later analysis over a large number of grains showed that the number of edges per grain converges to 6; for a small number of grains the edge effects – larger grains being more likely to terminate outside the current window – skewed the results). Figure 4 shows the distribution of number of edges per grain for the real and theoretical networks.
It should be noted that the number of edges per grain calculated for the 304 stainless steel was counted by eye, hence very short edges would not necessarily be resolved. This may explain the rather high number of grains with 3 edges amongst the 413 grains. Notwithstanding the limited data, the pseudo lattice is clearly much closer to the actual grain structure than a hexagonal lattice where every grain has 6 edges.

Having established that the pseudo lattice is a closer topological fit to a real grain structure than the hexagonal lattice, we expect that a real grain structure will have similar percolation properties to the pseudo lattice.

1.4 Discussion of expected results

Before analysing the percolation properties of the pseudo lattice, we consider how we might expect it to compare to standard lattices. Consider the percolation thresholds for the three most common lattices given in Table 1 (the values are for bond percolation):
The percolation threshold is strongly related to the number of bonds per vertex. This seems to be intuitive given that the number of bonds per vertex determines the density of paths in the lattice (consistent with standard texts, a percolation path is assumed to follow open bonds – e.g. [21]).

Gurland (see for example, [22]) proposed that for 3-D percolation, an average of 1.5 open bonds at each vertex was an approximate critical value independent of whether the graph was random or a regular lattice. In 2-D, it may be seen from table 1 that the number of open bonds connecting to each vertex at percolation is an average of approximately 2. This is another way of making the same observation that the percolation threshold is approximately 1/(number of bonds per vertex).

From this point of view, the pseudo lattice (and grain structure) has 1.5 bonds per vertex, and hence is expected to have a similar percolation threshold to the hexagonal lattice. This is consistent with the normal representation of a grain structure as a hexagonal lattice.

One must now question the effect of the random nature of the pseudo lattice when compared to the regular hexagonal lattice, and whether this will significantly change the expected percolation threshold. To aid the discussion we note that both the square and hexagonal lattices can be obtained by deleting 1 or 1.5 bonds per vertex, respectively from the standard triangle lattice (see Figure 5 and Figure 6).

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Number of bonds per vertex</th>
<th>1/(number of bonds per vertex)</th>
<th>Percolation threshold</th>
<th>Average number of open bonds meeting at each vertex at percolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular</td>
<td>3</td>
<td>0.33</td>
<td>0.347</td>
<td>2.08</td>
</tr>
<tr>
<td>Square</td>
<td>2</td>
<td>0.5</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>Hexagonal</td>
<td>1.5</td>
<td>0.67</td>
<td>0.653</td>
<td>1.96</td>
</tr>
</tbody>
</table>

Table 1: Percolation thresholds for standard lattices ([11]); note that the number of bonds per vertex is ½ the number of bonds meeting at each vertex.
Figure 5: Obtain a square lattice by deleting the dotted lines from the triangle lattice and translating every other horizontal line.

Figure 6: Obtain a hexagonal lattice by deleting the dotted bonds in the triangle lattice, and expanding the gap between the required vertices to arrive at hexagons.

Take the case of the square lattice. Deleting one bond per vertex of the triangular lattice removes 1/3 of the bonds; if these had been removed in a random manner, one would expect that $\frac{0.347}{1-1/3} = 0.52$ of
the remaining bonds should be open to arrive at the percolation threshold (consistent with the percolation threshold for the triangle lattice of 0.347). In fact the actual fraction requiring removal is 0.5. Hence the non-random removal of the first 1/3 of the bonds has not had a huge effect on the percolation threshold.

Similarly, for the hexagonal lattice, had we deleted 1/2 of the bonds in a random manner, we would expect that \( \frac{0.347}{1-0.5} = 0.694 \) of the remaining bonds should be open to arrive at the percolation threshold. In this case, the actual value is 0.653.

In the case of the pseudo lattice, since the bonds are removed in a somewhat random manner (as opposed to the structured removal of bonds to form the honeycomb lattice), one would expect that the percolation threshold will be closer to the value of 0.694. This was the starting hypothesis of the authors.

1.5 Percolation threshold results for the pseudo lattice

To determine an actual value for the percolation threshold for the pseudo lattice, a number of random lattices were created and the threshold found numerically for each lattice (with a search step size of 0.001). Various methods of deciding the percolation threshold were tried with very similar results. The chosen method was to take the value of \( P(HAR) \) that had an equal weight of percolating values below as the weight of non-percolating values above it. The weight was simply the distance of the value from the calculated percolation threshold.

Percolation was registered if a continuous path of HAR boundaries occurred from left to right AND / OR from bottom to top of the window. The size of the window was increased until convergence of the percolation threshold was obtained.

A typical set of results is illustrated in Table 2 for a single pseudo lattice.
Table 2: Typical data from percolation searches on a single pseudo lattice with varying P(HAR). For this case the calculated threshold is 0.7114. A ‘Percolation’ value of 0 / 1 indicates that percolation has not / has occurred (respectively) for this value of P(HAR).

<table>
<thead>
<tr>
<th>Percolation</th>
<th>P(HAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.6910</td>
</tr>
<tr>
<td>0</td>
<td>0.6920</td>
</tr>
<tr>
<td>0</td>
<td>0.6930</td>
</tr>
<tr>
<td>0</td>
<td>0.6940</td>
</tr>
<tr>
<td>0</td>
<td>0.6950</td>
</tr>
<tr>
<td>0</td>
<td>0.6960</td>
</tr>
<tr>
<td>0</td>
<td>0.6970</td>
</tr>
<tr>
<td>0</td>
<td>0.6980</td>
</tr>
<tr>
<td>0</td>
<td>0.6990</td>
</tr>
<tr>
<td>0</td>
<td>0.7000</td>
</tr>
<tr>
<td>0</td>
<td>0.7010</td>
</tr>
<tr>
<td>0</td>
<td>0.7020</td>
</tr>
<tr>
<td>1</td>
<td>0.7030</td>
</tr>
<tr>
<td>0</td>
<td>0.7040</td>
</tr>
<tr>
<td>1</td>
<td>0.7050</td>
</tr>
<tr>
<td>0</td>
<td>0.7060</td>
</tr>
<tr>
<td>1</td>
<td>0.7070</td>
</tr>
<tr>
<td>0</td>
<td>0.7080</td>
</tr>
<tr>
<td>1</td>
<td>0.7090</td>
</tr>
<tr>
<td>0</td>
<td>0.7100</td>
</tr>
<tr>
<td>1</td>
<td>0.7110</td>
</tr>
<tr>
<td>1</td>
<td>0.7120</td>
</tr>
<tr>
<td>1</td>
<td>0.7130</td>
</tr>
<tr>
<td>0</td>
<td>0.7140</td>
</tr>
<tr>
<td>1</td>
<td>0.7150</td>
</tr>
<tr>
<td>0</td>
<td>0.7160</td>
</tr>
<tr>
<td>1</td>
<td>0.7170</td>
</tr>
<tr>
<td>0</td>
<td>0.7180</td>
</tr>
<tr>
<td>0</td>
<td>0.7190</td>
</tr>
<tr>
<td>1</td>
<td>0.7200</td>
</tr>
<tr>
<td>1</td>
<td>0.7210</td>
</tr>
<tr>
<td>1</td>
<td>0.7220</td>
</tr>
<tr>
<td>1</td>
<td>0.7230</td>
</tr>
<tr>
<td>1</td>
<td>0.7240</td>
</tr>
<tr>
<td>1</td>
<td>0.7250</td>
</tr>
<tr>
<td>1</td>
<td>0.7260</td>
</tr>
<tr>
<td>1</td>
<td>0.7270</td>
</tr>
<tr>
<td>1</td>
<td>0.7280</td>
</tr>
<tr>
<td>1</td>
<td>0.7290</td>
</tr>
<tr>
<td>1</td>
<td>0.7300</td>
</tr>
</tbody>
</table>

Approximately forty values of P(HAR) were assessed on more than 10 different randomly generated pseudo lattices of approximately 6000 grains. A number of pseudo lattices of 16000 grains were also used to confirm the accuracy of the smaller models. The results were remarkably consistent with a standard deviation of less than 0.002 (0.25%).

The average percolation threshold value was 0.711. This was slightly higher than the expected value of 0.694 (by 2%).

To validate the model a standard hexagonal lattice was assessed in exactly the same manner, and a percolation threshold of 0.6513 obtained. This is slightly below the published value of 0.6527 (by 0.2%), but generally supports the accuracy of the models and methodology.

1.6 Application of triple junction constraints

It has been observed (see for example, [16]) that the misorientation values of three grain boundaries meeting at a triple junction are not completely independent. If $M_i$ is the misorientation matrix for the $i^{th}$ grain boundary, then:

$$M_1M_2M_3=I$$

where $I$ is the identity matrix.
The constraint would generally be applied in a natural manner by assigning random (3D) orientations to the grains, and calculating the resultant (2D) misorientation. For ease of application of the constraint in our case, it is convenient to deal solely with randomly assigned misorientations and applying the constraint derived from Eq. (1) as follows [16]:

\[ \theta_{\text{max}} \leq \theta_i + \theta_j \]  

(2)

\( \theta_{\text{max}} \) is the maximum misorientation of the three boundaries at a given triple junction, and \( \theta_i, \theta_j \) are the other two misorientations.

An arbitrary maximum misorientation of 22.5° was chosen for the exercise, and a misorientation was labelled as HAR if the angle was greater than 15°. This turned out to be a convenient combination that resulted in less effort required to satisfy Eq. (2). Note that the study in [16] reported that the percolation results on a standard lattice were insensitive to both actual crystal symmetry (and hence maximum misorientation), and defined HAR limit. A target \( P(\text{HAR}) \) was chosen, and a given angle randomly selected between 0° and 15° (i.e. LAR) or 15° and 22.5° (i.e. HAR) based upon this probability. If the resultant boundaries meeting at a given junction did not satisfy Eq. (2), the process was repeated. A limit was placed upon the number of attempts at each junction. However, the constraint was violated at less than 0.1% of the junctions.

The final \( P(\text{HAR}) \) that resulted from this algorithm was slightly higher than the target; thus the final value was the one used in the analysis.

To illustrate the effect of the constraint, and by way of validation, we define \( J_i \) (i=1,2,3) to be the probability that a given junction will have i LAR (i.e. non-HAR) boundaries meeting there. For example, if \( P(\text{HAR})=1 \), then \( J_3=J_2=J_1=0 \) since there are no LAR boundaries, and \( J_0=1 \). Values of \( J_i \) are plotted in Figure 7 for the unconstrained case, and for the constrained case.
Figure 7: $J_i$ for the unconstrained and constrained cases on a pseudo lattice. The graphs are for: $J_0$ – dotted; $J_1$ – dot-dash; $J_2$ - dashed; $J_3$ – solid.

These figures compare favourably with those in [16], and indicate that the method for choosing the grain boundary orientations is reasonable. To be more specific, the constrained values lie between those for the general and those for the fibre case described in [16], and closer to the general case. We will discuss these cases in more detail after presenting the results of the constrained pseudo lattice model.

As for the unconstrained case, several hundred percolation searches were undertaken on different pseudo lattices, and with varying P(HAR). Once again the results were extremely consistent (the standard deviation was 0.003). The effect of the constraint was to reduce the percolation threshold from 0.711 to 0.686.

To put this result in context we quote the results from several models on the standard hexagonal model reported in [12]; see Table 3. The fibre case represents a material with a preferred crystallographic axis (such as for an extruded material); the twinned case represents a material with significant twinning, and where the open / closed bonds are taken to be non-CSL / CSL respectively (i.e. P(HAR) refers to P(non-CSL)).
Table 3: Percolation thresholds ($P_c$) and approximate values of ($J_0+J_1$) for various cases.

<table>
<thead>
<tr>
<th>Model</th>
<th>$P_c$(HAR)</th>
<th>$J_0+J_1$ at $P$(HAR)=0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random hexagonal [18]</td>
<td>0.653</td>
<td>0.21</td>
</tr>
<tr>
<td>Fibre constrained hexagonal [16]</td>
<td>0.601</td>
<td>0.33</td>
</tr>
<tr>
<td>General constrained hexagonal [16]</td>
<td>0.676</td>
<td>0.29</td>
</tr>
<tr>
<td>Twinned constrained hexagonal [19]</td>
<td>0.659</td>
<td>0.27</td>
</tr>
<tr>
<td>Random pseudo lattice (this paper)</td>
<td>0.711</td>
<td>0.21</td>
</tr>
<tr>
<td>Constrained pseudo lattice (this paper)</td>
<td>0.686</td>
<td>0.29</td>
</tr>
</tbody>
</table>

$J_0$ and $J_1$ are the fraction of junctions with 3 and 2 HAR boundaries respectively. Thus, the value of $J_0+J_1$ gives the fraction of junctions that contribute to HAR paths. If the $J_0$ and $J_1$ junctions were distributed randomly through the material, it would clearly be the case that a higher value of $J_0+J_1$ would mean a higher probability of a percolating path, and therefore a lower value of $P_c$. However, for all of the constrained models the value of $J_0+J_1$ is higher than the respective random case; yet the value of $P_c$ is not consistently lowered due to the constraint.

It is observed in [16] that the application of the triple junction constraint tends to cluster the HAR bonds. In the case of the fibre texture the clusters tend to be ‘strings’, thus increasing the probability of a percolation path. Furthermore, in this case the value of $J_0+J_1$ is highest. Both of these facts lead to a lowering of $P_c$. In the case of the general texture, the clusters of HAR bonds form tighter groups, and actually decrease the probability of percolation, despite the rise in $J_0+J_1$. Hence the nature of the clustering appears to have a larger affect than the change in $J$ values.

In the case of the pseudo lattice, the clustering also results in a ‘patchier’ network than the random case. However, the clusters appear to help paths circumnavigate the occasional larger grains, rather than resulting in small loops of HAR bonds (see Figure 8) This, combined with the higher value of $J_0+J_1$, reduces the percolation threshold from the random case.
1.7 Summary

The objective of this study was the investigation of percolation on lattice-based networks that represented the topology of real grain structures more closely than the traditional hexagonal lattice.

A random pseudo lattice composed of triple junctions has been obtained from the standard triangle lattice, and is proposed as a closer fit to real 2-D grain structures. The average number of edges in a single real grain structure (0.48) was found to be lower than that for the pseudo lattice (0.56). This is likely to be due to effects such as sub grains (see [17]), but more data is required for a true comparison. However, the distribution of number of edges per grain for the real case and the pseudo lattice is reasonably close.

The percolation threshold for the pseudo lattice has been found to be 0.711. This is slightly higher than the expected value of 0.694, and significantly higher than the standard hexagonal lattice value of 0.653.

For percolation in 2-D, an approximate average value of 2 open bonds meeting at each vertex is required. The resultant threshold of 0.711 is equivalent to 2.13 open bonds at each vertex, which is slightly higher than the value for the triangle lattice.
We have argued that the percolation threshold would be close to the value of 0.694. This has proven to be the case. The reason for the slightly higher than expected value is presumably due to the obstacles created by the randomly placed large grains that necessitate longer paths on average to cross the sample; this results in the requirement for a higher percentage of open bonds for percolation.

The triple junction constraint – representing the relationship between misorientation angles at a triple junction for real grain boundaries – has been applied to the pseudo lattice, and results in the lowering of the percolation threshold to 0.686. This is the opposite of the effect seen on the hexagonal lattice, where the threshold was raised from 0.653 to 0.676. Thus the final separation of the constrained hexagonal and pseudo lattice thresholds turns out to be quite small.

The standard deviation for the calculated percolation thresholds across a series of randomly constructed pseudo lattices is very small, and indicates that the results from such structures will be stable.
2 Part II: Property Closures in Polycrystals

When it comes to material properties, a design engineer will typically have access to a table of discrete values relating to commercially available materials. Whether he undertakes a formal optimization process or not, he will generally balance off various properties (including cost) and arrive at the material that gives him the highest (lowest) values in the areas of greatest concern to him.

In essence, he is searching the space of possible properties for the material, and taking one – most likely on the boundary of the space – that best fits his requirements.

Clearly the space of properties that are possible for a given material is much broader than the discrete set of commercially available products. In an ideal world, a designer would have the option of searching the entire space of possible properties – or the property closure – for the materials he is using.

One step closer to that ideal world would be to have a mathematical tool that allows exploration of all possible properties, and defines the microstructures related to each of these. This section of the thesis will seek to define the boundary of a multidimensional property space (confined to 2-dimensions for this exercise), and give the two-point microstructure functions related to points on this boundary.

2.1 Formulation of the problem

Most materials of practical use are made up from subcomponents in a higher (or equal) state of order. Homogenization theories are based upon the assumption that the properties of the higher-order subcomponents are better understood, or are more uniform, than that of the bulk material.
In microstructure sensitive design (MSD), as applied to poly-crystals, the building blocks for the homogenization theories are the individual crystals, or grains. The crystallographic structure of these grains is assumed to be known, and the properties are governed by this structure (including knowledge of the grain material and phase) and the orientation of each grain relative to some global frame of reference.

As a simple example, take a copper crystal with known modulus in the various crystallographic directions [23] of:

<table>
<thead>
<tr>
<th>Direction</th>
<th>[100]</th>
<th>[110]</th>
<th>[111]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modulus (GPa)</td>
<td>66.7</td>
<td>130.3</td>
<td>191.3</td>
</tr>
</tbody>
</table>

If one were to assume that crystals are small and randomly oriented throughout the bulk material, then one might also assume that the modulus of the bulk material would be some kind of average of the three modulae given above. In fact this turns out to be the case, with the material modulus being approximately 120 GPa.

First order homogenization theories are derived from volume averaging techniques of this type (see for example [24]). The methodology results in estimates and bounds for an effective global property (such as modulus). These bounds give the property closure for the first order theory. The ‘closure’ is the envelope within which a certain property must lie.

The first order homogenization is the easiest method to obtain estimates for global properties in practice. This is because it is generally relatively easy to determine an approximate volume fraction for the subcomponents of the material - be it the volume fraction of grains within a certain orientation band, or the volume fraction of grains of different phases.

The volume fraction method gives no information about the position (or relative position) of the grains in the bulk material. Yet this information can have a dramatic effect on the global properties. For
example, assume that a tensile specimen is made up of 50% of a stiff isotropic material, and 50% of a soft isotropic material as in Figure 9.

![Figure 9](image)

Figure 9: Tensile specimens with two isotropic materials – the black material being stiff, the white material being soft

Clearly the modulus of the first specimen will be dominated by the stiff material, while the modulus of the second specimen will be dominated by that of the soft material. Hence the volume fraction property calculation can sometimes give a very poor estimate of material properties.

At the other extreme, if the properties and position of every grain in the material were known, then the bulk material properties could be accurately calculated. Between these extremes are descriptions of position and volume fraction of subcomponents of different orders.

As an example, the effective (or ‘global’) elasticity tensor, $C^*$, can be written as a sum of averages of functions of different orders $[25]$

$$C^* = \langle C \rangle - \langle C' \Gamma C' \rangle + \langle C' \Gamma C' \Gamma C' \rangle - ...$$  \hspace{1cm} (3)

Here, $C$ is the local tensor, $\langle \rangle$ refers to taking the ensemble average of a function, $C' = C - C^*$ is the ‘polarization’ of the local stiffness tensor with respect to a reference stiffness tensor, and $\Gamma$ is the
appropriate Green’s function. The first term on the right of the equation is the first order description, the second term is second order, and so on.

In this work we will refer to the second order description as containing information about both the volume fraction of subcomponents, and the probabilities of these subcomponents lying in certain locations with respect to each other. In mathematical terms, assume that \( g \) is a particular orientation, \( dg \) is a volume element (invariant volume) in orientation space about \( g \), \( r \) is a vector in real space, and \( dr \) is a volume element in vector space; then the second order description provides:

1. \( f(g) \) - the probability distribution function for orientations; hence \( f(g)dg \) is simply the volume fraction of subcomponents with orientation lying in the volume element \( dg \) about \( g \).
2. \( f_2(g,g'|r) \) - the probability distribution function for orientations separated by vector \( r \) in real space; hence \( f_2(g,g'|r)dgdg'dr \) is the volume fraction of subcomponents with orientation lying in the volume element \( dg \) about \( g \), and subcomponents with orientation in \( dg' \) about \( g' \), separated by a vector within \( dr \) about \( r \).

In the second order description, it is clear that the examples in Figure 9 are completely different structures, and hence should result in a better approximation to the actual modulus.

It can be shown [25] that in terms of the second order description, a property, \( P \), can be calculated using an equation of the form:

\[
P = N_{ss'}D_{s}^{n}D_{s'}^{n} + B^{n} \sum_{s=1}^{S} D_{s}^{n} \tag{4}
\]

subject to ‘conservation of mass’ constraints for each \( s \):

\[
\sum_{n=1}^{N} D_{s}^{n} = N \tag{5}
\]

and positivity constraints:

\[
D_{s}^{n} \geq 0 \tag{6}
\]
Here, the ‘s’ refers to the position in physical space, and the ‘n’ refers to the position in orientation space. The real and orientation spaces are discretised such that each s, n, refer to a finite volume in each space. The number of such sub volumes, or cells, is N in orientation space, and S in real space.

Equation (4) can be put in matrix form if each of the $D^s_n$ is given a unique identifying number. This can be done using the scheme below:

$$x_i = D^s_n \text{ where } i = (n - 1)S + s$$

Then Eq.s (4)-(6) can be re-written:

$$P = x^T A x + b^T x \quad (7)$$

subject to:

$$\sum_{n=1}^{N} x_{(n-1)S+s} = N \quad (8)$$

for each $s=1:S$, and:

$$x_i \geq 0 \quad (9)$$

for all i.

Equation (7) now has the form of a typical quadratic surface (see for example [26] and [27]) and can be solved using standard quadratic programming (see [28], [29], [30]).

We will first discuss multi-objective optimization associated with this problem (which is the main result for this part of the thesis); then we will look in more detail at the original problem and optimization techniques that might be applied to it.
2.2 Bi-objective optimization

As discussed earlier, the objective of this part of the thesis is to use the optimization techniques discussed later on to find the property closure for a number of properties. We illustrate the solution technique using two properties, but the methodology could readily be extended to higher dimensions.

Take a pair of physical properties, P and Q. If the property closure of feasible properties were strictly convex and closed in PQ-space, then there would exist unique points representing maximum and minimum values of P and Q. In a traditional design approach one would be looking for a microstructure design that in some sense optimised both P and Q.

If it were the case that the designer wanted to minimise both P and Q, then the Pareto front of points between the global minimum values of P and Q would be found, and the designer would choose a value along this front. The points represent ‘non-dominated’ values, in the sense that any microstructure that decreases the value of P must increase the value of Q and vice versa.

If the objective were changed to minimizing P and maximizing Q, a further quadrant of the property closure would be found, and so on to find all four quadrants.

2.2.1 Generalised weighted sum and adaptive NBI methods

A more elegant way of following this process is to generalise the weighted sum method used to find a set of evenly spaced Pareto front points ([31],[32],[33]). In a typical approach one defines a function:

\[ J = \lambda_1 P + \lambda_2 Q \]  

such that \( \lambda_1 + \lambda_2 = 1 \), and both are positive. One then minimises J, subject to the same constraints on \( x \) required for feasible P and Q.
To adapt this approach in a manner that is easy to visualise, we modify the definition of the weights, $\lambda_i$. Let $n = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$ such that $\lambda_1^2 + \lambda_2^2 = 1$; i.e. $n$ is unit length. Then for a point $(P,Q)$ in PQ-space, $J$ represents the dot product of $n$ with the vector defined by $\begin{bmatrix} P \\ Q \end{bmatrix}$; or, in other words, the projection of the vector $\begin{bmatrix} P \\ Q \end{bmatrix}$ in the direction $n$. Thus minimizing $J$ gives the point on the closure that has the minimum projection in direction $n$.

![Diagram](image)

Figure 10: Point (asterisk) found on the boundary of the property closure for an arbitrary $n$

If the direction of $n$ is varied to cover the whole unit circle, every point on the strictly convex closure can be found. We will refer to this method as a generalised weighted sum approach (GWS).

We demonstrate the approach by taking an example of a cubic sample made of 27 cells (i.e. each side is divided into 3 even lengths). The composite is assumed to be composed of two isotropic phases with the following parameters (Lame constants for the two phases):
\[ \lambda_\alpha = 70 \text{ GPa} \quad \lambda_\beta = 100 \text{ GPa} \]
\[ \mu_\alpha = 35 \text{ GPa} \quad \mu_\beta = 55 \text{ GPa} \]

The two properties to be calculated are the stiffness matrix parameters \( P = C_{11} \) and \( Q = C_{12} \).

The GWS algorithm is implemented using the Matlab QP solver [34]. Equation (10) is minimised for an initial number of evenly spaced directions for \( n \) in the unit circle. Further points are then obtained by identifying badly spaced points, and taking further directions for \( n \) between the previous values.

![Property Closure for \( C_{11} \) and \( C_{12} \)](image)

Figure 11: Points on the boundary of the property closure for \( C_{11} \) and \( C_{12} \) found using GWS approach

In this example we have stopped the algorithm after several iterations to demonstrate that points on ‘flat’ regions of the closure are more difficult to find, and have fewer points in the figure. Several authors ([35],[36],[37]) have discussed the two disadvantages of the weighted sum method highlighted by this example:

1. the resultant points may not be evenly spaced;
2. points on regions of the closure that are not strictly convex will not necessarily be found.
In practice we do not expect that the closure will be strictly convex, or even convex - in which case a significant modification to our approach is required. Two methods that can find points in the concave region of the closure are the normal boundary intersection (NBI) method [38], and the adaptive weighted sum method [33] (AWS). Both of these methods will find points in the concave region of the closure. However, the AWS method will only find non-dominated (Pareto) points. From a designer’s point of view, this may be an advantage as Pareto points may be the only design points of interest; and the AWS method should be chosen. However, since we set out to find all points on the closure we have chosen the NBI method.

One disadvantage with both of these methods is that they introduce non-linear constraints into the optimisation routing, thus barring quadratic programming as a potential solution method. Up until this point, all constraints have been linear, but once a constraint is placed in the PQ-space (which are quadratic functions of the microstructure) this linear property is lost. Of course, it is no theoretical hardship to move to a non-linear programming method such as SQP. The issue is the time required to solve the problem. For a relatively small problem in a real microstructure there might be tens of thousands of variables, and keeping the problem as simple as possible will be important.

For the example above we recorded the time taken to solve 100 optimisation steps (the same problem each time for this exercise) using 100 different directions of n. The times are plotted against each other for the QP and the SQP algorithms (Figure 12). One may see that the SQP algorithm generally takes longer to solve the problem (an average of 55% longer for this problem), and in some cases it takes 6 times as long. These results are for a reasonably small and well behaved problem; we would expect the difference to be more pronounced as the problem size increases and perhaps becomes less well behaved.
The NBI method is set up to solve a traditional Pareto-front minimisation problem where only one quadrant of the property closure is being sought. A chord is drawn between the point that minimizes $P$, and the point that minimizes $Q$. Lines normal to this chord are found, and the minimal point in the property closure is found along this line.

This scheme does not suit our requirements for a number of reasons, and we define an adaptive NBI routine (ANBI) as follows. Firstly, a number of evenly spread vectors, $n_i$, are defined in the unit circle. $J$ is minimised with respect to each of these vectors to find a series of points on the closure. We define a maximum tolerance on the ‘distance’ between points on the front. This is done with respect to each
parameter by taking the difference between the maximum and minimum values for P, and dividing by some factor (e.g. 7 in our examples); and similarly for Q. Thus the step in the P direction and the step in the Q direction must both lie within the respective tolerances. Starting at the first point on the closure we check the ‘distance’ between subsequent points. If it exceeds the tolerance we take a vector n halfway between the two the two values of n used to find these points. We minimise J for this value of n, still using quadratic programming; i.e. up to this point we are using the GWS method. If no new point is found, we switch to ANBI for these points.

For the ANBI step we first draw a line between the points, and find the unit normal to this line that points into the property closure. We then draw the bisecting line (i.e. in the direction of the normal, and halfway between the points). We define J using the normal, n (i.e. \( J = n^T \begin{bmatrix} P \\ Q \end{bmatrix} \)), and minimize J using SQP with the added constraint that the points (P,Q) must lie on the bisecting line. A schematic is given in Figure 14.

Figure 14: Schematic for finding a point on a concave region of the property closure between points \( p_k \) and \( p_{k+1} \) that have already been found

In this manner, we use linear constraints wherever possible, and introduce non-linear constraints only where the boundary is not strictly convex, or is almost flat.

If we repeat the previous example using the ANBI method, points in the flat region are easily found, as is demonstrated by Figure 15.
Figure 15: Points found using the GWS and ANBI methods

Note that the points are not perfectly spaced. This is due to the fact that the points found using the GWS method are not as evenly spread as those found using the ANBI method. This will always be the case since the ANBI method puts constraints on the spacing in PQ space, while the GWS method uses the original linear constraints with a modification to the function being optimized, and hence cannot accurately predict the spacing in PQ space.

To emphasize this fact, we demonstrate with a set of points found using only the ANBI method after the original 7 points were found using the GWS method. The result is a much more even spread (Figure 16). This approach takes significantly more computer time, however, due to the non-linear constraints (the difference is more noticeable with a larger number of variables; an example with 4 times the number of variables leads to a significant time difference).
2.2.2 Improvements to the method

Two improvements to the above method were tested. The first was simply to attempt multiple values of n before resorting to the ANBI method. A judgment was made in terms of the extra time required for the ANBI method, and thus a limit on initial GWS trials was imposed.

For example, the solution for the above example was found using only the ANBI method (after finding the first nine points using GWS). A total of 21 ANBI points were required to meet the constraint on the distance between points.

The same problem was repeated using GWS for the first attempt at finding a point between each pair of existing points. The number of ANBI points reduced to 11 (note that the surfaces of the closure are fairly flat, resulting in a high number of ANBI points).

When multiple attempts at the GWS methods were allowed (in this case, up to 5 attempts using different values of n), the number of ANBI points reduced to 7.
The second improvement was then to choose the coefficient of \( x \) that changed the most for the microstructure associated with the current two points on the closure. A (linear) constraint was then placed on this coefficient such that its value was fixed half way between the value at each point. Since this constraint is linear the GWS QP algorithm can still be used. This additional constraint reduced the required number of ANBI points to 4.

2.2.3 Accuracy of the purely linear solution

The equations for the properties, as developed above, involve a first and second order term. As one might expect in this type of expansion of terms, the first order term dominates the problem. In the equation:

\[
x b A x + P_T T
\]

for the example we are using, the coefficients of the matrix \( A \) are of the order 25 times smaller than the coefficients of \( b \).

If we consider only the problem: \( P' = b^T x \), the solution is much simpler than the quadratic case. Since the equation is now linear, and the feasible region is bounded by straight lines (i.e. the constraints are linear), the optimum points for \( P' \) must lie at the vertices of the feasible region. In essence, this means the each cell of the sample contains only one phase.

In the case of our example, there are only two phases. Furthermore, the equality constraint, \( x_i + x_j = 2 \) for \( i, j \), corresponding to a single cell, was incorporated into the problem. Hence each cell has only a single degree of freedom, and the coefficients of \( x \) are simple to optimise. To illustrate, take the example of minimising \( P \). Since we want the least positive value, if the corresponding coefficient of \( b \) is negative the value of the \( x \) coefficient is chosen to be 2 (it’s maximum value); for positive values of \( b \) the coefficient for \( x \) is chosen to be zero (its minimum value). In this manner an optimum microstructure is obtained.
While this is a particularly simple example, it demonstrates the desirability of optimising only the linear part of the original equation.

To find general points on the property closure, we proceed using the GWS method. Define

\[ J = \lambda_1 P' + \lambda_2 Q' \]

where \( P' \) and \( Q' \) are the linear parts of the property definition. Now minimise \( J \) for a series of \( n \) in the unit circle, as before. Once the optimal microstructures (i.e. vectors \( x \)) have been determined, the corresponding values of \( P \) and \( Q \) are determined using the original quadratic equation.

The resulting points lie exactly on the closure for \( PQ \) as illustrated in Figure 17.

![Property Closure for C_{11} and C_{12}](image)

Figure 17: Points (circles) determined from linear optimization, plotted on the full closure

While it is not expected that the linear optimisation approach will result in the exact closure for larger problems, this approach may give a first approximate. Apart from giving an approximate closure the points could also be used as the starting point for a genetic algorithm.
2.2.4 The genetic algorithm approach

Since the methods being developed for property closures are intended for use on large problems, an alternative approach to quadratic or other gradient based programming is to use genetic algorithms (GA). This approach is good for large numbers of variables where an approximate answer is sought, rather than a single optimum point.

Again, the traditional Pareto-front search can be adapted to find the whole property closure using an approach related to that given above. In this case, however, we search each quadrant (i.e. between the four optimal points for P and Q; clearly these points are not necessarily distinct, as is the case for the example given above), rather than producing the full closure in a single search. This is because the GA approach automatically spreads out the points along the closure for the searches in each of the four quadrants.

We adopt the max(min) approach, that assigns a fitness to each ‘design’ (or point in PQ-space) in such a manner that the optimisation naturally moves points towards the boundary of the closure while also spreading them out. Although there are many variations that can be made on the theme, we describe our approach briefly to highlight the main points of the method.

First, the PQ-space is searched using a QP algorithm to find microstructures that optimise P and Q (maximum and minimum for each). We define a generation size, G, and choose unit normal vector n in the direction [1, -1] (thus we are ‘minimizing P and maximizing Q’; note that whether or not n is unit length is irrelevant).

A ‘design’ is a microstructure of N*S variables. Reformulate the numbering for convenience as follows: $x_i = D_{is}$ where $i = (s - 1)N + n$. The genes are the microstructure variables associated with each cell in real space; hence a gene is of length N.
We randomly choose \((G-2)\) microstructures within the linear constraints, and add the microstructures that minimize \(P\) and maximize \(Q\) to give a total of \(G\). We then choose a tournament size, \(T\), and randomly select \(T\) out of the \(G\) designs. We choose the fittest design in this tournament as our first parent based upon the max(min) fitness ranking. For the \(i^{th}\) design, this is defined as:

\[
 f(i) = \max_{j \neq i}\left(\min\left((P_i - P_j)(Q_i - Q_j)\right)\right) 
\]

where \(P_i\) is the value of property \(P\) for the \(i^{th}\) design, etc.

We similarly find a second parent. The two children from these parents are defined starting from the parents themselves, and deciding whether to crossover their ‘genes’ based upon a crossover probability. If they are to be crossed over, a random crossover point, \(c\), is defined, and the first child is composed of the first \(c\) genes from the first parent, and the remainder from the second parent (and vice versa for the second child).

A mutation probability is defined, and if mutation is decided for a given child, then a random number is again chosen for each gene. If the number is below the mutation probability, the gene is switched to a different value randomly chosen from the feasible region of microstructure orientations (i.e. such that the original constraints are satisfied).

In order to ensure that values are spread out across the full quadrant of the closure, one must periodically re-inject the microstructures corresponding to the minimum value of \(P\) and the maximum value of \(Q\). This is done at the mutation stage where periodically two of the children are replaced by the ‘optimum’ microstructures.
The fitness of the parents and children are now found with respect to the whole group, and the fittest G are chosen to go on to the next generation (i.e. elitism is used).

Once the desired number of generations is reached, the process is repeated with n in the directions [1,1], [-1,1], [-1,-1] to find all four quadrants.

As an example, we demonstrate the results from 50 generations (of 100 parents in each generation) in each quadrant with the same example as that given above. We plot the actual closure found using the ANBI method to demonstrate the accuracy of the method. It can be seen that the results are good for even this modest number of generations. The time taken for the calculation is similar to that of the ANBI method for about 25 points.

![Figure 18: Genetic algorithm results (asterisks) vs. ANBI method (line)](image)

2.2.5 Conclusions for Bi-objective optimization

The link between property closures for a number of material properties and the related microstructures is an important map, allowing designers to (theoretically) optimize material characteristics against the required design application.
The second order properties functions are quadratic in form. One might find the boundaries of the closure by calculating the extreme values of $P$, and taking an even set of values between these two extremes. $P$ would then be constrained to each of these values, and the extreme values of $Q$ calculated for each such value of $P$. In essence, a grid search is applied to the closure.

However, this approach has two immediate disadvantages: 1) the constraints on $P$ apply a nonlinear constraint to the original problem, thus increasing the effort required to solve each step; and 2) points on concave regions of the closure would not necessarily be found.

A better approach has been demonstrated, using the idea of a generalized Pareto front. Quadratic programming is used wherever possible to increase the speed of the solution, using a generalized weighted sum (GWS) approach. Where the boundary is concave, or close to concave, the method is switched to an approach adapted from the normal boundary intersection method (which we call the ANBI method).

The approach has been demonstrated to be effective on an actual microstructure problem (albeit of limited scale).

Improvements to the method were suggested that reduce the required number of non-QP steps when finding the closure. While the reduction in effort may seem modest for the example problem, the incentive for increasing efficiency is driven by the knowledge that problems of practical use will be several orders of magnitude larger than the one demonstrated here.

Due to the nature of the problem, the second order term is significantly smaller than the first order term. It was demonstrated that for our example, solving only the linear part of the problem, and plotting the points associated with the resultant microstructures in PQ space gave an accurate property closure. It is not expected that for larger problems the linear optimal points will lie exactly on the second order closure, but they may give a rough estimate of the boundary. Furthermore, the linear optimization problem clearly
requires much less effort to solve – particularly given the structure of our problem where each cell in the sample space can be considered individually.

An alternative approach using a generalization of the maxmin method for genetic algorithms has also been demonstrated. The efficiency of this approach was found to be reasonable for the type of problem being considered, though not better than the gradient methods. However, it may be of more interest when the number of variables increases dramatically in problems of practical value.

2.3 General optimization techniques - the shape of the original problem

In order to discuss general optimization methods on the original problem, it is helpful to consider the ‘shape’ of the problem. This will be easier to visualize if we can write it in terms of a quadratic form. Take Eq. (7) and let \( y = x + x_0 \). We require:

\[
y^T A y = (x + x_0)^T A (x + x_0) = x^T A x + b^T x + c
\]

for some constant \( c \).

Hence

\[
x_0^T A x + x^T A x_0 = 2x_0^T A x = b^T x
\]

since \( A \) is symmetric. And therefore:

\[
2x_0^T A = b^T
\]

or

\[
2x_0^T = A^{-1} b^T
\]

Thus, to make this transformation, either \( A \) must be invertible, or \( b^T \) must be in the range of \( A \).

Assuming that the transformation can be made, the new problem is to optimize \( P' = P - c \):

\[
P' = y^T A y
\]

subject to the original constraints on \( x \). This is a quadratic form, and we take some standard results from the theory [26],[27].
The solution to Eq. (12) for each value of $P'$ is a conic centred about the origin (or some subspace containing the origin).

Assume the eigenvalues of $A$ are $\lambda_1 - \lambda_{NS}$, and order them such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_i > 0,$
$\lambda_{i+1} \leq \lambda_{i+2} \leq \ldots \leq \lambda_{i+j} < 0,$ (assume $\lambda_1 > 0$ and $P' \geq 0$ in the original equation; multiply by $-1$ if necessary; if this assumption cannot be made, there is clearly no solution), then the Inertia of $A$ is given by $In(A) = (i,j,k)$.

Then each value of $(i,j,k)$ corresponds to a different type of conic (i.e. there are $n(n+1)/2$ different types). For example, if $i=NS$, Eq. (12) would represent an ellipsoid with axes of length $\sqrt{\frac{P'}{\lambda_i}}$ for each value of $P'$.

Since $A$ is real and symmetric, the following options are open in terms of transformations on the form:

1. There exists an orthogonal matrix $U$ such that:
   $$U^T A U = D = diag(\lambda_1, \ldots, \lambda_n)$$
   i.e. $D$ has the same eigenvalues as $A$, and these are along the diagonal. Thus, this is simply a realignment of the axis to those of the eigenvalues. The new equation is a sum of squares. $A$ is said to be ‘similar’ to $D$.

2. There also exists (non-singular) matrix $C$ (not necessarily orthogonal) such that:
   $$C^T A C = D_0 = diag(1, 1, \ldots, -1, -1, \ldots, 0, 0)$$
   where there are 1’s for all positive eigenvalues, -1’s for all negative eigenvalues, and 0’s for all zero eigenvalues. Writing $Cz = y$, the new form, $z^T D_0 z$, is a sum of squares. $D_0$ is said to be ‘congruent’ to $A$. Thus $D_0$ has the same fundamental geometry as $A$ (i.e. if $A$ defines an ellipsoid, $D_0$ defines a circle, etc). The reason for this is that $D_0$ has the same ‘inertia’ as $A$ (number of positive, negative, and zero eigenvalues), and the inertia defines the geometry.
3. Furthermore, there exist matrices $F, G$ (not orthogonal) such that:

$$F^T_AG=D_1=\text{diag}(1,1,\ldots,0,0)$$

i.e. if $A$ is non-singular, then $D_1$ is the identity matrix. $D_1$ is said to be equivalent to $A$.

If $D_0 \neq D_1$ then $D_1$ does not have the same eigenvalues as $A$, and defines a completely different geometry. This transformation does not represent a coordinate transformation and hence is not useful in solving or understanding the problem.

A reminder of some definitions:

‘similar’ means lengths and angles are preserved (rotations, reflections) (orthogonal $U$)

‘congruent’ means a linear transformation has been applied to the space (i.e. $x=Cy$)

‘equivalent’ does neither

‘affine’ means points on lines stay on lines which is the same as a linear transformation plus a translation.

If the quadratic form were to be transformed to a sphere, the optimization routine simply needs to find the points in the transformed feasible region that are closest to (to give the minimum), and furthest from the origin (to give the maximum). However, the only way to transform an arbitrary conic into a sphere is option 3, which doesn’t preserve length, so doesn’t result in a useful modification of the original problem. Hence this ‘sphere’ route can only be taken if $A$ is positive definite.

### 2.4 Examples on quadratic forms

Since the original problem can often be transformed to a quadratic form, we will look at some examples where the problem starts from this point. Again, this will help to visualize the issues. Hence start with the problem:

$$P = x^T Ax \quad (13)$$
subject to:

\[ \sum_{n=1}^{N} x_{(n-1)S+n} = N \]

for each \( s=1:S \), and:

\[ x_j \geq 0 \]

At this point we might:

1. Incorporate the equality constraints
2. Transform to diagonal form (similarity transformation)
3. Transform to standard form (congruence transformation)

We will demonstrate simple examples of these options. For simplicity, take \( S=1 \).

The feasible region forms part of an \( N-1 \) dimensional hyper plane in \( N \)-D space (\( N \) = number of variables). If the equality \( \sum x_i = N \) is incorporated into the quadratic form, the new feasible region is the projection of the original feasible region into \( N-1 \) space. This region is a solid ‘pyramid’. Hence, there may not be a great deal gained by introducing the equality constraint (i.e. we still have a closed feasible region of the same dimensionality). However, we will demonstrate both options.

**Case 1:** the quadratic form represents an \( N \) dimensional ellipse (inertia = \( (N,0,0) \))

If one were to include the equality constraint in the equation for \( P \), the new equation would be that of an ellipse.

\( P = x_1^2 + 2x_2^2 + x_3^2 \)

s.t. \( x_1 + x_2 + x_3 = 3 \quad x_j \geq 0 \)

hence

\[ A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

and this is the equation of an ellipse (see Figure 19).
Now incorporate the equality constraint: \( x_1 + x_2 + x_3 = 3 \)

then:

\[
P = 2x_1^2 + 3x_2^2 + 2x_1x_2 - 6x_1 - 6x_2 + 9
\]

s.t.

\[
x_1 + x_2 \leq 3 \quad x_1, x_2 \geq 0
\]

where the first inequality is a consequence of incorporating the inequality \( x_3 \geq 0 \).

The feasible region is the solid triangle between the origin, and \( x_1=3 \) and \( x_2=3 \) on the respective axes. Since the equation is still that of an ellipse (with other conics, the result of incorporating the equality constraint will generally not result in a conic of the same form), the boundary is convex, and hence the maximum will occur at one of the corners of the solid triangle. Note that the new ellipse is not centered at the origin.

One may also put the new conic in standard form by, for example, completing the square. In that case the new equation becomes:

\[
P = y_1^2 + 2y_2^2 + \frac{261}{44}
\]

s.t.

\[
\frac{y_1}{\sqrt{2}} - \frac{y_2}{\sqrt{11}} \geq \frac{27}{22}, \quad \frac{y_2}{\sqrt{22}} \geq -\frac{3}{11} \quad \text{and} \quad \frac{y_1}{\sqrt{2}} + \frac{y_2}{\sqrt{11}} \leq \frac{27}{22}
\]
the feasible region now being a triangle that is not centered at the origin. But now the equation for $P$ is a circle centered at the origin; hence the maximum value is the furthest point of the feasible region from the origin (which should be easy to find).

One may also approach the ellipse case without incorporating the inequality. In this case the standard form of the equation is an $n$-sphere. The maximum value of $f$ is given by the furthest point from the origin in the transformed feasible region.

e.g. using the same example as previously, one may define
\[
U = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]
such that $U^T AU = I$. Hence the coordinate transformation is given by $y = Ux$

Thus in the new coordinates:

$$P = y_1^2 + y_2^2 + y_3^2$$

and the feasible region is similarly transformed such that the extreme points of the feasible region are at the points $y_1 = 3$, $y_2 = 3\sqrt{2}$, $y_3 = 3$ along the respective axes.

Since the new equation for $P$ is a sphere centered at the origin, the maximum is given by the furthest point from the origin in the feasible region – i.e. $y_1 = 0$, $y_2 = 3\sqrt{2}$, $y_3 = 0$

**Case 2:** the quadratic form is not an ellipse (i.e. it is one of the $2N(N-1)$-1 other geometries; the inertia is $(i,j,k)$ with $i<N$). The standard form of the quadratic form is now a sum of squares with coefficients of +1 or −1.

**Some observations on Case 2:**

For the $i$ variables with positive coefficients, the projection onto this $i$-dimension space is a sphere; hence, for each such subspace the maximum value of $f$ is clearly the furthest point from the origin in a slice of the feasible region in this subspace.
e.g. \[ P = x_1^2 + x_2^2 - x_3^2 \]

A figure of this conic is shown in Figure 20.

![Figure 20: Each slice in the x₁-x₂ plane is a circle; the maximum in this case is at x₃=0, and either x₁=3 or x₂=3 on their respective axes.](image)

We wish to determine whether an optimum point found in the subspace of xᵢ’s with positive coefficients is also a global optimum. In this example, if one starts from the boundary of the projected feasible region onto the subspace with xⱼ=0 for all xⱼ with non-positive coefficients, then the value of P will increase as the xⱼ are increased if and only if the slope of a line in the feasible region is less than 45 degrees from an xᵢ axis. Hence no simple formula can be found for determining the global optimum using a subspace of the entire space.

Similarly, we would like to know whether a knowledge of the eigenvectors / eigenvalues can be used to determine the global optimum.

The eigenvectors for a conic centered at the origin are perpendicular to the conic where they cross each surface (i.e. \( \nabla P \) is parallel to the eigenvector at any point along the line given by the eigenvector).

Hence:

- If there are no positive eigenvalues the global maximum value will obviously be negative (or zero), but this says nothing about the position of the maximum.
- If no eigenvectors associated with positive eigenvalues point into the feasible region, but there is a positive eigenvalue, the maximum will be along an edge of the feasible region.
• If the maximum eigenvector points across the feasible region then: a. if it points along a corner (i.e. along a coordinate axis) of the feasible region, the maximum will be at this point; b. if it doesn’t point along an axis but is perpendicular to the feasible region, then maximum will be at this point; c. if it is not perpendicular to the feasible region, and not along an axis then the maximum will be some distance from the intersection with the feasible region.

e.g. Take \( P = x_1^2 - x_2^2 \) and rotate by 30 degrees. Then
\[
A = \begin{bmatrix}
\cos^2 R - \sin^2 R & 2 \cos R \sin R \\
2 \cos R \sin R & \sin^2 R - \cos^2 R
\end{bmatrix}
\]
where \( R = 30 \) deg. Figure 21 demonstrates that the maximum point is not necessarily in the direction of the intersection of the largest eigenvector with the feasible region;

Figure 21: Example demonstrating that the optimum point for a quadratic surface is not necessarily determined by the extreme eigenvalues. In this case the heavy line represents the eigenvector associated with the maximum eigenvalue; the dotted line represents the feasible region; the asterisk is the position of the optimum point.
2.5 Optimization algorithms

It appears that there is no simple trick that enables us to optimize the property functions – mainly due to the fact that we have no guarantee that the property functions are convex. Hence we turn to traditional optimization techniques in this section.

We repeat the description of the problem to clarify our objective. We choose to minimize the objective function, $P$, for the sake of the exercise; but the approach would be equivalent if we chose to maximize $P$.

Objective: To minimize a quadratic function of many variables (relating to the two point CDF), with linear constraints.

Problem Statement:
Minimise

$$P = x^T A x + b^T x$$  \hspace{1cm} (14)

S.T.

$$\sum_{n=1}^{N} x_{(n-1)S+s} = N \hspace{1cm} \text{for each } s=1:S$$

and

$$x_i \geq 0 \hspace{1cm} \text{for all } i.$$

An obvious approach to this type of problem (quadratic function with linear constraints) is Quadratic Programming. There are various approaches that might suit the original problem. For example, one might search the space for a feasible point, and then use a simplex or projection method to minimize the function while remaining in the feasible area. However, in the literature (see for example [39]) it is noted that half of the effort expended in this approach is usually finding the feasible point. An alternative approach is to first determine the unconstrained optimum, and then solve the dual problem (related to the unfeasible space) to determine the optimal point for this problem, which turns out to be the minimal point on the boundary of the feasible region.
This approach can be divided in two phases:

1) Solve the unconstrained problem (using any method)
2) Finding the closest point in the feasible region (using the dual QP method)

In either case, it is helpful to review both unconstrained and constrained optimization methods. In many constrained algorithms, an ‘unconstrained’ approach (such as conjugate gradients) is used repeatedly during the process.

2.6 The unconstrained problem

We wish to minimise:

\[ P = x^T A x + b^T x \]  (15)

If \( A \) is symmetric, the problem may be rewritten by observing that the solution to Eq. (15) is equivalent to solving the problem:

\[ 2Ax = -b \]  (16)

Proof (adapted from [40]):

\[ \nabla P = \nabla (x^T A x) + \nabla (b^T x) = Ax + A^T x + b \]

Since \( A \) is symmetric, the right hand side equals: \( 2Ax + b \)

For a minimum \( \nabla P = 0 \), and hence: \( 2Ax = -b \)

If \( A \) is positive definite, and \( x \) satisfies \( 2Ax = -b \), then for any other point \( x + v \):

\[ P(x + v) = (x + v)^T A(x + v) + b^T (x + v) \]
\[ = x^T A x + b^T x + v^T A v + x^T A v + v^T A x + b^T v \]

For symmetric \( A \), and using \( 2Ax = -b \), this equals:

\[ P(x) + v^T A v - v^T b = P(x) + v^T A v \]
If $A$ is positive definite, then $v^T Av$ is positive for any $v$, hence $P(x+v)$ is greater or equal to $P(x)$, i.e. $P(x)$ is a global minimum.

Several key things to note here are:

1) $A$ is the Hessian of the original function.

2) The residual, $(-b - 2Ax)$, is the negative gradient of the functions, and hence the direction of steepest descent.

3) If $A$ is not positive definite, but is symmetric, the solution to $2Ax=-b$ will be a stationary point, a maximum or a minimum.

4) One may include the equality constraints from the original problem in the linear equation, writing the new equation as $2A_jx=b_j$.

5) If $A$ is not positive definite, but symmetric a positive definite problem may be formulated as follows: $2A_j^2y=-b$ and $x=Ay$.

2.6.1 Options for the unconstrained problem

Various ways in which the unconstrained problem may be solved are suggested here:

1) Gaussian Elimination (or other matrix method)

2) *Steepest descent

3) *Quasi Newton

4) *Conjugate gradient

5) *Minimal residual (conjugate gradient type)

6) Conjugate gradient with preconditioning

Note: * Indicates that we have developed code to solve by these methods.

The Gaussian elimination method was not implemented as part of this project, since it is relatively easy to predict the efficiency of the method. Furthermore, the convergence of the result (over the NS
iterations required) may be very poor compared to one of the other methods (i.e. it may not be until the last step that the result converges rapidly).

The other methods were each implemented using code developed by the author. The convergence of each method is compared.

The method of preconditioning was not fully implemented, and hence there are no significant results available for discussion. But the method itself is briefly reviewed, as it may be an important option for large sparse matrices.

To compare the methods a ‘sparse’ 50 x 50 matrix was generated. This was done by populating 20% (an arbitrary figure) of a 50 x 50 matrix, and then adding it to its transpose (to make it symmetric). The resultant matrix was squared to make it positive definite.

The matrix was inverted to find the ‘exact’ minimum, and the error at each step was the distance from this point. The first two variables (of the 50) were plotted against the number of iterations to demonstrate the convergence in this subspace. A maximum of 150 iterations were allowed in each case. The Matlab codes for each method are attached in the appendices.

2.6.2 Steepest descent

The unconstrained problem (either the original problem, Eq. (16), or with the equality constraints included) can be solved using the steepest descent method. This method had hardly started to converge after 150 iterations (Figure 22), demonstrating the highly elliptical nature of the hyper surface generated by the matrix.

The plot of the first two variables highlights the tedious path followed by the program. The code was developed based upon theory in [41] and a basic algorithm in [40].
Figure 22: Plotting the error (left) and the first two variable (right) for the steepest descent method

**Important points to note for this method:**

1) No line search is required (alpha is calculated exactly at each step)
2) This method converges in a stable, but VERY SLOW, manner
3) This method does not appear to be a reasonable option for large matrices of the type chosen.

2.6.3 Quasi-Newton method

As for steepest descent, the unconstrained problem can be solved using quasi Newton methods. In this case, the approximate inverse Hessian will eventually equal $A^{-1}$ (as already pointed out, the Hessian of the original function is $A$).

This can readily be proven. Suppose $N=A^{-1}$. Using the Newton method:

$$(x_i - x_0) = -H^{-1}\nabla f(x_0) = -A^{-1}(Ax_0 - b) = -x_0 + A^{-1}b$$

i.e. $x_i = A^{-1}b$, and $x_i$ is a solution to the original problem.

**Calculation of the step length**

Since the original function is quadratic, the step length in each direction for the quasi Newton method can be calculated exactly as follows. Let $s$ be the search direction. Then for the approximate Hessian inverse, $N$:

$$s = -N\nabla f = -N(Ax - b)$$

Let $x_i = x_0 + \alpha s$, then for a minimum in the search direction we require that $\frac{d}{d\alpha} f(x_i) = 0$
Hence: \( \nabla f \frac{dx_1}{d\alpha} = 0 \). But \( \frac{dx_1}{d\alpha} = s \) and \( \nabla f = (Ax - b) \). Using the minus of the gradient:

\[
(b - Ax)^T s = 0 \\
[b - A(x_0 + \alpha s)]^T s = 0
\]

i.e. Step length at each iteration:

\[
\alpha = \frac{(b - Ax_0)^T s}{(As)^T s}
\]

The results for this method are given below. Once again, the code is in the appendices (adapted from theory in [41], with exact step length calculated by the author).

![Figure 23: Plot of error (left) and the first two variables (right)](image)

**Important points to note for this method:**

1. As mentioned, the approximate inverse Hessian will eventually equal \( A^{-1} \) (since A is the Hessian of the original function) – this is a useful property for use in phase 2, where the inverse of A is required.

2. Step length can be calculated exactly for this method as well:

3. To prevent a non-positive definite matrix N, the BFGS update was used for N (past experience has shown that a non-positive definite N is easily obtained using the rank-1 update).

4. For an \( n \times n \) matrix, A, the method converges in \( n \) iterations (in this case, 50). As can be seen from Figure 5, the main convergence may come in the last few iterations.

5. The method is more expensive, from a computer memory point of view, than the conjugate gradient method due to the requirement to update the \( n \times n \) Hessian inverse at each iteration.
2.6.4 Conjugate gradient method

As for the quasi-Newton method, the conjugate gradient method will theoretically converge in $n$ iterations. At each iteration a new direction is chosen that is ‘conjugate’ to all previous directions, and hence there is a total of $n$ such directions. Conjugate simply means that the directions are ‘orthogonal’ using the operator $x^T A y$; i.e. $x$ is conjugate to $y$ if $x^T A y = 0$. Thus one may think of the directions as being perpendicular in a space where the inner product involves multiplying by $A$. If the elliptical nature of the hyperspace was ‘squashed’ until it became spherical, the directions would indeed be perpendicular in the usual sense.

In practice the algorithm is set up to carry rounding errors from one iteration to the next, and hence an exact solution is not generally achieved after $n$ iterations. Hanke [42] points out that there are renormalisation techniques available to ensure that conjugacy is preserved despite the round-off, but these are expensive. In our algorithm, we simply restarted the problem after so many iterations. Hanke states that for ill-posed problems completely different standards are required from the usual well-posed problems. Furthermore, for a dense matrix, $A$, the conjugate method is not as efficient or as accurate as direct Gaussian elimination (see [43]).

However, there are several advantages that may be particularly useful for our particular problem. Firstly, at each step the residual is minimised for the subspace of $R^n$ spanned by the residual vectors of all previous iterations; hence if the algorithm is stopped early the answer achieved up to that point is likely to be a good estimate of the minimum given the limited number of steps [43]. Secondly, it can be shown that the conjugate method fits a polynomial of order $i$ to the set of eigenvalues of $A$, where $i$ is the number of the current iteration. If these eigenvalues are clustered together (such as in a sparse matrix), a polynomial of lower order can give a good fit, and hence much fewer iterations are required to minimise the problem. This is also the case for multiple equivalent eigenvalues. Note that the eigenvalues are not actually calculated [40].
In our case, the conjugate gradient method performed slightly worse than the quasi-Newton method, as can be seen from the figures below. It may be that for larger, or sparser matrices the advantages of the conjugate gradient method would come to the fore.

Two different algorithms were used to test the conjugate gradient method. They have the same mathematical basis, but the first (based on [43]) was more efficient (making better use of values already calculated). However, this led to worse round-off error than the second version (based on [40]). Only the results from the second version are included, but the codes for both versions are included in the appendices.

![Figure 24: Plot of error (left) and the first two variables (right) for the conjugate gradient method](image_url)

**Important points to note for this method:**

1. At each step the residual is minimised for the subspace of $\mathbb{R}^n$ spanned by the residual vectors of all previous iterations; hence the method may be better than other methods if you have to stop early.

2. This method uses less memory than other methods since it does not need the Hessian calculation which is expensive!

3. It can be shown that the conjugate method fits a polynomial of order $i$ to the set of eigenvalues of $A$, where $i$ is the number of the current iteration. If these eigenvalues are clustered together, a polynomial of lower order can give a good fit, and hence much fewer iterations are required to minimise the problem. Hence, it should be good for sparse matrices.
4. Round-off error can be a big issue. Regular restarts will help, but for ill-posed problems more robust methods (as discussed in [42]) may be required.

2.6.5 Minimal residual method

Conjugate gradient methods can be thought of as operating over a mathematically defined space called a Krylov space. Various conjugate gradient methods may be obtained by defining different inner products over this space, simply by pre-multiplying by the matrix $A$ to different powers. For the normal conjugate gradient method, the power is 0.

The minimal residual method is a type of conjugate gradient method, but with the inner product of the space defined using $A$ to the power ‘one’. This leads to the residual at each iteration being minimal for the subspace of the Krylov space defined by all previous search directions. Hence, if the search is stopped early, this method should give the closest approximate of all conjugate gradient methods.

However, it is more expensive to implement (in terms of number of calculations per iteration).

![Figure 25: Error plot for Minimal Residual method](image)
In our implementation, while the algorithm consistently converged, it was much slower than the other cg methods. It is not clear why this was the case, and may mean that the particular method of implementation carried large errors for the 50 x 50 matrices tested.

2.6.6 Conjugate gradient with preconditioning

Given that the conjugate gradient method is most likely to be used only for large problems with sparse matrices, this type of matrix for A lends itself to what is called ‘preconditioning’. This means that A can be approximated by a matrix M that is easier to invert, making the problem:

\[ M^{-1}Ax = M^{-1}b \]

significantly easier to solve.

The conjugate gradient method used above was run on a problem with a proportion of the elements in A being two orders of magnitude away from the rest of the values in the matrix. This led to an ill-posed problem, as demonstrated by the poor convergence in Figure 26 using the standard conjugate gradient method.

![Figure 26: Error plot for the conjugate gradient method with a poorly conditioned matrix](image)

This demonstrates the need, or at least the potential, for preconditioning. According to [40], preconditioning is almost a requirement for large matrix problems.
Note that there are issues with the fact that $M^{-1}A$ may not be positive definite or symmetric (even for symmetric, positive definite $M$). However, it is possible to find a matrix $E$ such that $EE^T = M$, and such that $E^{-1}AE^T$ is symmetric and positive definite. Then the following equivalent problem is solved:

$$E^{-1}AE^T y = E^{-1}b \quad \text{and} \quad y = E^Tx$$

This method was not implemented, but may be promising for the type of problem at-hand.

### 2.7 Constrained problem

#### 2.7.1 Incorporation of equality constraints

As already discussed, for large problems there may be limited benefits obtained from inclusion of the equality constraint in the problem. However, for completeness, we briefly discuss how incorporation of the constraint may be applied. Starting from the main problem, reformulate the numbering for this task as follows: $x_i = D_i^n$ where $i = (s-1)N + n$. Then:

$$P = \sum_{i=1}^{NS} \sum_{j=1}^{NS} a_{ij} x_i x_j + \sum_{i=1}^{NS} b_i x_i$$

the first equality constraint states:

$$x_1 = N - x_2 - \ldots - x_N$$

hence (for symmetric $A$):

$$P = a_{11} (N - x_2 - \ldots - x_N)^2 + 2 \sum_{j=2}^{N} a_{1j} (N - x_2 - \ldots - x_N)x_j + \ldots$$

$$\sum_{i=2}^{NS} \sum_{j=2}^{NS} a_{ij} x_i x_j + b_1 (N - x_2 - \ldots - x_N) + \sum_{i=2}^{NS} b_i x_i$$

It is easy to see that a rearrangement of the terms will result in a new problem of the form:

$$P = x^T A' x + b' x + c$$

If the new matrix, $A'$, is also positive definite, the solution to the new problem is given by $2A'x = -b'$. All of the equality constraints may be included in this manner.
2.7.2 Quadratic programming

In order to gain in-depth experience with quadratic programming, original code was written based on Goldfarb [39], and is included in the appendices. According to the referenced paper, one half of the effort in many QP approaches is finding a feasible point. Goldfarb advocates a different approach where instead of looking for a feasible point, we find an unconstrained optimum, and then optimize a ‘dual’ problem in the infeasible region (giving a feasible optimum).

Code for this phase of quadratic programming was validated using the simple problem demonstrated in [39]. The solution demonstrated various features of the method. The original (constrained) problem is given by:

\[
2,0,0 \geq + \geq + \geq = \quad f(x) = 6x_1 + 2(x_1^2 - x_1x_2 + x_2^2)
\]

Thus, the unconstrained (phase 1) problem is given by:

\[
\begin{bmatrix}
4 & -2 \\
-2 & 4
\end{bmatrix} x = \begin{bmatrix}
6 \\
0
\end{bmatrix}
\]

A contour plot of the function f is shown in Figure 27, along with the feasible region, the unconstrained optimum, and the constrained optimum. For this simple problem the unconstrained optimum (phase 1) is found by inverting the matrix A. Phase 2 then finds points on the boundaries of the constraints that are tangent to the contours of f.

In the figure shown, the constrained optimum was found in one step from the unconstrained optimum. In practice this is not usually the case. Various larger problems were solved using this code. For the sake of clarity, the largest example demonstrated here is a 10 variable problem (i.e. A is 10 x 10). The unconstrained optimum was found close to zero, since the coefficients in vector b were small compared to those in A. The search for a feasible point begins at this optimum.
The constraints were that all variables be non-negative, and that their sum be greater than or equal to NS=10. This problem is then similar to the original property problem, with the equality constraints included.

Variables $x_1$ and $x_3$ are plotted in Figure 28. All other variables are held at the values of the constrained optimum. It may be seen, once again, that the constrained optimum (marked with a ‘*’) is at a point where the contour plot for $f$ is tangent to one of the constraints.

Note that although the path appears to enter the feasible region before arriving at the final optimum, it is obviously not feasible in the other variables at these points. It took 15 steps to converge in this case.

Figure 27: Example of optimisation of 2-variable quadratic using the dual method
Figure 28: Example of 10-variable QP problem; variables $1(=x_1)$ and $3(=x_2)$ are plotted. The starting point is the unconstrained optimum close to $(0,0)$. The path to the constrained optimum (marked with a ‘*’) is shown. Note that the feasible region is only defined as in the drawing when the optimum point is reached; at other points the effect of the other variables will change its shape.

**Important points to note for the QP Method**

- The inverse of the original matrix (i.e. the inverse of the Hessian) is required in the code using this dual method. It may be that an update method (using consecutive approximations to the inverse Hessian) could be used, but this was not explored by the authors. If the quasi-Newton method is used for phase 1, this matrix would be available.

- For positive definite $A$, the constrained optimum was found in a few more than NS steps for all of the values of NS tried (up to NS=50, although these are solved very rapidly – less than a second). For non-positive definite $A$ of significant size (NS>20), it was found that a solution is unlikely. In this case, the problem would need to be converted into a positive definite problem, as suggested at the beginning of the discussion on the unconstrained problem.

- The code was not optimized for efficiency; there is likely to be some margin that can be gained by minimizing the number of times values are calculated. But this may be marginal (based on an overall knowledge of how the code works).
• For small problems (at least up to 50 variables) the method appears to be very efficient, but is likely to take a large amount of memory for big problems (based upon the number of matrix operations required).

• This method is one of many QP methods. There may be a different method that suits the large matrices better. But to our knowledge this is as good as any.

2.8 Summary of optimisation techniques

• The original property optimization is a typical quadratic programming problem.

• Constrained quadratic function minimization may be approached in two parts: optimization of the unconstrained problem, followed by solution of the constrained problem.

• The unconstrained problem can be written as a linear problem: \( 2Ax = -b \)

• For the unconstrained solution, quasi Newton is the best method tried, but conjugate gradient may be better for certain types of sparse matrix.

• Preconditioning is likely to help for large sparse matrices, but was not fully tested.

• Quasi Newton also inverts the matrix, which is useful for phase 2

• Quadratic programming using a dual approach appears to be efficient (of order NS steps are required after finding the unconstrained optimum). But the method used is likely to be memory intensive for large problems.

• In practice, one would use an algorithm readily available, such as those in [34].
References


[34] MATLAB Version 7, The Mathworks, Inc., 2005


[40] Shewchuk J.R., An introduction to the conjugate gradient method without the agonizing pain, 
Edition 1¾, August 4, 1994, School of Computer Science Carnegie Mellon University Pittsburgh, 
PA 15213 Website: http://www.2.cs.cmu.edu/~quake-papers/painless_conjugategradient.pdf


[42] Hanke M., Conjugate gradient method for ill-posed problems, Longman Scientific & Technical, 
England 1995

1971
Appendix
Appendix 1: Percolation Codes and Software Architecture

The codes used for this section are in two parts – those codes relating to the new pseudo lattice, and related validation, and the code for testing for percolation on these structures.

The codes relating to the new pseudo lattice include the following. Code from files marked with an asterisk are included. Other files listed are variations on these basic files.

TRIOUT.M – outputs edge coordinates for a triangle lattice converted to a pseudo-lattice with only triple junctions, with specified probabilities for high and low angle boundaries (output file = triangle.m)

*HEXOUT.M – outputs edge coordinates for a hexagonal lattice with specified probabilities for high and low angle boundaries (output file = triangle.m)

*CONSTRAINT.M – same as TRIOUT.M, but adds the triple junction constraint found in Frary and Schuh 2004 to the probability of an edge being HAB.

TRIBASE.M – same as TRIOUT.M, but does not delete any vertices from the triangle lattice; hence it outputs a triangle lattice

*FINDPC.M – find the percolation threshold from data produced by PT4CONLOOP, etc; copy and paste data (****) into: A=[ **** ] in the main Matlab window from the vector qp output from PT4TRILOOP.M (1st column is 0 or 1 for percolation, 2nd column is value of p(HAB)) before running this file; copy and paste data from first percolation to last non percolation

*GRAINNUM.M – counts the number of grains and their edges from a pseudo lattice (it has the code for creating the pseudo lattice embedded), and lists all of the grains in matrix g, and outputs graph of probability vs. number of edges (against the Stainless steel data used in the 2005 Scripta Materiala paper by Jay Basinger)

EDGES.M – data for number of edges per grain in stainless steel sample (see GRAINNUM.M which now has the same data, making edges.m unnecessary).

The codes associated with testing for percolation was originally developed by Jay Basinger and are related as described below. Note that the codes are included as they are not referenced elsewhere.

Architecture of percolation analysis files (* all files in the diagram are listed):
PT4TRI.M
Code to check percolation status of a 2-D grain boundary network

Author(s):
Jay Basinger - 2004
Modifications by David Fullwood in 2005 to accept artificially generated grain networks.

Input
triangle.txt must be in the same directory, containing a matrix where each row contains (in text format with spaces between numbers):
x then y-coordinate of start of edge; x then y-coordinate of end of edge; ‘angle’ of edge

The angle is given by: 2 = high angle boundary, 1 = low angle boundary

This file is created by triout.m, triout2.m or hexout.m

Output
p=1 indicates percolation; p=0 indicates no percolation
graph of all HAB’s;
graph of clusters of HAB’s

Limitations
maximum of 4 (may even be 3, but I think it’s 4) edges meeting at a given vertex.

Names of previous versions
PercThreshold4Hex.m – previous versions used actual angles and csl attributes rather than simply 1 and 2 for low and high angle.

Related Programs
TRACEEDIT.M
HUBCHECK.M
CSL PLOTTER.M – loads in x,y data points from oim data – e.g. ‘everything.txt’ (or other) and plots out the grain boundaries

PT4TRILoop.M – same as PT4TRI.M, but with the code to change p(HAB) of the pseudo lattice file from TRIOUT.M embedded to allow it to change the probability of HAB between given limits within a loop (hence it can find the percolation threshold for a given pseudo-lattice or for the hexagonal lattice). In this case TRIOUT.M or HEXOUT.M must be run with ph=1 to give the basic lattice in triangle.m first.
PT4CONLOOP – same as PT4TRILOOP.M, but adds the triple junction constraint found in Frary and Schuh 2004 to the probability of an edge being HAB. Outputs vector qp giving p(hab) and whether percolation occurred (1) or not (0)

EVERYTHING.TXT – the grain network data reported in the scripta mat paper for the stainless steel sample (the everything-brandon.txt contains the same grain boundary data, but using the Brandon criterion for CSL definition); this data can be read by PercThreshold4Hex.m files
% hexagonal lattice output and plot
% dtf 17/2/05
% DOESN'T INCLUDE FIRST 2 ROWS / COLUMNS FROM EDGE
n=51; % odd number - number of rows / columns
p=0.0; % probability of HAB
rand('state',sum(100*clock)); % reset rand generator to random position
e=ones(n+1,n+1,3);
% now set up a loop to delete edges to make hexagonal equivalent
% lattice
for i=2:2:n-1  % set up triangular lattice
   for j=2:2:n-2
      xe(i,j,1)=i-1;
ye(i,j,1)=j+1;
x(i,j,2)=i+1;
y(i,j,2)=j+1;
x(i,j,3)=i+2;
y(i,j,3)=j;
   end
end
% now a loop to define HAB's with probability p
for j=1:n-1
   for i=j-2*floor(j/2)+1+2:n
      for k=1:3
         if e(i,j,k)==1; a=rand(1); if a<p; e(i,j,k)=2;end ; end
      end
   end
end
% now set up a loop to delete edges to make hexagonal equivalent
% lattice
for i=2:2:n-1  % set up triangular lattice
   for j=2:2:n-2
      if i/2==2*floor(i/4); e(i,j,1)=0;  e(i,j,3)=0; e(i,j,2)=1; else  e(i,j,1)=0; e(i,j,3)=1; e(i,j,2)=1;end %
      alternately delete edge 3, and 3,1
   end
end
% note - don't need extra assignments of 1 - used for setting all % to HAB - in which case assign the number 2
for i=3:2:n
   for j=2:2:n-1
      if (i+1)/2==2*floor((i+1)/4); e(i,j,2)=0;  e(i,j,3)=0; e(i,j,1)=1; else e(i,j,2)=0; e(i,j,3)=1; e(i,j,1)=1;end %
      alternately delete edge 2, and 2,3
   end
end
% now a loop to define HAB's with probability p
for j=1:n-1
   for i=j-2*floor(j/2)+1+2:n
      for k=1:3
         if e(i,j,k)==1; a=rand(1); if a<p; e(i,j,k)=2;end ; end
      end
   end
end

% put all edge data in rank 2 matrix ready for output
row=0;
for j=4:n-4
    for i=4:2:n-4
        ip=i;
        if 2*floor(j/2)==j; ip=i-1;end % j values odd if i even, and vv
        for k=1:3
            if e(ip,j,k)>0
                row=row+1;
                big(row,1)=ip;
                big(row,2)=j;
                big(row,3)=xe(ip,j,k);
                big(row,4)=ye(ip,j,k);
                big(row,5)=e(ip,j,k);
            end
        end
    end
end

fid=fopen('triangle.txt','w');
fprintf(fid,'%i %i %i %i %i
',big');
fclose(fid);

%return % comment-out to plot
figure
hold on
for i=2:2:n-1
    for j=1:2:n-2
        for k=1:3
            x=[i xe(i,j,k)];
            y=[j ye(i,j,k)];
            if e(i,j,k)==1; plot(x,y); end
            if e(i,j,k)==2; plot(x,y,'r'); end
        end
    end
end
for i=3:2:n
    for j=2:2:n-1
        for k=1:3
            x=[i xe(i,j,k)];
            y=[j ye(i,j,k)];
            if e(i,j,k)==1; plot(x,y); end
            if e(i,j,k)==2; plot(x,y,'r'); end
        end
    end
end

constraint.m

% pseudo lattice with constraint on triple junction angles
% from schuh 2004 - theta(max)<=theat1+theta2
% basic code taken from triout.m
% dtf 17th march 05
% DOESN'T INCLUDE FIRST 2 ROWS / COLUMNS FROM EDGE
n=59; % odd number
p=0.21; % probability of HAB
tmax=45; % maximum angle for misorientation from crystal symmetry
tm=tmax/2; % max angle for each theta
lab=15; % maximum angle for low angle boundary
constrain=1; % =0 to omit triple junction constraint
output=0; % =1 to send output to file ready for percolation search
yesplot=1; % =1 to plot data
debug=0; % printout debug data and pause each loop
rand('state',sum(100*clock)); % reset rand generator to random position
e=ones(n+1,n+1,3);
junct=zeros(1,4); % number of each type of triple junction (All LAB, 1 HAB, 2 HAB, 3 HAB)
viol=0; % number of violated constraints
for i=2:2:n-1
    for j=1:2:n-2
        xe(i,j,1)=i-1;
        ye(i,j,1)=j+1;
        xe(i,j,2)=i+1;
        ye(i,j,2)=j+1;
        xe(i,j,3)=i+2;
        ye(i,j,3)=j;
    end
end
for i=3:2:n
    for j=2:2:n-1
        xe(i,j,1)=i-1;
        ye(i,j,1)=j+1;
        xe(i,j,2)=i+1;
        ye(i,j,2)=j+1;
        xe(i,j,3)=i+2;
        ye(i,j,3)=j;
    end
end

% now set up a loop to randomly delete edges with the constraint that
% each vertex is a triple junction
for i=j:n-1
    for j=2*floor(j/2)+1+2:n
        if j==1; a=floor(rand(1)*3+1); else a=e(i-2,j,3)+e(i-1,j-1,2)+e(i+1,j-1,1); end% number of vertices
to delete (account for those at left and down already assigned)
        if a==3; e(i,j,1:3)=0;end
        if a==2; b=floor(rand(1)*3+1); e(i,j,1:3)=0; e(i,j,b)=1; end
        if a==1; b=floor(rand(1)*3+1); e(i,j,1:3)=0;e(i,j,b)=0;end
    end
end

% now a loop to define HAB's with probability p
if constrain==0 % don't add triple junction constraint
    for j=1:n-4
        for i=j-2*floor(j/2)+1+2:n
            for k=1:3
                if e(i,j,k)==1; a=rand(1); if a<p; e(i,j,k)=2;end ; end
            end
        end
    end
end
end
end
else
    % add triple junction constraint
    t=zeros(n+1,n+1,3); % set up matrix for misorientation angles of each edge
    for j=2:n-1
        %if j==2; endn=n-1; else endn=n-3; end
        for i=4:2:n-1
            ip=i;
            if 2*floor(j/2)==j; ip=i-1;end %j values odd if i even, and vv
            if i<5 | j==2 | ip==n-1 % initialise the bottom and left edges
                for k=1:3
                    if e(ip,j,k)==1; a=rand(1); if a<p; t(ip,j,k)=rand(1)*(tm-lab)+lab; else t(ip,j,k)=rand(1)*lab;
                        end ; end
                elseif max(e(ip,j,1:3))>0 % check whether there are any edges that require defining
                    keeptrying=0;
                    count=0;
                    while keeptrying==0 & count<50
                        count=count+1;
                        if count==50; viol=viol+1;end
                        for k=1:3
                            a=rand(1);
                            if a<p & e(ip,j,k)==1; t(ip,j,k)=rand(1)*(tm-lab)+lab;end %HAB
                            if a>=p & e(ip,j,k)==1; t(ip,j,k)=rand(1)*lab;end % LAB
                        end
                        thist=[t(ip,j,1) t(ip,j,2) t(ip,j,3) t(ip-2,j,3) t(ip-1,j-1,2) t(ip+1,j-1,1)]; % angles meeting at this vertex
                        if debug==1;
                            if max(e(ip+2,j,1:3))==0
                                [t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)]
                            end
                        end
                    end
                    if max(thist)<=sum(thist)-max(thist) & max([t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)])<=sum([t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)])-max([t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)])
                        keeptrying=1;
                    end
                    else
                        if max(thist)<=sum(thist)-max(thist)
                            keeptrying=1;
                        end
                    end
                end
            end
        end
    end
    if max(e(ip+2,j,1:3))==0 & ip<n-4 % if no edges to choose at the next vertex
        % check whether theta max for this and the next
        % vertex are less than the sum of the other two
        % thetas
        if max(thist)<=sum(thist)-max(thist) & max([t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)])<sum([t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)])-max([t(ip,j,3) t(ip+1,j-1,2) t(ip+3,j-1,1)])
            keeptrying=1;
        end
        else
            if max(thist)<=sum(thist)-max(thist)
                keeptrying=1;
            end
        end
    end
end
end
for k=1:3
    if t(ip,j,k)>lab; e(ip,j,k)=2; end
junct(ip,j)=sum([e(ip,j,1) e(ip,j,2) e(ip,j,3) e(ip-2,j,3) e(ip-1,j-1,2) e(ip+1,j-1,1)])-3; % number of HAB at this vertex
junct(junct(ip,j)+1)=junct(junct(ip,j)+1)+1; % keep tally of each type of vertex
end
end
end

% put all edge data in rank 2 matrix ready for output
if output==1
row=0;
for j=4:n-4
for i=4:2:n-4
ip=i;
if 2*floor(j/2)==j; ip=i-1;end %j values odd if i even, and vv
for k=1:3
if e(ip,j,k)>0
row=row+1;
big(row,1)=ip;
big(row,2)=j;
big(row,3)=xe(ip,j,k);
big(row,4)=ye(ip,j,k);
big(row,5)=e(ip,j,k);
end
end
end
end
end

fid=fopen('triangle.txt','w');
fprintf(fid,'%i %i %i %i %i
',big');
close(fid);
end

if yesplot==1 %plot data
figure
hold on
for i=2:2:n-1
for j=1:2:n-2
for k=1:3
x=[i xe(i,j,k)];
y=[j ye(i,j,k)];
if e(i,j,k)==1; plot(x,y); end
if e(i,j,k)==2; plot(x,y,'r'); end
end
end
end
end

for i=3:2:n
for j=2:2:n-1
for k=1:3
x=[i xe(i,j,k)];
y=[j ye(i,j,k)];
if e(i,j,k)==1; plot(x,y); end
if e(i,j,k)==2; plot(x,y,'r'); end
end
end
end

72
disp(['number of violated constraints' violation])

findpc.m

% work out weighted percolation threshold for each data set
% pc is defined as the point with equal weight percolation and
% non-percolation data on either side using a simple distance weight
% DTF 15th March 2005

% A = input('data'); % **** easiest way is to copy and paste A = ['**** '] into
% main matlab window from vector qp (1st col is 0 or 1 for percolation, 2nd column is value of p(HAB))
% before running this file
% copy and paste data from first percolation to last non percolation
flag = 0;
count = 0;
one = [];
zer = [];

for i = 1:length(A(:,1))
    if A(i,1) == 1 & flag == 0
        flag = 1;
        start = i;
    end
    if flag == 1
        count = count + 1;
        if A(i,1) == 0
            zer = [zer i - start];
        else
            one = [one i - start];
        end
    end
end

count = 0;
for z = 1:0.01:length(A(:,1))
    count = count + 1;
    LHS = 0;
    RHS = 0;
    for i = 1:length(one)
        if one(i) <= floor(z)
            LHS = LHS + (z - one(i));
        end
    end
    for i = 1:length(zer)
        if zer(i) >= ceil(z)
            RHS = RHS + (zer(i) - z);
        end
    end
    plotz(count) = z;
    plotp(count) = RHS - LHS;
end
end

plot(plotz,plotp)
zi=interp1(plotp,plotz,0);
pct=A(start,2)+zi*(A(2,2)-A(1,2))
pc=interp1(A(:,2),A(:,3),pct)

grainnum.m

% code for counting numbers of grains, and edges per grain in a % pseudo-lattice plot
% dtf 17/2/05
% updated 5/5/05
% DON'T INCLUDE FIRST 2 ROWS / COLUMNS FROM EDGE
n=59; % odd number; n=309 takes 1/2 hour; n=209 takes 5 minutes
p=0.69; % probability of HAB (not used in this code)
e=ones(n+1,n+1,3); % could update this in line with better approach in triout2.m ***
epass=zeros(n+1,n+1,3); % keeps track of how this edge has been passed (max once in each direction)
plotfig=1;
% first set up the triangle lattice - then make it pseudo by deleting edges
for i=2:2:n-1
    for j=1:2:n-2
        xe(i,j,1)=i-1;
        ye(i,j,1)=j+1;
        xe(i,j,2)=i+1;
        ye(i,j,2)=j+1;
        xe(i,j,3)=i+2;
        ye(i,j,3)=j;
    end
end

for i=3:2:n
    for j=2:2:n-1
        xe(i,j,1)=i-1;
        ye(i,j,1)=j+1;
        xe(i,j,2)=i+1;
        ye(i,j,2)=j+1;
        xe(i,j,3)=i+2;
        ye(i,j,3)=j;
    end
end

% now set up a loop to randomly delete edges with the constraint that % each vertex is a triple junction
for j=1:n-1
    for i=j-2*floor(j/2)+1+2:n
        if j==1; a=floor(rand(1)*3+1); else a=e(i-2,j,3)+e(i-1,j-1,2)+e(i+1,j-1,1); end% number of vertices to delete (account for those at left and down already assigned)
            if a==3; e(i,j,1:3)=0;end
            if a==2; b=floor(rand(1)*3+1); e(i,j,1:3)=0; e(i,j,b)=1; end
            if a==1; b=floor(rand(1)*3+1); e(i,j,b)=0;end
        end
    end
end
if plotfig==1  % for plot
    figure
    hold on
    for i=2:2:n-1
        for j=1:2:n-2
            for k=1:3
                x=[i xe(i,j,k)];
                y=[j ye(i,j,k)];
                if e(i,j,k)==1; plot(x,y); end
                if e(i,j,k)==2; plot(x,y,'r'); end
            end
        end
    end
    end
end % end plot

% algorithm to work out number of edges per grain
% 22/2/05
% dtf

jump(1,:)=[-2,0];   %steps for different edge types
jump(2,:)=[-1,1];
jump(3,:)=[1,1];
jump(4,:)=[2,0];
jump(5,:)=[1,-1];
jump(6,:)=[-1,-1];
gngrain=0;
g=zeros(n-8); %array for storing number of edges per grain
for j=4:n-4
    for ip=4:2:n-4
        i=ip;
        if 2*floor(j/2)==j; i=ip-1;end %j values odd if i even, and vv
        clear gtemp;
        nedge=0;        % reset number of edges for this grain
        gtemp(1)=i; % vertices for this grain (xvalues odd, y values even)
        gtemp(2)=j;
        pos(1)=i;   % current x position
        pos(2)=j;   % current y position
        loopflag=0;
        if e(i,j,3)>0;
            nedge=nedge+1;
            pos(1)=pos(1)+2;
            gtemp(3)=pos(1);
            gtemp(4)=pos(2);
        end
    end
end
edgetype=1; % edge you've just crossed relative to new vertex
if epass(i,j,3)==0; epass(i,j,3)=1; elseif epass(i,j,3)==-1; epass(i,j,3)=2; elseif epass(i,j,3)==2;
loopflag=2; end % this edge has been passed in positive direction
% if epass=-1 then it has already been passed in
neg dir - move on
while loopflag==0
    if min(pos)<3; break; end % if the grain is heading for the edge break out of loop
    if max(pos)>n-3; break; end % ditto
    % check edge types from this point
    locedge=zeros(1,6);
    if e(pos(1)-2,pos(2),3)>0; locedge(1)=1; end
    if e(pos(1),pos(2),1)>0; locedge(2)=1; end
    if e(pos(1),pos(2),2)>0; locedge(3)=1; end
    if e(pos(1),pos(2),3)>0; locedge(4)=1; end
    if e(pos(1)+1,pos(2)-1,1)>0; locedge(5)=1; end
    if e(pos(1)-1,pos(2)-1,2)>0; locedge(6)=1; end
    for k=1:5
        offset=edgetype+k; % look for edges in clockwise direction from current type
        if offset>6; offset=offset-6; end
        if locedge(offset)>0
            if 1<offset & 5>offset; % mark this edge as being passed if it belongs to this unit cell
                if epass(pos(1),pos(2),offset-1)==0; epass(pos(1),pos(2),offset-1)=1; elseif epass(pos(1),pos(2),offset-1)==-1; epass(pos(1),pos(2),offset-1)=2; elseif epass(pos(1),pos(2),offset-1)==2; loopflag=2; end % this edge has been passed in positive direction
            end
            pos=pos+jump(offset,:); % ditto
            edgetype=offset+3; % edge just crossed relative to new vertex
            if edgetype>6; edgetype=edgetype-6; end
            if 1<edgetype & 5>edgetype; % mark this edge as being passed if it belongs to this unit cell
                if epass(pos(1),pos(2),edgetype-1)==0; epass(pos(1),pos(2),edgetype-1)=1; elseif epass(pos(1),pos(2),edgetype-1)==-1; epass(pos(1),pos(2),edgetype-1)=2; elseif epass(pos(1),pos(2),edgetype-1)==2; loopflag=2; end % this edge has been passed in neg direction
            end
        end
    end
end
% look for bottom left point in grain
if loopflag==1; % not previously saved - so save this grain
    glen=length(gtemp)/2;
   ngrain=ngrain+1;
g(ngrain,1:2*glen+1)=[glen gtemp];
end
% pause
clear gtemp;
edge=0; % reset number of edges for this grain
rtemp(1)=i; % vertices for this grain (x values odd, y values even)
gtemp(2)=j;
pos(1)=i; % current x position
pos(2)=j; % current y position
loopflag=0;
if e(i,j,2)>0;
  nedge=nedge+1;
  pos(1)=pos(1)+1;
  pos(2)=pos(2)+1;
  gtemp(3)=pos(1);
  gtemp(4)=pos(2);
  edgetype=6; % edge you've just crossed relative to new vertex
  if epass(i,j,2)==0; epass(i,j,2)=1; elseif epass(i,j,2)==-1; epass(i,j,2)=-1; elseif epass(i,j,2)==2;
  loopflag=2; end % this edge has been passed in positive direction
end % if epass=-1 then it has already been passed in

% check edge types from this point
locedge=zeros(1,6);
if e(pos(1)-2,pos(2),3)>0; locedge(1)=1;end
if e(pos(1),pos(2),1)>0; locedge(2)=1;end
if e(pos(1),pos(2),2)>0; locedge(3)=1;end
if e(pos(1),pos(2),3)>0; locedge(4)=1;end
if e(pos(1)+1,pos(2)-1,1)>0; locedge(5)=1;end
if e(pos(1)-1,pos(2)-1,2)>0; locedge(6)=1;end

for k=1:5
  offset=edgetype+k; % look for edges in clockwise direction from current type
  if offset<6; offset=-offset-6; end
  if locedge(offset)>0
    if 1<offset & 5>offset; % mark this edge as being passed if it belongs to this unit cell
      if epass(pos(1),pos(2),offset-1)==0; epass(pos(1),pos(2),offset-1)=1; elseif
      epass(pos(1),pos(2),offset-1)==-1; epass(pos(1),pos(2),offset-1)=-1; elseif epass(pos(1),pos(2),offset-
      1)==2; loopflag=2; end % this edge has been passed in positive direction
    end
    pos=pos+jump(offset,:);
    edgetype=offset+3; % edge just crossed relative to new vertex
    if edgetype>6; edgetype=edgetype-6; end
    if 1<edgetype & 5>edgetype; % mark this edge as being passed if it belongs to this unit cell
      if epass(pos(1),pos(2),edgetype-1)==0; epass(pos(1),pos(2),edgetype-1)=1; elseif
      epass(pos(1),pos(2),edgetype-1)==-1; epass(pos(1),pos(2),edgetype-1)=-1; elseif epass(pos(1),pos(2),edgetype-1)==2;
      loopflag=2; end % this edge has been passed in positive direction
    end
    if pos==[gtemp(1) gtemp(2)]; if loopflag==0; loopflag=1;end
    break
  end
end
end
end

% look for bottom left point in grain
if loopflag==1; % not previously saved - so save this grain
  glen=length(gtemp)/2;
ngrain=ngrain+1;
end
g(ngrain,1:2*glen+1)=[glen gtemp];
end
% pause % before next i,j
end

gstats=g(:,1);
disp(['number of grains ' num2str(length(gstats))]);
disp(['mean number of edges per grain ' num2str(mean(gstats))]);

if plotfig==1
%figure
%hold on
% plot the grains that have been found over the complete plot as a check
for i=1:length(gstats)
    keepgo=1;
    j=0;
    while keepgo==1
        j=j+2;
        if j>length(g(1,:)) | g(i,j)==0
            keepgo=0;
            px=g(i,2);
            py=g(i,3);
        else
            px=g(i,j);
            py=g(i,j+1);
        end
        if j>2
            plot([pxold px],[pyold py],'r')
        end
        pxold=px;
        pyold=py;
    end
end

title('grains that have been counted are red')

% plot out number of edges data vs that for SS as given in scripta paper
% the following is the edge data for Stainless steel 304 with N=0.9013
% - see Jay Basinger et al Scripta Mat 2005 paper
edge=[4 5 6 5 4 6 3 5 5 10 5 4 5 4 4 5 5 5 4 4 10 4 5 4 4 4 3 7 3 9 6 3 5 5 4 9 6 4 6 5 3 6 4 4 5 9 3 6
4 7 3 3 6 3 10 ...
9 3 3 5 4 6 4 4 3 4 6 4 3 4 5 5 7 5 4 6 5 6 5 5 ...
4 3 3 5 5 6 3 3 3 8 8 4 3 11 5 7 5 6 5 4 7 4 4 5 4 4 4 3 5 4 6 7 3 5 14 3 6 6 5 5 5 4 4 5 4 5 4 3 5 7 8
4 7 ...
5 7 4 6 3 3 4 3 3 7 3 4 4 4 3 5 5 3 12 10 4 7 5 4 4 3 3 4 5 7 12 3 3 4 3 4 5 3 3 6 4 5 5 5 3 4 4 9 10
3 4 9 5 3 3 3 3 4 4 5 3 14 5 3 6 6 5 3 5 10 3 6 5 9 5 3 ...
4 5 5 3 4 4 3 10 3 4 4 3 5 6 5 4 3 5 4 4 5 7 4 3 3 7 3 10 6 4 5 3 6 4 3 4 3 4 3 4 3 8 4 3 5 4 6 3 4 3 3 ...
3 4 3 4 5 6 11 3 4 4 5 7 7 3 4 3 3 7 4 3 5 4 4 14 3 4 6 7 4 5 11 4 3 6 3 5 4 3 8 5 4 3 10 6 5 3 4 4 3 3 7 4
5 5 3 4 4 ...
3 3 4 4 3 3 7 3 13 4 3 11 3 3 3 4 5 4 3 5 3 4 3 4 4 4 3 6 5 8 3 7 5 4 5 5 6 5 3 6 4 4 5 3 6 4 6 6 4
4 3 3 4 7 5 3 4 3 4 4 6 3 7 8 5 3 4 3];

% the following for a generated triangle pseudo lattice:
tedge=gstats;

m=max([max(edge) max(tedge)]); % find max number of edges for both cases
pope=zeros(1,m); % set up vector to keep track of number of grains with each number of edges
for i=1:length(edge)
    n=edge(i);
    pope(n)=pope(n)+1;
end

popt=zeros(1,m); % set up vector to keep track of number of grains with each number of edges
for i=1:length(tedge)
    n=tedge(i);
    popt(n)=popt(n)+1;
end
tot=[pope'/length(edge) popt'/length(tedge)];
figure
bar(tot,'grouped')
title('Population of grains with n-edges for grain structure and pseudo lattice (red)')
xlabel('Number of edges per grain')
ylabel('Probability of having this number of edges')

PT4TRI.m

% percolation search algorithm by Jay Basinger - 2004
% all other code by David Fullwood - March 2005
% percolation data is read from triangle.txt which is written by hexout.m or
% constraint.m

% This is the main program who's goal is to take a scan (in the case of real data) and
% return the percolation threshold based on the average
% percentage at which percolation is found to occur from the
% top of a scan to the bottom, in other words, a cluster, or
% connected set of traces, runs from the top of the scan
% to the bottom. The user sets the number of iterations over
% which the first occurrences of percolation are found as the
% percentage is raised as well as the lower starting percentage
% This program will then return the averaged percentage found
% over the set number of iterations.

clear; close all;clc;
%warning off MATLAB:divideByZero
load 'triangle.txt';
x1=triangle(:,1);
y1=triangle(:,2);
x2=triangle(:,3);
y2=triangle(:,4);
gblength=ones(length(x1),1);
csl=triangle(:,5);
csl=csl*0;
angle=triangle(:,5);
numberoftraces=length(x1)
clear triangle;
%make x1 always less than x2 - for convenience
for c=1:length(x1)
    if (x2(c)<x1(c))
        d_temp=x1(c);
        x1(c)=x2(c);
        x2(c)=d_temp;
    end


\[
d_{\text{temp}} = y_1(c); \quad y_1(c) = y_2(c); \quad y_2(c) = d_{\text{temp}};
\]

end
end

% window size control
mx2 = max(x2);
mx1 = max(x1);
my2 = max(y2);
my1 = max(y1);
xn2 = min(x2);
xn1 = min(x1);
y2 = min(y2);
y1 = min(y1);
if mx2 > mx1
    xscanmax = mx2;
else
    xscanmax = mx1;
end
if my2 > my1
    yscanmax = my2;
else
    yscanmax = my1;
end
if xn2 > xn1
    xscanmin = xn2;
else
    xscanmin = xn1;
end
if y2 > y1
    yscanmin = y1;
else
    yscanmin = y2;
end
plotbool = 1; % if this equals one, it will plot the graphs
p = 3; % initiating p and giving it a value different from 0 or 1
LengthAve = 0;
NumFract = 0;
btype = 0; % if btype is 0 it filters out CSL and low angle boundaries, if 1
% it will filter out HAR boundaries i.e. susceptible boundaries

[NumFract, LengthAve, p, RclusterSN, RclusterSL, NmassClustS, LmassClustS] = psubc4(plotbool, x1, y1, x2, y2, xscanmin, xscanmax, yscanmin, yscanmax, gblength, csl, angle, btype);

RSN = RclusterSN';
RSL = RclusterSL';
NS = NmassClustS';
LS = LmassClustS';
LSSray = 1:20:(max(LS));
LShist = hist(LS, LSSray)';
NShist = 1:(max(NS));
NShist = hist(NS, NShist)';
% filtering and combining all data
FNShist = find(NShist > .01);
NewNShist = NShist(FNShist);
clear NShist;
NShist = NewNShist;
FLShist = find(LShist > .01);
NewLShist = LShist(FLShist);
clear NSHist;
NSHist=NewNsHist;
axis equal
p % if p=1 it percolated

\textbf{psubc4.m}

\begin{verbatim}
function
[NumFract,LengthAve,p,RelclusterN,RelclusterL,NmassClust,LmassClust]=psubc2(plotbool,cfx1,cfy1,cfx2,cfy2,xbegin,xend,ybegin,yend,gblength,cfangle,btype);
%take in unfiltered information, filter by random box size so ther is a
%list of only the traces that are in the box and their modified lengths.
%Recorede total length of all traces within the box. Filter out good traces
%and record length of bad traces. Send bad traces to percolation engine
%(perc.m) record LsubA(bad)/LsubA(All) as a percentage and return the
%LsubA(bad)/LsubA(all) as well as p=0 if there was no percolation in the
%particular window and p=1 if it did percolate accross the window.

LaAll=sum(gblength);
if btype==0
%characteristic filter - csl and LAR
  g=0; d=0;fx1=[];fx2=[];fy1=[];fy2=[];fgblength=[];tracelist=[];
  while g <length(cfcsl)
    g=g+1;
    if cfcsl(g)==0 %if the traces is not special and not low angle
      %make an array of the non-csl/HAR traces.
      if cfangle(g) > 1 % was 15 *** this is the standard 15 degree angle limit between
        %Low angle and high angle random boundaries
          d=d+1;
          fx1(d)=cfx1(g);
          fx2(d)=cfx2(g);
          fy1(d)=cfy1(g);
          fy2(d)=cfy2(g);
          fgblength(d)=gblength(g); %makes an array of the length of non-csl
          %traces lengths.
          tracelist(d)=trace(g);
        end
      end
    end %if cfcsl(g)==0
  end %while
end %if btype==0

if btype==1
%characteristic filter - non-CSL and HAR (filters them out)
  g=0; d=0;fx1=[];fx2=[];fy1=[];fy2=[];fgblength=[];tracelist=[];
  while g <length(cfcsl)
    g=g+1;
    if (cfcsl(g) ~= 0 | cfangle(g) <= 15)%if the trace is special and low angle
      %make an array of the csl/LAR traces.
      %this is the standard 15 degree angle limit between
      %Low angle and high angle random boundaries
        d=d+1;
        fx1(d)=cfx1(g);
        fx2(d)=cfx2(g);
        fy1(d)=cfy1(g);
        fy2(d)=cfy2(g);
        fgblength(d)=gblength(g); %makes an array of the length of non-csl
        %traces lengths.
        tracelist(d)=trace(g);
      end
    end %if cfcsl(g)==0
  end %while
end %if btype==1
\end{verbatim}
fgblength(d)=gblength(g); %makes an array of the length of csl
%traces lengths.
  tracelist(d)=trace(g);
end
end
% end characteristic (csl) filter
end %if btype =1
%this function is for the case where there are no susceptible traces in
%the window placed there.
if isempty(tracelist)==1 | tracelist(1)==0
  p=0;
  LengthAve=0;
  NumFract=0;
  return;
end
LaBad=sum(fgblength);
for k=1:length(fx1)%now that the accurate list
  %of susceptible or resistant trace ID numbers is compiled
  %this plots them
  xt(1)=fx1(k);
  xt(2)=fx2(k);
  yt(1)=-fy1(k);
  yt(2)=-fy2(k);
  if plotbool==1
    plot(xt,yt,'-r')
    hold on
  end
end
if plotbool==1; figure;end
NumFract=d/g; %instead of length percentage as in LengthAve, this is the number fraction of
%d(bad) to g(total) traces in a window.
LengthAve=LaBad/LaAll; %Percentage of bad grain boundaries in box

[RCusterN,RclusterL,NmassClust,LmassClust,p]=perc3(tracelist,cfx1,cfx2,cfy1,cfy2,xend,yend,plotbool,xbegin,ybegin);
return;

perc3.m

% function called from psubc4.m
% Jay Basinger
function
[RclusterN,RclusterL,NmassClust,LmassClust,I]=percolation(completernlist,x1,x2,y1,y2,xscanmax,yscanmax
max,plotbool,xscanmin,yscanmin);

tracelist = zeros(length(completernlist),8);
%this makes a more succinct list of only the traces we are interested in
%and their associated endpoints.
for b=1:length(completernlist)
  tracelist(b,1)=x1(completernlist(b));
  tracelist(b,2)=y1(completernlist(b));
  tracelist(b,3)=x2(completernlist(b));
  tracelist(b,4)=y2(completernlist(b));
  tracelist(b,5)=0;
  tracelist(b,6)=0;%status of x1 y1 end
tracelist(b,7)=0; % status of x2 y2 end ie, whether the ends have been connected
% 0 is for unconnected and 1 is for connected
tracelist(b,8)=1;
end

% Initializations
I=0;
h=1;
p=0;
plotcolor=hsv(200);
pc(1)=plotcolor(1,1);
pc(2)=plotcolor(1,2);
pc(3)=plotcolor(1,3);
u=1;
lc=0;
line=0;
linekeeper=0;
pcOC=1;
while h<=length(completernlist)
% Start with trace h and check to see if it has been connected, i.e.
% tracelist(h,6) and tracelist(h,7)==1 if it is connected. If it has been connected
% already add one to h and check again.
xycount=1;
if h>1
[p,rgyrn,rgyrl,Lmass,Nmass]=pcboxplotter3(xylist,pc,xscanmax,xscanmin,yscanmax,yscanmin,plotbool,yscanmin);
RclusterN(pcOC)=rgyrn;
RclusterL(pcOC)=rgyrl;
LmassClust(pcOC)=Lmass;
NmassClust(pcOC)=Nmass;
pcOC=pcOC+1;
if p==1
I=1;
end
end
if lc==0
linekeeper=line+linekeeper;
end
line=0;
lc=lc+1;
original=1;
[h,callagain]=tracecheck(tracelist,h);
while callagain==1
if h>(length(completernlist)) % if it looks like we've gone past the last
% item in the list then send it to pcboxplotter to see
% Sends a cluster to be plotted, get a radius of gyration, number fraction mass,
% and length mass
[p,rgyrn,rgyrl,Lmass,Nmass]=pcboxplotter3(xylist,pc,xscanmax,xscanmin,yscanmax,yscanmin,plotbool,yscanmin);
RclusterN(pcOC)=rgyrn;
RclusterL(pcOC)=rgyrl;
LmassClust(pcOC)=Lmass;
NmassClust(pcOC)=Nmass;
pcOC=pcOC+1;
if p==1
I=1;
end
end
return;
end
[h,callagain]=tracecheck(tracelist,h);
end %end while callagain loop
u=u+1; %this is just for changing the plot color
% Then store its length and pick x1 and y1 as the
% starting point for each h that is picked.
x=tracelist(h,1);
y=tracelist(h,2);
if u==max(size(plotcolor))+1
    u=1;
end
pc(1)=plotcolor(u,1); %changes the animal color each time
pc(2)=plotcolor(u,2);
pc(2)=plotcolor(u,3);
n=1; %initializes the hub number to 1
% create a hub for n=1 and make
% that match from the hublist and the hub structure.
hub(n,1)=x;
hub(n,2)=y;
hub(n,3)=1;
hub(n,4)=h;
hub(n,5)=0;
hub(n,6)=0;
hub(n,7)=0;
clear xylist;
tracelist(h,6)=1;
traceid=h;
while n > 0
    [hublist,hub,tracelist,line,n]=percloop(pc,original,tracelist,n,x,y,h,hub,line,traceid);
    newanimal=0;
    % perform hubcheck to see if the trace id's are the same. If they are then remove
    % the trace from the list, and move all the other id's down. Lower the status
    % of both hubs and add the length to the total. do not create another hub, instead
    % follow the remaining traceid and check its x and y point.
    % check for connecting traces
    % if there is 1:
    % the hub counter is not raised. add the length to the total, change the x and y
    % values to the unconnected end of the trace and begin again.
    if (length(hublist)==1 & hublist==0)
        both=0;
        if (tracelist(hub(n,4),7)==1 & tracelist(hub(n,4),6)==1)
            if (x==tracelist(hub(n,4),1) & y==tracelist(hub(n,4),2))
                x=tracelist(hub(n,4),3);
y=tracelist(hub(n,4),4);
            end
            if (x==tracelist(hub(n,4),3) & y==tracelist(hub(n,4),4))
                x=tracelist(hub(n,4),1);
y=tracelist(hub(n,4),2);
            end
        end
    end
    if (tracelist(hub(n,4),7)==1 & both==1)%if the x2 y2 are connected, then look at x1 y1 for connections
        x=tracelist(hub(n,4),1);
y=tracelist(hub(n,4),2);
    end
end
if (tracelist(hub(n,4),6)==1 & both==1) %if x1 y1 are connected, look at x2 y2 for connections
    x=tracelist(hub(n,4),3);
y=tracelist(hub(n,4),4);
end

traceid=hub(n,4);
tracelist(traceid,8)=0;
line=line+sqrt((tracelist(hub(n,4),1)-tracelist(hub(n,4),3))^2+(tracelist(hub(n,4),2)-tracelist(hub(n,4),4))^2);

xp(1)=tracelist(traceid,1);
xp(2)=tracelist(traceid,3);
yp(1)=tracelist(traceid,2);
yp(2)=tracelist(traceid,4);

hub(n,3)=hub(n,3)-1;
if plotbool==1
    plot(xp,yp,'Color',[pc(1) pc(2) pc(3)])
    hold on
end

xylist(xycount,1)=tracelist(traceid,1);
xylist(xycount,2)=tracelist(traceid,2);
xylist(xycount,3)=tracelist(traceid,3);
xylist(xycount,4)=tracelist(traceid,4);
xycount=xycount+1;
end

% if there are 2:
% create a hub(n) starting with 1 and adding 1 to n every time this is found. Store into it the x and y points as well as
% the number of connections and their trace id's. hub(n,4) and hub(n,5) are for
% storing those from the hublist.Also store all info into animalhublist which will not change by n's just
% keep increasing
% in order to store them all in a list aimed to check for hubs connecting
% to hubs.
% Next, choose the last trace to follow according to
% the hub(n,3), or status value.add the length of the trace followed.
% So, if the value is 2, it will choose to follow
% hub(n,5) trace to the unconnected point, which it will test for connections.
% before testing that point lower
% the status of the hub by one, which says that that branch has been
% connected.(begin again)
  if hub(n,3)==2

    both=0;
  if (tracelist(hub(n,3+(hub(n,3))),7)==1 & tracelist(hub(n,3+(hub(n,3))),6)==1) %if x2 y2 are connected, then look at x1 y1 for connections
    if (x==tracelist(hub(n,3+(hub(n,3))),1) & y==tracelist(hub(n,3+(hub(n,3))),2))
        x=tracelist(hub(n,3+(hub(n,3))),3);
y=tracelist(hub(n,3+(hub(n,3))),4);
    end
  if (x==tracelist(hub(n,3+(hub(n,3))),3) & y==tracelist(hub(n,3+(hub(n,3))),4))
        x=tracelist(hub(n,3+(hub(n,3))),1);
y=tracelist(hub(n,3+(hub(n,3))),2);
    end
    both=1;
  end
end
end
if tracelist(hub(n,3+(hub(n,3))),6)==1 %if x1 y1 are connected, look at x2 y2 for connections
    x=tracelist(hub(n,3+(hub(n,3))),3);
    y=tracelist(hub(n,3+(hub(n,3))),4);
end
traceid=hub(n,3+(hub(n,3)));
tracelist(traceid,8)=0;
xp(1)=tracelist(traceid,1);
xp(2)=tracelist(traceid,3);
yp(1)=tracelist(traceid,2);
yp(2)=tracelist(traceid,4);
if plotbool==1
    plot(xp,yp,'Color',[pc(1) pc(2) pc(3)])
    hold on
end
hub(n,3)=hub(n,3)-1;
line=sqrt((tracelist(traceid,1)-tracelist(traceid,3))^2+(tracelist(traceid,2)-tracelist(traceid,4))^2)+line;
n=n+1;
xylist(xycount,1)=tracelist(traceid,1);
xylist(xycount,2)=tracelist(traceid,2);
xylist(xycount,3)=tracelist(traceid,3);
xylist(xycount,4)=tracelist(traceid,4);
xycount=xycount+1;
end %if hub(n,3)>=2
% if there are 0:
% check status of hub. if it is zero lower n by one
% If n is negative one this is the end of the percolation animal set hub number one to x2, y2 of trace h and
begin again.
if ((hublist==0) | isempty(hublist)==1 )
    if original==1
        xylist(xycount,1)=x;
        xylist(xycount,2)=y;
        xycount=xycount+1;
    end
while(hub(n,3)==0)
    n=n-1;
    if n<=0
        [r,c]=size(xylist);
        if r==1
            clear xylist;
            xylist(xycount,1)=tracelist(traceid,1);
            xylist(xycount,2)=tracelist(traceid,2);
            xylist(xycount,3)=tracelist(traceid,3);
            xylist(xycount,4)=tracelist(traceid,4);
        end
        if h==length(completernlist)
            if lc==0
                linekeeper=line+linekeeper;
            end
            %Sends a cluster to be plotted, get a radius of gyration, number fraction mass,
            %and length mass
            [p,rgyrn,rgyrl,Lmass,Nmass]=pcboxplotter3(xylist,pc,xscanmax,xscanmin,yscanmax,yscanmin,plotbool,yscanmin);
            RclusterN(pcOC)=rgyrn;
            RclusterL(pcOC)=rgyrl;
            LmassClust(pcOC)=Lmass;
        end
    end
end
% if hublist==0 | isempty(hublist)==1 )
if original==1
    xylist(xycount,1)=x;
    xylist(xycount,2)=y;
    xycount=xycount+1;
end
while(hub(n,3)==0)
    n=n-1;
    if n<=0
        [r,c]=size(xylist);
        if r==1
            clear xylist;
            xylist(xycount,1)=tracelist(traceid,1);
            xylist(xycount,2)=tracelist(traceid,2);
            xylist(xycount,3)=tracelist(traceid,3);
            xylist(xycount,4)=tracelist(traceid,4);
        end
        if h==length(completernlist)
            if lc==0
                linekeeper=line+linekeeper;
            end
            %Sends a cluster to be plotted, get a radius of gyration, number fraction mass,
            %and length mass
            [p,rgyrn,rgyrl,Lmass,Nmass]=pcboxplotter3(xylist,pc,xscanmax,xscanmin,yscanmax,yscanmin,plotbool,yscanmin);
            RclusterN(pcOC)=rgyrn;
            RclusterL(pcOC)=rgyrl;
            LmassClust(pcOC)=Lmass;
        end
    end
end
NmassClust(pcOC)=Nmass;
pcOC=pcOC+1;
if p==1
  I=1;
end
return;
end
h=h+1;
clear hub;
clear length;
x=-1;
y=-1;
newanimal=1;
breatk;
end
end
if newanimal==1
  n=-1;
breatk;
end
if (tracelist(hub(n,3+(hub(n,3))),7)==1 & tracelist(hub(n,3+(hub(n,3))),6)==1)
  if (x==tracelist(hub(n,3+(hub(n,3))),1) & y==tracelist(hub(n,3+(hub(n,3))),2))
    x=tracelist(hub(n,3+(hub(n,3))),3);
y=tracelist(hub(n,3+(hub(n,3))),4);
  end
  if (x==tracelist(hub(n,3+(hub(n,3))),3) & y==tracelist(hub(n,3+(hub(n,3))),4))
    x=tracelist(hub(n,3+(hub(n,3))),1);
y=tracelist(hub(n,3+(hub(n,3))),2);
  end
  both=1;
end
if tracelist(hub(n,3+(hub(n,3))),7)==1%if the x2 y2 are connected, then look at x1 y1 for connections
  x=tracelist(hub(n,3+(hub(n,3))),1);
y=tracelist(hub(n,3+(hub(n,3))),2);
end
if tracelist(hub(n,3+(hub(n,3))),6)==1 %if x1 y1 are connected, look at x2 y2 for connections
  x=tracelist(hub(n,3+(hub(n,3))),3);
y=tracelist(hub(n,3+(hub(n,3))),4);
end
traceid=hub(n,3+(hub(n,3)));
line=sqrt((hub(n,1)-x)^2+(hub(n,2)-y)^2)+line;
tracelist(traceid,8)=0;
hub(n,3)=hub(n,3)-1;
xp(1)=tracelist(traceid,1);
xp(2)=tracelist(traceid,3);
yp(1)=tracelist(traceid,2);
yp(2)=tracelist(traceid,4);
if plotbool==1
  plot(xp,yp,'Color',[pc(1) pc(2) pc(3)])
  hold on
end
%run length and boundary finding stuff
xystlist(xycount,1)=tracelist(traceid,1);
xystlist(xycount,2)=tracelist(traceid,2);
xystlist(xycount,3)=tracelist(traceid,3);
xystlist(xycount,4)=tracelist(traceid,4);
xcycount=xycount+1;
end %if
original=0;
end %while n~=0
%lengthlist(h)=line; %this should store the the lengths of all clusters encountered
end %while h<length(completerlist)
'at the way end'
%Sends a cluster to be plotted, get a radius of gyration, number fraction mass,
%and length mass
[p,rgyrn,rgyrl,Lmass,Nmass]=pcboxplotter3(xylist,pc,xscanmax,xscanmin,yscanmax,plotbool,yscanmin);
RclusterN(pcOC)=rgyrn;
RclusterL(pcOC)=rgyrl;
LmassClust(pcOC)=Lmass;
NmassClust(pcOC)=Nmass;
pcOC=pcOC+1;
if p==1
   I=1;
end

perceloop.m

% function called from perc3.m
% Jay Basinger
function [hublist,hub,tracelist,line,n]=perceloop(pc,original,tracelist,n,x,y,hub,line,traceid);
hublist=[];
[r,c]=size(hub);
if (n>r & n~=1)
   hub(n,3)=0;
end
if (hub(1,3)~=0 & n<r & n==1)
   n=n+1;
   hub(n,3)=0;
end
[r,c]=size(tracelist);
if r < 8
   trlistlength=r;
else
   trlistlength=length(tracelist);
end
for i=1:trlistlength
   %compares x1 and y1 of the h loop to x1 & y1 of the i loop
   if ((traceid ~= i) & ((x <= tracelist(i,1)+.02 & x>= tracelist(i,1)-.02) & (y<=tracelist(i,2)+.02 &
      y>=tracelist(i,2)-.02)) & (tracelist(i,8)~=0))
      hub(n,3)=hub(n,3)+1;
      hublist(hub(n,3))=i;
      tracelist(i,6)=1; %setting the zero value to 1 shows that this point has been found to hub(n,3)
   end
   if ((traceid ~= i) & ((x <= tracelist(i,3)+.02 & x>= tracelist(i,3)-.02) & (y<=tracelist(i,4)+.02 &
      y>=tracelist(i,4)-.02)) & (tracelist(i,8)~=0))
      hub(n,3)=hub(n,3)+1;
      hublist(hub(n,3))=i;
      tracelist(i,7)=1; %setting the zero value to 1 shows that this point has been found to hub(n,3)
   end
end % % perform hubcheck to see if the trace id's are the same. If they are then remove
% the trace from the list, and move all the other id's down. Lower the status
% of both hubs and add the length to the total. do not create another hub, instead
% follow the remaining traceid and check its x and y point.
if isempty(hublist)==1
    hublist=0;
    return;
end
if (n==1 & original==1)
    if length(hublist)==1;
        hub(n,4)=h;
        hub(n,5)=hublist(1);
        'does it ever get here? ------ I mean really'
    end
    if length(hublist)==2;
        hub(n,4)=h;
        hub(n,5)=hublist(2);
    end
    if length(hublist)==3;
        hub(n,4)=h;
        hub(n,5)=hublist(2);
        hub(n,6)=hublist(3);
        'there is an animalhub(acount,7)'
    end
else
    if length(hublist)==1;
        if hublist(1)==0
            hub(n,4)=0;
            hub(n,3)=0;
        else
            hub(n,1)=x;
            hub(n,2)=y;
            hub(n,4)=hublist(1);
        end
    end
    if length(hublist)==2;
        hub(n,1)=x;
        hub(n,2)=y;
        if hublist(1)==0
            hub(n,4)=hublist(2);
            b=hublist(2);
            clear hublist;
            hublist=b;
        else
            hub(n,4)=hublist(1);
            hub(n,5)=hublist(2);
        end
    end
    if length(hublist)==3;
        hub(n,1)=x;
        hub(n,2)=y;
if hublist(1)==0
    hub(n,4)=hublist(2);
    hub(n,5)=hublist(3);
    hub(n,3)=hub(n,3)-1;
    a=hublist(2);
    b=hublist(3);
    clear hublist;
    hublist(1)=a;
    hublist(2)=b;
else
    hub(n,4)=hublist(1);
    hub(n,5)=hublist(2);
    hub(n,6)=hublist(3);
end
end
if length(hublist)==4;
    hub(n,1)=x;
    hub(n,2)=y;
    if hublist(1)==0
        hub(n,4)=hublist(2);
        hub(n,5)=hublist(3);
        hub(n,6)=hublist(4);
        hub(n,3)=hub(n,3)-1;
        a=hublist(2);
        b=hublist(3);
        c=hublist(4);
        clear hublist;
        hublist(1)=a;
        hublist(2)=b;
        hublist(3)=c;
    else
        hub(n,4)=hublist(1);
        hub(n,5)=hublist(2);
        hub(n,6)=hublist(3);
        hub(n,7)=hublist(4);
    end
    'there is an animalhub(acount,7)'
end
end

pcboxplotter.m

% called from perc3.m
% Jay Basinger
function [I,radgn,radgl,lmass,nmass] = pcboxplotter3(xylist,pc,xscanmax,xscanmin,yscanmax,plotbool,yscanmin);
I=0;
xmax=max(xylist(:,1));
ymax=max(xylist(:,2));
xmin=min(xylist(:,1));
ymin=min(xylist(:,2));
xdif=xmax-xmin;
ydif=ymax-ymin;
if (abs(yscanmax-ymax)<10 & abs(ymin-yscanmin)<10)
    I=1;
end

90
end
if (abs(xscanmax-xmax)<10 & abs(xmin-xscanmin)<10)
    I=1;
end

%radius of gyration:
%first find midpoints of all lines
mid = zeros(length(xylist(:,1))-1,3);
if length(xylist(:,1))==1
    'something is odd'
end
for k=2:length(xylist(:,1))
    n=k-1;
    mid(n,1) = (xylist(k,1)+xylist(k,3))/2; %this is the x midpoint
    mid(n,2) = (xylist(k,2)+xylist(k,4))/2; %y midpoint for each trace
    mid(n,3) = sqrt((xylist(k,1)-xylist(k,3))^2+(xylist(k,2)-xylist(k,4))^2);
end
xav = sum(mid(:,1).*mid(:,3))/(sum(mid(:,3)));
yav = sum(mid(:,2).*mid(:,3))/(sum(mid(:,3)));
xav = sum(mid(:,1).*mid(:,3))/(sum(mid(:,3)));
yav = sum(mid(:,2).*mid(:,3))/(sum(mid(:,3)));
r = zeros(length(mid(:,1))-1,1);
for m = 1:length(mid(:,1))
    r(m)=((mid(m,1)-xav)^2+(mid(m,2)-yav)^2);
end
hold on
lmass = sum(mid(:,3));
radgn = sum(r)/(length(r));
radgl = sum(r)/(lmass);
nmass = (length(xylist(:,1))-1);

tracecheck.m

% called from perc3.m
% Jay Basinger
function [m,callagain]=tracecheck(tracelist,h);
callagain=0;
if tracelist(h,7)==1 & tracelist(h,6)==1 %if both ends are connected
    callagain=1;
    m=h+1;
end
if tracelist(h,8)==0
    callagain=1;
    m=h+1;
end
if tracelist(h,7)==0 & tracelist(h,6)==0 %if neither end is connected
    m=h;
    callagain=0;
end
if tracelist(h,8)==1
    m=h;
    callagain=0;
end
Appendix 2: Optimization Codes

The main codes used in the second part of the thesis include:

**Biobjective Optimization:**

runnbi2.m – uses GWS and ANBI methods to find property closure for 2 properties  
macfun.m – called from runnbi2.m  
mycon.m – called from runnbi2.m  

lin.m – finds the optimal points to the linear portion of the properties closure problem and plots P and Q for these optimal microstructures.

genclose.m – uses a maxmin algorithm on a genetic algorithm to find the property closure.

**Property_closure_matrices.mat** – basic matrices developed by Massimiliano Binci (Drexel) for calculating C11 and C12 for a sample of two isotropic media; called from all the biobjective optimization files.

**General optimization routines:**

steepd.m – runs the steepest descent method on an unconstrained quadratic problem  
quasi.m – runs a quasi newton method on an unconstrained quadratic problem  
cgs.m – runs a conjugate gradient method on an unconstrained quadratic problem  
mrh – runs a minimal reduced gradient method on an unconstrained quadratic problem  
qp – runs a quadratic programming algorithm using the dual method on a constrained quadratic problem.
% Runs Matlab SQP solver for property closure of form:
% P = x'N11x+b1'x
% Q = x'N12x+b2'x
% from runparito.m...
% subfiles required are maxfun.m and mycon.m
% DT Fullwood 21 june 2005; final version 14th July 2005
% this version adjusts lambda to find different points on the front (GWS
% method); it then searches between the found points for further points
% until the distance between points meets the tolerance. It initially uses
% GWS with multiple trials allowed (up to pqfirst), and with a constraint
% on the maximum x coefficient to try and help find points.
% If this does not result in a new point, the ANBI method is used.

global N11 N12 b1 b2 lambda1 lambda2 p1x p2x p1y p2y
load property_closure_matrices.mat

Tstart=clock;   % time how long this takes

maxtol=1000;    % maximum allowable value for properties (anything above this is spurious)
numit=20;       % number of iterations to fill in missing points
numlam=53;      % number of initial values for lambda in J=\lambda1(P)+\lambda2(Q)
tolp=1e-3;      % tolerance on P for two values being equal
pqfirst=5;      % number of iterations using QP prog before resorting to NBI
distolfac=5;    % factor for defining max distance between points
bad=0;          % number of bad points

N11=Ahr1;       % set up the matrices from Massimiliano Bianci (Drexel) in my standard form
N12=Ahr2;
b1=Bh1;
b2=Bh2;
n=length(b1);

% constraints Cx<=a
C=[eye(n,n) -eye(n,n)]'; % constraints are simply that all variable greater than zero
a=[2*ones(n,1); -zeros(n,1)]; % and that the sum of the variables is greater than n
VLB=zeros(n,1); % set upper and lower bounds to the QP problem (not absolutely necessary
VUB=ones(n,1)*2; % since the constraints already set bounds
vec1=cos([1:numlam]*2*pi/numlam);  % set up the values for lambda - choose numlam vectors in unit
vec2=sin([1:numlam]*2*pi/numlam);  % minus would make it clockwise
x0=ones(length(b1),1); % starting point - chosen in feasible region
options=optimset('GradObj','on','Hessian','on','display','off','GradConstr','on','MaxIter',500,'TolX',1e-
10,'TolFun',1e-10);

warning off all
% find intial set of points in this section
puthere=0; % counter used to make sure only distinct points are kept
for i=1:length(vec1)
    puthere=puthere+1;
    flag=0;
    lambda1g(puthere)=vec1(i); % vec1 and vec2 give values that search a full circle around closure

    ...
\begin{verbatim}
lambda2g(puthere)=vec2(i);
lambda1=lambda1g(puthere);
lambda2=lambda2g(puthere);
H=(lambda1*N11+lambda2*N12)/2;
f=(lambda1*b2+lambda2*b2);
x=quadprog(H,f,C,a,[],VUB,VUB,x0,options);
xx(:,puthere)=x;
pp(puthere)=x'*N11*x+b1'*x;
if abs(pp(puthere))>1000; puthere=puthere-1;flag=1;end  % ignore daft answers
if puthere>1 & flag==0
    if abs(pp(puthere)-pp(puthere-1))<tolp
        puthere=puthere-1;  % write oer this point next time if it is not a new point
        flag=1;
    end
end
qp(puthere+flag)=x'*N12*x+b2'*x;
if abs(qp(puthere))>1000 & flag==0; puthere=puthere-1;end
end

% new code 25 june to find more points using GWS / ANBI pareto points
distolp=abs(max(pp)-min(pp))/distolfac;  % max distance allowed between points
distolq=abs(max(qp)-min(qp))/distolfac;

flag=1;
count=0;
countnbi=0;
while (flag==1)&(count<numit)
    flag=0;
count=ccount+1
    added=0;  % number of extra points added during the loop
    nlen=length(pp);
    for i=1:nlen-1
        j=i+added;
        if abs(pp(j+1)-pp(j))>distolp | abs(qp(j+1)-qp(j))>distolq   % if length between these points is too big...(find more points)
            flag=1;
            pqcount=1;
pqfound=0;
            while pqcount<pqfirst & pqfound==0
                [mval,mx]=max(abs(xx(:,j+1)-xx(:,j)));
                minx=min([xx(mx,j) xx(mx,j+1)]+0.5*mval);  % set min constraint for x(mx)
                maxx=min([xx(mx,j) xx(mx,j+1)])+0.5*mval);  % set max constraint
                lambda1=lambda1g(j)+(lambda1g(j+1)-lambda1g(j))/(pqcount^2*3);
                lambda2=lambda2g(j)+(lambda2g(j+1)-lambda2g(j))/(pqcount^2*3);
                if mx~=0  % if mx non-zero, you need to add these extra constraints on the variables
                    cvec=zeros(n,1);
                    cvec(mx)=1;
                    C0=[C; -cvec'; cvec'];
                    a0=[a; -minx; maxx];
                else
                    C0=C;
                    a0=a;
                end

                H=(lambda1*N11+lambda2*N12)/2;
f=(lambda1*b2+lambda2*b2);
\end{verbatim}
x=quadprog(H,f,C0,a0,[],[],VLB,VUB,x0,options);
p-temp=x'*N11*x+b1'*x;
if abs(p-temp)>maxtol; p-temp=0;end
q-temp=x'*N12*x+b2'*x;
if abs(q-temp)>maxtol; q-temp=0;end
if ((pp(j)<p-temp)&(p-temp<pp(j+1))) | ((pp(j)>p-temp)&(p-temp>pp(j+1))) % check new point is between previous 2
    pp=[pp(1:j) p-temp pp(j+1:nlen+added)];
    qp=[qp(1:j) q-temp qp(j+1:nlen+added)];
    xx=[xx(:,1:j) x xx(:,j+1:nlen+added)];
    lambda1g=[lambda1g(1:j) lambda1 lambda1g(j+1:nlen+added)];
    lambda2g=[lambda2g(1:j) lambda2 lambda2g(j+1:nlen+added)];
    added=added+1;
    pqfound=1;
else bad=bad+1; % keep track of number of rejected points
end
if pqfound==0
    lambda1= lambda1g(j+1)-(lambda1g(j+1)-lambda1g(j))/(pqcount^2*3);
    lambda2= lambda2g(j+1)-(lambda2g(j+1)-lambda2g(j))/(pqcount^2*3);
    if mx~=0 % if mx non-zero, you need to add these extra constraints on the variables
        cvec=zeros(n,1);
        cvec(mx)=1;
        C0=[C; -cvec'; cvec'];
        a0=[a; -minx; maxx];
        else
            C0=C;
            a0=a;
        end
    end
    H=(lambda1*N11+lambda2*N12)/2;
    f=(lambda1*b2+lambda2*b2);
    x=quadprog(H,f,C0,a0,[],[],VLB,VUB,x0,options);
p-temp=x'*N11*x+b1'*x;
if abs(p-temp)>maxtol; p-temp=0;end
q-temp=x'*N12*x+b2'*x;
if abs(q-temp)>maxtol; q-temp=0;end
if ((pp(j)<p-temp)&(p-temp<pp(j+1))) | ((pp(j)>p-temp)&(p-temp>pp(j+1)))
    pp=[pp(1:j) p-temp pp(j+1:nlen+added)];
    qp=[qp(1:j) q-temp qp(j+1:nlen+added)];
    xx=[xx(:,1:j) x xx(:,j+1:nlen+added)];
    lambda1g=[lambda1g(1:j) lambda1 lambda1g(j+1:nlen+added)];
    lambda2g=[lambda2g(1:j) lambda2 lambda2g(j+1:nlen+added)];
    added=added+1;
    pqfound=1;
else bad=bad+1; % keep track of number of rejected points
end
end
pqcount=pqcount+1;
end

if pqfound==0 % if no point found use generalized nbi method
    p1x=pp(j);
    p2x=pp(j+1);
    p1y=qp(j);
    p2y=qp(j+1);
    lambda1=-(p2y-p1y);
\[ \lambda_2 = p_2x - p_1x; \]
\[ x = \text{fmincon}(\text{macfun}, x, 0, C, a, [], [], [], [], @mycon, \text{options}); \]
\[ p_{temp} = x^*N_{11}x + b_1^*x; \]
if abs(p_{temp}) > maxtol; p_{temp} = 0; end
\[ q_{temp} = x^*N_{12}x + b_2^*x; \]
if abs(q_{temp}) > maxtol; q_{temp} = 0; end
if ((p_{j}) < p_{temp}) \& (p_{temp} < p_{j+1}) | ((p_{j}) > p_{temp}) \& (p_{temp} > p_{j+1})
\[ p = [p_{1:j} \quad p_{j+1:nlen+added}]; \]
\[ q = [q_{1:j} \quad q_{j+1:nlen+added}]; \]
\[ x = [x(:,1:j) \quad x_{j+1:nlen+added}]; \]
\[ \lambda_1g = [\lambda_1g(1:j) \quad \lambda_1g(j+1:nlen+added)]; \]
\[ \lambda_2g = [\lambda_2g(1:j) \quad \lambda_2g(j+1:nlen+added)]; \]
\[ \text{added} = \text{added} + 1; \]
\[ \text{countnbi} = \text{countnbi} + 1; \]
else bad = bad + 1;
end
end
end
end
end

\text{Tend} = \text{clock};
\text{etime}(\text{Tend}, \text{Tstart}) \quad \% \text{time elapsed}
bad \quad \% \text{number of bad points} \quad - \text{i.e. points rejected}
figure
\text{hold on}
P = p + 205.9291; \quad \% \text{translation for actual } P \quad - \text{see email from max (Massimiliano Binci) 27 June 05}
Q = q + 53.9587;
\text{plot}(P, Q, '*k')
\text{xlabel('Property } P = \text{C}_1_1 \text{ (MPa)')}
\text{ylabel('Property } Q = \text{C}_1_2 \text{ (MPa)')}
\text{title('Property Closure for \text{C}_1_1 \text{ and } \text{C}_1_2')}
\text{countnbi} \quad \% \text{number of nbi points}
\text{warning on all}

\text{macfun.m}

\% \text{function to minimize for max's matrix}
\% \text{called from runnbi2}
\% DT Fullwood 21st June 2005

function [floc, gradloc, hessloc] = macfun(xloc);

\text{global } N_{11} N_{12} b_1 b_2 \lambda_1 \lambda_2
\text{floc} = xloc^*(\lambda_1*N_{11} + \lambda_2*N_{12})xloc + (\lambda_1*b_1 + \lambda_2*b_2)^*xloc;

\text{if nargout} > 1 \quad \% \text{fun called with two output arguments}
\quad \text{gradloc} = 2*(\lambda_1*N_{11} + \lambda_2*N_{12})xloc + (\lambda_1*b_1 + \lambda_2*b_2); \quad \% \text{Gradient of the function}
\quad \text{evaluated at } x
\quad \text{if nargout} > 2
\quad \text{hessloc} = 2*(\lambda_1*N_{11} + \lambda_2*N_{12}); \quad \% \text{Hessian evaluated at } x
\text{end}
\text{end}
mycon.m
% non-linear inequalities for runnbi
% DT Fullwood 21st June 2005
% this file imposes the nonlinear constraints for the anbi method
% the equality constraint is that the resultant values must lie on a
% bisecting line between the current two points

function [c,ceq,GC,GCeq] = mycon(x,p1x,p1y,p2x,p2y)

global N11 N12 b1 b2 lambda1 lambda2 p1x p2x p1y p2y

% Nonlinear inequalities at x
% Nonlinear equalities at x
if nargout > 2  % mycon called with 4 outputs
    GC = [];  % Gradients of the inequalities
    GCeq = (p2y-p1y)*(2*N12*x+b2)+(p2x-p1x)*(2*N11*x+b1);  % Gradients of the equalities
end

lin.m
% optimization using only linear portion
% using Max's matrices
% DT Fullwood 1st July 2005
% note that this file MAXIMISES the property; it would be simple to
% minimize - just set b1=-b1, b2=-b2.

load property_closure_matrices.mat

N11=Ahr1;  % put max's matrices in standard form
N12=Ahr2;
b1=Bh1;
b2=Bh2;

n=length(b1);

numlam=1000;  % set up the values for lambda - choose numlam vectors in unit circle
vec1=cos([1:numlam]*2*pi/numlam);
vec2=sin([1:numlam]*2*pi/numlam);  % minus would make it clockwise

for i=1:length(vec1)
    lambda1=vec1(i);
    lambda2=vec2(i);
    b=lambda1*b1+lambda2*b2;
    for j=1:n
        if b(j)>0
            xx(j,i)=2;  % maximize property by choosing maximum value for positive b coeff.
        else
            xx(j,i)=0;
        end
    end
    pp(i)=b1'*xx(:,i);  % 1st order term only
    qp(i)=b2'*xx(:,i);
    pp2(i)=xx(:,i)'*N11*xx(:,i)+b1'*xx(:,i);  % full quadratic function
    qp2(i)=xx(:,i)'*N12*xx(:,i)+b2'*xx(:,i);
end
figure
hold on
P=pp2+205.9291;  % translation for actual P - see email from max (Massimiliano Binci) 27 June 05
Q=qp2+53.9587;
plot(P,Q,'ok')  % these give the correct closure

gendclose.m

% genetic algorithm to find property closure
% DT Fullwood 28 June 2005
% using max(min) method
% loads Massimiliano Bianci's matrices (Max from Drexel)

rand('state',sum(100*clock)); % reset rand generator to random position
load property_closure_matrices.mat

sample=100;         % number in each generation - make this an even number!!!
numgen=50;          % number of generations
tour=2;             % tournament size for parents
cross=0.9;          % probability for crossover (not much sense in not crossing due to elitism
mutate=0.1;         % probability for mutation
mval=2;             % maximum value for each x - problem specific (i.e. this is the constraint)

N11=Ahr1;           % put Max's matrices in my standard matrices
N12=Ahr2;
b1=Bh1;
b2=Bh2;
n=length(b1);

% find max and min points of P,Q for periodic insert into search
C=[eye(n,n) -eye(n,n)]; % constraints are simply that all variable greater than zero
a=[2*ones(n,1); -zeros(n,1)]; % and that the sum of the variables is less than n
VLB=zeros(n,1); % set upper and lower bounds to the QP problem (not absolutely necessary
VUB=ones(n,1)*2; % since the constraints already set bounds
options=optimset('display','off');
x0=ones(length(b1),1);  % starting point - chosen in feasible region
Pmin=quadprog(N11,b1,C,a,[],[],VLB,VUB,x0,options); % microstructure that minimises P
Qmin=quadprog(N12,-b1,C,a,[],[],VLB,VUB,x0,options);
Pmax=quadprog(-N11,-b1,C,a,[],[],VLB,VUB,x0,options);
Qmax=quadprog(-N12,-b2,C,a,[],[],VLB,VUB,x0,options);

lambdag1=[1 -1];    % for all 4 sides of a closure you need ++ +--+ --
lambdag2=[-1 1];    % for max's matrices there are only 2 sides

for outer=1:length(lambdag1)    % outer loop to step through each lambda
    for i=1:sample
        for j=1:n
            xx(j,i)=rand*mval;  % set up initial population of designs
        end
        p(i)=xx(:,i)*N11*xx(:,i)+b1'*xx(:,i); % property values for each design
        q(i)=xx(:,i)*N12*xx(:,i)+b2'*xx(:,i);
    end
xx(:,1) = Pmin;  % insert extreme designs
xx(:, sample) = Pmax;

lambda1 = lambdag1(outer);
lambda2 = lambdag2(outer);

% find fitness of this generation using max(min)
for j = 1:sample
    for k = 1:sample
        mink(k) = min([lambda1 * (p(j) - p(k)) lambda2 * (q(j) - q(k))]);
    end
    mink(j) = -1e20;  % make sure this value doesn't win
    [rank(j) posi(j)] = max(mink);
end

for i = 1:numgen
    for j = 1:sample/2  % create children
        for k = 1:tour    % set up tournament pools for parents
            poola(k) = ceil(rand * sample);
            poolb(k) = ceil(rand * sample);
        end
        [dummy1 dummy2] = min(rank(poola));  % find fittest parent
        parent1 = xx(:, poola(dummy2));
        [dummy1 dummy2] = min(rank(poolb));
        parent2 = xx(:, poolb(dummy2));
        % decide on crossover for creating children:
        if rand < cross  % crossover
            cpoint = ceil(rand * (n-2)) + 1;  % choose crossover point between 2 and n-1
            xx(:, sample+j*2-1) = [parent1(1:cpoint); parent2(cpoint+1:n)];
            xx(:, sample+j*2) = [parent2(1:cpoint); parent1(cpoint+1:n)];
        else               % don't crossover
            xx(:, sample+j*2-1) = parent1;
            xx(:, sample+j*2) = parent2;
        end
        if rand < mutate  % decide whether to mutate a child
            for k = 1:n  % mutate
                if rand < mutate
                    xx(k, sample+j*2-1) = rand * mval;
                end
                if rand < mutate
                    xx(k, sample+j*2) = rand * mval;
                end
            end
        end
    end % end of mutation
end % end of mutation

if rand < mutate  % keep adding max / min points periodically to spread out points
    xx(:, sample*2-2) = Pmin;  % note in this case Pmin = Qmin, etc so only need two
    xx(:, sample*2-1) = Pmax;
end

p(sample+j*2-1) = xx(:, j*2-1 + sample) * N11 * xx(:, j*2-1 + sample) + b1 * xx(:, j*2-1 + sample);
q(sample+j*2-1) = xx(:, j*2-1 + sample) * N12 * xx(:, j*2-1 + sample) + b2 * xx(:, j*2-1 + sample);
p(sample+j*2) = xx(:, j*2 + sample) * N11 * xx(:, j*2 + sample) + b1 * xx(:, j*2 + sample);
q(sample+j*2) = xx(:, j*2 + sample) * N12 * xx(:, j*2 + sample) + b2 * xx(:, j*2 + sample);
% elitism to choose best for next generation
for j=1:2*sample
    for k=1:2*sample
        mink(k)=min([lambda1*(p(j)-p(k)) lambda2*(q(j)-q(k))]);
    end
    mink(j)=-1e20;  % make sure this value doesn't win
    [rank(j) posi(j)]=max(mink);
end
[ordrank posj]=sort(rank);
for j=1:sample
    newxx(:,j)=xx(:,posj(j));
    newp(j)=p(:,posj(j));
    newq(j)=q(:,posj(j));
end
xx(:,1:sample)=newxx;
p(1:sample)=newp;
q(1:sample)=newq;
rank=rank(posj);
end % end loop for this generation

hold on
P=newp+205.9291;  % translation for actual P - see email from max (Massimiliano Binci) 27 June 05
Q=newq+53.9587;
xlabel('Property P = C_1_1 (MPa)')
ylabel('Property Q = C_1_2 (MPa)')
title('Property Closure for C_1_1 and C_1_2')
plot(P,Q,'*k')

end % end outer loop

steepd.m

% steepest descent method
% d fullwood, 8th April 05
% based on course notes by A Parkinson, ME EN 575, Winter 2005
% Also see...
% An Introduction to the Conjugate Gradient Method Without the Agonizing Pain Edition 1 1/4
% Jonathan Richard Shewchuk August 4, 1994
%
% solves Ax=b

clear
loada=0;    % set = 1 to load previously saved matrix
if loada==0
    s=0.2;  %percentage of matrix positions with non-zero values
    el=0.1;  %fraction of nonzero values with large values
    z=100;  %order of magnitude of large vs small values
    n=50;  % size of A
    A=zeros(n);
    b=zeros(n,1);
    for i=1:n
        for j=1:n
            if rand(1)<s
                if rand(1)<el
                    A(i,j)=rand(1)*z;
                end
            end
        end
    end
end
else
    A(i,j) = rand(1);
end
end
end
b(i) = rand(1)*z;
end
A = A + A'; % fill in other side of A
A = A * A; % make positive definite
b = b .* 2; % make b the same order of magnitude
else
    load sparsea;
n = length(b);
end

% now start actual optimisation
x0 = inv(A) * b; % actual solution for plotting error
eps = 0.00001; % tolerance on solution
x = zeros(n,1); % starting point
x = [0;0];

r = b - A * x;
i = 0;
imax = 150;
err = sqrt((x0 - x) .* (x0 - x));
xerror(1) = err;
delnew = r' * r;
delold = delnew;
while err > eps & i < imax
    i = i + 1;
    q = A * r;
    a = delnew / (r' * q);
    x = x + a * r;
    r = r - a * q;
    delold = delnew;
    delnew = r' * r;
    residual(i) = sqrt(r' * r); % normal definition of error
    err = sqrt((x0 - x) .* (x0 - x)); % this 'exact' error used for comparing methods
    xerror(i + 1) = err;
x1(i) = x(1); % array used to plot the first two variables
x2(i) = x(2);
end

figure
plot(xerror) % plot the error vs iterations
save sparsea A b
figure
plot(x1,x2)
% quasi newton on f=xAx-bx
% dtf 12 April 2005
% based on course notes by A Parkinson, ME EN 575, Winter 2005
% solves Ax=b

clear
loada=0; % set = 1 to load previously saved matrix
if loada==0
s=0.2; %percentage of matrix positions with non-zero values
el=0.1; %fraction of nonzero values with large values
z=100; %order of magnitude of large vs small values
n=50; % size of A
A=zeros(n);
b=zeros(n,1);
for i=1:n
    for j=i:n
        if rand(1)<s
            if rand(1)<el
                A(i,j)=rand(1)*z;
            else
                A(i,j)=rand(1);
            end
        end
    end
    b(i)=rand(1)*z;
end
A=A+A'; % fill in other side of A
A=A*A; % make positive definite
b=b.^2; % make b the same order of magnitude
else
    load sparsea;
n=length(b);
end

% start of optimisation
x0=inv(A)*b; % actual solution
eps=0.01; % tolerance on solution
n=length(A(:,1));
N=eye(n,n); % first value for inverse Hessian (identity)
s=b-A*x; % first search direction
err=sqrt((x0-x)'*(x0-x));
error(1)=err;
i=0;
while err>eps & i<10
    i=i+1;
    grado=A*x-b;
a=-grado'*s/((A*s)'*s);
delx=a*s;
x=x+delx;
grado=A*x-b;
gam=gradn-grado;
N=N+(1+gam'*N*gam/(delx'*gam))*delx*delx'/(delx'*gam)-(delx*gam'*N+gam*N*gam*delx')/(delx'*gam); s=S.N/gradn
    grado=gradn;

% end of optimisation
err = sqrt((x0-x)'*(x0-x));
exerror(i+1) = err;
x1(i) = x(1)  % arrays to store first two variables history
x2(i) = x(2)
    pause
end
plot(xerror)
figure
plot(x1,x2)
xp1 = [-100:100];
xp2 = xp1;
xp = [xp1;xp2];
k1 = length(xp1);
k2 = length(xp2);
for i = 1:k1  % calculate contour data
    for j = 1:k2
        xtemp = [xp1(i);xp2(j)];
        f(i,j) = xtemp'*A*xtemp - b'*xtemp;
    end
end
contour(xp1,xp2,f')

cgs.m

(note that a second version based on Reid is available, but not attached)

% conjugate gradient method
% d fullwood, 8th April 05
% An Introduction to the Conjugate Gradient Method Without the Agonizing Pain Edition 1 1/4
%Jonathan Richard Shewchuk August 4, 1994
% solves Ax=b

clear
loada = 0;  % loada=1 to load previously saved matrix
if loada == 0
    s = 0.2;  % percentage of matrix positions with non-zero values
    e = 0.1;  % fraction of nonzero values with large values
    z = 1;    % order of magnitude of large vs small values
    n = 50;   % size of A
    A = zeros(n);
    b = zeros(n,1);
    for i = 1:n
        for j = i:n
            if rand(1) < s
                if rand(1) < e
                    A(i,j) = rand(1)*z;
                else
                    A(i,j) = rand(1);
                end
            end
        end
    end
    b(i) = rand(1)*z;
end
A=A+A';  % fill in other side of A
A=A*A;  % make positive definite
b=b.^2; % make b the same order of magnitude
else
    load sparsea;
    n=length(b);
end

% now start actual optimisation
x0=inv(A)*b;  % actual solution - used for plotting error
eps=0.1;  % tolerance on solution
x=zeros(n,1);  % starting point
r=b-A*x;
p=r;
i=0;
imax=150;  % maximum iterations
err=1;
err=sqrt((x0-x)'*(x0-x));
xerror(1)=err;
delnew=r'*r;
delold=delnew;
while err>eps & i<imax
    i=i+1;
    q=A*p;
a=delnew/(p'*q);
x=x+a*p;
    if floor(i/20)*20==i  % restart every 20 iterations
        r=b-A*x;
        else
        r=r-a*q;
        end
    delold=delnew;
delnew=r'*r;
b=delnew/delold;
p=r+b*p;

    residual(i)=sqrt(r'*r);  % should use this error in normal use
    err=sqrt((x0-x)'*(x0-x));  % this error used to compare methods
    xerror(i+1)=err;
x1(i)=x(1);  % arrays used to plot first two variables
x2(i)=x(2);
end

figure  % plot error
plot(xerror)
save sparsea A b  % save A and b for future runs
figure  % now plot first two variable
plot(x1,x2)
hold on
plot(x1(1),x2(1),'*')  % start point
plot(x1(length(x1)),x2(length(x1)),'*r')  % end point
mrh.m

% minimum residual conjugate gradient method
% d fullwood, 11th April 05
% Martin Hanke - conjugate gradient type methods for ill posed problems

clear
loada=0; % set = 1 to load previously saved matrix
if loada==0
s=0.2; % percentage of matrix positions with non-zero values
el=0.1; % fraction of nonzero values with large values
z=100; % order of magnitude of large vs small values
n=50; % size of A
A=zeros(n);
b=zeros(n,1);
for i=1:n
    for j=i:n
        if rand(1)<s
            if rand(1)<el
                A(i,j)=rand(1)*z;
            else
                A(i,j)=rand(1);
            end
        end
    end
    b(i)=rand(1)*z;
end
A=A+A'; % fill in other side of A
A=A*A; % make positive definite
b=b.^2; % make b the same order of magnitude
else
    load sparsea;
    n=length(b);
end

% start of optimisation
x0=inv(A)*b; % actual solution
eps=0.1; % tolerance on solution
x=zeros(n,1); % starting point
r=b-A*x;
p=r;
q=A*p;
i=1;
imax=150;
err=1;
err=sqrt((x0-x)'*(x0-x));
xerror(1)=err;
delnew=r'*(A*r);
delold=delnew;
while err>eps & i<imax
    i=i+1;
a=delnew/(q'*q);
x=x+a*p;
if floor(i/20)*20==i % restart every 20 iterations
    r=b-A*x;
else
r=r-a*q;
end
delnew=r*(A*r);
b=delnew/delold;
p=r+bi*p;
q=A*r+bi*A*p;
delold=delnew;
residual(i)=sqrt(r'*r);
err=sqrt((x0-x)'*(x0-x));
xerror(i+1)=err;
x1(i)=x(1);
x2(i)=x(2);
end
figure
plot(xerror)
save sparsea A b
figure
plot(x1,x2)
hold on
plot(x1(1),x2(1),'*')
plot(x1(length(x1)),x2(length(x1)),'*r')

qp.m

% QP algorithm from
% Goldfarb
% A numerically stable dual method....
% Math. Program. V 27, 1983
% d t fullwood 12 April 05
% Minimises: f = 1/2 x' G x + b x
% subject to C' x - a >= 0
% clear
debug=0; % toggle debug helps in program

% problem setup is: is f=b'x+1/2 x'G x
% constraints C'x-a>=0 - note the transpose for C
% the following is a problem from Golfarb for validation
%G=[4 -2; -2 4];
%b=[6;0];
%C=[1 0; 0 1; 1 1];
%a=[0;0;2];

% random large problem....... 
s=1.0; %percentage of matrix positions with non-zero values
el=1.0; %fraction of nonzero values with large values
z=20; %order of magnitude of large vs small values
n=1000; % size of A
A=zeros(n);
b=zeros(n,1);
for i=1:n
for j=i:n
if rand(1)<s
if rand(1)<el
A(i,j)=rand(1)*z;
else
end
end
end
figure
plot(xerror)
A(i,j)=rand(1);
end
end
end
b(i)=rand(1)*z;
end
A=A+A'; % fill in other side of A
A=A*A; % make it positive definite
b=b.^2; % to make same order of magnitude as A
G=A;

% set up constraints for example large matrix
cv=ones(n,1);
C=[eye(n,n) cv]; % constraints are simply that all variable greater than zero
a=[zeros(n,1); n]; % and that the sum of the variables is greater than n

% start actual optimisation
% step 0
n=length(b);
m=length(C);
K=[1:m]; % inequality set
inG=inv(G);
x=-inG*b; % choose unconstrained optimum for first point
xp=x; % for plotting route
f=b'*x/2;
H=inG;
A=[];
k=[];
N=[];
q=0;
u=[];
r=0;
ii=1;
iimax=1000;
first=1; % for debugging

% step 1 - steps follow Goldfarb's outline
while ii<iimax % loop with maximum
s=C'*x-a; % s is the value of each constraint for this x (s>=0 means feasible)
V=[]; % V is the vector of non-active violated constraints
for i=1:m
if s(i)<0
notA=1;
for j=1:length(A)
if i==A(j)
notA=0;
end
end
if notA==1;
V=[V i];
end
end
VL=length(V);
if VL>0
p=V(ceil(rand(1)*VL));  % choose a non-active violated constraint
else
    return  %  no violated constraints so end
end
if first==1 & debug==1; p=2; first=0; end % for validating against goldfarb
sp=s(p);    % amount that the chosen constraint is violated
np=C(:,p);  % normal vector to constraint p
if q==0; u=[]; end
up=[u;0];   % lagrange multipliers for active constraint (including new chosen one)

Ap=[A p];   % add constraint to Ap
Np=[N np];  % add constraint normal to Np

% step 2
step1=0;
while step1==0
    z=H*np;    % step direction in primal space
    if q>0
        r=Ns*np;    % negative of step direction in dual space
    else
        r=[];   %not in alg, but required
    end
    if q==0 | r<=0 % beware! you must have the q==0 first or matlab misses this for r=[]
        t1=1e20;    % the ti are step lengths
    else
        t1=1e20;
        for i=1:q
            if r(i)>0 & up(i)/r(i)<t1
                t1=up(i)/r(i);  
                el=i;
            end
        end
        k=A(el);
    end
    if abs(z)==0
        t2=1e20;
    else
        t2=-sp/(z'*np);
    end
    t=min(t1,t2);
    if t>1e10
        disp('no solution')
        return
    end
    if t2>1e10
        up=up+t*[-r;1];
        flag=0;
        i=0;
        while flag==0;  % remove constraint k from list of active
            i=i+1;
            if A(i)==k
                flag=1;
                if i==1
                    A=A(2:length(A));
                end
            end
    end
end
N=N(:,2:length(A)+1);
up=up(2:length(up));
else if i==length(A)
   A=A(1:length(A)-1);
   N=N(:,1:length(A));
   up=up(1:length(up)-1);
else
   A=[A(1:i-1) A(i+1:length(A))];
   N=[N(:,1:i-1) N(:,i+1:length(A)+1)];
   up=[up(1:i-1); up(i+1:length(up))];
end
end
end
q=q-1;
if q==0;
   Ns=[];
   H=inG;
else
   Ns=inv(N'*inG*N)*N'*inG;    % update matrices
   H=inG*(eye(n,n)-N*Ns);
end
else
   x=x+t*z;    % new x position
   s=C'*x-a;
   sp=s(p);
   xp=[xp x];    % append x to plotting history
   f=f+t*z*np*(t/2+up(q+1));
   up=up+t*[-r;1];
   if t==t2
      u=up;
      A=[A p];
      N=[N np];
      q=q+1;
      Ns=inv(N'*inG*N)*N'*inG;
      H=inG*(eye(n,n)-N*Ns);
      g=G*x+b;
      step1=1;    % go back to step 1
   elseif t==t1
      flag=0;
      i=0;
      while flag==0    % remove constraint k from list of active
         i=i+1;
         if A(i)==k
            flag=1;
            if i==1
               A=A(2:length(A));
               N=N(:,2:length(A)+1);
               up=up(2:length(up));
            elseif i==length(A)
               A=A(1:length(A)-1);
               N=N(:,1:length(A));
               up=up(1:length(up)-1);
            else
               A=[A(1:i-1) A(i+1:length(A))];
               N=[N(:,1:i-1) N(:,i+1:length(A)+1)];
            end
         end
      end
   end
end
up=[up(1:i-1); up(i+1:length(up))];
end
end
end
q=q-1;
if q==0;
    Ns=[];
    H=inG;
else
    Ns=inv(N'*inG*N)*N'*inG;
    H=inG*(eye(n,n)-N*Ns);
end
end
if debug==1
    % put values you want to see here
    A
    u
    pause
end
end % end of step 2 loop
end % end of step 1 loop

% extra code to plot out contour graphs, etc for higher dimension runs
% copy and paste into matlab after running qp

x1=[-2*n:.1:2*n];
x2=[x1];
for i=1:length(x1)
    for j=1:length(x2)
        xtemp=[x1(i); x(2); x2(j); x(4:n)]; % x is the constrained optimum
        f(i,j)=0.5*xtemp'*G*xtemp+b'*xtemp;
    end
end
contour(x1,x2,f',20)
hold on
c1=zeros(length(x1));
c2=c1;
c3=n-x1-x(2)-sum(x(4:n));
plot(c1,x2,'k')
plot(x1,c2,'k')
plot(x1,c3,'k')
plot(xp(1,:),xp(3,:))
plot(xp(1,length(xp(1,:))),xp(3,length(xp(1,:))),'*')
xlabel('x1')
ylabel('x2')
title('QP algorithm using dual method')
1 Stauffer D, Aharony A, Introduction to Percolation Theory. 2nd ed., Bristol, PA: Taylor and Francis; 1992
8 Cottrell, Materials Science and Technology, Vol 6, 1989 pp 121
10 Wolf D, Merkle KL, Correlation between the structure and energy of grain boundaries in metals, in Materials Interfaces: Atomic-Level Structure & Properties, editors Wolf & Yip, Chapman & Hall, 1992
12 Raisanen VI, Quasistatic cracks and minimal energy surfaces, Physical review letters, vol 80 #2, 12 Jan 1998
20 Frary M, Schuh CA, Grain Boundary Networks: Scaling Laws, Preferred Cluster Structure, and Their Implications for Grain Boundary Engineering, Submitted to Acta Mat., January 2005
34 MATLAB Version 7, The Mathworks, Inc., 2005
36 Das, I.; Dennis, J.E. 1997: A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. Struct Optim 14, 63–69
41 Parkinson A, Course Notes to ME EN 575, Winter 2005, BYU Academic Publications