Prediction of Mechanical Properties of Aerogels using a Multifractal Multidimensional Multiscaling Approach

Fritz Andres Campo Schickler
West Virginia University

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
https://researchrepository.wvu.edu/etd/3069

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact research.repository@mail.wvu.edu.
Prediction of Mechanical Properties of Aerogels using a Multifractal Multidimensional Multiscaling Approach

Fritz Andres Campo Schickler

Dissertation submitted to the College of Engineering and Mineral Resources at West Virginia University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

Committee Members
Ever J. Barbero, Ph.D., (Chairperson and Advisor)
Cesar Barbero, Ph.D.,
Darran Cairns, Ph.D.,
Edward Sabolsky, Ph.D.,
Nick Wu, Ph.D.

Department of Mechanical and Aerospace Engineering Morgantown, WV, USA

2011

Keywords: Sol-gel; aerogel; hierarchical structures; fractals; scattering; homogenization techniques; computational materials
Abstract

Prediction of Mechanical Properties of Aerogels using a Multifractal Multidimensional Multiscaling Approach

Fritz Andres Campo Schickler

Aerogels, produced by sol-gel technologies, have several applications in sensors, high energy particle physics, catalysis, heat insulation, supercapacitors, heat storage devices, high efficiency windows, among others. These applications take advantage of the outstanding properties these materials present as a result of their structure. However, the low mechanical properties that these materials present as result of the process, limits their commercial applications. In this dissertation, it is investigated the relationship between the processing conditions and mechanical properties of these materials computationally.

The prediction of the effective properties for these materials is a daunting task because of their complex structure. Aerogels’s structure is not homogeneous nor periodic, but rather amorphous, nanostructured, and highly porous, making the traditional techniques used to study other materials inapplicable. This dissertation presents the prediction of mechanical properties of aerogels calculated by a novel Multifractal Multidimensional Multiscaling Approach (MMMA) developed here.

MMMA consists on recursively calculating the effective properties of the material along several scales. Since aerogels and structures produced by sol-gel technologies present a multifractal character, it is shown that MMMA is applicable to predict the effective properties of these materials.

The implementation of MMMA requires a fractal characterization of the structure. For this, computational scattering experiments were performed on structures resembling aerogels. The structures resembling aerogels were produced computationally incorporating the chemistry and the physical phenomena involved in the formation process.

MMMA was used to predict the mechanical properties of silica aerogels for different processing conditions. Thus, mechanical properties, scattering experiments, and processing conditions were investigated and correlated in this work.
I want to dedicate my work to Guillermo and Anita (my parents), and to Franz Nicolás (my twin brother). Without my family I simply would not be, and as a fractal, this work would not be either.
Acknowledgment

I want to thank many people that contributed directly or indirectly with this dissertation and the completion of my PhD. First, I want to thank Dr. Ever Barbero, that more than my advisor gave me the opportunity to dig into fundamental problems as the subject of this dissertation. From the office, Sandro Rivas who suffered major long talks about a diversity of problems related to the dissertation, as well as Joaquin Gutierrez, Adi Adumitroaie, and Juan Cruz. I also want to thank Marco Maurier who helped writing Fortran codes. I also want to thank the committee members for their comments and assistance. Last but not least I want to thank Dr. Boyd Edwards who besides introducing me into Non–linear dynamics, chaos and fractals, helped me as a committee member as long as the logistics allowed.
## Contents

 Acknowledgement ........................................ iv

 List of Figures ........................................... ix

 List of Tables ............................................. xi

 List of Symbols ........................................... xii

1 Introduction ............................................. 1

 1.1 Experimental Behavior of the fractal structure ........ 2

 1.2 Implications of a Hierarchical Structure ............... 5

2 Aggregation Algorithm .................................... 7

 2.1 Introduction ........................................... 7

 2.2 Aggregation model ...................................... 9

  2.2.1 Brownian Motion ................................. 11

  2.2.2 Reactions ......................................... 12

 2.3 Generated Sol Structures .............................. 19

 2.4 Discussion ............................................ 27

 2.5 Conclusions ........................................... 30

3 Scattering Response ..................................... 31

 3.1 Introduction .......................................... 31
CONTENTS

5.4.3 Multiscaling with the Principle of Similarity ........................................ 79
5.4.4 Constant Fractal Dimension ................................................................. 80
5.4.5 Variable Fractal Dimension ................................................................. 81
5.4.6 Experimental Fractal Dimension ........................................................... 81
5.4.7 Calculation of Young Modulus and Thermal Conductivity .................... 83
5.4.8 Sintering ................................................................................................. 84
5.5 Conclusions ............................................................................................... 86

6 Code Implementation ..................................................................................... 88
   6.1 Bird’s eye view ...................................................................................... 88
   6.2 Program blocks ..................................................................................... 89
       6.2.1 Aggregation Algorithm ................................................................. 91
       6.2.2 Scattering Algorithm .................................................................... 97
       6.2.3 Effective Mechanical Properties Algorithm ................................. 98
   6.3 Auxiliary Functions .............................................................................. 100
       6.3.1 Scattering Average ...................................................................... 100
       6.3.2 Effective Stiffness Average .......................................................... 102
       6.3.3 Hierarchical Calculation of Stiffness ............................................ 103
       6.3.4 FindPolymer ............................................................................... 103
       6.3.5 ElementNodesP .......................................................................... 104
       6.3.6 ElementNodes ........................................................................... 105
   6.4 Hierarchical Structure Reconstruction .................................................. 105
       6.4.1 HierarchicalStructure ................................................................... 106

7 Conclusions ................................................................................................. 107

Bibliography .................................................................................................... 109
A  Program Codes

A.1  Control Panel Code in Matlab ................................................. 120
A.2  Aggregation Algorithm Matlab Code ............................................ 122
A.3  Aggregation Algorithm Program in Fortran ............................. 126
A.4  Scattering Algorithm Matlab Code ............................................. 141
A.5  Scattering Code in Fortran .......................................................... 143
A.6  Effective Stiffness Calculation Algorithm Code in Matlab .......... 150
A.7  AveIq in Matlab ................................................................. 171
A.8  AveStiffness in Matlab ........................................................... 172
A.9  EffectStiffGenAlgorithm in Matlab ............................................. 173
A.10 FindPolymer in Matlab ............................................................ 174
A.11 ElementNodesP in Matlab ....................................................... 175
A.12 ElementNodes in Matlab .......................................................... 180
A.13 Generation of Hierarchical structure ........................................ 182

B  Homogenization: Stresses Point of View .................................. 184
# List of Figures

1.1 TEM image of silica aggregates [1] ................................................. 2
1.2 AFM of silica aerogel [2] ................................................................. 3
1.3 SEM of silica aerogel [2] ................................................................. 3
1.4 Experimental Young modulus vs. density ........................................... 4
1.5 Effect of primary particle (families 1-2) and structure (family 3) ............. 4
2.1 Flow chart of sol-gel algorithm .......................................................... 10
2.2 Geometry of the clusters modeled by [3] ............................................ 15
2.3 Energy Models .................................................................................. 17
2.4 Error of Models compared to the values from [3] .................................. 17
2.5 Network for $w = 1$, and two different correlation lengths, $L$ ............... 20
2.6 Coordination number distribution varying simulation size ....................... 21
2.7 Functionality distribution varying simulation size .................................. 21
2.8 Coordination number distribution, same density, varying particles ........... 22
2.9 Functionality distribution, same density, varying particles .................... 23
2.10 Coordination number distribution for several densities with $L = 10$ ........... 24
2.11 Functionality distribution for several densities with $L = 10$ .................... 24
2.12 Coordination number distribution for several reactivities ....................... 25
2.13 Functionality distribution for several reactivities ................................ 26
3.1 Cross-section of an aggregated structure inside the simulation box .......... 33
LIST OF FIGURES

3.2 Scattering Intensities for the samples in Table 3.1 ........................................... 41
3.3 Fractal dimension and critical percolation density vs. reactivity: .................. 46

4.1 Structure generated with $N = 400$ particles and reactivity $w = 1.0$. ............ 51
4.2 Von Misses plot for structure as gelled and fully connected structure ............. 55
4.3 Effective Young modulus vs. number of particles ............................................. 56
4.4 Effective Poisson ratio vs. number of particles ............................................... 56
4.5 Density $\rho$ vs. number of generations $n$ for different reactivities .................. 57
4.6 Effective Young modulus vs. reactivity .......................................................... 59
4.7 Effective Poisson ratio vs. reactivity ............................................................. 59
4.8 Effective Young modulus as a function of the effective density ...................... 60
4.9 Effective Poisson ratio as a function of the effective density ......................... 61
4.10 Power law exponent as a function of reactivity .............................................. 62

5.1 Intensity plot of reconstructed structure ......................................................... 70
5.2 Recursive procedure in order to calculate the stiffness .................................. 74
5.3 Density vs. Scale for different fractal dimensions .......................................... 80
5.4 Effective density as a function of scale .......................................................... 82
5.5 $q(r)$ and fractal dimension of an aerogel ....................................................... 82
5.6 Effective Young modulus and thermal conductivity ........................................ 84
5.7 Scaling exponent of thermal conductivity and Young modulus ..................... 85

6.1 Bird’s eye view of the effective properties calculation flow diagram ............. 90
6.2 Effective properties calculation flow diagram ................................................. 91
6.3 Effective properties calculation algorithm ....................................................... 92
6.4 Face numbers .................................................................................................. 100
6.5 Loading cases ............................................................................................... 101
List of Tables

2.1 Processing conditions from [3] .............................................. 13
2.2 Cluster formation energies from [3] and models ...................... 16
3.1 Percolation results and test conditions ................................. 40
4.1 Parametric test conditions .................................................. 55
B.1 Tests in order to calculate $C_{\alpha\beta}$ ................................ 187
List of Symbols

\( \phi \)  
\begin{align*}
diameter
\end{align*}

\( \rho \)  
\begin{align*}
density \text{ in mass per volume}
\end{align*}

\( E \)  
\begin{align*}
Young \text{ modulus}
\end{align*}

\( \Delta H \)  
\begin{align*}
enthalpy \text{ of formation}
\end{align*}

\( k_b \)  
\begin{align*}
Boltzmann \text{ constant}
\end{align*}

\( T \)  
\begin{align*}
absolute \text{ temperature}
\end{align*}

\( Z \)  
\begin{align*}
partition \text{ function}
\end{align*}

\( P_{Metro} \)  
\begin{align*}
occurrence \text{ probability of an event in Monte Carlo simulation}
\end{align*}

\( P_{BM} \)  
\begin{align*}
probability \text{ of choosing a particle to resemble Brownian motion}
\end{align*}

\( m_i \)  
\begin{align*}
mass \text{ of particle } i
\end{align*}

\( a_i \)  
\begin{align*}
cluster \text{ diameter}
\end{align*}

\( Z_{BM} \)  
\begin{align*}
Brownian \text{ motion partition function}
\end{align*}

\( q_{n_i}^{m} \)  
\begin{align*}
cluster \text{ with } n \text{ silicon atoms and } m \text{ oxygen bridges}
\end{align*}

\( E_i \)  
\begin{align*}
energy \text{ of particle } i
\end{align*}

\( n_c \)  
\begin{align*}
coordination \text{ number}
\end{align*}

\( E_{n_c} \)  
\begin{align*}
energy \text{ of particle as a function of number of coordination } n_c
\end{align*}

\( a_1 \)  
\begin{align*}
chemical \text{ potential of a particle}
\end{align*}

\( a_2 \)  
\begin{align*}
energy \text{ associated to } n_c, \text{ related to the surface energy}
\end{align*}

\( E_{A_f} \)  
\begin{align*}
energy \text{ of a particle after reaction}
\end{align*}
\[ E_{A_0} \] energy of a particle before reaction
\[ P_{\text{React}} \] probability of reaction if there is a collision between two particles
\[ \dot{w} \] extra energy required to form a bond when the \( n_c \) is increase by one
\[ \hat{Z} \] reaction partition function
\[ w \] reactivity
\[ N \] number of particles in a simulation box
\[ L \] size of simulation box
\[ D \] fractal dimension
\[ f \] functionality
\[ \lambda \] scale
\[ \lambda_{\text{max}} \] max scale associated to the simulation box \( L \)
\[ 2a_0 \] size of primary particle
\[ \xi \] correlation length
\[ \rho_{\text{ef}}^0 \] primary particle effective density
\[ N_0 \] number of particles to aggregate
\[ N_C \] number of clusters in simulation box
\[ d \] mean free path between clusters, or distance between clusters
\[ \rho_C \] number of cluster per unit volume
\[ N_{k/C} \] number of particles per cluster
\[ \rho_{\text{crit}} \] critical density for percolation
\[ t \] time
\[ \Delta t \] average time interval between collisions of two clusters
\[ (\cdot)^t \] quantity evaluated at time \( t \)
\[ t_{\text{per}} \] percolation time
\[ q \] wave number
\[ I(q) \] scattering intensity evaluated at \( q \)
$P(q)$  form factor

$S(q)$  scattering function

$r_{ij}$  distance between particle $i$ and $j$

$L_{\text{min}}$  primary particle size at a certain scale range

$R$  Calculated cluster size

$q_{\text{crit}}$  critical percolation wave number

$D_q$  fractal dimension as calculated from scattering experiments

$q_{\text{min}}$  minimum wave number

$I_0$  scattering intensity of at $q = q_{\text{min}}$

$S_{\text{cube}}$  scattering function calculated for a cubic lattice

$u_i$  uniform displacements

$\varepsilon_{ij}^0$  uniform strain

$x_i$  position of a particle

$\sigma_\alpha$  stress in contracted notation

$\sigma_{ij}$  stress

$C_{\alpha\beta}$  stiffness tensor in contracted notation

$C_{ijkl}$  stiffness

$U$  stored energy in volume $V$

$V$  volume

$U_{\text{tot}}$  total stored energy

$V_{\text{tot}}$  total volume

$\mathcal{H}$  homogenization operator

$N_s$  number of samples

$\mathcal{D}$  fractal dimension intrinsic of the aggregation process

$n$  number of generations

$n_0$  density as number of particles per volume
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_E$</td>
<td>Young modulus vs. density scaling exponent</td>
</tr>
<tr>
<td>$\nu$</td>
<td>Poisson ratio</td>
</tr>
<tr>
<td>$\Box$</td>
<td>simulation box</td>
</tr>
<tr>
<td>$\Box$</td>
<td>smallest discernible particle box</td>
</tr>
<tr>
<td>$T$</td>
<td>(fractal) transformation mapping</td>
</tr>
<tr>
<td>$\circ$</td>
<td>composed of</td>
</tr>
<tr>
<td>$L_{ij}$</td>
<td>vectors that define $\Box$</td>
</tr>
<tr>
<td>$\Delta L_{kl}$</td>
<td>vector that define $\Box$</td>
</tr>
<tr>
<td>$S$</td>
<td>structure</td>
</tr>
<tr>
<td>$\lambda_{ijkl}$</td>
<td>dimensionless scaling factor</td>
</tr>
<tr>
<td>$\mathcal{P}$</td>
<td>property</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>is map to</td>
</tr>
<tr>
<td>$\mathcal{P}_{Hl}$</td>
<td>eigenvectors of $\mathcal{H}$</td>
</tr>
<tr>
<td>$\tilde{h}_l$</td>
<td>eigenvalues of $\mathcal{H}$</td>
</tr>
<tr>
<td>$\alpha_{\Box l}$</td>
<td>projection of $\mathcal{P}<em>{\Box}$ on $\mathcal{P}</em>{Hl}$</td>
</tr>
<tr>
<td>$\alpha_{\Box l}$</td>
<td>projection of $\mathcal{P}<em>{\Box}$ on $\mathcal{P}</em>{Hl}$</td>
</tr>
<tr>
<td>$\mathcal{D}_{kl}^{m}$</td>
<td>property scaling exponent</td>
</tr>
<tr>
<td>$g(\lambda)$</td>
<td>multiscaling function</td>
</tr>
<tr>
<td>$\mathcal{D}_{\mathcal{P}}$</td>
<td>property scaling exponent for decoupled case</td>
</tr>
<tr>
<td>$\kappa_l$</td>
<td>thermal conductivity</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Silica aerogels are the result of the aggregation of silica forming a colloid that has a hierarchical structure. For this, a solution is prepared consisting on a precursor such as tetramethyorthosilicate (TMOS) or tetraethylorthosilicate (TEOS) in excess of water and some alcohol. Hydrolysis\(^1\) and condensation\(^2\) reactions are activated by the pH of the solution. Hydrolysis reactions create hydroxils (M-OH) dispersed in the solution which start moving with a brownian motion. The motion comes from the interaction with molecules moving due to their thermal energy which randomly transfer momentum to the particles. In fact, the velocity distribution of the particles follow an equilibrium distribution (Maxwell-Boltzmann distribution \[^{[4]}\]). These hydroxils collide and stick together due to condensation reactions releasing water. The aggregation of the particles forms clusters, also moving with brownian motion. Furthermore, these clusters can collide between them, forming bigger clusters, which again collide forming even bigger clusters, each time at a larger scale. If the amount of precursor is enough, the final clusters are large enough to form a network in the container taking the shape of the container. At this moment, a gel is formed. Extra time allows more particles to reinforce the structure, a process known as aging. Additionally, the liquid phase can be removed. To avoid breaking the very fine solid structure while evaporating the liquid phase,

\(^1\)Hydrolysis are reactions where the TEOS or TMOS breaks down into silanols also called silicon hydroxils.
\(^2\)The highly reactive hydroxils combine together aggreagating the silicon atoms by oxygen bonds.
a gentle removal can be achieved taking the liquid phase to its critical point. For this, first the liquid is replaced by alcohol, which is then taken to its critical point at which the surface tension is zero (implying that the removal of an atom from the liquid phase to the gas phase requires zero energy, therefore no capillary forces would appear) and vented out with carbon dioxide.

### 1.1 Experimental Behavior of the fractal structure

The hierarchical structure of aerogels has been seen using Transmission Electron Microscopy (TEM), Atomic Force Microscopy (AFM), and Scanning Electron Microscopy (SEM). The aggregation of silica is observable with the aid of image-processing, reconstructing the primary particle \[1\] as presented in Figure 1.1. Three sizes of aggregates are identifiable as encircled in Figure 1.1: 1) \(\phi 10–15nm\), 2) \(\phi 40–60nm\), and 3) \(\phi 120–150nm\). These are three hierarchical sizes at which the aerogel aggregated. The primary particle identification is better observed in
AFM as presented in Figure 1.2. Aggregations with sizes around $\phi 100-150nm$, $\phi 0.5-0.6\mu m$, and $\phi 2.0-2.1\mu m$ are identified. In Figure 1.3 using SEM, the primary particles are not easily identifiable. However, aggregations around $\phi 0.20-0.25\mu m$, and $\phi 1.5-2.0\mu m$ can be observed. Note that the small blobs are arranged into bigger blobs, which arrange into bigger blobs as well, characteristic of a hierarchical (fractal) structure, and a hierarchical aggregation.

Moreover, in Figure 1.4, the experimental values found by T. Woignier et. al. are plotted on the Ashby chart positioning aerogels into the materials realm. The family of aerogels display a power law behavior as expected from a fractal material with a slope of $3.7 \pm 0.2$. The density domain reported by Woignier goes from $\rho = 0.08g/cm^3$ to $0.5g/cm^3$. Note that the trend line (which is a straight line in a log-log plot), overlaps the known properties of amorphous silica. A similar behavior is observed in the strength-density and toughness-Young modulus plots (not presented here).
Figure 1.4: Experimental Young modulus vs. density from [5] of aerogels on Ashby charts [6].

Figure 1.5: Effect of primary particle (families 1-2) and structure (family 3).
1.2 Implications of a Hierarchical Structure

From the Ashby charts, varying the constitutive material, but keeping the same structure, translates the aerogel family depicted in Figure 1.3 from family 1 to family 2. No change between elements of the family occurs, but the overall effective property is affected similarly for all members of the family. If only the structure varies, family 2 could be modified to become family 3. In this case, elements of the family see their properties change relative to each other. This observation is summarized as the Principle of Similarity (PS), discussed in Chapter 5, being the foundation of the Multifractal Multidimensional Multiscaling Approach.

This dissertation is organized in the following way. Chapter 2 describes a novel algorithm to recreate sol-gel structures incorporating Brownian motion and chemical reactions. Chapter 3 describes the scattering response of the structures produced as in chapter 2. Scattering is a response dependent on the mass distribution of the material structure. Chapter 4 describes the mechanical response, in particular the elastic behavior from the structure as the ones reproduced as described in chapter 2. Here, the connectivity in the structure plays an important role. For both, chapter 3 and 4, the procedure is implemented for hierarchical structures. For these structures, an iterative procedure can be implemented predicting the material properties at a large scale starting with the known material properties at molecular scale. The formalism is described in detail in Chapter 5 where reconstructed material structures are reproduced. These formalism is the Multidimensional Multifractal Multiscaling Approach (MMMA) proposed here. Multidimensional stands for the dimensionality of the properties, as stiffness, that are tensors in a 3D, as well as the scaling that can be characterized by a tensor as well. Multifractal stands for the iterative procedure that may allow varying the the way how the property scales at each iteration. Multiscaling stands for the recursive procedure, that for a single material structure, yields to a variety of responses, such as conductivity, stiffness, density, etc. Finally in Chapter 6, the description of the computer

---

3 A family is defined here as the set of aerogels produced under similar processing conditions varying a single parameter, in this case, amount of precursor.

4 Amorphous silica has a density of \( \rho = 2.33 g/cm^3 \) and Young modulus of \( E = 80 \pm 20 GPa \) \cite{7}. 
codes developed for this dissertation are presented. The use of High Performance Parallel computing, interacting with engineering programs as Matlab\textsuperscript{8} and Ansys\textsuperscript{9} is described.

Throughout this dissertation the reader will see how and why MMMA proves to be an efficient technique to calculate the effective material properties in particular for hierarchical materials such as aerogels, as well as he will understand how different properties of a material can be correlated as the result of sharing the material’s structure. Even considering that the accuracy in the calculation depends on the accuracy in the reproduction of the material’s structure and the physics used to calculate the properties of the material which can vary with the scale, it is believed that MMMA is general enough to be a valuable tool in the calculation of material properties in materials science.
Chapter 2
Aggregation Algorithm

Abstract

A stochastic computational model for the gelation of a sol is explained and tested for the case of neutral silica aerogels. The computational model produces the final structure of the sol after gelation, using two of the several physical phenomena occurring during gelation of sols. Diffusion, represented by Brownian motion, is modeled by a random walk, and chemical reactions are incorporated through a stochastic aggregation model using a probability function; the later determined in terms of the processing conditions based on the knowledge of the cluster formation energies. The two phenomena are coupled by a Monte Carlo simulation. The analysis of the connected structure and its functionality is demonstrated for neutral silica aerogels. It is shown how the gelation process can be controlled to obtain different structures for different application requirements. The only parameters required by the model are the density and the processing conditions. The results of the model show that those parameters strongly affect the structure of the generated samples. Therefore, processing conditions could be selected to produce aerogels with structures tailored to specific applications, which would constitute a major achievement in aerogel fabrication.

2.1 Introduction

Sol-Gel technologies include a range of processes which start with a metal alkoxide solution, and lead to products such as dense films, aerogels, dense ceramics, uniform particles, and ceramic fibers[10]. Many of these products are made by producing first a sol, which is the metal
alkoxide solution with activated reactions of hydrolysis and condensation. The sol gelates into a wet gel, which is composed of a nanoporous solid network immersed in a liquid phase (solvent). The solid network can be separated from the solvent by evaporation of the liquid which leads to xerogels, or by supercritical drying which leads to aerogels. Supercritical drying does not break the fine solid structure present in the wet gel, thus imparting aerogels with unique physical properties. Because of their unique physical properties, aerogels have found applications in insulation, catalysts, sensors, fuel storage, Cherenkov detectors, lightweight optics, special effect optics, impedance matchers for transducers, energy absorbers, hypervelocity particle traps, ICs, and capacitors.

The structural characteristics of aerogels, therefore their properties, are mostly defined during the gelation of the sol. Processing conditions as temperature, pressure, the nature of the catalyst, and the ratio of precursor:catalyst:solvent, have a major impact on how the structure is formed. Although it is well known that the gelation process is a fundamental part of aerogel fabrication, it is not completely understood. In this paper, the process of formation of aerogel structures is investigated. A general computational model for the gelation of a sol and the formation of the aerogel structure is proposed. The results of this gelation model are verified for the case of silica aerogels. The resulting model can be used to predict the processing-property relationships for a sol gel derived material.

Different models have been proposed to describe the structure of aerogels. Some are based on experimental evidence, mostly scattering experiments, which work under the assumption that the structure is fractal. However, the fractal model is not always applicable. Other models, based on Percolation Theory (PT), allow to visualize the gelation when the network percolates. Additional tests can be done to the structures generated with PT, however, the predictions do not correlate well with experimental data for aerogels. Aggregation models, which are modified versions of PT models, can incorporate physical phenomena such as Brownian Motion (BM), reactions, or both combined. In limiting cases, aggregation models produce similar results.
CHAPTER 2. AGGREGATION ALGORITHM

to PT models [42]. Aggregation models generate structures that resemble aerogels [43, 44], however, the parameters of the models have not been directly related to the processing conditions. In this paper, the input parameters for the model are determined in terms of the processing conditions for aerogels, which allows us to use the computational model to predict the processing-property relationships for aerogels.

Other techniques such as molecular dynamics (MD) and ab-initio simulations have been used to create clusters of silica and aerogels [45, 46, 3, 47, 48, 49, 50] evaluating their response to different types of stimuli. However, these techniques do not allow the control of the features of the structure generated, and the formation process cannot be directly related to the processing conditions.

2.2 Aggregation model

The aggregation model proposed assumes that the gelation process consists of two physical phenomena: diffusion of particles in the sol, which can be described by a Brownian motion (BM) and condensation reactions that occur when the clusters collide. Hydrolysis is assumed to occur fast\footnote{There is evidence, for acidic conditions that hydrolysis reactions occur at least one order of magnitude faster than condensation reactions\cite{24}.}, which guarantees the presence of reactive particles from the beginning of the simulation. The algorithm allows to include the rate of hydrolysis incorporating a source of reactive particles, but that is not considered here since only the competition between the browninian motion and the condensation of particles in the aggregation process is investigated. Sintering and aging could also be investigated but these processes additional to the aggregation process.

Through a Monte Carlo (MC) simulation [51, 24], the different phenomena are modeled and coupled together. For this, each phenomena is associated to a set of possible events. The set of events to model the BM are differentiated from the reactions by the existence or not of a collision. Only when a collision occurs, a reaction can occur, but no BM occurs.
The algorithm starts by choosing a particle and a tentative direction of movement. Next, the algorithm evaluates if the cluster containing the particle collides or not with any other particle of the system. If no collision is detected, the cluster can move in the tentative direction reproducing BM. If a collision is detected, the cluster will stay at the same position, but it opens the possibility for a reaction to occur \(^2\). Next, the algorithm tests if a reaction occurs between the test particle and the target particle \(^3\). If there is no reaction, the cycle is repeated as described in the flow chart in figure 2.1.

\(^2\) The colliding particle in the moving cluster is called test particle. The particle that is hit by the test particle is the target particle.

\(^3\) Multiple simultaneous colliding sites are possible. In this case, the reaction probability is divided by the number of simultaneous collisions to account the reduction in the intensity (pressure) of the collision.
CHAPTER 2. AGGREGATION ALGORITHM

The Metropolis algorithm [51] is used to choose random events from the set of events that characterize each phenomenon, satisfying the physical laws and statistical mechanics. This algorithm is congruent to the Boltzmann postulate [52] which states that events at the same energy level occur with the same probability. In this way, all the events of the BM are equally probable to occur because there is no net change of the total energy of the system. The problem of BM is discussed in detail in section 2.2.1.

The Metropolis algorithm sorts the events based on the difference in the total energy (from a reference) of the system $\Delta H$ as it happens for reactions. For the set of all possible reactions that can occur, the probability of the $k$-th reaction to occur is given by

$$P_{Metro}^k = \frac{1}{Z} e^{-\frac{\Delta H_k}{k_b T}}$$  \hspace{1cm} (2.1)

where $k_b$ is the Boltzmann constant, $T$ is the temperature of the system, and $Z$ is the partition function for the reactions defined as

$$Z = \sum_j e^{-\frac{\Delta H_j}{k_b T}}$$  \hspace{1cm} (2.2)

In the section 2.2.2, reactions are discussed and related to the processing conditions through the energies of formations of the clusters, which can be directly related to the reaction energies $\Delta H_k$.

2.2.1 Brownian Motion

The BM is modeled stochastically by a random walk on a simple cubic lattice. The simple cubic lattice, defined by a lattice parameter length, implies that a particle located at a site can have a maximum of six particles in its vicinity, therefore, a particle can have a maximum of six bonded particles to it. The random walk consists on randomly choosing a cluster to move to its vicinity. A particular cluster is chosen to move when a particle that makes part of the cluster is chosen using a probability function. The probability function for all the
particles in the $i$-th cluster, $P_{BMi}$, is defined using the Einstein and Smoluchowski theory \[4,53]. This theory measures the drag by the medium of the cluster using an effective radius $a_i$ calculated in terms of its total surface area $A_i$ from $A_i = 4\pi a_i^2$. $A_i$ is calculated summing up, over all the particles of the cluster, all the sites in the vicinity of each particle without bonded particles to it, and multiplying by the primary particle size squared. Additionally, since the event of moving a whole cluster implies moving at once not only the particle that was chosen but all the particles associated to its cluster, the probability function is divided by the number of particles of the cluster, $m_i$. Finally, normalizing the probability function with a partition function $Z_{BM}$ over all the possible particles that could be randomly chosen, $P_{BMi}$ is written as

$$P_{BMi} = \frac{1}{Z_{BM}} \frac{1}{m_i a_i^{3/2}} \quad (2.3)$$

where

$$Z_{BM} = \sum_k \frac{1}{m_k a_k^{3/2}} \quad (2.4)$$

Although (2.3) does not include the processing conditions of the sol explicitly, the probability function is the result of the diffusivity (or viscosity), which are temperature dependent. The explicit dependence on temperature is canceled in the normalization, unless the temperature is not constant through the system. The only parameters that explicitly appear in the normalized $P_{BMi}$ in (2.3) are those that differentiate the clusters from each other, i.e., their sizes.

### 2.2.2 Reactions

The chemical reactions to aggregate particles are incorporated to the MC simulation when collisional events occur during the BM. If a collision occurs, a bond can form, which is equivalent to say that a reaction took place. The event of occurrence of a reaction is determined
The probability function \( P_{\text{React}} \), that depends on the change of total energy of the system, \( \Delta H \), following the Metropolis algorithm in (2.1). \( \Delta H \) can be measured at the states of the system after and before the reaction. In this way, calculating the total energy of the system for each state allows us to calculate \( \Delta H_k \) between all the possible reactions, each one identified by the subscript \( k \).

The total energy of formation for all the possible cluster formations can be found experimentally using calorimetry and spectroscopy such as Nuclear Magnetic Resonance [23, 54], or computationally using MD simulations and Quantum Mechanics calculations [3, 55, 56, 50, 47]. The calculation of the energy of the system, requires the energy of formation of the clusters and the clusters present in the solution. The energy of formation of the clusters is highly dependent on the conditions of the solution, i.e., the processing parameters for the solution. These parameters are the temperature and the concentration ratio of the species involved in the process. For this study, to show how the model works, the formation energies of several clusters are taken from [3]. The data in [3] corresponds to a single processing condition, but the model can be applied to other conditions, simply by changing the energy of formation of the clusters.

The processing condition of [3] are summarized in Table 2.1. Since no catalyst was added, these conditions yield a neutral silica aerogel. The energy of formation \( E_c \) of the clusters shown in figure 2.2 are presented in the table 2.2. The nomenclature \( q^n_m \), as it was introduced in [3], means that the cluster has \( n \) silicon atoms which have \( m \) bonded silicons (by oxygen bridges). Based on these data, the change of the total energy, \( \Delta H_k \), of the clusters due to a reaction (labeled as \( k \)) can be calculated as follows.

First, note that in a single reaction, only two particles have a change in their energy. The
other particles remain untouched, thus, note that the change of the total energy is solely due to the change of the energy of the colliding particles. This suggests to model the total energy of the clusters, $E_c$, as a summation of the energies of the individual particles, $E_i$ that compose each cluster, and to consider the energy of each particle dependent only on its state, which is described by the *coordination number*, i.e. the number of bonds that the particle has.

Two models to describe the total energy of the clusters after a reaction are proposed here,

- **Model 1**: Each particle has an energy $E_i$ which depends on its coordination number $n_c$. Therefore, the energy of a particle is one of the set \{$E_1, E_2, E_3, E_4$\}. Only four states are considered for the silica system since silicon can at most form four bonds. The energy required for five or six bonds is too high, therefore improbable.

- **Model 2**: $E_i$ linearly increases with $n_c$ as

  \[ E_{n_c} = a_1 + n_c a_2 \]

  where the value of $a_1$ is the energy associated with the existence of a particle, and it can be related to the chemical potential of the system at the processing conditions. The value of $a_2$ is the energy associated with the reduction of surface area due to the bonding between particles, related to the surface tension in the liquid-solid system (see sect. 2.2.2 for a refined model). Note how Model 2 is a simplified version of Model 1.

Knowing the formation energies of several clusters allows us to calculate the best fitting values of \{$E_1, E_2, E_3, E_4$\}, and $a_1$ and $a_2$. The parameters are found by the minimizing the error of a model

\[ [M] \vec{p} = \vec{E}_c \]  \hspace{1cm} (2.5)

where $[M]$ is the matrix of the coefficients of the equations in table 2.2 (based on Models
Figure 2.2: Geometry of the clusters modeled by [3]. Each bar represents an oxygen bond between two silicon atoms.
Table 2.2: Cluster formation energies from [3] and models.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>$E_c$ [kcal/mol]</th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1^2$</td>
<td>-9.381</td>
<td>2$E_1$</td>
<td>2($a_1 + a_2$)</td>
</tr>
<tr>
<td>$q_2^1 q_2^1$</td>
<td>-15.512</td>
<td>2$E_1 + E_2$</td>
<td>2($a_1 + a_2$) + ($a_1 + 2a_2$)</td>
</tr>
<tr>
<td>$q_3^1$</td>
<td>-4.763</td>
<td>3$E_2$</td>
<td>3($a_1 + 2a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1$</td>
<td>-38.146</td>
<td>2$E_1 + 2E_2$</td>
<td>2($a_1 + a_2$) + 2($a_1 + 2a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1$</td>
<td>-30.616</td>
<td>3$E_1 + E_3$</td>
<td>3($a_1 + a_2$) + ($a_1 + 3a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1 q_1^1$</td>
<td>-4.192</td>
<td>$E_1 + 2E_2 + E_3$</td>
<td>($a_1 + a_2$) + 2($a_1 + 2a_2$) + ($a_1 + 3a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1$</td>
<td>-25.715</td>
<td>4$E_2$</td>
<td>4($a_1 + 2a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1 q_1^1$</td>
<td>-43.587</td>
<td>2$E_1 + 3E_2$</td>
<td>2($a_1 + a_2$) + 3($a_1 + 2a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1 q_1^1$</td>
<td>-40.198</td>
<td>3$E_1 + E_2 + E_3$</td>
<td>3($a_1 + a_2$) + ($a_1 + 2a_2$) + ($a_1 + 3a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1$</td>
<td>-31.978</td>
<td>4$E_1 + E_4$</td>
<td>4($a_1 + a_2$) + ($a_1 + 3a_2$)</td>
</tr>
<tr>
<td>$q_3^1 q_3^1 q_1^1$</td>
<td>-20.049</td>
<td>$E_1 + 3E_2 + E_3$</td>
<td>($a_1 + a_2$) + 3($a_1 + 2a_2$) + ($a_1 + 3a_2$)</td>
</tr>
<tr>
<td>$q_3^1$</td>
<td>-12.582</td>
<td>5$E_2$</td>
<td>5($a_1 + 2a_2$)</td>
</tr>
</tbody>
</table>

1 and 2), $\vec{p}$ is the vector with the parameters of the model, and $\vec{E}_c$ is the vector containing the formation energies given in column 2 of table 2.2. The results of the fitting are presented in figure 2.3. The errors between the formation energy of the clusters predicted from reaction models 1 and 2, and the values calculated in [3] are shown in figure 2.4. The error decreases quickly as the size of the cluster (number of silicon atoms) increases. As a result, for the processing condition described in table 2.1, model 1 shows an increase in the energy associated to particles as its coordination number increases, which is in good agreement with the linear model 2. From now on, only model 2 is considered because it correctly captures the description of the energies of formation of the cluster under the mentioned processing conditions, and it is simpler than Model 1.

Two colliding particles forming a bond increase their coordination number $n_c$ by one. Therefore, the change of total energy of the system (for each new bond), $\Delta H_{AB}$, due to the reaction for the two colliding clusters, tagged as $A$ and $B$, is

$$\Delta H_{AB} = \Delta H_A + \Delta H_B$$

$$= (E_{Af} - E_{Ao}) + (E_{Bf} - E_{Bo}) \quad (2.6)$$

Using the Metropolis algorithm (2.1)
$E_c(n_c) = (-19.31 \pm 1.54) + (7.89 \pm 0.88)n_c$

Figure 2.3: Energy Models

Figure 2.4: Error of Models compared to the values from [3].
CHAPTER 2. AGGREGATION ALGORITHM

$$P_{\text{React}} = \frac{1}{Z} e^{-\frac{\Delta H_A}{k_b T}} e^{-\frac{\Delta H_B}{k_b T}}$$

(2.7)

According to model 2, $\Delta H_{AB}$, due to the reaction for the two colliding clusters is,

$$\Delta H_{AB} = 2a_2$$

(2.8)

which allows us to conclude that $P_{\text{React}}$ is a constant since $a_2$ is a constant.

Refined Model

As observed above, $P_{\text{React}}$ is constant under the processing conditions presented in table 2.1 (from [3]), but that may not be always the case[57]. What would happen if the processing conditions are modified? It becomes a daunting task to get all the possible reaction energies, $\Delta H_{AB}$, for all the possible processing conditions, and to analyze the different types of structures that could appear. Instead, a model for $\Delta H_{AB}$ is proposed here in order to study the structures parametrically. For a material system where every additional bond on a particle $i$ requires more energy than the previous one (having a coordination number $n_{ciA}$), the change in total energy of each cluster $A$ due to the reaction can be calculated as

$$\Delta H_A = a_2 + \hat{w} n_{ciA}$$

(2.9)

where $\hat{w}$ represents the extra energy needed (or the unused energy) to bond two particles compared to the linear model. The effective surface energy, $\hat{a}_2 = a_2 + \hat{w} n_{ci}$, is no longer a constant. As a result, if (2.9) is put into the probability of the Metropolis algorithm in (2.7), $P_{\text{React}}$ becomes,
\begin{equation}
\begin{split}
P_{\text{React}} &= \frac{1}{Z} e^{\frac{a_2 + \hat{w}_n c_A}{k_b T}} e^{-\frac{a_2 + \hat{w}_n c_B}{k_b T}} \\
&= \frac{1}{Z} e^{-\frac{2a_2}{k_b T}} e^{-\frac{\hat{w}_n c_A}{k_b T}} e^{-\frac{\hat{w}_n c_B}{k_b T}} \\
&= \frac{1}{Z} w^{n_A} w^{n_B}
\end{split}
\tag{2.10}
\end{equation}

Since $Z = Z e^{\frac{2a_2}{k_b T}}$ is constant, and defining the reactivity as $w = e^{-\frac{\hat{w}}{k_b T}}$, the probability of reaction can be written as a function of the reactivity. Note that for particular case of $\hat{w} = 0$, $w = 1$, and $P_{\text{React}}$ is a constant, matching the processing condition and the cluster formation energies calculated in [3] and given in table 2.1.

The final expression in (2.10) is similar to the one used by [39] with the difference that functionality was used in [39] instead of the coordination number to calculate $P_{\text{React}}$. The coordination number counts the formation of bonds for a particle. Therefore, this parameter accounts the energy state of the particle. On the other hand, the functionality, defined as the number of particles that are in the vicinity, bonded or not, is not directly related to the energy state. As it is shown (section 2.3), the functionality and coordination number distributions are different for the same structure. Furthermore, the response of the structure correlates with either functionality or coordination number distribution, as it is the case of density and elastic response, respectively. This becomes crucial for tailoring a material for a particular application, when it is desired to optimize one property or another.

### 2.3 Generated Sol Structures

Two structures with 251 and 1326 particles with simulation box length 10 and 20, respectively, were simulated to achieve densities\footnote{The density of amorphous silica is 2200 [kg/m$^3$].} of 2200 [kg/m$^3$] · (251)/10$^3$ = 552.2 [kg/m$^3$] and 2200 [kg/m$^3$] · (1326)/20$^3$ = 364.65 [kg/m$^3$]. These two samples were chosen to satisfy the relationship in (2.11) given by a fractal dimension, $D = 2.4$, as found experimentally for
Figure 2.5: Network for $w = 1$, and two different correlation lengths, $L$. 

(a) $N = 251$ and $L = 10$

(b) $N = 1326$ and $L = 20$
Figure 2.6: Coordination number distribution for $N = \{251, 1326\}$ and $L = \{10, 20\}$ with $w = 1$

Figure 2.7: Functionality distribution for $N = \{251, 1326\}$ and $L = \{10, 20\}$ with $w = 1$
neutral aerogels \cite{27}. Since only one particle fits in a simulation box of size equal to the lattice parameter, \((N(1) = 1)\), the number of particles in a simulation box of size \(L\) can be calculated using (2.11). The number of particles for the large sample was chosen to take a reasonable computer simulation time.

\[
N(L) = L^D
\]  

(2.11)

The simulation box length, which is dimensionless, can be converted to real length units by multiplying the lattice parameter length of 0.4 nm \cite{27} estimated by scattering experiments. Periodic boundary conditions are used for the algorithm, and the reported error is the standard deviation calculated from 24 samples.

The structure percolates when the correlation length is chosen equal to or greater than the size of the simulation box as evidenced in figure 2.5 for the reactivity chosen to reproduce neutral silica aerogels with the processing conditions of table 2.1. Percolating means that the formed structure spans the entire system\cite{51}. The simulation box length is also the correlation length which measures the size of the clusters \cite{27} for networks that percolate \cite{58}.

The number of coordination distribution and the functionality distribution are presented

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{coordination_number_distribution.png}
\caption{Coordination number distribution for different number of particles holding the same initial density.}
\end{figure}
in figures 2.6 and 2.7. All particles aggregated into a single cluster congruent with a 0% of isolated particles. If there were isolated particles, the percentage of 0-coordination number, and 0-functionality number would be different from 0.

The coordination ($n_c$) and functionality ($f$) distributions measure different aspects of the structure evidenced by the different results obtained in figures 2.6 and 2.7. The $n_c$ distribution shows how the structure is connected. 35% of the structure consists of dead ends, i.e. particles with a single bond. Another 35%, with $n_c = 2$, are particles that act as elbows and linear bridges. Higher coordination numbers associated to branches are less likely to be seen, explaining an average coordination number of 2.0. The $f$ distribution shows something different but not contradictory. Mostly all particles are surrounded by 2, 3, or 4 particles making it a compact structure explaining an average functionality of 2.8.

Next, when the correlation length is chosen much smaller than the size of the simulation box, the simulation yields aggregates that not percolate. The initial density is chosen to be lower than the one predicted for a correlation length corresponding to the size of the simulation box. When the system aggregates, locally the density increases creating isolated
Figure 2.10: Coordination number distribution for several densities with $L = 10$.

Figure 2.11: Functionality distribution for several densities with $L = 10$. 
clustering that do not percolate. Due to the nature of the simulated aggregation process when the aggregates do not percolate, it is thus possible to study the structure without adverse influence of the boundary conditions.

The influence of the size of the simulation box on the aggregation process is considered simulating systems with 20, 30, 40, 100 and 400 particles while keeping the same density. There is no significant difference between the $n_c$ distributions when varying the number of particles while maintaining the initial density $^5$ for a single reactivity as presented in figure 2.8. The coordination number distribution behaves similarly to that of a denser sample as compared to figure 2.6.

The $f$ distribution changes with the number of particles maintaining the initial density constant as presented in the figure 2.9. The percentage of 1 and 2 functionality numbers decrease with the increase of the number of particles, while for 4, 5, and 6 functionalities, the percentage increases. This translates into a more compact structure with the increase of the number of particles.

Furthermore, the effect of the density $^6$ is explored by modeling systems with dimensionless

---

$^5$ The initial density used here is defined as the ratio of the number of particles and the volume of the simulation box because as the simulation evolves, the density is no longer uniform in space.

$^6$ The real density is equal to the dimensional density times the mass of amorphous silica that occupies the volume of the primary particle.
density of $40/10^3$, $100/10^3$, $200/10^3$, $400/10^3$, $706/10^3$ for a constant size of the simulation box. Changing the density of the samples does not cause a significant difference in the $n_c$ distribution as it is showed in figure 2.10. Again, the $n_c$ distribution resembles the one in figure 2.6. However, the $f$ distribution is affected by the initial density as shown in figure 2.11. The functionality distribution is shifted towards higher functionality as the density is increased creating more compact structures. The same effect occurs in figure 2.7.

Additionally, the algorithm is used to investigate the different structures achieved by varying the reactivity. The $n_c$ and $f$ distributions are investigated for reactivities of 0.05, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6 for a constant dimensionless density of $40/40^3$.

The coordination number and functionality distributions are presented in figures 2.12 and 2.13 respectively. There is a significant difference in coordination number distributions for different reactivities. The same happens for the functionality distributions, however, the amount of triple-coordinated particles increases reaching a maximum at $w = 0.8$, and then decreases. For tetra-, penta-, and hexa-coordinated particles the tendency is not completely clear, but they seem to follow the same trend as triple-coordinated particles for higher reactivities. The $f$ distribution behaves with a similar trend as the $n_c$ distribution, but the values...
in the coordination number distributions seem to be higher compared to the functionality
distribution for lower numbers (almost doubled for $n_c = 1$ compared to $f = 1$), while they
seem to be lower for higher numbers.

2.4 Discussion

Coordination number and functionality distributions shown in figures 2.6 and 2.7 respectively, measure different information about the structure of the neutral silica aerogel. The
different responses of the structure, such as the mechanical response and the scattering of
a coherent beam, depend on the connected structure or the mass distribution, respectively.
However, the invariance in the results for different simulation box lengths scaling the num-
ber of particles with (2.11) are evidence of the fractal structure as result of the aggregation
process.

The samples in figures 2.6, 2.7, 2.8, 2.9, 2.10, and 2.11 were chosen to observe the struc-
ture of neutral silica. The functionality tends to be higher than the coordination number
in average. For very large samples, with low density and large correlation lengths, it is ex-
pected that the functionality distribution converges to the coordination number distribution.
However, the functionality tends to be higher, meaning that not all of the particles in the
vicinity of a particle are bonded to it. This also means that for the critical density when the
system still percolates, the structure can arrange into a configuration conserving the corre-
lation number distribution that minimizes energy with the minimum density given by the
functionality distribution.

Achieving a lower density than the critical in a percolating structure would mean the
development of a different coordination number distribution during the aggregation process
which no longer minimizes the energy. So, in this case of lower densities, scattered clusters
appear without percolating. Each cluster will have a correlation number distribution which
minimizes the energy. On the contrary, a higher density implies the allocation of the excess of
particles into the remaining spaces, intertwining the forming clusters, affecting the functionality distribution, shifting the average towards higher numbers. Nevertheless, the minimization of the energy is satisfied by the invariance of the correlation number distribution.

Invariance of the coordination number and functionality distributions for the neutral silica samples is presented in the figure 2.6 and 2.7. For the bigger sample, whose simulation box is twice as large, each part of the structure is connected and positioned in a similar way as its parts. In other words, if a part of the structure is taken, it will resemble the structure generated with less particles, i.e. a subpart with box length 10 of the sample with box length 20, resembles the simulation with box length 10. This is evidence of the fractal (self-affine) structure of neutral silica aerogels which is replicated with the aggregation model proposed here. The invariance of the coordination number distribution remains as long as the reactivity is fixed (as observed in figures 2.6, 2.8, and 2.10), which is valid too for samples whose correlation length is smaller than the size of the simulation box.

In contrast, the \( f \) distribution changes as presented in figures 2.9 and 2.11. In figure 2.9, the density is constant by increasing the number of particles and the size of the box. As the clusters are aggregating, bigger clusters react with bigger clusters leaving bigger spaces. To reach the same density at a large scale, the clusters must be denser (or more compact) compared to the case with less particles. In other words, since the scale jump in a larger system is greater than in a smaller system (\( L \) is larger), the vacancies that appear by the aggregation of the clusters are compensated by more compact structures as the result of the apparent reduced volume available perceived by the particles when a cluster with empty spaces is formed. This explains how larger functionality numbers appear for larger number of particles for the same initial density. In figure 2.11 a similar effect is seen, but directly related to the addition of more particles that must be fitted in the spaces as the density is incremented for the same simulation box.

Combined, figure 2.9 and 2.11 explain figure 2.7. Since [27] found a decreasing relationship of the density with the correlation length (\( \rho \propto L^{-1.67} \)), the system creates empty spaces
to cope with the decreasing density. In contrast to the previous cases, the reduction of the density is stronger than the reduction of the density explained by the spaces created by the aggregation of big clusters, so the structure needs to create less compact configurations as the correlation length, $L$, is increased.

Additional processes can be used to make the coordination number distribution converge to the functionality distribution for higher densities, creating the missing bonds. Aging and sintering applied to the structure as gelled, allow to create missing bonds between the particles and its neighbors. The $f$ distribution is fixed during these processes since the functionality does not depend on bonding. In fact, the $f$ distribution becomes the limit of the $n_c$ distribution as the number of bonds is increased. So, aged and sintered structures will have the same $f$ and $n_c$ distributions, and only one distribution is enough to characterize them.

Changing the reactivity of the process to generate samples with constant density produced interesting results (figures 2.12 and 2.13). The reactivity determines which types of reactions are more prone to occur, favoring more linear structures for lower reactivities as presented in figure 2.12. In this way, for low reactivities, about 80% of the particles are elbows or linear structures. A more branched structure is formed as the reactivity is increased, first, raising the percentage of 3-coordinated particles, and then, for higher reactivities, higher order coordination numbers become more important. Note that for reactivities higher than 1, meaning $\hat{w} < 0$, it is easier to aggregate higher coordinated particles as observed in (2.9). On the other hand, for reactivities lower than 1, lower coordination numbers are favored. For instance, samples generated with processes at a low reactivity ($w = 0.05$) the structure has above 80% of its particles as double-coordinated particles after the gelation process and above 60% after a complete aging, which means that the structure is mostly formed by long fibers. For structures generated using processes with high reactivity ($w = 1.6$) the amount of single-coordinated particles (broken ends) is about 50%, which reduces to just over 10% after complete aging. On the other hand for the range of reactivities simulated, it seems that
$w$ does not have a strong effect on the amount of triple-coordinated particles after complete aging, all the structures having between 25% and 30% of these particles.

### 2.5 Conclusions

The effect of reactivity and density on the structure of the aerogel and the link established here between these parameters, as well as the processing conditions of the sol, expand the possibilities for designing aerogel structures according to application requirements. The structure as gelled depends uniquely on the density and the reactivity. For instance, aerogels expected to have high thermal resistivity can be fabricated using conditions favoring high reactivity, and not allowing aging of the samples. The final structure will have a large amount of broken ends, which will create a cumbersome path for the heat to diffuse through. Another example could be the fabrication of aerogels expected to have good structural integrity, and to be able to withstand loads. For this case a structure with a small percentage of broken ends is desired; therefore a process with low reactivity should be used. Aging and/or sintering the samples would have a positive effect on integrity, because it will increase the amount of branching points in the structure, due to an higher percentage of triple- and tetra-coordinated particles, along with a reduction of the broken ends.
Chapter 3

Scattering Response

Abstract

Scattering of sol-gel structures is investigated computationally. Sol-gels are recreated through an aggregation algorithm incorporating Brownian motion and chemical reactions. Using the fractal character of sol-gels, the concept of recursion is introduced as a tool to perform multi-scale computation of the response of sol-gels through the different scales from the atomic level to the macro scale. The concept is illustrated with the prediction of scattering intensity. The relationship between scattering intensity and functionality is investigated, noting that the later is a function of the Brownian motion and chemical reactivity. Computational simulation tools are developed to predict scattering intensity as a function of density and reactivity, the former represented by the number of particles, or clusters, in the simulation box. Then, the results are correlated to an analytical model that reveals the critical wave number, or critical scale, at which percolation occurs.

3.1 Introduction

Advanced products such as dense films, Aerogels, super capacitors, and dense ceramics can be fabricated with Sol-gel technology \cite{10, 11, 12, 13, 16, 18, 19, 22, 26}. This technology uses colloidal aggregation \cite{31, 30, 27, 28, 59, 2, 60, 37, 38, 39, 45, 14, 17, 24, 61, 29, 40, 41}, which after the removal of the liquid phase leaves a solid ceramic structure \cite{23, 27}. Extracting the liquid phase of the colloid gently, e.g. by supercritical drying, leaves a solid structure with unique physical properties \cite{23, 12, 30, 31, 5}. 
The outstanding properties of gel-derived materials are the result of a) the physical properties of the base material and b) the unique structure of the resulting material [29, 62]. To characterize the structure of gel-derived materials, Small Angle Neutron Scattering (SANS) and Small Angle X-Ray Scattering (SAXS) have been used extensively [43, 44, 63, 42, 64, 27, 5, 32, 65] revealing their fractal structure along several length scales [27].

In many cases, physical properties of the gel-derived structures can be explained by their fractal structure [29, 62]. However, fractal theory is not always applicable, in particular when it becomes inadequate to associate a fractal range to the structure, or when within a proved fractal range the response is not explained by classical fractal theory [5, 32].

In this work, it is proposed that some responses of gel-derived materials depend on the connectivity of the structure while other responses depend on the mass distribution of the structure. In this manuscript, density and scattering intensity are shown to depend on the mass distribution. For this, density and scattering intensity of computer generated structures that resemble gels and Aerogels are evaluated. On the other hand, Mechanical response of gels and Aerogels and their relationship to scattering are investigated in Part II [66].

3.2 Computer-generated Structures

The structures at scale $\lambda \in [1, \lambda_{max}]$ are generated by an aggregation algorithm explained in detail in [61]. Here $\lambda_{max}$ stands for the maximum scale at the first generation of the multi scale algorithm, a scale that is given by the size of the simulation box $L = \lambda_{max}(2a_0)$, where $2a_0$ is the size of the primary particles. First, particles are randomly positioned at the sites of a cubic lattice inside the simulation box. Then, a particle is chosen randomly to move in the lattice, in order to reproduce the Brownian motion that occurs in a forming colloid satisfying Einstein-Smoluchowsky theory [4, 53]. For two colliding particles, the probability of forming a bond is determined by the reactivity $\omega$ and their coordination numbers $n_{cA}$ and $n_{cB}$. If the bond is formed, the clusters containing the colliding particles bond into a single,
CHAPTER 3. SCATTERING RESPONSE

Figure 3.1: Cross-section of an aggregated structure inside the simulation box, at scale $\lambda \in [1, \lambda_{\text{max}}]$. Larger cluster. Periodic Boundary Conditions (PBC) are used to delimit the simulation box. The algorithm ends when all particles form a single cluster. At this point we say that all particles have aggregated, which is not the same as saying that all particles have bonded with their neighbors. For that one would have to age/sinter the structure. Typical structures are depicted in Figure 3.1 for low and high reactivity $w$.

Reactivity is a measure of the increase or decrease of additional energy required to form a new bond as a function of the number of bonds already formed. Using this concept, the probability of reaction of two particles is calculated using the the Metropolis algorithm [61]. In this way, different structures are formed by varying the reactivity. Longer branched structures are formed for lower reactivities, and more compact structures are formed for higher reactivities. The particular case of neutral Aerogels was found to correspond to a reactivity of $w = 1$.

The resulting structure has a functionality\(^1\) distribution which not always coincides with the coordination number\(^2\) distribution. However, both distributions can be modified as follows. When sintering and/or aging a sol, the coordination number distribution approaches the functionality distribution, as bonds appear between the particles that are next to each other. Also, as the initial density is lowered, the functionality distribution approaches the coordination number distribution (since all particles that are neighbors tend to bond when

---

\(^1\)The functionality counts how many particles are next to a particle.

\(^2\)The coordination number counts how many particles are bonded to a particle.
the density is low). Thus, responses such as stiffness, which are associated to the connectivity of the structure, are expected to be related to the coordination number rather than to the mass distribution, which is measured by the functionality. On the other hand, responses such as scattering intensity, which are associated to mass distribution, are expected to be related to functionality.

In this paper, scattering intensity (as a measure of mass distribution) is investigated for sol-gels. Mechanical properties (as a measure of the connectivity of the structure) and their relationship to scattering, are investigated in Part II [67].

### 3.3 Correlation Length

The correlation length $\xi$ is a measure of the size of the clusters that have aggregated during the gelation process. The relation between correlation length and density of the cluster is developed in this section for structures with constant fractal dimension. In a simulation box of size $L$, the aggregation process initially consists of $N_0$ primary particles of size $2a_0$ that aggregate. The primary particles may be molecules like $SiO_2$, or cluster of molecules with a known size and density. The effective density is $\rho_{ef}^0 = N_0/L^3$, where $N_0$ is the number of particles in the simulation box. Conservation of mass implies that the effective density is constant through the aggregation process. However, during aggregation, gaps are created as the clusters bond without perfect match. Therefore, the density of the forming clusters decreases due to the incorporation of vacancies, the size of the clusters $\xi$ increases, and the number of clusters $N_C$ decreases. Due to the decrease in the number of clusters in the simulation box, the average distance between clusters $d$ increases. In this way, the density of the cluster can be calculated as

$$\rho_C = \frac{N_C}{L^3} = \frac{1}{d^3} \quad (3.1)$$

Note that the number of particles $N_0$ can be calculated as
\[ N_0 = N_C N_{k/C} \quad (3.2) \]

where \( N_{k/C} \) is the number of particles per cluster. Furthermore, assuming that the aggregation process leads to a fractal structure, \( N_{k/C} \) follows a power law as

\[ N_{k/C} = \left( \frac{\xi}{2a_0} \right)^D \quad (3.3) \]

where \( D \) is the fractal dimension of the clusters.

At the end of the aggregation process, when only one cluster is found inside the simulation box, \( N_C = 1 \). Therefore, (3.3) can be rewritten as

\[ \frac{\xi}{2a_0} = N_0^{1/D} \quad (3.4) \]

From (3.1), it is concluded that the mean free path \( d \) becomes equal to the size of the simulation box, i.e. \( d = L \).

Note that if \( \xi < L \), the structure will not percolate and the effective density \( N_0/L^3 \) would be lower than the density of the cluster \( N_0/\xi^3 \).

Since \( \xi \) is limited by \( d \), the system percolates for \( L = \xi = d \), thus defining the critical percolation density

\[ \rho_{crit} = L^{D-3} \quad (3.5) \]

In other words, if \( \xi \) is chosen to be at least \( d \), the cluster spans the entire simulation box, connecting opposite faces of the simulation box. Using a dimensionless system, we have \( 2a_0 = 1 \), \( L' = L/(2a_0) \), and \( \rho' = (2a_0)^3 \rho \) Then, \( \rho'_{crit} = L'^{D-3} \).
3.4 Percolation Kinematics

In this section, the evolution of a cluster during the aggregation process is described in terms of the number of particles per cluster $N_{k/C}$ and the number of clusters $N_C$ in the simulation box.

The aggregation process is assumed to be isotropic. At time $t$, in average, there are $N_{k/C}^{(t)}$ particles per cluster and $N_C^{(t)}$ clusters in a simulation box of volume $L^3$. Let $\Delta t$ be the time it takes for the clusters to collide and bond with a second cluster, thus doubling the number of particles in each cluster while cutting in half the number of clusters in the simulation box. Then,

$$\begin{align*}
N_{k/C}^{t+\Delta t} &= 2N_{k/C}^t \\
N_C^{t+\Delta t} &= \frac{N_C^t}{2}
\end{align*}$$

Using (3.3) in (3.6) and (3.1) in (3.7), the evolution of the correlation length $\xi$ and the separation between clusters $d$ is given by

$$\begin{align*}
\xi^{t+\Delta t} &= 2^{1/D} \xi^t \\
d^{t+\Delta t} &= 2^{1/3} d^t
\end{align*}$$

Noting that $d^0 = LN_0^{1/3}$ and $\xi^0 = 2a_0$, the recurrence can be solved for a time $t = n\Delta t$ as

$$\begin{align*}
\xi^{n\Delta t} &= 2^{n/D} N_C (2a_0) \\
d^{n\Delta t} &= 2^{n/3} LN_0^{-1/3}
\end{align*}$$
Since $\xi$ and $d$ share the value $n$, the parameters are correlated by

$$\xi(t) = 2a_0 \left( \frac{d(t)}{L N_0^{-1/3}} \right)^\frac{1}{D}$$ (3.12)

If $\xi(t) = L$ at any time during the aggregation process, the structure percolates. This is the proposed percolation criterion.

It can be shown that if the separation between the clusters $d(t)$ is equal to $L$, when there is only one cluster in the simulation box, the size of the clusters satisfy $\xi(t) = L$, and the structure critically percolates.

When percolation starts at $t = t_{per}$, $\xi(t_{per}) = L$, and the distance between clusters is

$$d(t_{per}) = \left( \frac{N_0}{L^3} \right)^{-1/3} \left( \frac{L}{2a_0} \right)^\frac{2}{3}$$ (3.13)

Note that $d(t = t_{per})$ may be smaller, equal, or greater than $L$. Thus, the number of clusters $N_C(t = t_{per})$ inside the simulation box can be calculated as

$$N_C(t = t_{per}) = N_0 \left( \frac{2a_0}{L} \right)^D$$ (3.14)

If the percolation criterion is not satisfied ($\xi(t) < L$) with one cluster ($N_C = 1$) aggregating all the particles, then $\xi/(2a_0) = N_0^{(1/D)}$, and $d = L$. The isolated cluster does not percolate.

If the percolation criterion is satisfied ($\xi = L$) with one cluster ($N_C = 1$), the aggregation process ends as soon as the structure percolates ($t = t_{per}$).

If the percolation criterion is satisfied and $N_C > 1$, then there are multiple clusters still available that can further aggregate. If allowed to further aggregate in a larger simulation box, the final cluster would be larger than the simulation box ($\xi > L$).

From this discussion, three regimes are identified:

1. When $\xi < L$, isolated clusters are suspended in the simulation box.
2. When $\xi = L$, the density is the critical percolation density, and the structure spans the simulation box.

3. When $\xi > L$, the structure spans the simulation box before all particles are aggregated.

### 3.5 Scattering Intensity

Scattering has been widely used to characterize the structure of aerogels and disordered systems \[43, 44, 63, 42, 64, 27, 5, 32, 65\]. Researchers use scattering techniques to identify the size of the primary particle $2a_0$ and the size of the clusters formed, the latter known as the correlation length $\xi$. Also, scattering experiments support the fractal nature of aerogels by evidencing a power law between the scattered intensity $I(q)$ and the modulus of the scattering wave vector $q$

$$I(q) \propto q^{-D_q}$$  \hspace{1cm} (3.15)

where $D_q$ is the fractal dimension associated to the intensity.

The scattering intensity is obtained by multiplying the scattering function\footnote{The scattering function accounts the interference of the scattered beams caused by the mass distribution of the scattering centers.} $S(q)$ by the form factor\footnote{The form factor accounts for the scattering of the beam by each of the scattering centers.} $P(q)$ of the scattering centers and the number of particles as follows \[64\]

$$I(q) = NP(q)S(q)$$  \hspace{1cm} (3.16)

where $N$ is the number of particles. The scattering function $S(q)$ can be calculated as

$$S(q) = \frac{1}{N} \sum_i^N \sum_j^N m_i m_j \frac{\sin(qr_{ij})}{qr_{ij}}$$  \hspace{1cm} (3.17)
where \( r_{ij} = |r_i - r_j| \) is the relative distance between the centers, \( r_i \) and \( r_j \), for the particles tagged as \( i \) and \( j \), and \( m_i, m_j \) are the scattering amplitudes proportional to the mass of the scattering centers \( i \) and \( j \), respectively.

In (3.17), \( \frac{1}{N} \sum_i \) corresponds to the average over \( N \) particles of the total intensity of the interference between the scattered beams by particle \( i \) and all the other particles (summation over \( j \)).

The calculation of the scattering intensity of large structures can be done using their fractal character. By fractal character it is understood that the structure (mass distribution) is similar at different scales [68, 69]. Similarity implies that for each entity at a certain scale, there is another entity identified in a similar way at a larger scale. Considering a structure at two scales, a small scale \( \lambda \in [1, \lambda_{\text{max}}] \) and at large scale \( \lambda' \in [\lambda_{\text{max}}, \lambda^2_{\text{max}}] \), we have

\[
\lambda' = \lambda \lambda_{\text{max}}
\]  

where \( \lambda_{\text{max}} = L_{\text{max}}/L_{\text{min}} \), \( L_{\text{min}} \) is the size of the primary particle at the smallest scale of the aggregation process.

The smallest discernible particle is called primary particle. Its properties are those of the precursor material, which in this study is \( \text{SiO}_2 \). The primary particle is identified at the small scale range at \( \lambda = 1 \). At any subsequent scale, the primary particle is the cluster generated at the previous scale. This means that the primary particle size at the larger scale is of the size of the cluster at the previous smaller scale. The primary particle at the subsequent scale is named primary cluster and is defined at \( \lambda' = \lambda_{\text{max}} \).

The scattering function (3.17) is valid for the range

\[
\frac{\pi}{2L_{\text{max}}} < q < \frac{\pi}{2L_{\text{min}}}
\]  

where \( L_{\text{max}} = \lambda_{\text{max}} L_{\text{min}} \) is the size of the simulation box, and \( L_{\text{min}} \) is the primary cluster size at each scale. Thus, (3.19) suggests that the calculation of the scattering intensity can
be separated into wave number ranges $\delta_i = [\frac{\pi}{2L_{\min}^{(i)}}, \frac{\pi}{2L_{\max}^{(i)}}]$.

$$I(q) = NS(q)P^{(i)}(q)$$  \hspace{1cm} (3.21)

where $i = \{i : q \in \delta_i\}$. The form factor $P^{(i)}(q)$ corresponds to the form factor of the primary cluster at the scale range $(i)$. Then, it can be reconstructed by recursion starting from $P^{(0)} = P_0(q)$ corresponding to the form factor of the primary particles, using

$$P^{(i)}(q) = P^{(i-1)}(q)\left(\frac{\pi/(2L_{\max}^{(i)})}{\pi/(2L_{\min}^{(i)})}\right)^{-D_q^{(n-1)}} = P^{(i-1)}(q)\lambda_{\max}^{D_q^{(n-1)}}$$  \hspace{1cm} (3.22)

with $D_q^{(n-1)}$ the fractal dimension calculated for $\delta_i$.

This study assumes $P_0^{(i)}(q) = 1$. Only the scattering function $S(q)$ is required to calculate the scattering intensity for any value of $q$ using (3.21) and (3.22).
Figure 3.2: Scattering Intensities for the samples in Table 3.1. Rows, from top to bottom, correspond to N=100, 200, 400, 800, 1600, and 1728 particles. Columns, from left to right, correspond to reactivities w=0.1, 0.2, 0.4, 0.8, 1.0, 1.2, and 1.6.
Scattering intensities for the samples in Table 3.1 using (3.17) are shown in Figure 3.2. In the Figure, $D$ is the fractal dimension calculated from (3.24) after conditioning the signal with (3.25), $2a_0$ is the cluster radius, and $q$ is the dimensionless wave number. The scattering profile $I(q)$ predicted by the computational experiment (simulation) is shown with thick solid lines. An approximation using the model described in Section 3.6 is displayed with thin lines. Among the thin lines, continuous lines indicate percolated structures and discontinuous lines represent non-percolated structures.

### 3.6 Analytical Model

An analytical model for the scattering intensity is proposed in this section. The model consists on dividing the scattering range into two intervals: 1) $q < q_{\text{crit}}$ where the scattering of the structure resembles the scattering of a continuum solid, and 2) $q > q_{\text{crit}}$ where the scattering of the structure resembles the scattering of isolated particles. Based on the three regimens identified in Section 3.4, the critical percolation wave number $q_{\text{crit}}$ is here defined as

$$q_{\text{crit}} = \frac{\pi}{2\xi} \tag{3.23}$$

### 3.6.1 Continuum Regime

For a simulation box of volume $L^3$, if the correlation length is larger than the size of the simulation box, $\xi > L$, the aggregating clusters percolate, thus, for $q < q_{\text{crit}}$ the structure looks homogeneous with the fractal dimension of a solid. Since the effective density does not change as a function of the wave number, the measured fractal dimension is $D_q = 3$. Hence, the scattering intensity follows (3.15), i.e., $I(q) \propto q^{-3}$.

This case is dominant when the number of particles $N$ is large and thus most of the sites are occupied, as illustrated by thin, continuum lines in Figure 3.2.
3.6.2 Isolated Clusters Regime

For \( q > q_{\text{crit}} \), the scattering intensity resembles that of isolated spheres \([65, 64]\) of radius \( R = \xi/2 \). An attenuation factor \((q/q_0)^{-(3-D_q)}\) is proposed here to take into account the mass reduction due to the fractal character of the clusters. Hence, the proposed scattering intensity model satisfies

\[
I(q) = \left(3\sin qR - qR \cos qr\right)^2 \left(\frac{q}{q_0}\right)^{-(D_q-3)}
\]  

(3.24)

In Figure 3.2, this regime is illustrated by thin, discontinuous lines. This regime is dominant for low number of particles. In this way, the fractal dimension of the clusters can be determined by fitting (3.24) to the scattering intensity calculated by the computer simulation (thick, solid lines in the Figure).

Polydispersity, due to the cluster radius variability is incorporated by averaging the scattering intensity (3.24) for a log-normal distribution of radii, using a procedure similar to \([65, 63]\). This is made possible by simulating 32 specimens for each of the 42 samples depicted in Figure 3.2.

3.6.3 Critical Percolation Regime

The isolated cluster regime transitions into the continuum regime at the critical percolation state with \( q_{\text{crit}} \) in the range given by (3.19). At this state, the structure is a collection of clusters forming a highly porous structure.

In Figure 3.2, the critical percolation is identified when \( q_{\text{crit}} \) enters into the simulation wave number range given by (3.19), i.e. \( 2R = L \). Also, during this regime, \( D = D_q \).

3.6.4 Scattering Results

Simulation of scattering intensity is reported in this section. The influence of the reactivity and precursor density on the scattering intensity was investigated by modelling the scattering
function of structures recreated using the algorithm described in Section \( \text{3.2} \). The fractal dimension inside the cluster and the radius of the cluster were measured by fitting the parameters of the analytical model presented in Section \( \text{3.6} \).

Border effects are removed in the same way they are removed experimentally \([65, 64]\). That is, the measured logarithmic intensity is subtracted from the signal \( S_{\text{cube}}(q) \) produced by an homogeneous sample, which in this work is a simulation box with all sites occupied (fractal dimension equal to 3). In this way, the reported intensity corresponds to

\[
I(q) = I_0 \frac{S(q)}{S_{\text{cube}}(q)} \left( \frac{q}{q_0} \right)^{-3}
\]

(3.25)

where \( I_0 = 1 \) for \( q_0 = q_{\text{min}} \).

The scattering intensity \( I(q) \), fractal dimension \( D_q \), and equivalent radius of the clusters \( R_{eq} \) are calculated for the conditions in Table \( \text{3.1} \) and presented in Figure \( \text{3.2} \). The reported wave number \( q \) is dimensionless calculated as \( q = 2\pi/\lambda \). Note that since \( P(q) = 1 \), the scattering intensity and the scattering function have the same values.

Thirty-two samples are used to average\(^5\) the scattering results for each of the 42 conditions studied. Since the most accurate measurement of the fractal dimension occurs for low number of particles, it is assumed that the fractal dimension for larger number of particles is the same, if the reactivity is the same. A standard deviation of 50% of the average cluster radius was used for polydispersity. In order to avoid a discontinuity between the sphere model and the continuous model, a log-linear interpolation is used from the \( q_{\text{crit}}/2 \) to \( q_{\text{crit}} \).

Note that the proposed analytical model for scattering intensity fits the behavior well in Figure \( \text{3.2} \). That is, the thin lines representing the analytical model of Section \( \text{3.6} \) overlap large regions if not all of the simulation results (thick lines). The regions that are not overlapped, are explained by the transition interval of the analytical model from the isolated cluster regime to the continuum regime. Note that there is an increase of the calculated

\(^5\)An average is calculated for the structures generated as the result of aleatory character of the algorithm yielding a polydisperse population of clusters.
fractal dimension and a reduction of the sphere radius with the increase of reactivity.

The criterion for percolation presented in Section 3.4 is used to determine if the structures percolated or not, which is then reported in Table 3.1. A tendency to percolate is observed as the reactivity decreases and the number of particles increases.

3.7 Discussion

For larger values of $q$, the small structural features influence the scattering results, whereas for smaller values of $q$, the larger features are responsible for the scattering intensity. Then, selecting a particular interval of wave numbers allows us to characterize the type of features present at a certain scale interval. However, the scale intervals and the wave number intervals are inversely proportional.

The peaks in Figure 3.2 are the result of the scattering of the clusters as isolated spheres (or scattering of the pores between the clusters). As the number of particles in the simulation box increases, a continuum phase appears, thus reducing the effect of the pores. As revealed by the analytical model, as the number of particles $N$ increases, the continuum regime becomes dominant compared to the isolated clusters regime. On the other hand, as the number of particles is reduced below the critical percolation density, the isolated clusters regime becomes dominant.

An increase in the number of particles increases the size of the associated cluster since each cluster is made of more particles. However, the fractal dimension of the clusters is not expected to change considerably since the same chemical reactions and Brownian motion occur as long as the clusters do not percolate. That is, for values of $q > q_{\text{crit}}$, all structures with the same reactivity have the same fractal dimension regardless of the number of particles.

Structures that become continuum at a certain scale, remain continuum for larger scales. Similarly, structures that do not percolate will remain disconnected at larger scales as well. Only structures that critically percolate can change their regime at larger scales, thus re-
Figure 3.3: Fractal dimension (continuous line) and critical percolation density (discontinuous line) as a function of reactivity.

As the reactivity increases, more compact structures are observed, congruent with the definition of reactivity and results in [61]. The fractal dimension and critical density, \( n_0 = N_0/L^3 \) as a function of the reactivity for critical percolation are presented in Figure 3.3. Since higher fractal dimensions correspond to more compact structures, reduced radii are expected for the clusters yielding higher critical densities. For the range of reactivities studied, there is a monotonic increase of fractal dimension and critical density. However, achieving compact structures with fractal dimension of 3 should require infinite reactivity. Furthermore, for reactivities several orders of magnitude smaller, almost linear structures are expected with fractal dimensions close but greater than one.

---

6The critical density is defined as the number of primary particles in the simulation box at the scale range considered.
3.8 Conclusions

The mass distribution in the sol-gel determines alone the scattering intensity. Then, the fractal character of the mass distribution causes the scattering intensity to have a fractal character as well. Also, the recursion algorithms proposed in order to calculate the effective scattering response of the structure at different scale ranges can be used as a consequence of as the mass distribution having fractal character. Therefore, the scattering intensity evaluated at a certain wave number can be understood as the effective form factor of the structure at the corresponding scale. In addition, the effective form factor can be investigated by analyzing the structure in a limited scale range, with a reduced number of particles as compared to solving an involved model spanning a broad range of scales, which would be prohibitively expensive. The results in the behavior of the functionality distribution reported by [61] were confirmed through scattering experiments by comparing the fractal dimensions, and finding higher fractal dimensions, i.e. more compact structures, for higher reactivities.
Chapter 4

Mechanical Response

This chapter is an article submitted to the Journal of Non-Crystalline Solids: E.J. Barbero and F.A. Campo. Sol-Gel Simulation–II: Mechanical Response.

Abstract

A novel computational procedure is proposed to predict the outstanding mechanical properties of sol-gel structures. An aggregation algorithm incorporating Brownian motion and chemical reactions is used to recreate the sol-gel structure at molecular scale. Just like in the physical colloidal aggregation process, the computational aggregation process produces structures with fractal features. Such fractal character leads to a recursion algorithm to calculate mechanical properties at any scale using a recursive multi scale approach. The mechanical properties are then predicted at each scale by calculating the effective properties using the finite element method. It is shown how the mechanical properties depend on the coordination number distribution, which is a function of the Brownian motion and chemical reactivity used to recreate the structures. Then, it is shown that mechanical properties can be correlated to scattering intensity of sintered and/or aged structures.

4.1 Introduction

Sol-gel technology, using colloidal aggregation, provides an efficient, green-manufacturing path for the fabrication of ceramic products [31, 30, 2, 60, 37, 38, 39, 45, 14, 17, 24, 40, 41]. Dense films, Aerogels, supercapacitors, and dense ceramics, to name some, are manufactured with this technology [10, 11, 12, 13, 16, 18, 19, 22, 26]. After extracting the liquid phase from the colloid, a solid ceramic structure remains. The resulting solid structure presents unique physical properties that are exploited for the fabrication of advanced products.
CHAPTER 4. MECHANICAL RESPONSE

Gel-derived materials have been tested and characterized mechanically. In many cases, the mechanical properties, density, and other physical properties can be explained by their fractal structure \[29, 62\]. However, fractal theory is not always applicable, in particular when either it becomes inadequate to associate a fractal range to the structure, or when, within a given fractal range, the response is not explained by classical fractal theory \[5, 32\].

In this paper it is proposed that different responses (density, scattering, mechanical response) of gel-derived materials depend on different characteristics of the connected structure (coordination and/or functionality). In Part I \[67\], density and scattering intensity are shown to depend on the mass distribution, represented by the functionality distribution. In this paper, the stress-strain behavior is shown to depend also on the connectivity of the structure, represented by the coordination number distribution. For this purpose, density and mechanical response of computer generated structures that resemble gels and Aerogels are studied.

4.2 Computer generated structures

The aggregation algorithm used to generate structures that resemble Aerogels is explained in detail in \[61\]. Structures are generated inside a simulation box of size \(L = 2a_0\lambda_{\text{max}}\), where \(2a_0\) is the size of the primary particles, and \(\lambda_{\text{max}}\) the maximum scale analyzed. The scale \(\lambda\) is in the range \(\lambda \in [1, \lambda_{\text{max}}]\).

The structure is generated by the aggregation of particles initially located randomly in a cubic lattice inside a simulation box. The periodic boundary conditions are used to treat the simulation box as a representative volume element (RVE, \[70, \S 4.1.2\]). Particles are randomly chosen to move in the lattice, thus simulating Brownian motion. The coordination number and reactivity of the colliding particles (or clusters of particles) controls whether or not colliding particles/clusters bond or not, satisfying the Einstein-Smoluchowski conditions \[4, 53\]. For high reactivities, reactions between clusters having many bonds are favored, whereas for low reactivity, reactions between clusters with few bonds are favored. Thus,
the generated structure is more compact for higher reactivities, and more linear for low reactivities, as shown in [61] and confirmed by scattering experiments in [67].

4.3 Elastic Behavior

The elastic behavior of a structure is measured subjecting it to a set of uniform strain tests. A uniform displacements filed is defined as [71]

\[ u_i = \varepsilon_0^i (x_j - x_{j0}) \]  

(4.1)

where \( x_j \) is the position vector of any point, e.g. the position of a particle, and \( x_{j0} \) is a reference point where the displacement is zero. Then, uniform strains are defined as

\[ \varepsilon_{ij} = \frac{1}{2} (u_{i,j} + u_{j,i}) = \varepsilon_0^{ij} \]  

(4.2)

where \( u_{i,j} = \partial u_i / \partial x_j \). Using contracted notation [72, (1.9)] as

\[ \alpha = \begin{cases} 
  i & \text{for } i = j \\
  9 - (i + j) & \text{for } i \neq j 
\end{cases} \]  

(4.3)

yields the strain tensor \( \varepsilon_\alpha = \varepsilon_{ij} \), the stress tensor \( \sigma_\alpha = \sigma_{ij} \), and the stiffness tensor \( C_{\alpha\beta} = C_{ijkl} \). The total energy \( U(\varepsilon) \) stored in the system is measured simulating the structure using the Finite Element Method [9]. A typical structure generated using the aggregation algorithm is shown in Figure 4.1(a) and its geometrical representation using the finite element method is shown in Figure 4.1(b).

For small strains, the total energy stored in the system can be expanded with a Taylor series [72] as follows,

\[ \frac{1}{V} U(\varepsilon) \approx \varepsilon_\alpha \left( \frac{1}{2} C_{\alpha\beta}^{ij} \right) \varepsilon_\beta \]  

(4.4)

\[ \quad \text{1The first order derivatives vanish at the equilibrium state because the exerted force at that point is zero.} \]
where $V$ is the volume of the simulation box. Note that each of the components of stiffness tensor $C_{lm}$ can be extracted using its symmetry

$$C_{lm} = \frac{1}{2} (C'_{lm} + C'_{ml}) = C_{ml}$$

and using the fact that the stiffness can be derived from the potential function $U$ as

$$C_{lm} = \frac{1}{V} \frac{\partial^2 U}{\partial \varepsilon_l \partial \varepsilon_m}$$

To measure the 21 stiffness constants of the anisotropic structure, 21 tests combining all the possible $u_{ij}^k$ are performed, thus recreating all the possible pairs $\varepsilon_\alpha \varepsilon_\beta$. From the set of energies $U^k$ obtained, the solution of the system

$$\frac{1}{V} U^k = \varepsilon_\alpha^k C_{\alpha\beta} \varepsilon_\beta^k$$

gives the values of $C_{\alpha\beta}$.

Assuming that the material is composed of a random aggregation of structures like the ones produced with the algorithm described in Section 4.2, when subjected to a load, the strain-energy is absorbed in average by each of the composing structures proportionally to
their volume. In other words, the energy density can be assumed spreads homogeneously, satisfying the equipartition energy theorem \cite{51, 73} and the Boltzmann postulate \cite{52, 74}.

Hence, the total energy is the summation of the energy of the parts as

\[
U_{\text{tot}}(\varepsilon) = \sum U(\varepsilon)
\]

\[
V_{\text{tot}} \mathcal{H}(C_{\alpha\beta}) \varepsilon_{\alpha} \varepsilon_{\beta} = \sum V^{(e)} C_{\alpha\beta}^{(e)} \varepsilon_{\alpha} \varepsilon_{\beta}
\]

(4.8)

Therefore, for \( V_{\text{tot}} = \sum V^{(e)} \), for structures with the same volume, and for \( N_s \) samples, \( N_s = V_{\text{tot}}/V^{(e)} \), the total effective stiffness matrix can be calculated as

\[
\mathcal{H}(C_{\alpha\beta}) = \frac{1}{N_s} \sum_{\varepsilon}^{N_s} C_{\alpha\beta}^{(e)}
\]

(4.9)

where \( \mathcal{H}() \) represents the proposed process of homogenization and averaging.

According to the argument posed in \cite[§4]{67}, the fractal character makes sense only for the critical percolation concentration, meaning that the mechanical response needs to be calculated at this concentration. As discussed in \cite[§4]{67}, the aggregation process can be classified into three regimes depending on the number of particles \( N \) inside the simulation box and the fractal dimension \( D \): i) if there are less particles than a critical percolation density \cite[eqs. 5 and 12]{67}, the final aggregate results in isolated clusters suspended inside the simulation box, ii) if there are just enough particles, the aggregate percolates, i.e., it spans the simulation box and connects the faces of the box, and iii) if there are more particles than the critical percolation density, the simulation produces more compact structures, with fractal dimension higher than what is should be—that is, without the limitations of the simulation box, the physical colloid would aggregate a larger cluster with critical percolation density. Thus, because of the constraint imposed by the simulation box, only at the critical percolation density the structure can aggregate with a fractal dimension congruent with the chemistry

\footnote{The Boltzmann postulate states that microstates with equal energy have the same probability of occurrence.}
of the aggregation process. Since the simulation box has fixed dimensions, an interpolation allows us to find the mechanical response at the critical percolation density.

Using the fractal character of the structure, the effective density is

$$\rho = \rho_0 \left( \frac{\xi}{2a_0} \right)^{D-3} \quad (4.10)$$

where $\xi$ is the correlation length determined by the aggregation parameters [67, §3–4], and $\rho_0$ is the density of the primary particle. Note, that the correlation length of the clusters can be written as

$$\frac{\xi}{2a_0} = \left( \frac{L}{2a_0} \right)^n \quad (4.11)$$

where $n$ is the number of times (generations) that the structure is composed of itself. Hence,

$$n = \frac{\log \frac{\xi}{2a_0}}{\log \frac{L}{2a_0}} \quad (4.12)$$

Therefore, (4.10) is rewritten as

$$\frac{\rho}{\rho_0} = n_0 \left( \frac{L}{2a_0} \right)^n \quad (4.13)$$

In a more general scheme, assuming that $n$ is an integer, at each generation $(i)$, (4.13) can be calculated from the recurrence

$$\rho^{(i+1)} = \rho^{(i)} n_0(w) \quad (4.14)$$

with $\rho^{(0)} = \rho_0$.

Using the procedure explained in Section 4.3, the stiffness can be calculated sequentially at each generation by introducing the effective stiffness calculated during the previous generation as the stiffness of the primary cluster for the current generation. This means that the effective

Formally, $n$ is not necessarily an integer.
CHAPTER 4. MECHANICAL RESPONSE

stiffness can be calculated from the proposed homogenization $H$ and recurrence as

$$C_{\alpha\beta}^{(i+1)} = H(C_{\alpha\beta}^{(i)})$$  \hfill (4.15)$$

with $C_{\alpha\beta}^{(0)} = C_{\alpha\beta0}$. In this way, the relationship between the mechanical response and density can be calculated for a correlation length several orders of magnitude larger than the simulation box, i.e. $\xi >> L$.

4.3.1 Elastic Behavior

Silica Aerogels are simulated for the conditions in Table 4.1 using amorphous silica as primary particle at molecular level. An example of the resulting structure is presented in Figure 4.1(a). For Aerogels in this work, Young modulus $E = 80\, GPa$ and Poisson’s ratio $\nu = 0.22$ is used \mbox{[75]}. The simulation is performed using ANSYS \mbox{[9]} for two cases: 1) as-gelled structures and 2) fully connected structures. In as-gelled structures, the bonds are the result of the aggregation process; not all of the potential bonds between colliding particles are formed. In fully connected structures, the missing bonds of the as-gelled structure are allowed to materialize, either by ageing or sintering.

Von Misses stress contour plots are shown in Figure 4.2 for both types of structures subjected to $\epsilon_{xy} = 1$. Note that the as-gelled structure has missing bonds that prevent the structure from effectively transferring stress, while the fully connected structure displays a more uniform state of stress as a result of the ability of the structure to transfers stress along all possible paths.

After averaging over the 32 specimens for each sample of the 42 combinations possible using the conditions listed in Table 4.1 the effective mechanical material behavior was found to be isotropic.

For reactivity $w = 1$, effective Young modulus and effective Poisson ratio$^4$ as a function of the number of particles are presented in Figures 4.3 and 4.4 respectively. As-gelled structures

$^4E = 1/S_{11}$ and $\nu = -S_{12}/S_{11}$, where $S = C^{-1}$ is the compliance tensor.
### Table 4.1: Parametric test conditions

<table>
<thead>
<tr>
<th>$N$</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
<th>1600</th>
<th>1728</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.8</td>
<td>1</td>
<td>1.2</td>
</tr>
<tr>
<td>$\lambda_{\text{max}}$</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$N$ is the number of particles in the simulation box of size $L = 2a_0\lambda_{\text{max}}$, and $w$ is the reactivity.

**Figure 4.2**: Von Misses plot for structure as gelled and fully connected structure for $w = 1.0$ and $N = 400$. 

are produced by the aggregation algorithm described in [61]. Fully connected structures are then obtained by simulated sintering and/or aging.

Mechanical properties are shown in Figures 4.3 and 4.4 for different generations. Note that the fully connected structures present a higher Young modulus and Poisson’s ratio compared to the structure as gelled for the same conditions. Also, in fully connected structures, both Young modulus and Poisson’s ratio decrease with decreasing number of particles. However, in as-gelled structures, only the Young modulus decreases with decreasing number of particles.

With each generation, the Young modulus reduces a constant amount in logarithmic scale (Figure 4.3). This suggests that the Young modulus follows a power law as a function of generations. This is not the case for the Poisson’s ratio. In addition, note that for fully connected structures, as the number of particles reaches the limit in the simulation box, i.e. 1728 particles for $L = 12$, the properties approach the properties of the constituent material. On the contrary, for the structures as gelled, this trend is not observed because there are

---

5Sintering and aging are simulated by adding the missing bonds between neighboring particles.
Figure 4.3: Effective Young modulus for the structures as a function of the number of particles.

Figure 4.4: Effective Poisson ratio for the structures as a function of the number of particles.
missing bonds. Aging/sintering lets the structure form all its bonds and the properties of the primary particle are recovered. The effect of aging/sintering is seen over the entire range of $N$ in Figure 4.3.

As $N \to 0$, the stiffness should approach zero. For very low values of $N$, the splines used to fit the data in Figure 4.3 do not capture such trend exactly, but the computational evaluation of the stiffness does captures this effect as the homogenization over an empty simulation box produces zero stiffness.

Density $\rho$ vs. the generation number $n$ are shown in Figure 4.5 for different reactivities. It can be seen that density reduced with each successive generation. This is because the effective density of a cluster is less than the density of the primary particle, and each successive generation uses the cluster of the previous generation as primary particle. Also, for higher reactivities, the effective density is higher as the fractal dimension is higher [67].
As the density reduces with generations, so do the mechanical properties, as shown in Figures 4.3 and 4.6. Of course this effect is more notable when the number of particles is small because then the effective density at each generation is much smaller than the density of the primary particle. Recursion just magnifies this effect, as it is clearly seen on the left of Figures 4.3 and 4.6.

Another observation can be made regarding percolation. In Figure 4.3, the vertical line indicates the critical number of particles in the simulation box for which the structure percolates, as revealed by scattering simulation in [67]. The structures to the left of the vertical line do not percolate, and present an accentuated decrease in the Young modulus compared to the structures that percolate. The mechanical properties of the fully connected structures reduce faster than the as-gelled ones because un-percolated structures are disconnected, and thus inefficient to generate mechanical properties. Therefore, in this range, aged/sintered structures behave closer to as-gelled structures, the later having even more disconnected structure. On the contrary, to the right of the critical percolation density line, percolated aged/sintered structures are more efficient in carrying load, with higher mechanical properties, whereas as-gelled structures have some amount of disconnected structure.

In contrast to the Young modulus trend observed, Poisson’s ratio is not affected appreciably by percolation, as shown in Figure 4.4, since the strain coupling between two directions is a local effect. Percolation, which is not a local effect, measures the connectivity that the structure generates between faces of the simulation box.

Although not displayed in Figure 4.3 and 4.4, similar trends were observed for other values of reactivity.

Effective Young modulus and effective Poisson’s ratio are shown in Figures 4.6 and 4.7 as a function of reactivity, at the critical percolation concentration. For all reactivities, there is a slight improvement of the Young modulus and Poisson’s ratio as the reactivity increases. Note how the properties of the fully connected structure are higher than the properties of the as-gelled structure, for every generation.
Figure 4.6: Effective Young modulus as a function of the reactivity, at critical percolation concentration.

Figure 4.7: Effective Poisson ratio as a function of the reactivity for the critical percolation concentration.
In Figures 4.8 and 4.9, effective Young modulus and effective Poisson’s ratio are presented as a function of effective density. The power law relationship between effective Young modulus and effective density is clearly seen in Figure 4.8. Young modulus and Poisson’s ratio are clearly higher for fully connected structures than for as-gelled structures. Also, for fully connected structures, Young modulus and Poisson’s ratio are higher as the reactivity increases at constant density, whereas the opposite is observed for as-gelled structures.

4.3.2 Discussion

Effective Young modulus $E$ of the as-gelled and fully connected structures as a function of effective density $\rho$, shown in Figure 4.9, can be represented by the model equation

$$\frac{E}{E_0} = \left(\frac{\rho}{\rho_0}\right)^{b_E}$$

(4.16)

Values of the scaling exponent $b_E$ as a function of reactivity are presented in Figure 4.10. Similarly to [29, 45], the scaling exponent is found in the range 2–4. It is clear that the connectivity is crucial for the mechanical response at a given mass distribution. In
other words, a parameter measuring the degree of connectivity between particles is required to describe the mechanical properties. Such parameter decreases when the connectivity increases due to ageing/sintering.

In Figure 4.10, the change in power law exponent $b_E$ for as-gelled structures with reactivity $w > 1$ is explained by the change of the type of favored reactions during the formation of bonds that affect elastic response, as well as by how compact is the structure for varying density. The coordination number changes in a non trivial way with the reactivity [61]. As the reactivity grows, the number of isolated particles increases, the number of elbows and linear structures reduces, and the number of particles with three bonds reaches a maximum around $w = 1$. Particles with higher coordination also grow. The increase in isolated particles makes the structure less efficient in carrying load. The reduction in elbows and linear structures also contributes to lowering the mechanical properties, since the structure would carry load more efficiently if more bonds were loaded. However, the improvement in the mechanical response competes with the increasing density brought about by increasing reactivity. As a consequence, the mechanical properties improve at a lower pace with the reactivity in as-gelled structures compared to the density, which leads to an increase of $b_E$ in Figure 4.10.
On the contrary, for fully connected structures, mechanical properties improve at a similar pace to density.

For fully connected structures, the creation of the missing bonds reinforces the structure, yielding lower scaling exponents compared to as-gelled structures. For aged/sintered structures, the behavior is no longer determined by the coordination number distribution but rather by the functionality distribution, since the coordination number distribution approaches the functionality distribution \[61\]. Even with similar trends in the way how the functionality changes with reactivity, the changes are smoother. The visible effect on the mechanical behavior must be smoother as well as evidenced in Figures 4.8, 4.9, and 4.10.

In principle, a direct correlation between mechanical response and scattering intensity seems not to exist. The mechanical response depends directly on the connectivity of the structure whereas scattering intensity depends directly on mass distribution. However, for fully connected structures, as the coordination number distribution approaches the functionality distribution, mechanical properties are seen to depend on functionality, thus mass distribution. At this stage, scattering intensity and mechanical behavior of fully connected structures become directly correlated. For aged/sintered structures, all that is needed in
order to predict their mechanical response is available in their scattering response.

Future work may entail using the percentage of missing bonds between colliding particles to correlate the coordination to the functionality distribution. In doing so, one would be able to predict the mechanical behavior of as-gelled structures using their scattering response. So far, the use of scattering experiments allows us to partially predict the mechanical response of sol-gel structures by the measuring their correlation length and fractal dimension. However, it is emphasized that mechanical behavior cannot be fully derived from scattering experiments unless connectivity and mass distribution are correlated. The exception to this observation are sintered and/or aged structures where the coordination distribution and the functionality distribution coincide.

4.4 Conclusions

The mechanical properties of the sol-gel vary, depending on the aggregation process, as a function of reactivity and amount of precursor, thus determining the connectivity and correlation length of the structure. In this work, the two limiting cases where studied; the as-gelled structure with the lowest mechanical properties and the fully connected structure with the highest mechanical properties. Additional post-processing to the sol-gel such as aging or sintering, allowing them to form the missing bonds in the structure, produces a structure where scattering intensity and mechanical response are fully correlated to each other. As the scattering intensity, the mechanical response reveals a fractal character. Thus, the calculation of mechanical response for a large structure with a large amount of features can be performed using a recursive procedure. The variation of mechanical properties is a complex function of the coordination number distribution. However, qualitative insight explains how mechanical properties improve with reduction of reactivity, in particular for as-gelled structures, due to the reduction of inefficient particles acting as dead-ends. Making a mechanically efficient structure is then a compromise between the mechanical improvement
due to structure arrangement, improved connectivity, and density.
Chapter 5

Effective properties of hierarchical structures

This chapter is an article in preparation: F.A. Campo and E.J. Barbero. Multifractal Multidimensional Multiscaling of effective properties for hierarchical structures.

Abstract

A novel formalism for the calculation of effective properties of hierarchical structures is proposed. It consists on the recursive calculation of the effective multidimensional properties over a larger volume determined by a stretch tensor. For materials with a fractal character, the recurrence becomes repetitive and characterized by a set of scaling exponents, as well as a set of eigenvectors proper of each individual property. If the material structure is not fractal, a multifractal approach can be still implemented where the property scaling exponents change with the scale in the recurrence. This process is known as multiscaling. Mass distributions are reconstructed based on the results of a particle aggregation algorithm. The multiscaling formalism is illustrated for uncoupled properties. Relationships between properties are evidenced as materials share the same material structure. Finally, some applications of the formalism are presented as the prediction of Young modulus and thermal conductivity from scattering experiments, and the explanation of power law behavior in sintered materials.

5.1 Introduction

Prediction of material properties plays an important role in the development of devices. Such design requires a concurrent engineering practice simultaneously incorporating geometry and material design as it is the case for composite materials [72, 70].

Modeling of large structures is performed well with the use of the finite element method where the input consists of geometry, loading conditions, and constitutive material properties...
Using homogenization techniques, the constitutive properties of materials with an internal structure can be calculated as it is the case of fabrics and materials with features at a single scale. However, materials as wood, spider silk, aerogels, etc., present features along a broad scale range, making homogenization cumbersome due to the large amount of information required.

There is still lack of understanding on the role that the structure plays at different scales in hierarchical materials. Nonetheless, if the analysis is kept within a narrow scale range, different methods have proven to be effective, such as Density Functional Theory (DFT) at nanoscales, atomistic modeling and molecular dynamics for several millions of atoms, and continuum modeling at macroscales. This paper presents an alternative way to reduce the amount of information required to calculate the effective properties of a material over a broad scale range.

For this, materials are not observed in linear scale but rather in logarithmic scale. It is proposed to describe hierarchical materials such as sol-gels using recursive functions in a similar procedure as periodic materials such as crystals are described using periodic functions. In particular, Iterated Function Systems (IFS) are used for recursive functions.

The formalism consists of two parts: 1) finding the IFS that describes the material hierarchically, and 2) using the IFS for the calculation of the effective properties recursively, known as multiscaling. The former part of the formalism is inspired by an extended fractal image compression formalism to three dimensional structures, whereas the latter part is based on homogenization.

The IFS needs the definition of scaling factors which quantify the scale at which the structure is observed. The scaling factors together with the homogenization procedure, conceptualized by an operator, allows stating the Principle of Similarity (PoS), which links properties with structure (see section 5.3). The PoS yield property scaling exponents which characterize the properties as a function of the scale. When the property is the mass, the scal-

\[\text{[72]}\].
ing exponent is called Fractal Dimension\textsuperscript{81, 82}. As a consequence, a power law relationship is predicted for properties, function of the property scaling exponents.

The PoS extended for an infinite number of scale ranges, wherein the scaling exponents vary with the scale, yields to the multifractal description of hierarchical structure as presented in section 5.4.2. The effective properties can be recovered exactly by using the multiscaling procedure presented in section 5.4.3. Since different properties of a material share the unique material structure geometry, the different properties are correlated by relationships between the property scaling exponents, in particular correlated to the fractal dimension.

To conclude, a set of applications and examples are presented in section 5.4 where the properties of the materials are calculated using the proposed formalism.

### 5.2 Hierarchical Description of the Material

The hierarchical description of a material consists of finding a set mapping functions $\mathcal{T}$, that applied repeatedly to a metric space approximate the structure $\mathcal{S}$ as proven by the Collage Theorem\textsuperscript{69, 83, 84}.

Let’s assume that the structure to analyze is contained in a box $\square$ defined by the vectors $L_{ij}$. $\mathcal{S}_{\square}$ denotes the part of the structure $\mathcal{S}$ inside $\square$. Additionally, the size of the smallest discernible box $\Box$ is defined by $\Delta L_{kl}$. Thus, a dimensionless scaling factor $\lambda_{ijkl}$ can be defined as

$$L_{ij} = \lambda_{ijkl}\Delta L_{kl} \quad (5.1)$$

Since $\lambda_{ijkl}$ scales the dimensions of $\Box$ to the dimensions of $\square$, the scaling procedure can be extended by defining intermediate steps or generations as follows:
\[ \square^{(0)} = \square \]
\[ \square^{(n_i+1)} = \square^{(n_i)} \]
\[ \square^{(n_i)} = T^{(n_i)}(\square^{(n_i)}) \]
\[ \square^{(n)} = \square \]  

(5.2)

where \( n_i \) is the generation, and \( n \) is the total number of generations used to describe the structure. \( T \) denotes the mapping which in terms of the boxes dimensions yields to the definition of a set of scaling factors \( \lambda_{ijkl}^{(n_i)} \) as follows,

\[ L_{ij}^{(0)} = \Delta L_{kl} \]  

(5.3)

\[ L_{ij}^{(m+1)} = \lambda_{ijkl}^{(m)} L_{kl}^{(m)} \]  

(5.4)

Hence, the structure \( S \) can be hierarchically described as

\[ S \approx S_{\square} = S_{\square^{(n)}} = S_{T^{(n)}}(\square^{(n)}) = S_{T^{(n)}}(\square^{(n-1)}) = S_{T^{(n)} \circ T^{(n-1)}}(\square^{(n-2)}) = \ldots = S_{T^{(n)} \circ T^{(n-1)} \circ \ldots \circ T^{(1)}}(\square^{(0)}) \]

\[ S \approx S_{T^{(n)} \circ T^{(n-1)} \circ \ldots \circ T^{(1)}}(\square) \]  

(5.5)

For fractal structures, (5.5) becomes convenient as \( T^{(n_i)} \) becomes the same regardless
CHAPTER 5. EFFECTIVE PROPERTIES OF HIERARCHICAL STRUCTURES

of the generation (or scale). However, as proven for fractal image compression algorithms [69, 83, 84, 68], the hierarchical description approximates any structure requiring a larger number of generations as well as the change of $T(n_i)$ as a function of scale.

5.2.1 Reconstruction of the Hierarchical Structure of Aerogels

Using (5.5), the structures of silica aerogels can be reproduced. This procedure is named reconstruction. The reverse procedure used to get the necessary parameters for the reconstruction of the hierarchical structure is named deconstruction.

[61] presents a hierarchical aggregation model, where, due to the characteristics of the process, a fractal structure is found to aggregate at a critical density [67, 58]. From the aggregation model, different mappings are found as a function of the processing conditions identified by the density, $\rho/\rho_0 = N/\lambda^3$, and reactivity, $w$, the later containing the chemical information of the aggregation process. The set of possible mappings is called a pool. The probability of occurrence of one mapping is the same for all mappings.

The structure of neutral silica aerogel is reconstructed using (5.2) as follows. An homogeneous structure with $\rho/\rho_0 = 1$ is chosen as $S_{\Box}$. The first generation is obtained by applying a transformation chosen randomly from the pool corresponding to the critical density $\rho_{\text{crit}}/\rho_0$ ($N \approx 400$ and $\lambda = 12$), and a neutral reactivity $w = 1$ to $\Box$. The transformations replace the box to be transformed with the mapping. If the box is empty, the mapping produces an empty box as well. On the contrary, if the box has density $\rho/\rho_0 = 1$ as depicted in figure 5.1(a), it replaces the box with the mapping containing pores as depicted in figure 5.1(b).

A second generation is obtained by applying randomly chosen transformations from the pool corresponding to the critical density and neutral reactivity to the boxes with $\rho/\rho_0 = 1$ as depicted in figure 5.1(c). A third generation is obtained repeating the same procedure used for the second generation applied to the second generation structure as depicted in figure 5.1(d).

Note that at each generation, the overall effective density is reduced by a factor $N/\lambda^3$ as pores are created subsequently at smaller scales. This means that at the third generation,
the smallest box with $\rho/\rho_0 = 1$ has a size $2a_0 = 1/12^3 L$, where $L$ is the size of $□$. Inverting, the size of $□$ is $L = (2a_0) \cdot 12^3$. The density of $□$ is unity, and the effective density, corresponding to the density of $□$, is $1/((12^3)^3)$ times the density of $□$.

In the next section, the hierarchical description of the material response is proposed by stating the Principle of Similarity (PoS). Note that the properties of the material are the measurement of the material response. Thus, the PoS relates properties to the hierarchical description of the material.
5.3 Principle of Similarity (PoS)

The *Principle of Similarity* (PoS) is postulated as follows. For $S_\Box$ made of primary particles $S_\Box$, an effective property $P_\Box$ and the property $P_\Box$ are related by

$$P_\Box = H(\lambda_{ijkl})P_\Box$$  \hspace{1cm} (5.6)

$H$ is the homogenization operator of the property $P_\Box$, which only depends on the structure. $H$ consists of calculating the effective property of the material inside $\Box$ as the result of the interaction between all particles $\Box$ characterized by the property $P_\Box$. $P$ can be a higher order tensor, $P = P_{k1k2...kn}$.

An example of the use of the PoS for the elastic properties of silica aerogels is presented in [66]. In [66], (5.6) was calculated directly using ANSYS[9]. However, the separation of $P_\Box$ into the multiplication of $H(\lambda_{ijkl})$ and $P_\Box$ has a mathematical structure that yields to power laws of the properties as a function of the scale.

Without loss of generality, if it is possible to find a set of eigenvectors, $P_{Hl}$, and corresponding eigenvalues $\tilde{h}_l$ for $H$, the properties can be written as a linear combination of the eigenvectors as follows,

\begin{align*}
P_\Box &= \sum_l \alpha_\Box l P_{Hl} \\
P_\Box &= \sum_l \alpha_\Box l P_{Hl}
\end{align*}

Thus, the PoS stated in (5.6) can be rewritten as a set of scalar functions,

$$\alpha_\Box m = \alpha_\Box m \tilde{h}_m$$  \hspace{1cm} (5.9)

Using (5.1), one notes that the functional dependencies of $\alpha_\Box m$, $\alpha_\Box m$, and $\tilde{h}_m$ are $\alpha_\Box m =$

\footnote{The effective properties are apparent properties of a region when tested on the boundary of the region.}
$\alpha_m(L_{ij})$, $\alpha_m = \alpha_m(\Delta L_{kl})$, and $\tilde{h}_m = \tilde{h}_m(\lambda_{ijkl})$. Then, the partial derivatives of $\alpha_m$ over $\lambda_{ijkl}$ and $\Delta L_{kl}$ yield

$$\frac{\partial \alpha_m}{\partial \lambda_{ijkl}} = \sum_{i'j'} \frac{\partial \alpha_m}{\partial L_{i'j'}} \frac{\partial L_{i'j'}}{\partial \lambda_{ijkl}} = \frac{\partial \alpha_m}{\partial L_{ij}} \Delta L_{kl}$$

$$\Rightarrow \frac{\partial \alpha_m}{\partial L_{ij}} \Delta L_{kl} = \frac{\partial \tilde{h}_m}{\partial \lambda_{ijkl}} \alpha_m$$ (5.10)

and

$$\frac{\partial \alpha_m}{\partial \Delta L_{kl}} = \sum_{ij} \frac{\partial \alpha_m}{\partial L_{ij}} \frac{\partial L_{ij}}{\partial \Delta L_{kl}} = \sum_{ij} \frac{\partial \alpha_m}{\partial L_{ij}} \lambda_{ijkl}$$

$$\Rightarrow \sum_{ij} \frac{\partial \alpha_m}{\partial L_{ij}} \lambda_{ijkl} = \tilde{h}_m \frac{\partial \alpha_m}{\partial \Delta L_{kl}}$$ (5.11)

Multiplying (5.11) by $\Delta L_{kl}$, and summing over $kl$,

$$\sum_{kl} \sum_{ij} \frac{\partial \alpha_m}{\partial L_{ij}} \lambda_{ijkl} \Delta L_{kl} = \sum_{kl} \tilde{h}_m \frac{\partial \alpha_m}{\partial \Delta L_{kl}} \Delta L_{kl}$$ (5.12)

and replacing (5.10) into (5.12),

$$\sum_{kl} \left( \sum_{ij} \frac{\partial \tilde{h}_m}{\partial \lambda_{ijkl}} \lambda_{ijkl} \right) \alpha_m - \tilde{h}_m \frac{\partial \alpha_m}{\partial \Delta L_{kl}} \Delta L_{kl} = 0$$ (5.13)

Thus, separating the functions depending on $\lambda_{ijkl}$ from the functions depending on $\Delta L_{kl}$,

$$\frac{1}{\tilde{h}_m} \sum_{ij} \frac{\partial \tilde{h}_m}{\partial \lambda_{ijkl}} \lambda_{ijkl} = \frac{1}{\alpha_m} \frac{\partial \alpha_m}{\partial \Delta L_{kl}} \Delta L_{kl} = D^m_{kl}$$ (5.14)
Since the left and middle part of (5.14) are functions of different variables, the equality is only satisfied if both parts are equal to a constant $D^{m}_{kl}$ which are the Property Scaling Exponents.

Thus,

$$\frac{1}{\alpha^{m}_{kl}} \frac{\partial \alpha^{m}_{kl}}{\partial \Delta L_{kl}} \Delta L_{kl} = D^{m}_{kl}$$

(5.15)

and

$$\frac{1}{h^{m}_{m}} \sum_{ij} \frac{\partial h^{m}_{m}}{\partial \lambda_{ijkl}} \lambda_{ijkl} = D^{m}_{kl}$$

(5.16)

(5.15) and (5.16) allow the determination of the Property Scaling Exponents.

Solving (5.15),

$$\alpha^{m}_{kl} = \prod_{kl} \Delta L^{m}_{kl}$$

(5.17)

and solving (5.16),

$$h^{m}_{m} = \prod_{kl} \sum_{ij} \lambda^{m}_{ijkl}$$

(5.18)

Putting (5.7), (5.8), (5.17), and (5.18) into (5.6),

$$\mathcal{P}_{\square} = \sum_{m} \left( \mathcal{P}_{\square} \cdot \mathcal{P}_{\square m} \right) \mathcal{P}_{\square m} \prod_{kl} \sum_{ij} \lambda^{m}_{ijkl}$$

(5.19)

(5.19) defines the mapping $\mathcal{P}_{\square} \rightarrow \mathcal{P}_{\square}$.

Recursively making

$$\mathcal{P}_{\square}^{(n)} \rightarrow \mathcal{P}_{\square}^{(n+1)}$$

(5.20)

the property $\mathcal{P}$ can be calculated to the desired scale by setting $\mathcal{P}_{\square}^{(n+1)}$ to $\mathcal{P}_{\square}^{(n)}$, and
calculating $P_{\Box(n_i+1)}$ using (5.6) or (5.19).

A scheme of the procedure for a mechanical perturbation is presented in figure 5.2. When the structure is subjected to a mechanical perturbation, the structure transmits mechanical perturbations to the elements at a smaller scale as the result of the arrangement and connectivity of the structure. The perturbation on the smaller elements transmit a mechanical perturbation further to elements at smaller scales. This process continues until the smallest elements are perturbed.

The information about how the material structure $S_{\Box(n_i)}$ and the way how its parts interact to give a determined response is contained in $D_{kl}^{m(n_i)}$ and the eigenvectors $P_{Hm}^{(n_i)}$. For material structures with self-affinity, i.e., structures containing the same features at different scales as fractal structures, $D_{kl}^{m(n_i)}$ and $P_{Hm}^{(n_i)}$ do not change with the scale, meaning that transmission of the perturbations from one scale to another is invariant. If the material structure is not self-affine, the hierarchical approach can be still used, but the way how the perturbation is transmitted varies with the scale. Thus, the pairs $\left(\Box^{(n_i)}, (D_{kl}^{(n_i)}, P_{H}^{(n_i)})\right)$ describe the structure’s behavior hierarchically as a multifractal where the property scaling exponents vary with the scale.

In the next section, the solution of (5.19) is derived when the properties are uncoupled.
5.4 Applications to Uncoupled Properties

Properties such as density $\rho$, Young modulus $E$, thermal conductivity $\kappa_t$, and so on, can be calculated by simplifying (5.19) to one dimension. Making $i = j = k = l = m = 1$, the math is simplified with a reduction of formalism generality. For example, coupled properties such as the Poisson ratio $\nu$ require the use of (5.19) as shown in [66].

Rewriting (5.19), yields (5.33). For clarity, the derivation is repeated for the uncoupled case.

5.4.1 Principle of Similarity (PoS)

The proportionality between $\mathcal{P}(\lambda r_0)$ and $\mathcal{P}(r_0)$ is given by a function $g(\lambda)$ only dependent on the scaling factor $\lambda = r/r_0$.

Mathematically speaking,

$$\mathcal{P}(r) = g(\lambda)\mathcal{P}(r_0) \quad (5.21)$$

For example, two materials with exactly the same structure, but two primary particles of different densities, one the double of the other one, will have a density the former the double of the other one. In other words, a property $\mathcal{P}(\lambda r_0)$ that varies with scale $\lambda$ and the primary particle size $r_0$ is separable into $g(\lambda)\mathcal{P}(r_0)$.

For $\lambda \in [1, \lambda_{\text{max}}]$, the measured property $\mathcal{P}(r)$ follows a power law of the scale. To see this, note that $r = \lambda r_0$. Then, (5.21) can be rewritten as

$$\mathcal{P}(\lambda r_0) = g(\lambda)\mathcal{P}(r_0) \quad (5.22)$$

Taking a partial derivative\(^3\) of (5.22) with respect to $\lambda$,

\(^3\)To measure how much the property $\mathcal{P}$ changes with respect to the scaling $\lambda$ or the change of the primary particle size $r_0$.\n
\[
\frac{\partial P}{\partial \lambda} = r_0 \frac{\partial P}{\partial (\lambda r_0)} = \frac{\partial g}{\partial \lambda} P(r_0)
\]  
(5.23)

and taking a partial derivative of (5.22) with respect to \( r_0 \),

\[
\frac{\partial P}{\partial r_0} = \lambda \frac{\partial P}{\partial (\lambda r_0)} = g(\lambda) \frac{\partial P}{\partial r_0}
\]  
(5.24)

Taking the ratio between (5.23) and (5.24), yields

\[
\frac{\frac{\partial P}{\partial \lambda}}{\frac{\partial P}{\partial r_0}} = \frac{r_0}{\lambda} = \frac{\frac{\partial g}{\partial \lambda} P(r_0)}{g(\lambda) \frac{\partial P}{\partial r_0}}
\]  
(5.25)

Rearranging the second equality,

\[
\frac{r_0}{P(r_0)} \frac{\partial P}{\partial r_0} = \lambda \frac{\partial g}{g(\lambda)} \frac{\partial \lambda}{\partial r_0} = D_P
\]  
(5.26)

where \( D_P \) is a constant because the first term is a function of \( r_0 \), whereas the second is a function of \( \lambda \), and noting that the only way that two functions with different arguments satisfy an equality is when both are equal to a constant. Then,

\[
\frac{\partial g}{g} \frac{1}{D_P} = \frac{\partial \lambda}{\lambda}
\]  
(5.27)

\[
\frac{\partial P}{P} \frac{1}{D_P} = \frac{\partial r_0}{r_0}
\]  
(5.28)

Integrating (5.27) yields a power law on the scale,

\[
g = g_0 \left( \frac{\lambda}{\lambda_0} \right)^{D_P}
\]  
(5.29)

Since \( P(r) = P(r_0) \) when \( \lambda = 1 \), it is concluded that \( \lambda_0 = 1 \) and \( g_0 = 1 \), so (5.29) can be rewritten as
\[ g = \lambda^D \]  

(5.30)

Integrating (5.28) yields

\[ P = P_0 \left( \frac{r'}{r_0} \right)^D \]  

(5.31)

Since \( P = P(r) \) when \( r'_0 = r_0 \), it is concluded that \( P_0 = P(r_0) \) and \( r'_0 = \lambda r_0 \), so (5.31) can be rewritten as

\[ P = P(r_0) \lambda^D \]  

(5.32)

Using (5.30) or (5.32), (5.22) can be rewritten as

\[ P(\lambda r_0) = \lambda^D P(r_0) \]  

(5.33)

Thus, (5.33) allows us to define the *Property Scaling Exponent* \( D \). There are as many \( D \)'s as there are properties and they can be related ones as function of others. For example, the density scales with a power law \( D_\rho = D_N - 3 \) where \( D_N \) is the scaling exponent of for the number of primary particles \( N \), also called *Fractal Dimension* [82, 84]. Thus, the unknown function \( g(\lambda) \) is a power function of the scale characterized by the property scaling exponent.

Note that the scale range where \( D_P \) is constant, \([1, \lambda_{max}]\), can be chosen as small as desired. When the range \([1, \lambda_{max}]\) is made infinitesimally small, the list of scaling exponents required to describe the structure becomes large, and the property scaling exponent becomes a function of the scale.

Structures described by many scaling exponents are known as *multifractals* [85, 86, 87, 58, 58, 84]. Thus, it becomes convenient to use a *Property Scaling Exponent Function* to described the scaling exponents. The later, evaluated at the desired scale gives the *Local Property Scaling Exponent*.

\[ ^4 \text{This result also appears in renormalization theory and homogeneous functions}^{[52]} \]
5.4.2 Local Property Scaling Exponents

The definition of scaling exponents can be generalized for each size $r$, or scale $\lambda$, defining a Local Property Scaling Exponent for the property (LPSE). First, note that the effective property at scale $\lambda$, is $P(r)$. Assuming an infinitesimal scale range where (5.33) applies,

$$
 r + dr = \lambda r_0 + d\lambda r_0 \\
 = \lambda r_0 + \frac{d\lambda}{\lambda} \lambda r_0 \\
 = \lambda r_0 (1 + \frac{d\lambda}{\lambda}) \\
 = r\lambda' 
$$

(5.34)

The effective property at $r + dr$ (scale $\lambda + d\lambda$) can be calculated replacing $r_0 \rightarrow r$ and $\lambda \rightarrow \lambda'$ in (5.33). Thus, the LPSE can be defined as the property scaling exponent that characterizes the scaling of $P$ from $r$ to $r + dr$. Hence, the change in $P$ can be calculated as follows,

$$
 P(r + dr) = (1 + \frac{d\lambda}{\lambda})^{D_p} P(r) \\
 - P(r) = (1)^{D_p} P(r) 
$$

(5.35)

$$
 \frac{dP}{dr} dr = P(r) D_p \frac{d\lambda}{\lambda} 
$$

Since $\frac{dx}{x} = d\ln x$, the LPSE can be solved and written as,

$$
 D_p = D_p(\lambda) = \frac{d\ln P}{d\ln \lambda} 
$$

(5.36)

Since $r_0$ is known,

$$
 d\ln \lambda = \frac{d\lambda}{\lambda} = \frac{dr}{r} = d\ln r 
$$

(5.37)

Therefore, the definition of the LPSE, $D_p$, presented in (5.36) can be rewritten as
\[ D_P = D_P(r) = \frac{d \ln \mathcal{P}}{d \ln r} \]  

\( D_P \) defines how the logarithmic property \( \ln \mathcal{P} \) changes as a function of the logarithmic scale \( \ln \lambda \).

### 5.4.3 Multiscaling with the Principle of Similarity

(5.38) is calculated at \( r \), thus, the value of \( D_P \) is locally constant within \( (r, r + dr] \). Then, there is a function \( D_P(\lambda) \) that is defined over the scale range characterizing the structure, and the effective properties at each scale are the result of solving (5.38).

Describing the material by \( n \) scale ranges \( \Delta^{(i)} = (r_{i-1}, r_i] \), wherein the LSE \( D_P^{(i)} \) is constant, allows us to define the local scaling factor \( \lambda_i(r) \) as

\[
\lambda_i(r) = \begin{cases} 
1 & \text{for } r < r_{i-1} \\
\frac{r}{r_{i-1}} & \text{for } r_{i-1} < r \leq r_i \\
\frac{r_i}{r_{i-1}} & \text{for } r \geq r_i
\end{cases}
\]  

(5.39)

Multiscaling of \( \mathcal{P} \) can be written as

\[
\mathcal{P}(r) = \mathcal{P}(r_0) \prod_{i=1}^{n} \lambda_i^{D_P^{(i)}}(r) \]  

(5.40)

Using \( \prod e^{a_i} = e^{\sum a_i} \), and using \( e^{\ln \lambda_i} = \lambda_i \), (5.40) can be rewritten as

\[
\mathcal{P}(r) = \mathcal{P}(r_0) e^{\sum_{i=1}^{n} D_P^{(i)} \ln \lambda_i(r)} \]  

(5.41)

Only the scaling factors \( \lambda_i(r) = \{\lambda_l(r) : l = \{1, \ldots, k\}\} \), where \( k \) is such that \( r > r_l \), are relevant. The terms \( \ln \lambda_i(r) \) do not contribute to the summation in (5.41) for \( i > k \) because \( \lambda_i(r) = \ln 1 = 0 \). Thus, the summation finishes at \( i = k \).
Formally, the summation for infinitesimal increments becomes an integral as follows,

\[ P(r) = P(r_0) e^{\int_{\ln r_0}^{\ln r} \ln \lambda = 0} \int P(\lambda) d\ln \lambda \]

\[ (5.42) \]

### 5.4.4 Constant Fractal Dimension

When the fractal dimension \( D_N(r) = D_1 \) is constant, one can calculate the number of particles \( N \) and particle density \( \rho \) as a function of the scale.

Using (5.42) to calculate \( N \),

\[ N(r) = N_0 e^{D_1 \int_{r_0}^{r} \ln r'} = N_0 e^{D_1 \ln (r/r_0)} = N_0 \left( \frac{r}{r_0} \right)^{D_1} \]

\[ (5.43) \]

For \( \rho = N/r^3 \) and \( \rho_0 = N_0/r_0^3 \), the density follows,

\[ \rho(r) = \rho_0 \left( \frac{r}{r_0} \right)^{D_1-3} \]

\[ (5.44) \]

As depicted in the Figure 5.3, the density changes with the scale for different fractal dimensions. As the fractal dimension is lower, the density approaches zero faster. For \( D = 3 \), the density is the one corresponding to a dense object, which is invariant to scale.
**5.4.5 Variable Fractal Dimension**

Let’s consider a material that is fully dense, $D_N = 3$, for $r < r_0$. Then, the material becomes porous with fractal dimension $D(r) = D_1 < 3$ for $r_0 \leq r < r_1$. Later, the fractal dimension drops to $D_N(r) = D_2 < D_1$ for $r_1 \leq r < r_2$. Finally, the fractal dimension drops to $D_N = 0$, the dimension of an isolated. Then,

$$D_N(r) = \begin{cases} 
3 & \text{for } r \leq r_0 \\
D_1 & \text{for } r_0 \leq r \leq r_1 \\
D_2 & \text{for } r_1 \leq r \leq r_2 \\
0 & \text{for } r_3 \leq r
\end{cases} \quad (5.45)$$

Next, (5.42) is used to calculate the density. Integrating into the proper limits,

$$\frac{\rho}{\rho_0} = \begin{cases} 
1 & \text{for } r \leq r_0 \\
\left(\frac{r}{r_0}\right)^{D_1-3} & \text{for } r_0 \leq r \leq r_1 \\
\left(\frac{r_1}{r_0}\right)^{D_1-3}\left(\frac{r}{r_1}\right)^{D_2-3} & \text{for } r_1 \leq r \leq r_2 \\
\left(\frac{r_1}{r_0}\right)^{D_1-3}\left(\frac{r_2}{r_1}\right)^{D_2-3}\left(\frac{r}{r_2}\right)^{-3} & \text{for } r_3 \leq r
\end{cases} \quad (5.46)$$

As an example, the calculated density is depicted in Figure 5.4 for $D_1 = 2.5$, $D_2 = 1.5$, $r_0 = 1$, $r_1 = 10$, $r_2 = 100$, $r_3 = 1000$.

**5.4.6 Experimental Fractal Dimension**

It has been shown experimentally [28] that several (but not all) Aerogels have pair correlation functions (PCF) $g(r)$ that are compliant with
Figure 5.4: Effective density as a function of scale.

Figure 5.5: Pair correlation function $g(r)$ and fractal dimension of an aerogel. The domain of the plot goes to a value of $r = 2\xi$.

$$g(r) = A\phi^2 e^{-r/\xi} \left(\frac{r}{\xi}\right)^D ; \quad \phi = \left(\frac{\xi}{2a_0}\right)^{D-3}$$

with $A = 1.8 \pm 0.13$, $a_0 \approx 0.4[\text{nm}]$, $\xi = 6 \times 10^{-6}[\text{m}]$, and $D = 2.4$

(5.47)

The Pair Correlation Function is proportional to the density as function of the distance from a selected particle. Thus, (5.38) is applied to calculate the fractal dimension from the PCF as presented in figure 5.5. As the material becomes less dense, the fractal dimension
decreases from 2.4 to a value close to 0.

For a value of $r$ that approaches the characteristic size $\xi$ of the cluster [66], also known as correlation length, the fractal dimension is $D_N \approx 1.5$. Note that the fractal dimension of a linear structure, like wires, is one, and the fractal dimension for an isolated point is zero. The fractal dimension of a flat surface is two, and as crumpling a sheet of paper, crumpling a surface yields to the growth of the fractal dimension from 2 to 3 as the structure becomes more compact. Thus, the Aerogel structure looks as a crumpled sheet of paper at short scales. As the scale grows, the structure becomes similar to a network of wires which at large scales break resembling dust.

5.4.7 Calculation of Young Modulus and Thermal Conductivity

Once the fractal dimension $D_N$ is known, one can reconstruct the Young modulus and thermal conductivity. For Aerogels, [29] proposes that the Young modulus and the thermal conductivity satisfy,

$$\frac{E}{E_0} = \left( \frac{r}{r_0} \right)^{D_N - 5} \tag{5.48}$$

$$\frac{3}{\beta} \frac{\kappa_t}{\kappa_{t0}} = \left( \frac{r}{r_0} \right)^{\frac{D_N - 5}{2}} \tag{5.49}$$

were $\beta$ is a parameter in the range 1.3–1.7 that enhances the thermal conductivity in aerogels to account the large inner surface. Using (5.36), the relationship between the property scaling exponents and fractal dimension satisfy,

$$\mathcal{D}_E = D_N - 5 \tag{5.50}$$

$$\mathcal{D}_{\kappa_t} = \frac{D_N - 5}{2} \tag{5.51}$$
Figure 5.6: Effective Young modulus and thermal conductivity.

Using (5.42), (5.50) and (5.51), the specific elasticity \( E/E_0 \) and the specific thermal conductivity \( 3/\beta \kappa_t/\kappa_{t0} \) are obtained as presented in figure 5.6.

Note that for \( r = \xi \approx 6 \times 10^{-6}[m] \), corresponding to \( \rho/\rho_0 \approx g(\xi)/g_0 = 0.0023 \), the specific values of the elasticity and thermal conductivity are \( E/E_0 \approx 4.6 \times 10^{-10} \) and \( 3/\beta \kappa_t/\kappa_{t0} \approx 2.1 \times 10^{-5} \). Comparing to the data compiled in [45], \( E/E_0(\rho/\rho_0) \approx 6.23 \times 10^{-9} \), the value predicted is nominally lower than that from the experimental values reported in [45] for the same density and fractal dimension.

Varying the apparent density of the aerogel by varying the sizes of the clusters determined by the correlation length \( \xi \) in (5.47), it is found a a power law relationship between the Young modulus and density. The same behavior is found for the thermal conductivity and density.

Varying the fractal dimension \( D \) in (5.47), the property scaling exponent for the Young modulus and thermal conductivity is found to vary as presented in figure 5.7. Note that for \( D = 2.4 \), the exponent for the Young modulus coincides with the values reported in [45]. Then, the fractal dimension affects the mechanical and thermal properties of the aerogel.

5.4.8 Sintering

Sintering, one of the processes used to manufacture aerogels, ceramics and metal powders, consists on rising the temperature up to about 75% of the melting temperature. This enhances and activates mass transport mechanisms, e.g., solid diffusion, which reinforce bonding...
Figure 5.7: Exponent of the effective thermal conductivity and Young modulus power laws.

between the particles. Mass redistribution, which occurs at the smallest scales, increases the size of the primary particles, meaning that the structure becomes more compact, and denser at the short scale as corroborated experimentally by SAXS \cite{59}. In order to conserve the total mass of the structure, the structure at larger scales shrinks retaining the shape.

Assume two equal samples with primary particle \( r_0 \) which are sintered differently. For the first sample, the primary particle size increases to \( r_0^{(1)} \) whereas for the second sample, the primary particle size increases to \( r_0^{(2)} \) with \( r_0^{(1)} > r_0^{(2)} \). The reconstruction of any uncoupled property using (5.42), yields

\[
\mathcal{P}_i^{(2)}(r) = \mathcal{P}_i(r_0^{(2)}) e^{\int_{r_0^{(2)}}^r D_{\mathcal{P}_i}(r') d\ln r'}
\]

\[
\mathcal{P}_i^{(1)}(r) = \mathcal{P}_i(r_0^{(1)}) e^{\int_{r_0^{(1)}}^r D_{\mathcal{P}_i}(r') d\ln r'}
\]

Then, taking the ratio of the effective properties of the two samples,

\[
\frac{\mathcal{P}_i^{(2)}(r)}{\mathcal{P}_i^{(1)}(r)} = \frac{\mathcal{P}_i(r_0^{(2)})}{\mathcal{P}_i(r_0^{(1)})} e^{\int_{r_0^{(1)}}^{r_0^{(2)}} D_{\mathcal{P}_i}(r') d\ln r'}
\]  

(5.53)

Assuming that \( D_{\mathcal{P}_i}(r) \) is approximated by a constant (\( D_{\mathcal{P}_i}(r) \approx D_{\mathcal{P}_i} \)) in the range \( r_0^{(2)} < r < r_0^{(1)} \), the integral part in (5.53) yields
\[ \int_{r_0^{(2)}}^{r_0^{(1)}} \mathcal{P}_i(r') d\ln r' = \left( \frac{r_0^{(1)}}{r_0^{(2)}} \right)^{\mathcal{D}_i} \] (5.54)

For identical constitutive materials, \( \mathcal{P}_i(r_0^{(2)}) = \mathcal{P}_i(r_0^{(1)}) \). Hence, a power law relationship if found,

\[ \frac{\mathcal{P}_i^{(2)}}{\mathcal{P}_i^{(1)}} = \left( \frac{r_0^{(1)}}{r_0^{(2)}} \right)^{\mathcal{D}_i} \] (5.55)

Physical evidence of (5.55) is presented in [59]. In particular,

\[ \frac{\rho_i^{(2)}}{\rho_i^{(1)}} = \left( \frac{r_0^{(1)}}{r_0^{(2)}} \right)^{D-3} \] (5.56)

for \( D = D_N \) the fractal dimension at small scales, (close to the primary particle sizes). Thus,

\[ \frac{\mathcal{P}_i^{(1)}}{\mathcal{P}_i^{(2)}} = \left( \frac{\rho_i^{(1)}}{\rho_i^{(2)}} \right)^{\mathcal{D}_i} \] (5.57)

Physical evidence of this power law is presented in [5, 29, 32, 89] for thermal, elastic, and failure properties of sintered materials.

### 5.5 Conclusions

The multidimensional multifractal multiscaling is a complementary approach to predict properties of materials. The formalism allows the calculation of multidimensional properties in multidimensional materials. Characterizing the material structure in the logarithmic scale reduces the amount of information required as the structure is described by how the structure is composed, very convenient for hierarchical materials. If the characterization of the material changes with the scale, it is said that the material is multifractal. Describing the structure by how it is composed leads to a recursive procedure. These recursion is called multiscaling. As
a consequence, power laws describe the behavior of the structure characterized by property scaling exponents.

Affecting the hierarchical character of a material yields power law relationships as observed in sintering and Aerogels. These effects can be calculated computationally but also experimentally. In particular, properties are related as the result of share the same structure.
Chapter 6

Code Implementation

Abstract

In this chapter the implementation codes used to multiscale the effective properties of sol-gels and the codes to recreate the hierarchical structure are described. First, the multiscaling scheme is presented describing a broad view of its implementation. A detailed description of the functions developed in this dissertation is presented followed by the commented codes in Matlab and Fortran. Then, the code used to recreate the hierarchical structure is described.

A hierarchical description of the code is developed from a bird’s eye view going by steps into the details.

6.1 Bird’s eye view

A bird’s eye view of the project flow diagram is presented in figure 6.1. The calculation of the material effective properties is the result of the chemistry involved in the aggregation process of the sol-gel, the concentration of the species that react forming the colloid (as discussed in Chapter 1), and the material properties of the constituent materials (as discussed in Chapter 2). Also, the size of the container that determines the correlation length $\xi$ of the clusters is incorporated through the boundary conditions for the model. The model (Black Box) proposed takes these input parameters and finds the effective mechanical properties and scattering intensity.

Detailing the parameters as presented in the flow diagram in figure 6.2, the concentration of the precursors, the chemistry and periodic boundary conditions are fed into the model as a
set of reactivity $w$, simulation box size $L$, and number of particles $N$, for periodic boundary conditions $PBC$.

In this chapter, the implementation of the program blocks is detailed.

6.2 Program blocks

The model (Black Box), is implemented in Matlab using functions represented by blocks. An example of the Control Panel used in Matlab incorporating the blocks is presented.

In $[N, L, w]$, the matrix containing all the possible tests planned to do is written where $N$ is the number of particles to aggregate in the box with size $L$, for the reactivity $w$. $ganymed-ssh2$ needs to be installed at the Matlab workspace once, in order to establish communication with the cluster. The AggregationAlgorithm, ScatteringAlgorithm and EffectiveStiffnessAlgorithm functions return a list of the structures, a list of the scattering functions, and a list of the effective stiffness matrices, respectively, used to construct the Black Box presented in figure 6.3. Additional operations based on these functions are performed as AveIq which calculates the average over the samples with the same conditions $(N,L,w)$ of the scattering function. Similarly, the function AveStiffness calculates the average stiffness. The function EffectStiffGenAlgorithm recursively calculates the stiffness of each generation using the same structure obtained from the aggregation algorithm but using as properties the effective properties calculated at the previous generation, starting from the constituent material properties.

Since the computational resources of single machines are not enough to calculate the large amount of samples required, a Beowulf Cluster computer was used under Cluster Rocks 5.2 (Chimichanga) operating system when possible. The cluster consisted of 4 computers each one with 8 Intel(R) Xeon(R) E5506 at 2.13Ghz cores and 12Gb of RAM, and a front-node with 4 cores and 12Gb of RAM. This cluster, was called from functions in matlab using the java library $ganymed-ssh2-250$ which allows connecting to the cluster as a client.

\footnote{Rocks, based on CentOS from Linux is a package to easily build Beowulf cluster computers.}
through a secure shell interface from Matlab\footnote{MATLAB (R), Copyright 1984-201X MathWorks, Inc. All rights reserved.}, and upload-download of files.

Inside Matlab, functions describing each one of the blocks in figure\footnote{Figure 6.1: Bird's eye view of the effective properties calculation flow diagram.} was written. For the aggregation algorithm, a fortran90 using Open source Message Passing Interface library (MPICH2) \footnote{MPICH2} code was developed. Matlab creates a batch file with the conditions to be tested, it copies the aggregation code as well as the batch file to the cluster. The batch file is executed in the cluster compiling and creating the folders required for the simulations. Later, Matlab downloads the results, decompresses the folders, gathers the information into the Matlab workspace, and cleans the folder directories in the computer as well as in the cluster.

Similarly, the scattering algorithm was written in fortran90 using MPICH2, which is copied with the structure to be analyzed, from Matlab to the cluster. The code is executed, and the results are brought back into the Matlab workspace.

Also, due to the limited capability of Matlab to solve Partial Diferential Equations (PDE), Ansys\footnote{ANSYS, Copyright 1975-201X ANSYS, Inc. All rights reserved.} was used. An input file for Ansys is generated from Matlab. Then, Ansys is called in a batch mode from Matlab, and finally the results are brought back into the workspace.
6.2.1 Aggregation Algorithm

In [A.2] the aggregation algorithm is reproduced. This code requires the use of the cluster computer.

- **Function**: This function, in Matlab, gets structures for processing conditions.

- **Input**:

  1. **NLw**: This is a matrix containing at each row the parameters for a test. Each test is specified by the number of particles (first column), the length of the simulation box (second column), and the reactivity (third column).

  2. **SimulationParameters**: This is a list of a structure. Each item of the list corresponds to each of the tests in **NLw**. The structure has three fields:
Figure 6.3: Zooming the effective properties calculation flow diagram algorithm.
(a) `SimulationParameters(i).Tmax`: Maximum number of iterations so the aggregation algorithm will stop at some time.

(b) `SimulationParameters(i).OnlyLast`: If true, only the last structure of the aggregation algorithm is stored and returned. If false, every moment that the in the aggregation algorithm a couple of particles form a bond, this step is stored.

(c) `SimulationParameters(i).np`: The number of processors (samples) to make. This number can be higher than the number of cores in the cluster since the Cluster-Rocks(5.2) Distribution, using the operating system creates processes which are scheduled efficiently over the amount of processors available.

3. `datacluster`: Contains the necessary information to open a SSH-2 session with the cluster computer:

   (a) `datacluster.host`: contains the name (`char Type`) of the host computer.

   (b) `datacluster.user`: contains the user (`char Type`) to access the host computer.

   (c) `datacluster.password`: contains the password of the user (`char Type`) to access the host computer.

4. `downloadpath`: contains the folder name (`char Type`) to download temporarily the files from the cluster after results are obtained.

- **Output:**

  1. `StructureOut`: contains the list (as cells) of data structures resulting from the conditions `NLw` for each node. The dimension of the cell list is `{iTest, Node + 1}` where `iTest` is the number of the test, and `Node` is the node for the conditions `NLw[iTest,:)` going from 0 to `np-1`. Each item of the data structure contains:

     (a) `StructureOut{iTest,Node +1}.Structure` contains a matrix where the columns correspond to the X, Y, and Z position in the lattice of each particle.
(b) \texttt{StructureOut\{iTest, Node +1\}.Connectivity} contains a matrix where the first column indicates what particle is at $x+$, the second column indicates what particle is at $x-$, the third column indicates what particle is at $y+$, the fourth column indicates what particle is at $y-$, the fifth column indicates what particle is at $z+$, the sixth column indicates what particle is at $z-$. If the number is negative it indicates the particle is bonded. If there is no particle, a zero appears. The seventh column indicates the coordination number of the particle, i.e. how many particles are bonded to each particle.

(c) \texttt{StructureOut\{iTest, Node +1\}.MassArea} contains a matrix where the first column indicates the mass of the cluster containing each particle, the second column indicates the area of the cluster containing each particle, and the third column indicates the coordination number of each particle.

(d) \texttt{StructureOut\{iTest, Node +1\}.w} indicates the reactivity used for the simulation.

(e) \texttt{StructureOut\{iTest, Node +1\}.L} indicates the simulation box size used for the simulation.

(f) \texttt{StructureOut\{iTest, Node +1\}.N} indicates the number of particles used for the simulation nominally. The total number of particles vary since the code, in average puts $N$ particles in the cubic lattice, but since positioning particles is a random process, the real number of particles vary. The real number of particles used in the simulation can be calculated as the length of the position vectors, for example.

(g) \texttt{StructureOut\{iTest, Node +1\}.downloadpath} contains the name of the download folder.

2. \texttt{errmsg} is a whole number to establish if an error occurred inside the program. 0 means that no problem occurred.
• Requirements: The ganymed−ssh−250 service needs to be initialized at the workspace before using this function. The folder for this java-library needs to be located at the folder where the function code is saved. The program AggregationAlgorithm.f90 needs to be at the current folder too. The cluster used must have the ssh2-client service working, MPICH2 installed with the corresponding daemon running in order to execute the commands mpirun. Also, mpif90 needs to be installed, usually part of MPICH2. The file machines.txt is expected to be uploaded at the user home folder. This file contains a list of the names of the slave computers to be used in the parallel simulation.

AggregationAlgorithm.f90

The code AggregationAlgorithm.f90 is presented [A.3]

• Function: This is a Fortran90 program to run at the cluster-computer which from an input file with simulation conditions, produces, using a MonteCarlo simulation, structures in a cubic lattice under periodic boundary conditions.

• Input: InputParameters.txt This is a text file containing the following input parameters:

1. \( w \) the reactivity.
2. \( N \) the number of particles to simulate (in average).
3. \( L \) the size of the simulation box.
4. \( T_{\text{max}} \) the maximum number of iterations allowed.
5. OnlyLast If true, the program only reports the last structure, if false, the program saves all steps when a couple of particles formed a new bond.
6. OUTPUT_FileName the name of the output file, usually Structure.dat
- Output: The program creates and organizes folders corresponding to each independent simulation, from Node00 to NodeXX where XX corresponds to np−1. Inside each folder corresponding to each node, folders are created at each recorded time enumerated from 1 to the final time. The iteration time in the simulation corresponding to each time is recorded in InTime.dat. Inside each time, the files Structure.dat, Connectivity.dat and MassArea.dat are saved.

1. Structure.dat contains the list of positions \([X, Y, Z]\) for each particle. It is saved as a matrix where column one corresponds to the \(X\) position, column two to the \(Y\) position, and column three to the \(Z\) position.

2. Connectivity.dat contains a matrix where the first column indicates what particle is at \(x^+\), the second column indicates what particle is at \(x^-\), the third column indicates what particle is at \(y^+\), the fourth column indicates what particle is at \(y^-\), the fifth column indicates what particle is at \(z^+\), the sixth column indicates what particle is at \(z^-\). If the number is negative it indicates the particle is bonded. If there is no particle, a zero appears. The seventh column indicates the coordination number of the particle, i.e. how many particles are bonded to each particle.

3. MassArea.dat contains a matrix where the first column indicates the mass of the cluster containing each particle, the second column indicates the area of the cluster containing each particle, and the third column indicates the coordination number of each particle.

4. InTime.dat recorded iteration times at which a bond was formed.

- Requirements: MPICH2 is expected to be installed at the cluster. The input file InputParameters.txt is required, as well as machines.txt containing the names of the computers of the cluster used for the simulation.
6.2.2 Scattering Algorithm

In \[\text{A.3}\] the scattering algorithm is presented. This code requires the use of the cluster computer.

- **Function**: This Matlab function calculates the scattering function for a list of structures \{NoiTests, NoNodes\} where NoiTests is the number of tests for a certain condition \((N, L, w)\), and the number of samples in NoNodes using a cluster computer. The range of wave numbers is chosen by the size of the structure and 200 points are calculated. This parameter, inside the code is Nvals.

- **Input**

  1. **StructureIn**: A list of structures contained in a data structure as the output in section \[\text{6.2.1}\] is required.
  2. **datacluster**: the same as section \[\text{6.2.1}\]
  3. **downloadpath**: the same as section \[\text{6.2.1}\]

- **Output**

  1. **Iq\{iTest, Node + 1\}**: produces a list of intensities, for each test iTest and for each sample Node + 1.

- **Requirements**: Needs **SCATT_MPI.f90** uploaded and compiled as **Scatt.exe** with mpif90 in the cluster in a folder /home/user/ScatteringExp. Also the file **machines.txt** containing the names of the slave computer to use of the cluster is required in /home/user/ScatteringExp. The cluster needs MPICH2. Needs gamymed-ssh2-250 installed in the Matlab workspace.
SCATT-MPI.f90

The code SCATT_MPI.f90 is presented in [A.5]

- **Function**: Calculates the scattering function of a structure in a cluster computer using High Performance Computing through using MPIC2.

- **Input**:

  1. InputParameters.txt:
     
     (a) **Input_Structure**: name of the file where the structure to be analyzed is.
     
     (b) **Input_Qs**: name of the file where the wave number values for the scattering experiment are.
     
     (c) **Output_FileName**: name of the output file where the scattering function is saved.

  2. **Input_Structure**: contains in a matrix form the positions of the particles for the scattering experiment where each column corresponds to the coordinates \([X, Y, Z]\).

  3. **Input_Qs**: Contains a column vector with the wave vectors to test for the scattering experiment on the structure in **Input_Structure**.

- **Output**: **Output_FileName**: contains a first column vector with the wave vectors tested, and a second column with the scattering function calculated for the corresponding wave vector.

- **Requirements**: Needs MPICH2, and the files **InputParameters.txt**, **Input_Structure**, and **Input_Qs**.

### 6.2.3 Effective Mechanical Properties Algorithm

In [A.6] the effective mechanical properties algorithm code is presented.
• **Function:** this Matlab function calculates the effective stiffness for a list of structures given the material properties of the particles. The function calculates the total energy accumulated by the structure for several loading cases.

• **Input:**

1. **StructureIn:** A list of structures contained in a data structure as the output in section 6.2.1 is required.

2. **FEMParameters:** Parameters required by the Finite Element Method and Ansys:

   a. **FEMParameters.nameAnsys:** Name of the input file to run with Ansys in batch mode.

   b. **FEMParameters.dUtot:** Magnitude of the total (max) deformation to apply to the loading tests at the surfaces.

   c. **FEMParameters.du:** vector containing the magnitude of the total (max) deformation to apply at each loading case.

   d. **FEMParameters.mode:** loading case, ranging from 1 to 9 as depicted in figure 6.5

   e. **FEMParameters.type:** During the simulation, for each loading case, the type is (1) if the results are added to the previous results, (0) if the results overwrite the previous ones. In this manner, the 21 constants for an anisotropic material are calculated. Count the number of *new* tests corresponding to the number of 0’s.

   f. **FEMParameters.dumode:** is a matrix containing, for each loading test, the set of deformations, modes and types of loading.

   g. **FEMParameters.dirpath:** temporary folder where Ansys is doing the simulation.

   h. **FEMParameters.nameout:** is the name of the solution file produced by Ansys.
(i) `FEMParameters.AnsysPath`: is the path of the folder where the Ansys executable is located.

(j) `FEMParameters.ESize`: Size of the meshing in arbitrary units. A particle has 1 unit of length, and the size of the simulation box has $L$ units.


- *Output*:

  1. `EffStiffness`: list of the effective stiffness matrix for \{NoTests, NoNodes\}.

- *Requirements*: Uses a license of Ansys (mechanical) in batch mode.

The positioning of the 6 faces of the cube used for this function are depicted in figure 6.4. A note about the theorem by calculated directly from the elastic properties and not from energy is presented in B. However, the energy approach explained in Chapter 2 allows the effective properties calculation of nonlinear materials.

### 6.3 Auxiliary Functions

#### 6.3.1 Scattering Average

In A.7 the code to average the scattering functions over samples sharing the same processing \(w, N, L\) conditions is presented.
Figure 6.5: Loading cases.
• **Function**: Calculates the Average of the scattering function over the nodes (samples) np.

• **Input**:

  1. \( I_q \): List of scattering functions as output from section 6.2.2.

• **Output**:

  1. \( \text{AveI}_q \): List of average scattering functions for the processing conditions \{NoTests\}.

• **Requirements**: None

### 6.3.2 Effective Stiffness Average

In $A.8$ the code to average the effective stiffness over samples sharing the same processing \((w,N,L)\) conditions is presented.

• **Function**: calculates the Average Stiffness over the number of nodes (samples) np.

• **Input**:

  1. \( \text{EffStiffness} \): As the output of section 6.2.3.

• **Output**:

  1. \( \text{EffAveStiffness} \): List of average stiffness for the processing conditions \{NoTests\}.

• **Requirements**: None
6.3.3 Hierarchical Calculation of Stiffness

In A.9 the code to calculate the subsequent effective stiffness starting over samples sharing the same processing (w,N,L) conditions is presented during several generations is presented.

- **Function**: calculates over several generations the effective stiffness of a structure recurrently using the Average effective properties calculated at the previous generation as the properties of the constituent material.

- **Input**:
  1. **StructureIn**: Structures as output of section 6.2.1.
  2. **FEMParameters**: Finite Element Parameters as section 6.2.3.
  3. **Stiffness**: Constituent material stiffness.
  4. **Nohierarchies**: Number of generations to be evaluated.

- **Output**:
  1. **StiffGeneration**: List of average stiffness for the processing conditions \{NoTests, Nohierarchies\}.

- **Requirements**: The same as section 6.2.1

6.3.4 FindPolymer

In A.10 the code to calculate the list of particles (monomers) forming a polymer staring from a pivot particles based on their connectivity is presented.

- **Function**: List of particles (monomers) forming a polymer staring from a pivot particles based on their connectivity is presented.

- **Input**:
1. \texttt{n1}: number of pivot particle

2. \texttt{ConnectingMatrix}: Matrix containing the connection between particles. Position \((i, j)\) es equivalent to the relationship between particle \(i\) and particle \(j\). It takes a value of 1 if the particles are connected, and 0 if not.

- \textit{Output}:

  1. \texttt{polymer}: list of particles forming a polymer pivoted in particle \(n_1\).

- \textit{Requirements}: None

### 6.3.5 ElementNodesP

- \textit{Function}: The code in \ref{sec:A.11} calculates the elements connected each element for the fully connected structure.

- \textit{Input}:

  1. \texttt{Connection\_Matrix}: the same as \texttt{ConnectingMatrix} in section \ref{sec:6.3.4}

  2. \texttt{Particles}: is a data structure list containing the positions of particles.

    \begin{align*}
    \texttt{Particles(i).PosX} & \text{ contains the coordinate } x \text{ of the particle } i \text{ position,} \\
    \texttt{Particles(i).PosY} & \text{ contains the coordinate } y \text{ of the particle } i \text{ position, and} \\
    \texttt{Particles(i).PosZ} & \text{ contains the coordinate } z \text{ of the particle } i \text{ position.}
    \end{align*}

- \textit{Output}:

  1. \texttt{Element\_Nodes}: Returns a matrix with a list of all the nodes that connect each element. Note that the elements have the order given by the function \texttt{FindPolymer}, i.e. the list \texttt{polymer}.

  2. \texttt{polymer}: list of particles forming the connected polymer.

- \textit{Requirements}: None
6.3.6 ElementNodes

- **Function:** The code in A.12 calculates the elements connected each element for a structure connected according to a ConnectingMatrix.

- **Input:**
  1. **Connection_Matrix:** the same as ConnectingMatrix in section 6.3.4.
  2. **Particles:** is a data structure list containing the positions of particles. 
     - `Particles(i).PosX` contains the coordinate $x$ of the particle $i$ position,
     - `Particles(i).PosY` contains the coordinate $y$ of the particle $i$ position, and
     - `Particles(i).PosZ` contains the coordinate $z$ of the particle $i$ position.

- **Output:**
  1. **Element_Nodes:** Returns a matrix with a list of all the nodes that connect each element. Note that the elements have the order given by the function FindPolymer, i.e. the list polymer.
  2. **polymer:** list of particles forming the connected polymer.

- **Requirements:** None

6.4 Hierarchical Structure Reconstruction

The reconstruction of the hierarchical structure is made by recurrently transforming the mass contained in subdomains for a desired number of iterations (or generations). The transformation consists on first determining if the subdomain in consideration has at least a critical density. If the threshold density is achieved, the homogeneous mass in the subdomain is changed by a distribution of mass organized as the structures obtained from previous algorithms. If the density is lower than the threshold density, the subdomain is cleared.
6.4.1 HierarchicalStructure

- **Function:** The code in [A.13] calculates the density distribution for a structure composed hierarchically by several generations.

- **Input:**
  1. **dd:** is a structure with three fields containing the resolution of the 3D grid where the mass distribution is going to be calculated. \texttt{dd.x} contains the resolution in the \textit{x} axis, \texttt{dd.y} contains the resolution in the \textit{y} axis, and \texttt{dd.z} contains the resolution in the \textit{z} axis.
  2. **Generaciones:** contains the target number of generations.
  3. **Structure:** the same as in section 6.2.1.
  4. **L:** the size of the simulation box.
  5. **iTest:** the test number as is section 6.2.1

- **Output:**
  1. **Intensity:** Returns a 3D matrix the dimensions determined by \texttt{dd} containing the density distribution as the result of iterating \texttt{Generaciones} number of times.

- **Requirements:** None
Chapter 7

Conclusions

Many of the detailed conclusions are presented in each chapter. Here, only an overall conclusion is presented.

The effective properties of materials can be calculated describing them hierarchically. This means that characterizing the structure response requires few parameters at each scale which are the scaling exponents and the eigenvectors associated to the effective property operator $\mathcal{H}$.

Fractal materials present the same scaling exponents at each scale. For non-fractal materials, the scaling exponents change with the scale leading to the multifractal description of the material. In both cases, materials can be considered hierarchically.

Multiscaling, the process for the calculation of the effective properties of a hierarchical material, is a recursive procedure. Multidimensional properties such as stiffness may couple the property components from one scale to the next one. Thus, the procedure can be involved, however, using FEM and homogenization techniques allow us to easily solve the problem computationally.

The effective stiffness is strongly dependent on the connected structure and the mass distribution. As the structure is more connected, it becomes stiffer. As the mass distribution is arranged to make the material structure more compact, thus improving the connectivity
of the structure, the density increases too. Then, improving the stiffness by making the structure more compact may not improve the stiffness versus density relationship.

Scattering intensity allows measuring the critical percolation density. At the critical percolation density, a structure produced from an aggregation process develops a fractal structure. At this density, the scattering intensity can be calculated using an iterative procedure limited by the correlation length. Then, scattering experiments in the lab, which allow the determination of the correlation length and fractal dimension, give enough information to characterize the mass distribution of a sol-gel. For aged/sintered structures, the mass distribution is the only requirement for the mechanical response calculation explaining how scattering experiments and the elastic behavior are strongly correlated. Nonetheless, sol-gels that are not aged/sintered do not present a strong correlation between the mechanical response and the mass distribution. Therefore, another parameter measuring the amount of missing bonds (effectiveness in the bonding) is required to derive the mechanical response directly from scattering experiments.

The aggregation process can be modeled stochastically incorporating the Brownian motion and chemical reactions. Chemical reactions form bonds, which depending on their nature, favor different types of structures. Characterizing the chemical reactions by a reactivity that measures the difficulty in forming an additional bond, allows the parametrization of the aggregation process. The calculation of the reactivity can be done by fitting calorimetric data from real experiments or \textit{ab-initio} simulations. Then, the reactivity becomes a function of the reactants and their molar fractions.

In a general conclusion, visualizing materials hierarchically proves to be a powerful tool in computational materials, in particular for materials with a fractal or multifractal character. Modeling materials under the hierarchical scope, called here \textit{Multifractal Multidimensional Multiscaling}, is believed to be general.
Bibliography


BIBLIOGRAPHY


[90] University of California., “Cluster Rocks 5.2 (Chimichanga) distribution..”


Appendix A

Program Codes
A.1 Control Panel Code in Matlab

Each cell, beginning with [%], can be executed independently.

```matlab
% created by: Fritz A Campo MAE@WVU 04/15/2011
%
installssh = false;
%% This cell executes the AggregationAlgorithm.f90 in the cluster
clc
% Data spool
w = 1.0; % Save this as a vector
L = 5;
N = 48;
NLw = [N,L,w];
for i = 1:length(w)
    SimulationParameters(i).Tmax = 100000;
    SimulationParameters(i).OnlyLast = 1;
    SimulationParameters(i).np = 32;
end

installssh = true;
datacluster.host='XXX.XXX.XXX.XX'; % Cluster IP
datacluster.user='XUSERX';
datacluster.password='XXPASSWORDXX';
downloadpath = 'C:\TempPhD';
[StructureOut ,errmsg] = AggregationAlgorithm( NLw, SimulationParameters,datacluster, downloadpath);
display('Aggregation ... ready!');

%% Installing java ganymed-ssh2 services
if ~installssh
    currentDir = pwd;
    cd([currentDir,'\ganymed-ssh2-build250']); % The dell
    javaaddpath('ganymed-ssh2-build250.jar')
    cd(currentDir);
    import ch.ethz.ssh2.SCPClient;
    import java.io.BufferedReader;
    import java.io.IOException;
    import java.io.InputStreamReader;
    import ch.ethz.ssh2.Connection;
    import ch.ethz.ssh2.Session;
    import ch.ethz.ssh2.ChannelCondition;
    installssh = true;
end

%datalcluster.host='XXX.XXX.XXX.XX'; % Cluster IP
datacluster.user='XUSERX';
datacluster.password='XXPASSWORDXX';
downloadpath = 'C:\TempPhD';
[StructureOut ,errmsg] = AggregationAlgorithm( NLw, SimulationParameters,datacluster, downloadpath);
display('Aggregation ... ready!');

%% Scattering Experiments
[Iq] = ScatteringAlgorithm(StructureOut,datacluster, downloadpath);
%% Mechanical Experiments
% Defining initial Stiffness
nu=0.22;
E=80; %GPa
% Material Properties of Amorphous Silica Ref. Freund, L.B. and Suresh, S.
% "Thin film materials", Cambridge University Press, 2003, pg. 96
S(1,1)=1/E; % [S] is Compliance in Barbero()
S(1,2)=nu/E;
S(2,1)=nu/E;
S(1,3)=nu/E;
S(3,1)=nu/E;
S(2,2)=1/E;
S(2,3)=nu/E;
S(3,2)=nu/E;
S(3,3)=1/E;
S(4,4)=2*(1+nu)/E;
S(5,5)=2*(1+nu)/E;
S(6,6)=2*(1+nu)/E;
```
Stiffness = inv(S);

FEMParameters.nameAnsys = 'AnsysInput.dat';
FEMParameters.dirpath = downloadpath;
FEMParameters.nameout='dataOut.dat';
FEMParameters.AnsysPath = 'C:\Program Files\Ansys Inc\v90\ANSYS\bin\intel\ansys90';
FEMParameters.AnsysPath = 'C:\Program Files (x86)\Ansys Inc\v90\ANSYS\bin\intel\ansys90';
FEMParameters.dUtot = 1;
FEMParameters.dUtot = ones(1,54);
FEMParameters.mode=[7,8,5,9,7,8,5,4,6,4,6,7,8,4,6,5,9,3,3,7,8,5,9,3,4,...
6,2,7,8,2,5,9,2,4,6,2,3,1,1,7,8,1,5,9,1,4,6,1,3,1,2];
FEMParameters.type=[0,1,0,1,0,1,0,1,1,0,1,1,0,1,1,0,0,1,1,0,1,0,1,0,1,...
1,0,0,1,1,0,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1];
FEMParameters.dumode=[FEMParameters.du',FEMParameters.mode',FEMParameters.type'];
FEMParameters.ESize = 1;
FEMParameters.FullyConnected = true;
[FeffStiffnessFC]=EffectiveStiffnessAlgorithm(StructureOut, FEMParameters, Stiffness);
FEMParameters.FullyConnected = false;
[FeffStiffnessAG]=EffectiveStiffnessAlgorithm(StructureOut, FEMParameters, Stiffness);

% Data Analysis

% Scattering
% Average of Scattering Function
[IqAve] = AveIq(Iq);

% Mechanical
% Average of Stiffness
[EffAveStiffnessFC]= AveStiffness(EffStiffnessFC);
[EffAveStiffnessAG]= AveStiffness(EffStiffnessAG);

% Hierarchical Mechanical Generations
Nohierarchies = 4;
FEMParameters.FullyConnected = true;
[StiffGenerationFC] = EffectStiffGenAlgorithm(StructureOut,FEMParameters, Stiffness, Nohierarchies);
FEMParameters.FullyConnected = false;
[StiffGenerationAG] = EffectStiffGenAlgorithm(StructureOut,FEMParameters, Stiffness, Nohierarchies);
A.2 Aggregation Algorithm Matlab Code

function \[\text{StructureOut} , \text{errmsg}\] = AggregationAlgorithm( \text{NLw}, \text{SimulationParameters}, \text{datacluster}, \text{downloadpath})
% Created by Fritz A Campo @ MAE WVU 13/04/2011
% This function Executes the AggregationAlgorithm.f90 in a Linux Cluster
% It:
% 1) creates a Batch File to execute in Linux: "SimBatch-dateXXXX-.txt"
% 2) opens an SSH
% 3) copies: a) AggregationAlgorithm.f90
% b) machines.txt This is the file for the machines to
% run of the cluster
% 4) compiles using OpenMPI the file%
% 5) executes the Batch File: a) one can choose to end executing of wait
% leave the cluster processing closing the connection, c) or leave the
% cluster processing with the creation of a nohup.out file
% 6) Downloads files to downloadpath
% 7) cleans cluster

\text{errmsg} = 0;
\text{Structure} = {};
\text{MassArea} = {};
\text{Connectivity} = {};
[rowNLw, colNLw] = size(NLw);

% Create Batch File for cluster
\text{UserinCluster} = \text{datacluster}.user;
\text{Nvals} = NLw(:,1);
\text{Lvals} = NLw(:,2);
\text{wvals} = NLw(:,3);
\text{datestr} = date;
\text{fid} = fopen(['SimBatch', datestr, '.txt'], 'w');
\text{downloadlistfiles} = {};
for test = 1:rowNLw
  try
    fprintf(fid, ['# Test No.', num2str(test), ' ============ ', datestr, ' ============== 
']);
    fprintf(fid, ['TargetDIRNAME=MechW', num2str(1000*wvals(test)), 'N', num2str(Nvals(test)), 'L', num2str(Lvals(test)), '; 
']);
    downloadlistfiles = {downloadlistfiles{:}, ['MechW', num2str(1000*wvals(test)), 'N', num2str(Nvals(test)), 'L', num2str(Lvals(test))]};
    fprintf(fid, '\text{SourceCode}="AggregationAlgorithm"; \n');
    fprintf(fid, ['w="', num2str(wvals(test)), '" \n']);
    fprintf(fid, ['N="', num2str(Nvals(test)), '" \n']);
    fprintf(fid, ['L="', num2str(Lvals(test)), '" \n']);
    fprintf(fid, ['Tmax="', num2str(SimulationParameters(test).Tmax), '" \n']);
    fprintf(fid, ['OnlyLast="', num2str(SimulationParameters(test).OnlyLast), '" \n']);
    fprintf(fid, ['cd /home/', UserinCluster, '/$TargetDIRNAME'; \n']);
    fprintf(fid, ['mkdir /home/', UserinCluster, '/$TargetDIRNAME'; \n']);
    fprintf(fid, ['cp /home/frcampo/$SourceCode.f90 /home/', UserinCluster, '/$TargetDIRNAME'; \n']);
    fprintf(fid, ['mpif90 $SourceCode.f90 -o /home/', UserinCluster, '/$TargetDIRNAME/$SourceCode.f90' \n']);
  catch
    fprintf(fid, ['# Changing Parameters File \n']);
    fprintf(fid, ['cd /home/', UserinCluster, '/$TargetDIRNAME'; \n']);
    fprintf(fid, 'rm -r InputParameters.txt; \n');
    fprintf(fid, 'touch "InputParameters.txt"; \n');
    fprintf(fid, 'echo "w $w" >> "InputParameters.txt"; \n');
    fprintf(fid, 'echo "$N" >> "InputParameters.txt"; \n');
    fprintf(fid, 'echo "$L" >> "InputParameters.txt"; \n');
    fprintf(fid, 'echo "Tmax $Tmax" >> "InputParameters.txt"; \n');
    fprintf(fid, 'echo "OnlyLast $OnlyLast" >> "InputParameters.txt"; \n');
  end
end
APPENDIX A. PROGRAM CODES

fprintf(fid,'echo " " >> "InputParameters.txt"; \n \n');

fprintf(fid,'"Parallel Computing \n');
fprintf(fid,'"mpiexec -n ',num2str(SimulationParameters(test).np),' -machinefile machines.txt sg.exe; \n');
fprintf(fid,'cd ..; \n');
fprintf(fid,'tar -czf "$TargetDIRNAME.tar.zip" "$TargetDIRNAME" \n');
fprintf(fid,'echo "Decompress with tar -xzf $TargetDIRNAME.tar" \n');
catch ME
ermsg = 1;
end
end
fclose(fid);

currentDir=pwd;
% ssh service must be installed in Matlab-workspace
import ch.ethz.ssh2.SCPClient;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import ch.ethz.ssh2.Connection;
import ch.ethz.ssh2.Session;
import ch.ethz.ssh2.StreamGobbler;
import ch.ethz.ssh2.ChannelCondition;

% Setup the channel with the remote server SCP
try
    channel = Connection(datacluster.host);
    channel.connect();
catch
    error('SCPTOMATLAB could not connect to the...
        'remote machine %s ...'],datacluster.host);
end
% Check the authentication for login...
isAuthenticated = channel.authenticateWithPassword(datacluster.user,datacluster.password);
if(~isAuthenticated)
    error...
    ('SCPTOMATLAB could not authenticate the',...
        'SSH connection...');
end

% Uploading AggregationAlgorithm.f90 and machines.txt and Batch file:
% ['SimBatch',datestr,'.txt']
scp1 = SCPClient(channel) %Open session
try
    scp1.put([currentDir,'
',machines.txt'], ['/home/',UserinCluster])
    scp1.put([currentDir,'
',SimBatch',datestr,'.txt'], ['/home/',UserinCluster])
display('files uploaded to cluster... ready')
catch ME
    errmsg = 3;
end

% Executing in Cluster
channelssh2 = channel.openSession(); %Open execution session
try
    CmdStr = ['bash ',SimBatch',datestr','.txt ']
    %CmdStr = ['nohup bash ',SimBatch',datestr','.txt & ']
    %CmdStr = ['less nohup.out'];
channelssh2.execCommand(CmdStr);
channelssh2.waitForCondition(ChannelCondition.EOF,100000);
display('files executed at cluster... ready')
catch ME
  errmsg = 4;
end
channelssh2.close();

% Downloading files
localedownloadfolder = downloadpath
for i = length(downloadlistfiles)
  try
    scp1.get(['/home/',UserinCluster,'/',downloadlistfiles{i},'.tar.zip'],
            localdownloadfolder);
catch ME
  errmsg = 5;
end
display([['downloaded file: ',downloadlistfiles{i},'.tar.zip ... ready']]);
end

% Clean in Cluster
try
  CmdStr = ['rm AggregationAlgorithm.f90 '];
  channelssh2 = channel.openSession();channelssh2.execCommand(CmdStr);
  channelssh2.waitForCondition(ChannelCondition.EXIT_SIGNAL,100000);channelssh2.close();
display(CmdStr);
  CmdStr = ['rm ',SimBatch',datestr','.txt '];
  channelssh2 = channel.openSession();channelssh2.execCommand(CmdStr);
  channelssh2.waitForCondition(ChannelCondition.EXIT_SIGNAL,100000);channelssh2.close();
display(CmdStr);
  for i = length(downloadlistfiles)
    CmdStr = ['rm ',downloadlistfiles{i},'.tar.zip '];
    channelssh2 = channel.openSession();channelssh2.execCommand(CmdStr);
    channelssh2.waitForCondition(ChannelCondition.EXIT_SIGNAL,100000);channelssh2.close();
display(CmdStr);
    CmdStr = ['rm -r ',downloadlistfiles{i}];
    channelssh2 = channel.openSession();channelssh2.execCommand(CmdStr);
    channelssh2.waitForCondition(ChannelCondition.EXIT_SIGNAL,100000);channelssh2.close();
display(CmdStr);
  end
display('files deleted at cluster... ready')
catch ME
  errmsg = 6;
end
channel.close()

% Decompressing downloaded data
% Im using 7-Zip program (WinRaR can be used to,, be sure to have in in
% the PATH for Windows
for i=1:length(downloadlistfiles)
  eval(['!7z x -r ',downloadpath,'\',downloadlistfiles{i},'.tar.zip -o',downloadpath]);
  eval(['!7z x -r ',downloadpath,'\',downloadlistfiles{i},'.tar -o',downloadpath]);
  eval(['!del ',downloadpath,'\',downloadlistfiles{i},'.tar']);
  eval(['!del ',downloadpath,'\',downloadlistfiles{i},'.tar.zip']);
end

% Retriving Data: Connectivity, MassArea, Structure
% Investigating cases in directory to retrive
caseDir = downloadpath;
directories=dir(caseDir);
counter=0;
for i=3:length(directories)
if directories(i).isdir
    counter=counter+1;
    wPos=find(cases{counter}=='W');
    nPos=find(cases{counter}=='N');
    lPos=find(cases{counter}=='L');
    ws{counter}=str2num(cases{counter}(wPos+1:nPos-1))/1000;
    Ns{counter}=str2num(cases{counter}(nPos+1:lPos-1));
    Ls{counter}=str2num(cases{counter}(lPos+1:end));
end

% np counting
nprocessors = 0;
Cdir = dir(fullfile(caseDir, cases{counter}));
for dirs=1:length(Cdir)
    if Cdir(dirs).isdir
        nprocessors = nprocessors + 1;
    end
end
np{counter} = nprocessors - 2; % subtrace /. and /..

% Bringing data to Matlab
for iTest=1:length(cases)
    for Node=0:np{iTest}-1
        a=dir(fullfile(caseDir, cases{iTest}, char(filesep), 'NODE', num2str(Node, '%2.2d'), char(filesep)));
        cd(fullfile(caseDir, cases{iTest}, char(filesep), 'NODE', num2str(Node, '%2.2d')));
        ftl(fullfile(caseDir, cases{iTest}, char(filesep), 'NODE', num2str(Node, '%2.2d')));
        Structure{iTest,Node+1} = load(fullfile(ftl, '\', 'Structure.dat')); % [X,Y,Z]
        Connectivity{iTest,Node+1} = load(fullfile(ftl, '\', 'Connectivity.dat')); % [x+ x- y+ y- z+ z-] [Coordination]
        MassArea{iTest,Node+1} = load(fullfile(ftl, '\', 'MassArea.dat')); % [Mass Area] [Coordination]
    end
end

cd(currentDir);

for iTest=1:length(cases)
    for Node = 0:np(iTest)-1
        StructureOut{iTest,Node+1}.Structure = Structure{iTest,Node+1};
        StructureOut{iTest,Node+1}.Connectivity = Connectivity{iTest,Node+1};
        StructureOut{iTest,Node+1}.MassArea = MassArea{iTest,Node+1};
        StructureOut{iTest,Node+1}.w = ws{iTest};
        StructureOut{iTest,Node+1}.L = Ls{iTest};
        StructureOut{iTest,Node+1}.N = Ns{iTest};
        StructureOut{iTest,Node+1}.downloadpath = downloadpath;
    end
end

% cleaning temporal downloading directory
for i=1:length(downloadlistfiles)
    eval(['!del ', downloadpath, '\', downloadlistfiles{i}]);
end
A.3 Aggregation Algorithm Program in Fortran

!******************************************************** COMMENTS ********************************************************
! Created by Fritz A Campo. Modified from sg5_MPI.f90
!
! Compile with: >>mpif90 _code__ -o _OutputFile___
!
! Run this program on Barbero Cluster with:
! >>mpiexec -n 24 -machinefile machines.txt _File2Execute_
! (Note: or -np)
!
! machines.txt contains the names of the machines:
! (file begins)
! compute-0-0
! compute-0-1
! compute-0-2
! (file ends)
!
! Required files:
! - InputParameters.txt
! Contains the parameters of the simulations
!
! Creates several folders at different times for each
! of the nodes saving the evolution of the CLUSTER Formation
!
! Update: MM/DD/YYYY | Details
! =========== =========== | ===========
! 01/20/2010 | - Creates Folders to save evolution
! 01/25/2010 | - Brownian Probability BP 1/(am^3/2)
! 01/28/2010 | - Output includes Total particles
! 02/01/2010 | - Total Mass & Total Area and
! | Coordination Evolution.
! 02/28/10 | - Reduction of number of reaction per
! | motion, to increase effect of low reactivity
! 03/02/10 | - reduction of reaction probability due
! | multiple point collision
! 03/12/10 | - Fixing possible reaction concept/
! | No scattering and maintaining the same seed
! | making a single repeatable test
! 03/18/10 | - Changing FindingNeighbors()
! 03/19/10 | - Creating Functions to clean code
! 03/22/10 | - Error FOUND: in ChoosingParticle, there were
! | particles that were not supposed to be chosen: {0,N+1}
! 03/22/10 | - Cleaning of code from tests
! 03/31/10 | - Removing and fixing T Folder Creation and
! | SelfDetermining SampleT and saving Times
! 04/11/10 | - Saving data with no time but sequentially
! 06/09/10 | - Qvals are determined by L itself, and connectivity is
! | saved.
! 13/04/11 | - Clean version + Commenting
!********************************************************

module vars
IMPLICIT NONE

! Declaration of Basic Parameters
REAL,PARAMETER::PI=3.1415926535

integer :: seed

! +++++ PUBLIC VARIABLES +++++
! Input Parameters (Be careful with these ones)
INTEGER, SAVE :: N, L, Tmax ! No Particles, Size of Lattice, Max t
REAL, SAVE :: w ! Reactivity
INTEGER, SAVE :: sampleT ! Sampling Rate
CHARACTER(len=256) :: INPUT_FileName
INTEGER, SAVE :: OnlyLast

! Program variables
! ParticleInfo Structure  [X Y Z][x+ x- y+ y- z+ z-][Mass Area]
! For all particles
! [X Y Z] is the position of the particle
! [x+ x- y+ y- z+ z-] are the neighbors of the particle at the
! corresponding positions
! [Mass Area] is the mass and area of the polymer to
! which the particle is part of
INTEGER, SAVE, DIMENSION(:,,:), POINTER :: ParticlesInfo
INTEGER, SAVE, DIMENSION(:,,:), POINTER :: tempParticlesInfo ! just for input purposes

LOGICAL, SAVE::Cycle_Condition ! cycling condition satisfied while t=Tmax and MaxMass < N
REAL, SAVE, DIMENSION(:,), POINTER :: BPArray ! Probability distribution of choosing a particle
! depending of the Mass and Area
INTEGER, SAVE, DIMENSION(:,), POINTER :: polymer1, polymer2 ! Test Polymers
INTEGER, SAVE :: NPoli, NPoli2 ! Lengths of polymers
INTEGER, SAVE :: NoCollision ! Number of collisions a polymer is subjected to
INTEGER, SAVE, DIMENSION(:,,:), POINTER :: NCollision ! Array that contains the pairs of colliding particles
integer, SAVE :: MaxMass ! Maximum mass of a polymer
logical :: WasThereReaction

! Scattering Variables
!INTEGER, SAVE :: No_Q ! Number of values for the scattering experiment
!REAL, SAVE, DIMENSION (:), pointer :: Q_vals ! Values for the scattering experiment

! MPI variables/parameters
integer, SAVE :: rank,ierr,numprocs,rc

! Debug Variables
INTEGER, DIMENSION(8) :: BugINT
REAL :: BugREAL0, BugREAL1

end module vars

! ********************* PROGRAM ******************** ********
program sg2010_MPI

use vars
implicit none
include 'mpif.h'

integer :: tag=1
integer stat(MPI_STATUS_SIZE)

! Functions
integer :: ChooseParticle, Random16generator ! Chooses particles and direction according to BPArray[]
logical :: PolymerCollision ! Establishes if a polymer collides or not
real :: Energy ! Evaluates Energy of the system based on the number of coordination distribution

! Variables in the Main Program
integer :: t,SavingT,i,j,k,status ! dummy counters
integer :: Parti, dir ! Chosen dir and particle per cycle
logical :: Collide ! Stores if the polymer containing Parti collides with something
CHARACTER(len=100) :: InTime_FileName
! Starting MPI
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )

call createnode(rank) ! Creates Node folder to save data for each node (rank)
call init_random_seed(rank) ! Initiates RandomGenerator function independently for each node

INPUT_FileName='InputParameters.txt' ! Name of the Parameters file
call Input_Parameters() ! Brings Parameters of the simulation from file
call Initialize() ! Puts about N particles in lattice of size L
print *, 'No.Particles', N, ' @ ', rank ! Each node creates its own number of particles close to the number in the parameters

! Definition of Pointers
allocate(tempParticlesInfo(N,11))
allocate(BPArray(N))
allocate(polymer1(N))
allocate(polymer2(N))
allocate(NCollision(N,2))

WRITE(chID,'(i2.2)') rank
InTime_FileName = 'InTime.dat'
InTime_FileName = 'NODE'/chID//'/InTime_FileName
OPEN (UNIT=5,FILE=InTime_FileName, STATUS='REPLACE', &
ACTION='WRITE', IOSTAT=status)

! Initial Step t=0
 t = 0
SavingT = 1
IF (OnlyLast == 0) THEN
 call createnodetime(rank, SavingT)
call SavingData(rank, SavingT)
call SavingMassAreaData(rank, SavingT)
 WRITE (*,*) t
endif

Cycle_Condition=.TRUE.
call FindNeighbors()
call BP()

MaxMass=1
WasThereReaction = .false.
! Evolution starts!!!!

do while(Cycle_Condition)
 WasThereReaction = .false.
 Parti=ChooseParticle()
dir=Random16generator()
Collide = PolymerCollision(Parti, dir)

IF (.not.Collide) THEN ! Move polymer1 of particles in the direction dir
call MovePolymer(dir)
ELSE ! There was no movement of polymer1 because there is at least a collision
call ReactPolymer(dir)
call Sintering()
EndIF ! Accepted movement or reaction (new configuration)

 t=t+1
IF (WasThereReaction) THEN ! Saving data and plotting info:
  write (*,*) 'T=', t, 'MaxMass= ', MaxMass, '/', N, ' @ ', rank
  IF (OnlyLast == 0) THEN
  WRITE (*,*) t
endif
SavingT = SavingT + 1
call createnodetime(rank, SavingT)
call SavingData(rank, SavingT)
call SavingMassAreaData(rank, SavingT)
endif
endif
IF (t==Tmax) Cycle_Condition=.FALSE.
enddo !cycling
print *, '------- t_Tot = ', t, ', NPart=',N,' @ ',rank
IF (OnlyLast==1) THEN
WRITE (5,*) t
SavingT = SavingT + 1
call createnodetime(rank, SavingT)
call SavingData(rank, SavingT)
call SavingMassAreaData(rank, SavingT)
endif

close(5)
! Releasing Pointers and closing MPI
dallocate(ParticlesInfo)
dallocate(BPArray)
dallocate(polymer1)
distribute(polymer2)
call MPI_FINALIZE(ierr)
end program sg2010_MPI

!***************************************************************
FUNCTIONS
***************************************************************

! This is the place to include Sinterization or sticking of neighbors from the new whole polymer
! - According to a probability given by e^{-Qsin/kT} for sintering, evaluate the following code
!call FindPolymer1(Parti)
!Area=ParticlesInfo(Parti,11)
!do i=1,NPol1 !Start Sintering
! do j=1,6 !over directions
! IF (ParticlesInfo(polymer1(i),3+j) > 0) THEN
! IsInPolymer = ISPartjPolymer1( ParticlesInfo(polymer1(i),3+j) )
! IF (IsInPolymer) THEN
! ParticlesInfo(ParticlesInfo(polymer1(i),3+j),3+InvDir(j) ) =
! ParticlesInfo( ParticlesInfo( polymer1(i),3+j),3+InvDir(j) )
! ParticlesInfo(polymer1(i),3+j)=-ParticlesInfo(polymer1(i),3+j)
! Area = Area - 2 !Recalled from the last Polymer1
! endif
! endif
! enddo !over directions
!enddo !Sintering
!DO i=1,NPol1
! ParticlesInfo(polymer1(i),11)=Area
!ENDDO

logical function PolymerCollision(Pivot, direction)
! Is the polymer where Pivot is part of collides if moving in this 'direction'?
use vars
implicit none

INTEGER, INTENT(IN) :: Pivot, direction
integer :: i
logical :: NotCollide

integer :: InvDir
logical :: ISPartjPolymer1

NoCollision = 0
NCollision = 0
NotCollide =.true.
call FindPolymer1(Pivot)
do i=1,NPol1 ! over all particles of Polymer1
   if (ParticlesInfo(polymer1(i),3+direction) <= 0) THEN ! Particles
      does Not collide if there is a bonded particle
      NotCollide = .true. .and. NotCollide
   elseif (ISPartjPolymer1( ParticlesInfo(polymer1(i),3+direction) ) ) THEN ! does not
      collide if there is a particle not bonded directly but part of the polymer1
      NotCollide = .true. .and. NotCollide
   else ! it collides
      NotCollide = .false.
      NCollision = NoCollision + 1 ! Add one collision to the count
      NCollision(NCollision,1)= polymer1(i) ! save the colliding particles to test later
      NCollision(NCollision,2) = ParticlesInfo(polymer1(i),3+direction)
   endif
enddo ! TestAcceptance
PolymerCollision = .not.NotCollide
return
end function PolymerCollision

subroutine ReactPolymer(direction)
! Evaluates is the colliding particles form a bond
use vars
implicit none

integer, intent(in) :: direction
integer :: i, k
integer :: Mass, Area
logical :: OneReaction

! Functions
integer :: InvDir
logical :: Reaction, ISPartjPolymer1

OneReaction = .false.
do i=1, NoCollision ! Evaluate for all the particles that collide if there is a new bond
   IF ( Reaction(NCollision(i,1), NCollision(i,2)) ) THEN
      WasThereReaction = .true.
      IF (.not.OneReaction) OneReaction = .true. ! first reaction?
      call FindPolymer1(NCollision(i,1)) ! Find new polymer pivot on tested particle
   endif
   IF (ISPartjPolymer1(NCollision(i,2))) THEN ! Updates Are a if colliding particle
      Area=ParticlesInfo(NCollision(i,1),11)-2
      DO k=1,NPol1 ! Updating Pol 1
         ParticlesInfo(polymer1(k),11)=Area
      ENDDO
      ParticlesInfo(NCollision(i,1),3+direction)= - NCollision(i,2) ! Creation of new bond
      ParticlesInfo(NCollision(i,2),3+InvDir(direction))= - NCollision(i,1)
   ELSE ! Updates Area if colliding particle is not part of polymer1
      call FindPolymer2(NCollision(i,1,2)) ! Finds polymer to attach
      Area = ParticlesInfo(NCollision(i,1,1),11) + ParticlesInfo(NCollision(i,2,1),11)-2 ! Updates area and mass
      Mass = ParticlesInfo(NCollision(i,1,1),10) + ParticlesInfo(NCollision(i,2,1),10)
      MaxMass = max(MaxMass,Mass)
      IF (MaxMass == N) Cycle_Condition = .FALSE. ! if all particles for a single cluster, stop algorithm: this interferes with sintering.. easy to fix
      DO k=1,NPol1 ! Updating Polymer 1
         ParticlesInfo(polymer1(k),11)=Area
         ParticlesInfo(polymer1(k),10)=Mass
      ENDDO
      DO k=1,NPol2 ! Updating Polymer 2
         ParticlesInfo(polymer2(k),11)=Area
         ParticlesInfo(polymer2(k),10)=Mass
      ENDDO
   endif
enddo
!
APPENDIX A. PROGRAM CODES

ENDDO

ParticlesInfo(NCollision(i,1),3+direction)= - NCollision(i,2) ! creates bond
ParticlesInfo(NCollision(i,2),3+InvDir(direction)) = - NCollision(i,1)
endif ! is or not in polymer1
endif !Reaction
IF (OneReaction) call BP() ! If it is the first reaction, update
Brownian probabilities (not at every time, because is very time consuming)
ENDIF

end subroutine ReactPolymer

END FUNCTION Reaction

SUBROUTINE MovePolymer(direction)
! Moves polymer1 in this direction
use vars
! implicit none

integer, intent(in) :: direction
integer :: i

DO i=1,NPol1
  IF (direction==1) THEN
    IF (ParticlesInfo(polymer1(i),1)==L) THEN
      ParticlesInfo(polymer1(i),1)=1
    ELSE
      ParticlesInfo(polymer1(i),1)=ParticlesInfo(polymer1(i),1)+1
    ENDIF
  ELSEIF (direction==2) THEN
    IF (ParticlesInfo(polymer1(i),1)==1) THEN
      ParticlesInfo(polymer1(i),1)=L
    ELSE
      ParticlesInfo(polymer1(i),1)=ParticlesInfo(polymer1(i),1)-1
    ENDIF
  ENDIF
END DO
Elseif (direction==3) THEN
  IF (ParticlesInfo(polymer1(i),2)==L) THEN
    ParticlesInfo(polymer1(i),2)=1
  ELSE
    ParticlesInfo(polymer1(i),2)=ParticlesInfo(polymer1(i),2)+1
  ENDIF
Elseif (direction==4) THEN
  IF (ParticlesInfo(polymer1(i),2)==1) THEN
    ParticlesInfo(polymer1(i),2)=L
  ELSE
    ParticlesInfo(polymer1(i),2)=ParticlesInfo(polymer1(i),2)-1
  ENDIF
Elseif (direction==5) THEN
  IF (ParticlesInfo(polymer1(i),3)==L) THEN
    ParticlesInfo(polymer1(i),3)=1
  ELSE
    ParticlesInfo(polymer1(i),3)=ParticlesInfo(polymer1(i),3)+1
  ENDIF
Elseif (direction==6) THEN
  IF (ParticlesInfo(polymer1(i),3)==1) THEN
    ParticlesInfo(polymer1(i),3)=L
  ELSE
    ParticlesInfo(polymer1(i),3)=ParticlesInfo(polymer1(i),3)-1
  ENDIF
ELSE
  write (*,*) 'End of the world'
ENDIF
ENDDO !Updating Polymer1 particles

call FindNeighbors() !Updating new neighbors
end subroutine MovePolymer

***************************************************************************

LOGICAL FUNCTION ISPartjPolymer1(Partj)
! Is Partj part of polymer1?
use vars
IMPLICIT NONE

INTEGER, INTENT(IN) :: Partj
INTEGER :: i

ISPartjPolymer1 = .FALSE. ! Innocent unless proven the opposite
  i = 0
DO while (i< NPol1 .and. .not. ISPartjPolymer1)
  i=i+1
  IF (polymer1(i)==abs(Partj)) ISPartjPolymer1 = .TRUE.
ENDDO
RETURN
END FUNCTION ISPartjPolymer1
***************************************************************************

***************************************************************************

SUBROUTINE FindPolymer1(Pivot)
! Recurrently finds the polymer starting from the neighbors bonded to the pivot
use vars
IMPLICIT NONE

INTEGER, INTENT(IN) :: Pivot
INTEGER :: posn, dir,i,j, NNewList
INTEGER, DIMENSION(6) :: NewList
LOGICAL :: ISPartPolymer1temp
APPENDIX A. PROGRAM CODES

posn=1
polymer1 = (/ (0,i=1,N) /)
polymer1(posn)=Pivot
Npol1=1

DO while(posn <= Npol1) ! if all the neighbors of all the particles of the polymer are already included, that is the polymer
    NewList = (/ (0,i=1,6) /)
    NNewList = 0
    DO dir=1,6 ! check from neighbors which are bonded
        IF (ParticlesInfo(polymer1(posn),3+dir) < 0 ) THEN
            ISPartPolymertemp = .FALSE.
            j = 0
            DO while (j<NPol1 .and. .not. ISPartPolymertemp) ! check that particles are not repeated
                j=j+1
                IF (polymer1(j)== -ParticlesInfo(polymer1(posn),3+dir)) &
                    ISPartPolymertemp = .TRUE.
            ENDDO
            IF (.NOT.ISPartPolymertemp) THEN ! add to list to add
                NNewList = NNewList+1
                NewList(NNewList) = -ParticlesInfo(polymer1(posn),3+dir)
            ENDIF
        ENDIF
    END DO
    IF (NNewList >= 1) THEN ! add list to polymer1
        polymer1(Npol1+1:Npol1+NNewList) = NewList(1:NNewList)
        Npol1=Npol1+NNewList
    END IF
    posn=posn+1 ! proceed to next particle and check which of the bonded particles can be added
END DO

END SUBROUTINE FindPolymer1

!**********************************************************
!**********************************************************
SUBROUTINE FindPolymer2(Pivot)
! see FindPolymer1
use vars
IMPLICIT NONE

INTEGER, INTENT(IN) :: Pivot
INTEGER :: posn, dir, i, j, NNewList
INTEGER, DIMENSION(6) :: NewList
LOGICAL :: ISPartPolymertemp

posn=1
polymer2 = (/ (0,i=1,N) /)
polymer2(posn)=Pivot
Npol2=1

DO while(posn <= Npol2) ! if all the neighbors of all the particles of the polymer are already included, that is the polymer
    NewList = (/ (0,i=1,6) /)
    NNewList = 0
    DO dir=1,6 ! check from neighbors which are bonded
        IF (ParticlesInfo(polymer2(posn),3+dir) < 0 ) THEN
            ISPartPolymertemp = .FALSE.
            j = 0
            DO while (j<NPol2 .and. .not. ISPartPolymertemp) ! check that particles are not repeated
                j=j+1
                IF (polymer2(j)== -ParticlesInfo(polymer2(posn),3+dir)) &
                    ISPartPolymertemp = .TRUE.
            ENDDO
            IF (.NOT.ISPartPolymertemp) THEN ! add to list to add
                NNewList = NNewList+1
                NewList(NNewList) = -ParticlesInfo(polymer2(posn),3+dir)
            ENDIF
        ENDIF
    END DO
    IF (NNewList >= 1) THEN ! add list to polymer2
        polymer2(Npol2+1:Npol2+NNewList) = NewList(1:NNewList)
        Npol2=Npol2+NNewList
    END IF
    posn=posn+1 ! proceed to next particle and check which of the bonded particles can be added
END DO

END SUBROUTINE FindPolymer2
NEWLIST(NNEWLIST) = -PARTICLESINFO(POLYMER2(POSN),3+DIR)
ENDIF
END IF
END DO

IF (NNEWLIST >= 1) THEN
  POLYMER2(NPOL2+1:NPOL2+NNEWLIST) = NEWLIST(1:NNEWLIST)
  NPOL2=NPOL2+NNEWLIST
ENDIF
POSN=POSN+1
END DO

END SUBROUTINE FINDPOLYMER2

!****************************
!***************************************************************

INTEGER FUNCTION CHOOSEPARTICLE()
! Choose a particle with probability distribution BPArray
USE vars
IMPLICIT NONE

INTEGER :: i ! dummy counter
REAL :: AccumulativeSum
REAL :: valuer

AccumulativeSum=0.0
CALL RANDOM_NUMBER(valuer)
!value=rand()
i = 1
DO while (AccumulativeSum < valuer .and. AccumulativeSum<=1.0 .and. i<N)
  AccumulativeSum=AccumulativeSum+BPArray(i)
i=i+1
END DO

CHOOSEPARTICLE = i
RETURN
END FUNCTION CHOOSEPARTICLE

!***************************************************************
SUBROUTINE BP()
! Calculates probability distribution following Brownian motion
USE vars
IMPLICIT NONE
INTEGER :: i
REAL :: suma

suma=0.0
! Brownian Model
DO i=1,N
  ! This is the Brownian Model Based of Einstein-Smoluchowski (MAZO)
  BPArray(i)=1.0/(PARTICLESINFO(i,10)*PARTICLESINFO(i,11)**(1.0/6.0))
  ! This is a Stokes friction model.
  ! BPArray(i)=1.0/PARTICLESINFO(i,10)**(1.0/6.0)
  suma=suma+BPArray(i)
END DO

BPArray=BPArray/suma

END SUBROUTINE BP

!***************************************************************
SUBROUTINE FINDNEIGHBORS()
! Updates for all particles at unbonded directions which particles are at the neighborhood
USE vars
IMPLICIT NONE

INTEGER :: i,j, dirij !dummy counter
INTEGER :: Xi, Xj, Yi, Yj, Zi, Zj ! dummy coordinates
LOGICAL :: PositionRight

INTEGER :: InvDir

WHERE (ParticlesInfo(:,4:9) > 0 )
    ParticlesInfo(:,4:9) = 0
END Where

DO i=1,N
    Xi=ParticlesInfo(i,1)
    Yi=ParticlesInfo(i,2)
    Zi=ParticlesInfo(i,3)
    DO j=1,N
        IF (j/=i) THEN
            Xj=ParticlesInfo(j,1)
            Yj=ParticlesInfo(j,2)
            Zj=ParticlesInfo(j,3)
            IF (Xj == 0 .or. Yj == 0 .or. Zj == 0) THEN
                write (*,*), 'Error in position @ FindNeighbors (j)'
                STOP
            endif
            IF (ParticlesInfo(i,4)==0) THEN
                IF (PositionRight(Xi,Xj,1,L) .AND. Yi==Yj .AND. Zi==Zj) &
                    ParticlesInfo(i,4)=j !right i.e. x+
            endif
            IF (ParticlesInfo(i,5)==0) THEN
                IF(PositionRight(Xi,Xj,-1,L) .AND. Yi==Yj .AND. Zi==Zj) &
                    ParticlesInfo(i,5)=j !left i.e. x-
            endif
            IF (ParticlesInfo(i,6)==0) THEN
                IF (Xi==Xj .AND. PositionRight(Yi,Yj,1,L) .AND. Zi==Zj) &
                    ParticlesInfo(i,6)=j !right i.e. y+
            endif
            IF (ParticlesInfo(i,7)==0) THEN
                IF (Xi==Xj .AND. PositionRight(Yi,Yj,-1,L) .AND. Zi==Zj) &
                    ParticlesInfo(i,7)=j !left i.e. y-
            endif
            IF (ParticlesInfo(i,8)==0) THEN
                IF (Xi==Xj .AND. Yi==Yj .AND. PositionRight(Zi,Zj,1,L)) &
                    ParticlesInfo(i,8)=j !left i.e. z+
            endif
            IF (ParticlesInfo(i,9)==0) THEN
                IF (Xi==Xj .AND. Yi==Yj .AND. PositionRight(Zi,Zj,-1,L)) &
                    ParticlesInfo(i,9)=j !left i.e. z-
            endif
        endif
    END DO !j
END DO ! i
END SUBROUTINE FindNeighbors

LOGICAL FUNCTION PositionRight(X1,X2,ToTheRight,L)
! Evaluates if a particle X2 is at the right (=1) or left (-1) of particle X1
! for Periodic Boundary Conditions
IMPLICIT NONE
INTEGER, INTENT(IN) :: X1, X2, ToTheRight, L

If (ToTheRight==1) THEN
    IF (X2-X1 == 1 ) THEN
        PositionRight = .TRUE.
    ELSEIF ( (X1==L) .and. (X2==1) ) THEN
APPENDIX A. PROGRAM CODES

PositionRight = .TRUE.
ELSE
PositionRight = .false.
ENDIF
ELSEIF (ToTheRight == -1) THEN
IF ( X1-X2 == 1 ) THEN
PositionRight = .TRUE.
ELSEIF ( (X2==L) .and. (X1==1) ) THEN
PositionRight = .TRUE.
ELSE
PositionRight = .false.
ENDIF
ENDif
RETURN
END FUNCTION PositionRight

!**************************************************************
! ================ GENERAL Standard Functions =========== ======
!**************************************************************
INTEGER FUNCTION Random16generator()

REAL::HARVEST
INTEGER :: valuer

CALL RANDOM_NUMBER(HARVEST)
valuer=6*HARVEST+1
Random16generator=valuer
RETURN
END FUNCTION Random16generator

!**************************************************************
!**************************************************************
INTEGER FUNCTION InvDir(num)
! finds inverse direction of direction num
INTEGER, INTENT(IN) :: num

IF (num==1) THEN
InvDir = 2
ELSEIF (num==2) THEN
InvDir = 1
ELSEIF (num==3) THEN
InvDir = 4
ELSEIF (num==4) THEN
InvDir = 3
ELSEIF (num==5) THEN
InvDir = 6
ELSEIF (num==6) THEN
InvDir = 5
ELSE
write(*,*) 'Error INVDIR'
STOP
ENDIF
RETURN
END FUNCTION InvDir

!**************************************************************
!**************************************************************
INTEGER FUNCTION CoordinationNumber(num)
! Counts the number of particles bonded to a particle
use vars
IMPLICIT NONE

INTEGER, INTENT(IN) :: num
INTEGER :: i, counter

counter=0

do i=1,6
   IF (ParticlesInfo(num,i+3)<0) THEN
      counter=counter+1
   end if
end do

CoordinationNumber=counter

RETURN
END FUNCTION CoordinationNumber

!**********************************************************

REAL FUNCTION Energy()
! Calculates energy based on the number of formed bonds
USE vars
IMPLICIT NONE

integer :: i
integer :: CoordinationNumber

Energy=0.0

do i=1,N
   Energy=Energy+1.0*(CoordinationNumber(i))
end do

Energy = -Energy

RETURN

END FUNCTION Energy

!**********************************************************

! STARTING FUNCTIONS

SUBROUTINE Input_Parameters() 
use vars
IMPLICIT NONE

INTEGER::status
CHARACTER(10)::label01,label02,label03,label04,label05,label06

OPEN (UNIT=1,FILE=INPUT_FileName, STATUS='OLD', ACTION='READ', &
IOMAIN=status)
!Was the OPEN successful?
IF (status==0) THEN !open successful
   read(1,*) label01,w
   !write(*,*) label01,w
   read(1,*) label02,N
   !write(*,*) label02,N
   read(1,*) label03,L
   !write(*,*) label03,L
   read(1,*) label04,Tmax
   !write(*,*) label04,Tmax
   read(1,*) label05, OnlyLast
ENDIF
APPENDIX A. PROGRAM CODES

!write(*,*) label05,OUTPUT_FileName
ELSE
    WRITE(*,'(1X,A,A,I6)') INPUT_FileName, \
        ' Failed open. Failed-status = ', status
END IF
CLOSE (1)

END SUBROUTINE Input_Parameters

!**************************************
!****************************************
SUBROUTINE Initialize()
! Affected public variables:
! 1) ParticlesInfo (initialized)
! 2) N (initialized)

USE vars
IMPLICIT NONE

INTEGER::istat
INTEGER::i,j,k
REAL::value
REAL::LocProb
INTEGER::counter

!CALL Random_seed()
counter = 0
LocProb=1.0*N/(1.0*L)**3.0 !defining probability of locating a particle at each site

!WRITE (*,*) '
allocate(tempParticlesInfo(1,11))
tempParticlesInfo(1,:)=(/ (0 ,i=1,11) /)
DO i=1,L
    DO j=1,L
        DO k=1,L
            CALL RANDOM_NUMBER(value)
            !value=rand()
            IF (value.le.LocProb) THEN !Adds a particle at position (i,j,k)
                counter=counter+1
                ALLOCATE(ParticlesInfo(counter,11), STAT=istat)
                ParticlesInfo(i:counter-1,:)=-tempParticlesInfo
                deallocate(tempParticlesInfo)
                ParticlesInfo(counter,1)=i ! PosX
                ParticlesInfo(counter,2)=j ! PosY
                ParticlesInfo(counter,3)=k ! PosZ
                ParticlesInfo(counter,4)=k ! neighbor +x if negative it is bonded. 0 means NONE
                ParticlesInfo(counter,5)=k ! neighbor -x if negative it is bonded. 0 means NONE
                ParticlesInfo(counter,6)=k ! neighbor +y if negative it is bonded. 0 means NONE
                ParticlesInfo(counter,7)=k ! neighbor -y if negative it is bonded. 0 means NONE
                ParticlesInfo(counter,8)=k ! neighbor +z if negative it is bonded. 0 means NONE
                ParticlesInfo(counter,9)=k ! neighbor -z if negative it is bonded. 0 means NONE
                ParticlesInfo(counter,10)=1 ! Mass of the polymer where it is contained
                ParticlesInfo(counter,11)=6 ! Area of the polymer where it is contained
                allocate(tempParticlesInfo(counter,11))
tempParticlesInfo=ParticlesInfo
deallocate(ParticlesInfo)
            END IF
        END DO !z
    END DO !y
END DO !x
!WRITE (*,*) '
allocate(ParticlesInfo(counter,11))
ParticlesInfo=-tempParticlesInfo
deallocate(tempParticlesInfo)
N=counter !Redefining the number of particles

END SUBROUTINE Initialize
SUBROUTINE init_random_seed(ID)
INTEGER, intent(IN) :: ID
INTEGER :: i, kn, clock
INTEGER, DIMENSION(:), ALLOCATABLE :: seed
CALL RANDOM_SEED(size = kn)
ALLOCATE(seed(kn))
CALL SYSTEM_CLOCK(COUNT=clock)
seed = clock+1E6*ID + 37 * (/ (i - 1, i = 1, kn) /)
CALL RANDOM_SEED(put = seed)
DEALLOCATE(seed)
END SUBROUTINE

SUBROUTINE createnode(ID)
INTEGER, intent(IN) :: ID
integer :: res
CHARACTER*2 :: chID
character*256 :: command
WRITE(chID,'(i2.2)') ID
command='mkdir '//'NODE'//chID
call system(command)
END SUBROUTINE

SUBROUTINE createnodetime(ID, t)
INTEGER, intent(IN) :: ID, t
integer :: res
CHARACTER*2 :: chID
CHARACTER*8 :: cht
character*256 :: command
WRITE(chID,'(i2.2)') ID
WRITE(cht,'(i8.8)') t
command='mkdir '//'NODE'//chID//' '/cht
call system(command)
END SUBROUTINE

SUBROUTINE SavingData(ID, t)
use vars
IMPLICIT NONE
INTEGER, intent(in) :: ID, t
CHARACTER(len=100) :: OUTPUT_FileName
INTEGER :: i,j, status! dummy counter
APPENDIX A. PROGRAM CODES

CHARACTER*2 :: chID
CHARACTER*8 :: cht

WRITE(chID,'(i2.2)') ID
WRITE(cht,'(i8.8)') t

OUTPUT_FileName = 'Structure.dat'
OUTPUT_FileName = 'NODE'//chID//'/'//cht//'/'//OUTPUT_FileName
OPEN (UNIT=1,FILE=OUTPUT_FileName, STATUS='REPLACE', &
ACTION='WRITE', IOSTAT=status)

DO i=1,N
  WRITE (1,*) ParticlesInfo(i,1:3)
END DO
CLOSE(1)

END SUBROUTINE SavingData

SUBROUTINE SavingMassAreaData(ID, t)
  ! Affected public variables:
  ! (none)
  use vars
  IMPLICIT NONE

  INTEGER, intent(in) :: ID, t
  CHARACTER(len=100) :: OUTPUT_FileName, OUTPUT_FileName_Con
  INTEGER :: i,j, status! dummy counter
  CHARACTER*2 :: chID
  CHARACTER*8 :: cht
  INTEGER :: CoordinationNumber

  WRITE(chID,'(i2.2)') ID
  WRITE(cht,'(i8.8)') t

  OUTPUT_FileName_Con = 'Connectivity.dat'
  OUTPUT_FileName = 'MassArea.dat'
  OUTPUT_FileName_Con = 'NODE'//chID//'/'//cht//'/'//OUTPUT_FileName
  OPEN (UNIT=1,FILE=OUTPUT_FileName, STATUS='REPLACE', &
  ACTION='WRITE', IOSTAT=status)
  OPEN (UNIT=2,FILE=OUTPUT_FileName_Con, STATUS='REPLACE', &
  ACTION='WRITE', IOSTAT=status)

  DO i=1,N
    WRITE (1,*) ParticlesInfo(i,10:11), CoordinationNumber(i)
    WRITE (2,*) ParticlesInfo(i,4:9), CoordinationNumber(i)
  END DO

  CLOSE(1)
  CLOSE(2)

END SUBROUTINE SavingMassAreaData
A.4 Scattering Algorithm Matlab Code

function [Iq] = ScatteringAlgorithm(StructureIn,datacluster, downloadpath)
% written by Fritz A Campo @ WVU MAE 04/14/2011
% This function calculates the Scattering Function for the structures in
% the cell Structure, np cell of number of processors used, L size of box
% cell used for the simulation of Structure
% 
% The program: 0) Creates tempStructure.dat file from Structure
% 1) Copies Program and files in Cluster
% 2) Compiles Program
% 3) Execute simulation
% 4) Brings data back
% 5) Cleans Cluster
% 6) Go to back to 0 till all the cells are covered.
% 
% RETURNS: 1) Sq cell of Scattering functions
% 2) q cell of wave vectors
% 
% REQUIREMENTS:
% uses SCATT_MPI.f90
% uses ganymed-sshbuild250
% the directory /home/,’UserinCluster,’/ScatteringExp needs to exist at
% the cluster with SCATT_MPI.f90 compiled (mpif90 SCATT_MPI.f90 -o Scatt.exe)
% machines.txt also needs to be at /home/,’UserinCluster,’/ScatteringExp

[NoTests, np] = size(StructureIn);
% Setup the connection with the remote server SCP
% ssh service must be installed in Matlab-workspace
import ch.ethz.ssh2.SCPClient;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import ch.ethz.ssh2.Connection;
import ch.ethz.ssh2.Session;
import ch.ethz.ssh2.StreamGobbler;
import ch.ethz.ssh2.ChannelCondition;
try
    channel = Connection(datacluster.host);
    channel.connect();
catch
    error(’[Error: SCPTOMATLAB could not connect to the...
        ’ remote machine %s ...’],datacluster.host);
end
% Check the authentication for login...
isAuthenticated = channel.authenticateWithPassword(datacluster.user,datacluster.password);
if(~isAuthenticated)
    error...
        ([’Error: SCPTOMATLAB could not authenticate the’,...
        ’ SSH connection...’]);
end

counter=0;
Nqvals=200;

% Open session
scp1 = SCPClient(channel);
% Execution of Scattering over cases
for iTest=1:NoTests
 for Node=0:np-1
 % Create Structure.dat file
 dlmwrite([downloadpath,’\Structure.dat’], StructureIn(iTest,Node+1).Structure);
 % uploading Structure file to /home/,’UserinCluster,’/ScatteringExp
 scp1.put([downloadpath,’\Structure.dat’], ['/home/,’datacluster.user,’/ScatteringExp'])

 % create qvals
\begin{verbatim}
q0=pi/(2*StructureIn{iTest,Node+1}.L);
qf=pi/2;
qvals=10.^(log10(q0):(log10(qf)-log10(q0))/(Nqvals-1):log10(qf));
dlmwrite([downloadpath, '\qvals.dat'], qvals, 'precision', '%10.4E');

% uploading Q values
scp1.put([downloadpath, '\qvals.dat'], ['/home/', datacluster.user, '/ScatteringExp']);

% running cluster using MPI
CmdStr=['cd ScatteringExp; mpirun -np ', num2str(np), ' -machinefile machines.txt Scatt.exe; cd ..'];
channelssh2 = channel.openSession();
channelssh2.execCommand(CmdStr);
channelssh2.waitForCondition(ChannelCondition.EOF, 100000);
channelssh2.close();

% bringing results
scp1.get(['/home/', datacluster.user, '/ScatteringExp/output.dat'], downloadpath);
display(['download... ', '/home/', datacluster.user, '/ScatteringExp/output.dat']);

% Importing data to Matlab
Iq{iTest,Node+1}=load([downloadpath, '\output.dat']);
display(['Node ', num2str(Node)]);
end
end

% Closing connections
channel.close();

% Cleaning downloadpath dir
eval(['!del ', downloadpath, '\*.dat']);
\end{verbatim}
A.5 Scattering Code in Fortran

! SCATT_MPI.f90
!
Created by: Marco Maurier and Fritz A Campo @ WVU July 2009
Compile with >>mpif90 SCATT_MPI.f90 -o Scatt.exe
Run with >> mpirun -machinefile /home/frcampo/ScatteringExp/machines.txt -np 32 Scatt.exe
!
! Modifications:
!
Date | Comments
| =========== | ========================================================
| 02/09/10 | Sorting of Output and Reviewing and Input file name
| 02/12/10 | Output and fixing resolution

******************************************************************************
MODULES: VARS
******************************************************************************

module vars
IMPLICIT NONE
! Declaration of Basic Parameters
REAL,PARAMETER::PI=3.14159265358979323846

!Files
CHARACTER(len=20)::INPUT_FileName
CHARACTER(len=20)::Input_Structure
CHARACTER(len=20)::Input_Qs
CHARACTER(len=20)::Output_FileName

! Declaration of Particle Structure and Q_Vals variables
INTEGER :: No_Particles, No_Q
REAL, DIMENSION(:,::), pointer :: ParticleArray
REAL, DIMENSION (:), pointer :: Q_vals, TempQ, outputArray
real, dimension(:,::), pointer :: recvArray, Iq
!mpi variables
integer :: rank,ierr,numprocs,rc
integer :: fh1,fh2,fh3,modeRead,modeWrite
SAVE
end module vars

******************************************************************************
PROGRAM: SCATTERING
******************************************************************************

PROGRAM SCATT_MPI
USE vars
IMPLICIT NONE
include 'mpif.h'
! Variables
INTEGER:: istat
INTEGER:: iqstat, itempstat
integer :: i, j, k, l, tag=1, bufsize=20
integer stat(MPI_STATUS_SIZE), num
integer :: rankStart
integer :: rankEnd
integer :: Copypos

call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
write (*,*) 'Rank = ',rank,' numprocs = ',numprocs,' ierr = ',ierr

! ============ Body of SCATTERING ============

! Only root reads files
if(rank==0) then
  ! write(*,*) '======== Parallel Scattering Code ======='
  INPUT_FileName='InputParameters.txt'
  CALL Input_Parameters()
  CALL Input_Structure_File(Input_Structure)
end if

! Broadcast the number of particles
call MPI_BCast(No_Particles,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

! Nodes allocate space before receiving array
if(rank>0) allocate(ParticleArray(No_Particles,3))
! Broadcast array of particles
call MPI_BCast(ParticleArray,No_Particles*3,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

! Only root reads files
if(rank==0) then
  CALL Input_Q(Input_Qs)
end if

! Broadcast the number of Q values
call MPI_BCast(No_Q,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

! Nodes allocate space before receiving array
if(rank>0) allocate(Q_vals(No_Q))
! Broadcast array of Q values
call MPI_BCast(Q_vals,No_Q,MPI_REAL,0,MPI_COMM_WORLD,ierr)

! Evaluation
CALL Scatt(rank)

! ============ Bringing Data to root ============

if (rank>0) then
  call MPI_Send( outputArray, (rankEnd(rank)-rankStart(rank)+1) ,MPI_REAL, 0, tag, MPI_COMM_WORLD,ierr )
end if

if (rank==0) then
  allocate(Iq(No_Q))
  ! Copy values into Iq
  Copypos=1
  ForRoot:do k=1,(rankEnd(0)-rankStart(0)+1)
    Iq(Copypos)=outputArray(k)
    Copypos=Copypos+1
  end do ForRoot
  NotRoot:do k=1,numprocs-1
    allocate(recvArray(rankEnd(k)-rankStart(k)+1) )
  end do NotRoot
end if
call MPI_Recv(recvArray,rankEnd(k)-rankStart(k)+1,MPI_REAL, k ,tag, MPI_COMM_WORLD,stat,ierr)
FromRoot:do l=1,rankEnd(k)-rankStart(k)+1
   Iq(Copypos)=recvArray(l)
   Copypos=Copypos+1
end do FromRoot
end do NotRoot
end if

! ============= SAVING DATA =============

!Root receives arrays and writes to output file
if (rank==0) then !Root Saves
   open (UNIT=10, FILE=Output_FileName,STATUS='REPLACE',ACTION='WRITE',IOSTAT=istat)
   Saving:do i=1,No_Q
      write(10,*) Q_vals(i),Iq(i)
   end do Saving
   close(10)
   WRITE(*,*),'Output: ',Output_FileName
deallocate(Iq)
end if !Root Saves

call MPI_FINALIZE(rc)

!All nodes and root deallocate
deallocate(ParticleArray,Q_vals,outputArray)
END PROGRAM SCATT_MPI

************************************************************************

!SUBROUTINES: -Input_Structure_File ( format [ X Y Z ] )
!PURPOSE: Input data from file = 'filename'

! -Input_Q ( format [ X ] )
!PURPOSE: Input Q values from file = 'filename'

! -ScatteringQ
!PURPOSE: Returns I(q) for a given value q for the struct

! -Scatt
!PURPOSE: Calculates I(q) for the set of q's

************************************************************************

SUBROUTINE Input_Parameters()

use vars
IMPLICIT NONE

INTEGER::status
CHARACTER(10)::label01,label02,label03,label04,label05
OPEN (UNIT=1,FILE=INPUT_FileName, STATUS='OLD', ACTION='READ', IOSTAT=status)
!Was the OPEN successful?
fileopen: IF (status==0) THEN !open successful
   read(1,*) label01,Input_Structure
!   write(*,*) label01,w
   read(1,*) label02,Input_Qs

END SUBROUTINE Input_Parameters
SUBROUTINE Input_Parameters

  use vars
  IMPLICIT NONE

  CHARACTER(len=*)::FileName
  INTEGER::status, istat, i
  REAL:: tempPosX, tempPosY, tempPosZ

  WRITE(*,*), 'Input Structure from: ', FileName

  OPEN (UNIT=9, FILE=FileName, STATUS='OLD', ACTION='READ', IOSTAT=istat)
  !Was the OPEN successful?
  IF (istat==0) THEN !open successful
    No_Particles=0
    DO
      READ (9,*,IOSTAT=istat), tempPosX, tempPosY, tempPosZ
      IF (istat/=0) EXIT
      No_Particles=No_Particles+1
    ! WRITE(*,*), tempPosX, tempPosY, tempPosZ
    END DO
    REWIND (Unit=9)
    ALLOCATE (ParticleArray(No_Particles,3))
    do i=1, No_Particles
      read(9,*,iostat=istat) tempPosX, tempPosY, tempPosZ
      ParticleArray(i,1)=tempPosX
      ParticleArray(i,2)=tempPosY
      ParticleArray(i,3)=tempPosZ
    END DO
  ELSE
    WRITE (*,'(1X,A,A,I6)'), FileName, 'File open failed-status = ', istat
  END IF

  CLOSE(9)

END SUBROUTINE Input_Parameters

SUBROUTINE Input_Structure_File(FileName)

  use vars
  IMPLICIT NONE

  CHARACTER(len=*)::FileName
  INTEGER::status, istat, i
  REAL:: tempPosX, tempPosY, tempPosZ

  WRITE(*,*), 'Input Structure from: ', FileName

  OPEN (UNIT=9, FILE=FileName, STATUS='OLD', ACTION='READ', IOSTAT=istat)
  !Was the OPEN successful?
  IF (istat==0) THEN !open successful
    No_Particles=0
    DO
      READ (9,*,IOSTAT=istat), tempPosX, tempPosY, tempPosZ
      IF (istat/=0) EXIT
      No_Particles=No_Particles+1
      ! WRITE(*,*), tempPosX, tempPosY, tempPosZ
    END DO
    REWIND (Unit=9)
    ALLOCATE (ParticleArray(No_Particles,3))
    do i=1, No_Particles
      read(9,*,iostat=istat) tempPosX, tempPosY, tempPosZ
      ParticleArray(i,1)=tempPosX
      ParticleArray(i,2)=tempPosY
      ParticleArray(i,3)=tempPosZ
    END DO
  ELSE
    WRITE (*,'(1X,A,A,I6)'), FileName, 'File open failed-status = ', istat
  END IF

  CLOSE(9)

END SUBROUTINE Input_Structure_File

SUBROUTINE Input_Q(FileName)

  use vars
  IMPLICIT NONE
APPENDIX A. PROGRAM CODES

CHARACTER (len=*) , INTENT(IN) :: FileName
INTEGER:::iierr, istat, i
REAL:: tempQval

WRITE(*,*), 'Input Q vals from: ', FileName

OPEN (UNIT=9,FILE=FileName,STATUS='OLD', ACTION='READ', IOSTAT=iierr)
! Was the OPEN successful?
IF (iierr==0) THEN ! open successful
   No_Q=0
   filesize:DO
      READ (9,*,IOSTAT=iierr),tempQval
      IF (iierr/=0) EXIT
      No_Q=No_Q+1
   END DO filesize
   ALLOCATE(Q_vals(No_Q))
   REWIND(UNIT=9)
   DO i=1,No_Q
      read(9,*,iostat=iierr), tempQval
      !write(*,*) tempQval
      Q_vals(i)=tempQval
   END DO
ELSE
   WRITE (*,'(1X,A,A,I6)') FileName,' File open failed-status = ',iierr
END IF
CLOSE(9)
END SUBROUTINE Input_Q

!********************
REAL FUNCTION ScatteringQ(qva)
! This is the basic Scattering Code: for a q it returns a I_q given
! No_Particles and ParticleArray defined in vars (global)
use vars
IMPLICIT NONE
REAL,PARAMETER::epsilon=0.00000001

!REAL, INTENT(OUT)::Iqexit
REAL, INTENT(IN)::qva
INTEGER::i,j, counter ! Counters
REAL :: Contribution, rij, Iqout, qvalor
REAL :: deltax, deltay, deltaz

Iqout=0.0
counter=0
qvalor=qva

DO i=1,No_Particles-1
   DO j=i+1,No_Particles
      deltax=1.0*ParticleArray(i,1)-1.0*ParticleArray(j,1)
      deltay=1.0*ParticleArray(i,2)-1.0*ParticleArray(j,2)
      deltaz=1.0*ParticleArray(i,3)-1.0*ParticleArray(j,3)
      rij=(deltax**2+deltay**2+deltaz**2)**0.5
      Contribution = 0.0
      IF ( qvalor*rij .le. epsilon ) THEN
         Contribution=2.0
      END IF
      Iqout=Iqout+Contribution
   END DO
END DO
ELSE
    Contribution=2.0*(sin(1.0*qvalor*rij)/(qvalor*rij))
END IF

Iqout=Iqout+Contribution
counter=counter+1
END DO

!If ( mod(i,10000) == 0.0 ) THEN
! println(*,*) (sin(1.0*qvalor*rij)/(qvalor*rij)) / (1.0*No_Particles)
!endif

END DO

Iqout=(Iqout/(1.0*No_Particles)+1.0)/(1.0*No_Particles) !Normalization of the Intensity
!write(*,*) qvalor,Iqout, No_Particles, counter

!IF (Iq<0.0) THEN
!   Iq=0.0
!END

ScatteringQ=Iqout
return

END FUNCTION ScatteringQ

SUBROUTINE Scatt()
USE vars
IMPLICIT NONE

integer :: tag=1
integer::i,j,rankEnd,rankStart, NoQs !Dummy counter
real::I_q, Qin,ScatteringQ

!rankEnd=(rank+1)*(No_Q/numprocs)
!rankStart=(rank*(No_Q/numprocs))+1
allocate(outputArray(rankEnd(rank)-rankStart(rank)+1)) ! This needs to be improved.
   It works if No_Q is a multiple of the numprocs
NoQs = rankEnd(rank)-rankStart(rank) +1
j=1
ScattEval:do i=rankStart(rank),rankEnd(rank)
    Qin=Q_vals(i)
    I_q=ScatteringQ(Qin)
    write(*,*) Qin, I_q,'item=', j,'from ', NoQs
    outputArray(j)=I_q
    j=j+1
end do ScattEval

END SUBROUTINE Scatt

SUBROUTINE rankStart(compu)
USE vars
IMPLICIT NONE

INTEGER, INTENT(IN) :: compu
integer :: temporankStart

temporankStart=compu*INT( No_Q/numprocs )
APPENDIX A. PROGRAM CODES

IF (compu<MOD(No_Q,numprocs)) THEN
  temporankStart=temporankStart+(compu+1)
ELSE
  temporankStart=temporankStart+(MOD(No_Q,numprocs)+1)
END IF

!write(*,*) compu,No_Q,temporankStart
rankStart=INT(temporankStart)
return

END FUNCTION rankStart

!**************************************
!**************************************

INTEGER FUNCTION rankEnd(compu)

USE vars
IMPLICIT NONE

INTEGER, INTENT(IN) :: compu
integer :: rankStart
integer :: temporankEnd

temporankEnd=rankStart(compu) + INT( No_Q/numprocs )

IF (compu<MOD(No_Q,numprocs)) THEN
  temporankEnd=temporankEnd+1
END IF

temporankEnd=temporankEnd-1
!write(*,*) compu,No_Q,temporankEnd
rankEnd=temporankEnd
return

END FUNCTION rankEnd

!**************************************


function [EffStiffness]=EffectiveStiffnessAlgorithm(StructureIn, FEMParameters, Stiffness)
% written by Fritz A Campo, MAE @ WVU 04/15/2011
% This program calculates the effective stiffness of the structures in
% StructureOut cell
% 1) Inputs data from StructureIn to Monomer,
% 2) Creates AnsysInput file.
% 3) Evaluates AnsysInput file
% 4) Brings energy data to Matlab and returns
% 5) Evaluates new Effective stiffness
% Requires: FindPolymer.m, ElementNodesP.m, ElementNodes.m,

[NoTests, np] = size(StructureIn);
% Preparing input file for ANSYS
% nameAnsys = FEMParameters.nameAnsys;
% dUtot = FEMParameters.dUtot;
% du = FEMParameters.du;
% mode = FEMParameters.mode;
% type = FEMParameters.type;
% dumode = FEMParameters.dumode;
% dirpath = FEMParameters.dirpath;
% nameout = FEMParameters.nameout;
% AnsysPath = FEMParameters.AnsysPath
ESize = FEMParameters.ESize;

for iTest = 1:NoTests;
  for Node = 0: np - 1
    % Temporal variable Monomer, and sizes L’s
    for i = 1:length(StructureIn{iTest,Node+1}.Structure(:,1))
      Monomer(i).PosX=StructureIn{iTest,Node+1}.Structure(i,1)-1;
      Monomer(i).PosY=StructureIn{iTest,Node+1}.Structure(i,2)-1;
      Monomer(i).PosZ=StructureIn{iTest,Node+1}.Structure(i,3)-1;
    end
    LX=StructureIn{iTest,Node+1}.L-1;
    LY=StructureIn{iTest,Node+1}.L-1;
    Lz=StructureIn{iTest,Node+1}.L-1;
    LX=StructureIn{iTest,Node+1}.L;
    LY=StructureIn{iTest,Node+1}.L;
    Lz=StructureIn{iTest,Node+1}.L;

    face1=find(StructureIn{iTest,Node+1}.Structure(:,1)==LX);
    face2=find(StructureIn{iTest,Node+1}.Structure(:,1)==1);
    face3=find(StructureIn{iTest,Node+1}.Structure(:,2)==LY);
    face4=find(StructureIn{iTest,Node+1}.Structure(:,2)==1);
    face5=find(StructureIn{iTest,Node+1}.Structure(:,3)==Lz);
    face6=find(StructureIn{iTest,Node+1}.Structure(:,3)==1);

    % creating ConnectivityMatrix
    long = length(Monomer);
    ConnectingMatrix=zeros(long,long);
    for i = 1:long
      for j = 1:6
        if (StructureIn{iTest,Node+1}.Connectivity(i,j) < 0)
          ConnectingMatrix(i,-StructureIn{iTest,Node+1}.Connectivity(i,j))=1;
        end
      end
    end
  end
end
% Find all elements that are effectively connected to the surface and
% delete the rest

disconnecting PBC's
for i=1:length(face1)
    for j=1:length(face2)
        if ConnectingMatrix(face1(i),face2(j))==1
            ConnectingMatrix(face1(i),face2(j))=0;
            ConnectingMatrix(face2(j),face1(i))=0;
        end
    end
end
for i=1:length(face3)
    for j=1:length(face4)
        if ConnectingMatrix(face3(i),face4(j))==1
            ConnectingMatrix(face3(i),face4(j))=0;
            ConnectingMatrix(face4(j),face3(i))=0;
        end
    end
end
for i=1:length(face5)
    for j=1:length(face6)
        if ConnectingMatrix(face5(i),face6(j))==1
            ConnectingMatrix(face5(i),face6(j))=0;
            ConnectingMatrix(face6(j),face5(i))=0;
        end
    end
end

newlist=[];
for i=1:length(face1)
    pol=[];
    pol=FindPolymer(face1(i),ConnectingMatrix);
    for j=1:length(pol)
        if isempty(find(newlist==pol(j)))
            newlist=[newlist,pol(j)];
        end
    end
end
for i=1:length(face2)
    pol=[];
    pol=FindPolymer(face2(i),ConnectingMatrix);
    for j=1:length(pol)
        if isempty(find(newlist==pol(j)))
            newlist=[newlist,pol(j)];
        end
    end
end
for i=1:length(face3)
    pol=[];
    pol=FindPolymer(face3(i),ConnectingMatrix);
    for j=1:length(pol)
        if isempty(find(newlist==pol(j)))
            newlist=[newlist,pol(j)];
        end
    end
end
for i=1:length(face4)
    pol=[];
    pol=FindPolymer(face4(i),ConnectingMatrix);
    for j=1:length(pol)
        if isempty(find(newlist==pol(j)))
            newlist=[newlist,pol(j)];
        end
    end
end
APPENDIX A. PROGRAM CODES

end
for i=1:length(face5)
    pol=[];
    pol=FindPolymer(face5(i),ConnectingMatrix);
    for j=1:length(pol)
        if isempty(find(newlist==pol(j)))
            newlist=[newlist,pol(j)];
        end
    end
end
for i=1:length(face6)
    pol=[];
    pol=FindPolymer(face6(i),ConnectingMatrix);
    for j=1:length(pol)
        if isempty(find(newlist==pol(j)))
            newlist=[newlist,pol(j)];
        end
    end
end
Monomer2=Monomer;
clear Monomer
for i=1:length(newlist)
    Monomer(i).PosX=Monomer2(newlist(i)).PosX;
    Monomer(i).PosY=Monomer2(newlist(i)).PosY;
    Monomer(i).PosZ=Monomer2(newlist(i)).PosZ;
end
clear Monomer2
ConnectingMatrix2=zeros(length(newlist),length(newlist));
for i=1:length(newlist)
    for j=i+1:length(newlist)
        if ConnectingMatrix(newlist(i),newlist(j))==1
            ConnectingMatrix2(i,j)=1;
            ConnectingMatrix2(j,i)=1;
        end
    end
end
ConnectingMatrix=ConnectingMatrix2;
clear ConnectingMatrix2

% Now, find the elements whose associated polymers are well constrained

% Writing Material to file
fid = fopen([FEMParameters.dirpath,char(filesep),FEMParameters.nameAnsys],'w');
if FEMParameters.FullyConnected
    try
        % Opening Preprocessor: Definition of Element type, Material Properties
        fprintf(fid,'! ANSYS v9.0 Input File \n');
        fprintf(fid,'! Created by: Fritz Andres Campo Schickler \n');
        fprintf(fid,'! Date: %s \n',date);
        vectime=clock;
        fprintf(fid,'! Time: %d:%d:%2.0f \n \n',vectime(4),vectime(5),vectime(6));
        fprintf(fid,'/SOLU \n');
        fprintf(fid,'ANTYPE,0 \n');
        fprintf(fid,'NLGEOM,1 \n');
        %fprintf(fid,'AUTOTS,ON \n');
        fprintf(fid,'NSUBST,10,,,OFF \n');
        fprintf(fid,'/prep7 \n');
    catch
        fprintf(fid,'%s
',ferror);
    end
end

% Change of ANSYS Working Directory: It is the Matlab Working directory
% plus the dirpath directory

% nonlinear behaviour
%
fprintf(fid,'/prep7 \n');
fprintf(fid,'\n');
APPENDIX A. PROGRAM CODES

%------- Element Type and Material Properties
fprintf(fid,'ET,1,SOLID64 \n');
fprintf(fid,'R,1, %E,%E,%E,%E,%E \n 
',MechanicalParameters.Area,MechanicalParameters.Izz,...
MechanicalParameters.Iyy,MechanicalParameters.Tkz,MechanicalParameters.Tky);

fprintf(fid,'TB,ANEL,1,1,21,0 \n');
fprintf(fid,'TBTEMP,0 \n');

fprintf(fid,'TBDATA,1,%E,%E,%E,%E,%E,%E \n',Stiffness(1,1),Stiffness(1,2),Stiffness(1,3),Stiffness(1,4),Stiffness(1,5),Stiffness(1,6));
fprintf(fid,'TBDATA,7,%E,%E,%E,%E,%E,%E \n',Stiffness(2,2),Stiffness(2,3),Stiffness(2,4),Stiffness(2,5),Stiffness(2,6),Stiffness(3,3));
fprintf(fid,'TBDATA,13,%E,%E,%E,%E,%E,%E \n',Stiffness(3,4),Stiffness(3,5),Stiffness(3,6),Stiffness(4,4),Stiffness(4,5),Stiffness(4,6));
fprintf(fid,'TBDATA,19,%E,%E,%E \n',Stiffness(5,5),Stiffness(5,6),Stiffness(6,6));
fprintf(fid,'\n');
fprintf(fid,'/SOLU \n');
fprintf(fid,'ANTYPE,0 \n');
fprintf(fid,'NLGEOM,1 \n'); % Large Deformations
fprintf(fid,'FINISH \n');

%------------------ Writing Structure to file ------------------

%------- Introduction of Nodes (KeyPoints) and definition of connecting
%Elements (Solids)
[Element_Nodes,polymer]=ElementNodesP(ConnectingMatrix,Monomer); % simple filling of array
[[Element_Nodes,polymer]=ElementNodesP(ConnectingMatrix,Monomer); % All are
% connected

% ---- writing Elements (Solids) into file ----
long=length(Element_Nodes(:,1)); % Not All particles need to be connected somehow
PosNodes=[];
Node_counter=1;
for i=1:long
    for j=1:8
        if Element_Nodes(i,j)==Node_counter
            switch j
            case 1
                fprintf(fid,'K,%d,%f,%f,%f \n',Node_counter,...
                Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY-0.5,
                Monomer(polymer(i)).PosZ-0.5);
                PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
                Monomer(polymer(i)).PosY-0.5,
                Monomer(polymer(i)).PosZ-0.5,Node_counter];
            case 2
                fprintf(fid,'K,%d,%f,%f,%f \n',Node_counter,...
                Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY+0.5,
                Monomer(polymer(i)).PosZ-0.5);
                PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
                Monomer(polymer(i)).PosY+0.5,
                Monomer(polymer(i)).PosZ-0.5,Node_counter];
            case 3
                fprintf(fid,'K,%d,%f,%f,%f \n',Node_counter,...
                Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY+0.5,
                Monomer(polymer(i)).PosZ-0.5);
                PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
                Monomer(polymer(i)).PosY+0.5,
                Monomer(polymer(i)).PosZ-0.5,Node_counter];
            case 4
                fprintf(fid,'K,%d,%f,%f,%f \n',Node_counter,...
                Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY-0.5,
                Monomer(polymer(i)).PosZ-0.5);
                PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
                Monomer(polymer(i)).PosY-0.5,
                Monomer(polymer(i)).PosZ-0.5,Node_counter];
            end
        end
    end
end
APPENDIX A. PROGRAM CODES

case 5
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,
Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
case 6
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,
Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
case 7
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,
Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
case 8
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,
Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
end %end switch case
Node_counter=Node_counter+1;
end % end if for Node generation
end % end for

fprintf(fid,'V, %d, %d, %d, %d, %d, %d, %d, %d 
',Element_Nodes(i,1),
Element_Nodes(i,2),Element_Nodes(i,3),Element_Nodes(i,4),
Element_Nodes(i,5),Element_Nodes(i,6),Element_Nodes(i,7),Element_Nodes(i,8));
end % end for elements

fprintf(fid,'
');
% Defining size of mesh and meshing (before applying loads)
fprintf(fid,'ESIZE,%g 
',ESize);
fprintf(fid,'VMESH,ALL 
');
fprintf(fid,'
');
% Defining Loads Depending on loading case
fprintf(fid,'/SOLU 
' );
fprintf(fid,'NSEL,ALL 
');
Numt=0;
%fprintf(fid,'PIVCHECK, OFF \n' ); % For the case of underconstrained model

for constrain=1:longconstrain
du=dumode(constrain,1);
mode=dumode(constrain,2);
fprintf(fid,'
');
end %end for
if constrain>1
  switch dumode(constrain,3)
  case 1
    fprintf(fid,'NSEL,ALL \n');
    fprintf(fid,'DCUM,ADD \n');
    AddString='DCUM,ADD \n';
  case 0
    fprintf(fid,'NSEL,ALL \n');
    fprintf(fid,'DCUM \n');
    AddString='DCUM \n';
  end
else
  fprintf(fid,'NSEL,ALL \n');
  AddString='DCUM \n';
end % adding BC's or replacing them
fprintf(fid,AddString);
switch mode
  case 1
dU=du*(Lx+1);
  fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
  fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
  fprintf(fid,'D,ALL, UX,%g \n',dU);
  fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
  fprintf(fid,'D,ALL,ALL,0 \n');
  fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
  fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
  fprintf(fid,'D,ALL, UX,%g \n',(X-(-0.5))*du*(Lx+1)/(Lx+1));
end % along x direction
  case 2
    dU=du*(Ly+1);
    fprintf(fid,'NSEL,S,LOC,Y,%g \n',Ly+0.5);
    fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
    fprintf(fid,'D,ALL, UX,%g \n',dU);
    fprintf(fid,'NSEL,S,LOC,Y,%g \n',-0.5);
    fprintf(fid,'D,ALL,ALL,0 \n');
    fprintf(fid,'NSEL,S,LOC,Y,%g \n',Ly+0.5);
    fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
    fprintf(fid,'D,ALL, UX,%g \n',(X-(-0.5))*du*(Ly+1)/(Ly+1));
end % along x direction
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 2
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 5
fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 6
fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
end % along y direction

case 3
dU=du*(Lz+1);
%Face 5 is being pulled uz
fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',Lz+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n \n',dU);
%Face 6 is totally constrained
fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',-0.5);
fprintf(fid,'D,ALL, ALL,0 \n');
% Face 1, 2, 3, 4 have the same stretching varying the position X
% dU(Z)=(Z-(-0.5))*du*(Lz+1)/(Lz+1)
for Z=-0.5+ESize:ESize:Lz+0.5-ESize
  fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',Z);
  fprintf(fid,'NSEL,R,LOC,Y,%g \n',Ly+0.5);
  fprintf(fid,'NSEL,R,LOC,X,%g \n\',-0.5+ESize,Lx+0.5-ESize);
  fprintf(fid,'D,ALL, UX,0,,,,UY \n');
  fprintf(fid,'D,ALL, UZ,%g \n \n',(Z-(-0.5))*du*(Lz+1)/(Lz+1));
  fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',Lz+0.5);
  fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
  fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
  fprintf(fid,'D,ALL, UY,%g \n \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
end % along z direction

case 4
dU=du*(Lz+1);
%Face 5 is being pulled uy
fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',Lz+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',dU);
%Face 6 is totally constrained
fprintf(fid,'NSEL,S,LOC,Z,Xg \n\',-0.5);
fprintf(fid,'D,ALL, ALL,0 \n');
% Face 1, 2, 3, 4 have the same stretching varying the position X
% dU(Z)=(Z-(-0.5))*du*(Lz+1)/(Lz+1)
for Z=-0.5+ESize:ESize:Lz+0.5-ESize
    % Face 1
    fprintf(fid,'NSEL,S,LOC,X,%g\n',Lx+0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 2
    fprintf(fid,'NSEL,S,LOC,X,%g\n',-0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 3
    fprintf(fid,'NSEL,S,LOC,Y,%g\n',Ly+0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'NSEL,R,LOC,X,%g,%g\n',-0.5+ESize,Lx+0.5-ESize);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 4
    fprintf(fid,'NSEL,S,LOC,Y,%g\n',-0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'NSEL,R,LOC,X,%g,%g\n',-0.5+ESize,Lx+0.5-ESize);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
end % along z direction

case 5
    dU=du*(Lz+1);
    % Face 5 is being pulled ux
    fprintf(fid,'NSEL,S,LOC,Z,%g\n',Z);
    fprintf(fid,'D,ALL, UY,0,,,,UY\n');
    fprintf(fid,'D,ALL, UX,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 6 is totally constrained
    fprintf(fid,'NSEL,S,LOC,Z,%g\n',-0.5);
    fprintf(fid,'D,ALL, ALL,0\n');
end

% Face 1, 2, 3, 4 have the same stretching varying the position X
% du(Z)=(Z-(-0.5))*du*(Lz+1)/(Lz+1)

for Z=-0.5+ESize:ESize:Lz+0.5-ESize
    % Face 1
    fprintf(fid,'NSEL,S,LOC,X,%g\n',Lx+0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 2
    fprintf(fid,'NSEL,S,LOC,X,%g\n',-0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 3
    fprintf(fid,'NSEL,S,LOC,Y,%g\n',Ly+0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'NSEL,R,LOC,X,%g,%g\n',-0.5+ESize,Lx+0.5-ESize);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 4
    fprintf(fid,'NSEL,S,LOC,Y,%g\n',-0.5);
    fprintf(fid,'NSEL,R,LOC,Z,%g\n',Z);
    fprintf(fid,'NSEL,R,LOC,X,%g,%g\n',-0.5+ESize,Lx+0.5-ESize);
    fprintf(fid,'D,ALL, UX,0,,,,UZ\n');
    fprintf(fid,'D,ALL, UY,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
end % along z direction

case 6
    dU=du*(Ly+1);
    % Face 3 is being pulled ux
    fprintf(fid,'NSEL,S,LOC,Y,%g\n',Ly+0.5);
    fprintf(fid,'D,ALL, UY,0,,,,UY\n');
    fprintf(fid,'D,ALL, UX,%g\n,(Z-(-0.5))*du*(Lz+1)/(Lz+1);');
    % Face 4 is totally constrained
    fprintf(fid,'NSEL,S,LOC,Y,%g\n',-0.5);
fprintf(fid,'D,ALL,ALL,0 \n');
% Face 1, 2, 5, 6 have the same stretching varying the position X
% dU(Y)=(Y-(-0.5))*du*(Ly+1)/(Ly+1)
for Y=-0.5+ESize:ESize:Ly+0.5-ESize
% Face 1
fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UY 
');
fprintf(fid,'D,ALL, UZ,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 2
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UY 
');
fprintf(fid,'D,ALL, UZ,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 5
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UY 
');
fprintf(fid,'D,ALL, UZ,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 6
fprintf(fid,'NSEL,S,LOC,Z,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UY 
');
fprintf(fid,'D,ALL, UZ,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
end % along y direction
case 7
dU=du*(Ly+1);
%Face 3 is being pulled ux
fprintf(fid,'NSEL,S,LOC,X,%g \n',Ly+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UY,0,,,,UZ 
');
fprintf(fid,'D,ALL, UX,%g 
 
',dU);
%Face 4 is totally constrained
fprintf(fid,'NSEL,S,LOC,Y,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,ALL,0 \n');
% Face 1, 2, 5, 6 have the same stretching varying the position X
% dU(Y)=(Y-(-0.5))*du*(Ly+1)/(Ly+1)
for Y=-0.5+ESize:ESize:Ly+0.5-ESize
% Face 1
fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g \n',Y);
fprintf(fid,'D,ALL, UY,0,,,,UZ 
');
fprintf(fid,'D,ALL, UX,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 2
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g \n',Y);
fprintf(fid,'D,ALL, UY,0,,,,UZ 
');
fprintf(fid,'D,ALL, UX,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 5
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UY,0,,,,UZ 
');
fprintf(fid,'D,ALL, UX,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
% Face 6
fprintf(fid,'NSEL,S,LOC,Z,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UY,0,,,,UZ 
');
fprintf(fid,'D,ALL, UX,%g 
 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
end % along y direction
case 8
dU=du*(Lx+1);
%Face 1 is being pulled uy
fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',dU);
%Face 2 is totally constrained
fprintf(fid,'NSEL,S,LOC,X,%g \n\n',-0.5);
fprintf(fid,'D,ALL,ALL,0 \n');
% Face 3, 4, 5, 6 have the same stretching varying the position X
% dU(X)=(X-(-0.5))*du*(Lx+1)/(Lx+1)
for X=-0.5+ESize:ESize:Lx+0.5-ESize
% Face 3
fprintf(fid,'NSEL,S,LOC,Y,%g \n',Ly+0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 4
fprintf(fid,'NSEL,S,LOC,Y,%g \n\n',-0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 5
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'NSEL,R,LOC,Y,%g,%g
',-0.5+ESize,Ly+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 6
fprintf(fid,'NSEL,S,LOC,Z,%g \n\n',-0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'NSEL,R,LOC,Y,%g,%g
',-0.5+ESize,Ly+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
end % along x direction
case 9
dU=du*(Lx+1);
%Face 1 is being pulled uz
fprintf(fid,'NSEL,S,LOC,X,%g \n\n',Lx+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',dU);
%Face 2 is totally constrained
fprintf(fid,'NSEL,S,LOC,X,%g \n\n',-0.5);
fprintf(fid,'D,ALL,ALL,0 \n');
% Face 3, 4, 5, 6 have the same stretching varying the position X
% dU(X)=(X-(-0.5))*du*(Lx+1)/(Lx+1)
for X=-0.5+ESize:ESize:Lx+0.5-ESize
% Face 3
fprintf(fid,'NSEL,S,LOC,Y,%g \n',Ly+0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 4
fprintf(fid,'NSEL,S,LOC,Y,%g \n\n',-0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 5
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'NSEL,R,LOC,Y,%g,%g
',-0.5+ESize,Ly+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 6
fprintf(fid,'NSEL,S,LOC,Z,%g \n\n',-0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'NSEL,R,LOC,Y,%g,%g
',-0.5+ESize,Ly+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
end % along x direction
end %switch mode
fprintf(fid,'NSEL,ALL \n');
APPENDIX A. PROGRAM CODES

fprintf(fid,' \n');
\%display('Writting Boundary Conditions....Ready!!');

% Solving
if constrain+1<=longconstrain
    if dumode(constrain+1,3)==1
        \% implement solve
        Numt=Numt+1;
        fprintf(fid,'TIME,%d \n',Numt);
        fprintf(fid,'SOLVE \n');
    end
else
    \% implement solve
    Numt=Numt+1;
    fprintf(fid,'TIME,%d \n',Numt);
    fprintf(fid,'SOLVE \n');
end \%end solving and timing. No time increase is done if the constrains are added!!!
end \%constrain for loop
fprintf(fid,'FINISH \n');

\%Post Processing given that the solution was done previously
fprintf(fid,'*dim,USimulation,array,%d \n',Numt);
fprintf(fid,'/post1 
');
fprintf(fid,'SET,FIRST \n');
for constrain=1:Numt
    fprintf(fid,'ETABLE,UTOTENE,SENE, \n');
    fprintf(fid,'SSUM,UTOTENE \n');
    fprintf(fid,'ETABLE,VOLUMEN,VOLU, \n');
    fprintf(fid,'SSUM,VOLUMEN \n');
    fprintf(fid,'*GET,UTOT,SSUM,,ITEM,UTOTENE \n');
    fprintf(fid,'USimulation(%d)= UTOT \n',constrain);
    fprintf(fid,'SET,NEXT \n');
end \%data retriving

\%Writing results file: solution.dat
fprintf(fid,'*cfopen, solution.dat \n');
fprintf(fid,'*vwrite,USimulation(1) \n');
fprintf(fid,'(G) \n');
fprintf(fid,'*cfclos 
');
fprintf(fid,'FINISH \n');
\%end \%try
elseif ~FEMParameters.FullyConnected
\%try
\% Opening Preprocessor: Definition of Element type, Material Properties
\% and Geometry.
fprintf(fid,'
');
fprintf(fid,'! ANSYS v9.0 Input File 
');
fprintf(fid,'!Created by: Fritz Andres Campo Schickler 
');
fprintf(fid,'!Date: %s 
',date);
vectime=clock;
fprintf(fid,'!Time: %d:%d:%2.0f 
 
',vectime(4),vectime(5),vectime(6));
\%nonlinear behaviour
\{%
fprintf(fid,'/SOLU 
' );
fprintf(fid,'ANTYPE,0 
');
fprintf(fid,'NLGEOM,1 
' );
%fprintf(fid,'AUTOTS,ON 
' );
fprintf(fid,'NSUBST,10,,,OFF 
' );
fprintf(fid,'/prep7 
' );
\%
fprintf(fid,'!*cfopen, solution.dat \n');
fprintf(fid,'*vwrite,USimulation(1) \n');
fprintf(fid,'(G) \n');
fprintf(fid,'*cfclos 
');
fprintf(fid,'FINISH \n');
\%end \%try
\%else
\% FEMParameters.FullyConnected
\%try
\% Opening Preprocessor: Definition of Element type, Material Properties
\% and Geometry.
\%\% fprintf(fid,'! ANSYS v9.0 Input File \n');
\%\% fprintf(fid,'!Created by: Fritz Andres Campo Schickler \n');
\%\% fprintf(fid,'!Date: %s \n',date);
\%\% vectime=clock;
\%\% fprintf(fid,'!Time: %d:%d:%2.0f \n ',vectime(4),vectime(5),vectime(6));
\%\%
\%\% \%nonlinear behaviour
\%\% \{%
\%\% fprintf(fid,'/SOLU \n' );
\%\% fprintf(fid,'ANTYPE,0 \n');
\%\% fprintf(fid,'NLGEOM,1 \n' );
\%\% fprintf(fid,'AUTOTS,ON \n ' );
\%\% fprintf(fid,'NSUBST,10,,,OFF \n' );
\%\% \%}
%------ Element Type and Material Properties
fprintf(fid,'ET,1,SOLID64 
');
fprintf(fid,'R,1, %E,%E,%E,%E,%E 
',MechanicalParameters.Area,MechanicalParameters.Izz,...
  MechanicalParameters.Iyy,MechanicalParameters.Tkz, MechanicalParameters.Tky);
fprintf(fid,'TB,ANEL,1,1,21,0 
');
fprintf(fid,'TBTEMP,0 
');
fprintf(fid,'TBDATA,1,%E,%E,%E,%E,%E,%E 
',Stiffness(1,1),Stiffness(1,2),Stiffness(1,3),Stiffness(1,4),Stiffness(1,5),Stiffness(1,6));
fprintf(fid,'TBDATA,7,%E,%E,%E,%E,%E,%E 
',Stiffness(2,2),Stiffness(2,3),Stiffness(2,4),Stiffness(2,5),Stiffness(2,6),Stiffness(3,3));
fprintf(fid,'TBDATA,13,%E,%E,%E,%E,%E,%E 
',Stiffness(3,4),Stiffness(3,5),Stiffness(3,6),Stiffness(4,4),Stiffness(4,5),Stiffness(4,6));
fprintf(fid,'TBDATA,19,%E,%E,%E 
',Stiffness(5,5),Stiffness(5,6),Stiffness(6,6));
fprintf(fid,'
');
fprintf(fid,'
');
fprintf(fid,'/SOLU 
');
fprintf(fid,'ANTYPE,0 
');
fprintf(fid,'NLGEOM,1 
'); %Large Deformations
fprintf(fid,'FINISH 
');

%Writting Strucutre to file
%------ Introduction of Nodes (KeyPoints) and definition of connecting
%Elements (Solids)
[Element_Nodes,polymer]=ElementNodesP(ConnectingMatrix,Monomer);
[Element_Nodes,polymer]=ElementNodes(ConnectingMatrix,Monomer); % All are
%connected

% ---- writting Elements (Solids) in file ----
long=length(Element_Nodes(:,1)); % All particles MUST be connected somehow
PosNodes=[];
Node_counter=1;
for i=1:long
  for j=1:8
    if Element_Nodes(i,j)==Node_counter
      switch j
        case 1
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ-0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
            Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ-0.5,Node_counter];
        case 2
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ-0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
            Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ-0.5,Node_counter];
        case 3
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ-0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
            Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ-0.5,Node_counter];
        case 4
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ-0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
            Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ-0.5,Node_counter];
        case 5
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ+0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
            Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ+0.5,Node_counter];
        case 6
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ+0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
            Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ+0.5,Node_counter];
        case 7
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ+0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
            Monomer(polymer(i)).PosY+0.5,
            Monomer(polymer(i)).PosZ+0.5,Node_counter];
        case 8
          fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
            Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ+0.5);
          PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
            Monomer(polymer(i)).PosY-0.5,
            Monomer(polymer(i)).PosZ+0.5,Node_counter];
      end
    end
  end
end
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
case 6
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
Monomer(polymer(i)).PosX+0.5,Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX+0.5,
Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
case 7
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
Monomer(polymer(i)).PosY+0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
case 8
fprintf(fid,'K,%d,%f,%f,%f 
',Node_counter,...
Monomer(polymer(i)).PosX-0.5,Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5);
PosNodes=[PosNodes;Monomer(polymer(i)).PosX-0.5,
Monomer(polymer(i)).PosY-0.5,
Monomer(polymer(i)).PosZ+0.5,Node_counter];
end
Node_counter=Node_counter+1;
end % end for Node generation
end % end for
fprintf(fid,'V, %d, %d, %d, %d, %d, %d, %d, %d 
',Element_Nodes(i,1),...
Element_Nodes(i,2),Element_Nodes(i,3),Element_Nodes(i,4),...
Element_Nodes(i,5),Element_Nodes(i,6),Element_Nodes(i,7),Element_Nodes(i,8));
end % end for elements
;display('Writting Elements & Nodes....Ready!!');
fprintf(fid,'
');

% Defining size of mesh and meshing (before applying loads)
fprintf(fid,'ESIZE,%g 
',ESize);
fprintf(fid,'VMESH,ALL 
');
fprintf(fid,'
');

% Defining Loads Depending on loading case
fprintf(fid,'/SOLU 
' );
fprintf(fid,'NSEL,ALL 
');
Numt=0;
for constrain=1:longconstrain
    du=dumode(constrain,1);
    mode=dumode(constrain,2);
    fprintf(fid,'\n');
    if constrain>1
        switch dumode(constrain,3)
case 1
% fprintf(fid,'NSEL,ALL \n');
% fprintf(fid,'DCUM,ADD \n');
AddString='DCUM,ADD \n';
case 0
% fprintf(fid,'NSEL,ALL \n');
% fprintf(fid,'DCUM \n');
AddString='DCUM \n';
end
else
fprintf(fid,'NSEL,ALL \n');
AddString='DCUM \n';
end % adding BC's or replacing them
fprintf(fid,AddString);
switch mode
case 1
dU=du*(Lx+1);
% Face 1 is being pulled uy
fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
fprintf(fid,'D,ALL, UX,%g \n \n',dU);
% Face 2 is totally constrained
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'D,ALL,ALL,0 \n');
% Face 3, 4, 5, 6 have the same stretching varying the position X
% dU(X)=(X-(-0.5))*du*(Lx+1)/(Lx+1)
for X=-0.5+ESize:ESize:Lx+0.5-ESize
fprintf(fid,'NSEL,S,LOC,Y,%g \n',Lx+0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
fprintf(fid,'D,ALL, UX,%g \n \n',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 4
fprintf(fid,'NSEL,S,LOC,Y,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
fprintf(fid,'D,ALL, UX,%g \n \n',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 5
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
fprintf(fid,'D,ALL, UX,%g \n \n',(X-(-0.5))*du*(Lx+1)/(Lx+1));
% Face 6
fprintf(fid,'NSEL,S,LOC,Z,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,X,%g \n',X);
fprintf(fid,'D,ALL, UY,0,,,,UZ \n');
fprintf(fid,'D,ALL, UX,%g \n \n',(X-(-0.5))*du*(Lx+1)/(Lx+1));
end % along x direction
case 2
dU=du*(Ly+1);
% Face 3 is being pulled ux
fprintf(fid,'NSEL,S,LOC,Y,%g \n',Ly+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',dU);
% Face 4 is totally constrained
fprintf(fid,'NSEL,S,LOC,Y,%g \n',-0.5);
fprintf(fid,'D,ALL,ALL,0 \n');
% Face 1, 2, 5, 6 have the same stretching varying the position X
% dU(Y)=(Y-(-0.5))*du*(Ly+1)/(Ly+1)
for Y=-0.5+ESize:ESize:Ly+0.5-ESize
fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));
end for
APPENDIX A. PROGRAM CODES

% Face 2
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n',(Y-(-0.5))*du*(Ly+1)/(Ly+1));

% Face 5
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));

% Face 6
fprintf(fid,'NSEL,S,LOC,Z,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Y,%g \n',Y);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',(Y-(-0.5))*du*(Ly+1)/(Ly+1));

end % along y direction

case 3
dU=du*(Lz+1);

% Face 5 is being pulled uz
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',dU);

% Face 6 is totally constrained
fprintf(fid,'NSEL,S,LOC,Z,%g \n',-0.5);
fprintf(fid,'D,ALL,ALL,0 
');

% Face 1, 2, 3, 4 have the same stretching varying the position X
% dU(Z)=(Z-(-0.5))*du*(Lz+1)/(Lz+1)
for Z=-0.5+ESize:ESize:Lz+0.5-ESize

% Face 1
fprintf(fid,'NSEL,S,LOC,X,%g \n',Lx+0.5);
fprintf(fid,'NSEL,R,LOC,Z,%g \n',Z);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(Z-(-0.5))*du*(Lz+1)/(Lz+1));

% Face 2
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Z,%g \n',Z);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(Z-(-0.5))*du*(Lz+1)/(Lz+1));

% Face 3
fprintf(fid,'NSEL,S,LOC,Y,%g \n',Ly+0.5);
fprintf(fid,'NSEL,R,LOC,Z,%g \n',Z);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(Z-(-0.5))*du*(Lz+1)/(Lz+1));

% Face 4
fprintf(fid,'NSEL,S,LOC,Y,%g \n',-0.5);
fprintf(fid,'NSEL,R,LOC,Z,%g \n',Z);
fprintf(fid,'NSEL,R,LOC,X,%g,%g
',-0.5+ESize,Lx+0.5-ESize);
fprintf(fid,'D,ALL, UX,0,,,,UY \n');
fprintf(fid,'D,ALL, UZ,%g \n 
',(Z-(-0.5))*du*(Lz+1)/(Lz+1));

end % along z direction

case 4
dU=du*(Lz+1);

% Face 5 is being pulled uy
fprintf(fid,'NSEL,S,LOC,Z,%g \n',Lz+0.5);
fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
fprintf(fid,'D,ALL, UY,%g \n 
',dU);

% Face 6 is totally constrained
fprintf(fid,'NSEL,S,LOC,Z,%g \n',-0.5);
fprintf(fid,'D,ALL,ALL,0 
');

% Face 1, 2, 3, 4 have the same stretching varying the position X
% dU(Z)=(Z-(-0.5))*du*(Lz+1)/(Lz+1)
for Z=-0.5+ESize:ESize:Lz+0.5-ESize

% Face 1
APPENDIX A. PROGRAM CODES

for Z = -0.5 + ESize: ESize: Ly + 0.5 - ESize
% Face 1
fprintf(fid, 'NSEL, S, LOC, X, Xg 
\n', Lx + 0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg 
\n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));

% Face 2
fprintf(fid, 'NSEL, S, LOC, X, Xg \n \n', -0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));

% Face 3
fprintf(fid, 'NSEL, S, LOC, Y, Xg \n \n', Ly + 0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));

% Face 4
fprintf(fid, 'NSEL, S, LOC, Y, Xg \n \n', -0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));

% Face 5 is being pulled ux
fprintf(fid, 'NSEL, S, LOC, Z, Xg \n \n', Lz + 0.5);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', dU);
fprintf(fid, 'D, ALL, ALL, 0 
');
% Face 6 is totally constrained
fprintf(fid, 'NSEL, S, LOC, Z, Xg \n \n', -0.5);
fprintf(fid, 'D, ALL, ALL, 0 
');
% Face 1, 2, 3, 4 have the same stretching varying the position X
% \n
for Z = -0.5 + ESize: ESize: Ly + 0.5 - ESize
% Face 1
fprintf(fid, 'NSEL, S, LOC, X, Xg \n \n', Lx + 0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));
% Face 2
fprintf(fid, 'NSEL, S, LOC, X, Xg \n \n', -0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));
% Face 3
fprintf(fid, 'NSEL, S, LOC, Y, Xg \n \n', Ly + 0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));
% Face 4
fprintf(fid, 'NSEL, S, LOC, Y, Xg \n \n', -0.5);
fprintf(fid, 'NSEL, R, LOC, Z, Xg \n \n', Z);
fprintf(fid, 'D, ALL, UX, 0, ..., UZ 
\n');
fprintf(fid, 'D, ALL, UY, Xg \n \n', (Z - (-0.5)) * du * (Lz + 1) / (Lz + 1));
end % along z direction

case 6
% Face 3 is being pulled ux
fprintf(fid, 'NSEL, S, LOC, Y, Xg \n \n', Ly + 0.5);
fprintf(fid, 'D, ALL, UX, 0, ..., UY 
\n');
fprintf(fid, 'D, ALL, UZ, Xg \n \n', dU);
fprintf(fid, 'D, ALL, ALL, 0 
');
% Face 4 is totally constrained
fprintf(fid, 'NSEL, S, LOC, Y, Xg \n \n', -0.5);
fprintf(fid, 'D, ALL, ALL, 0 
');
% Face 1, 2, 5, 6 have the same stretching varying the position X
APPENDIX A. PROGRAM CODES

\[ \% \text{dU}(Y) = (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1) \]

for \( Y = -0.5 \pm \text{ESize} : \text{ESize} : Ly + 0.5 - \text{ESize} \)

\% Face 1
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, X, %g \n', Lx + 0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UY \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UZ, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

\% Face 2
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, X, %g \n', -0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UY \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UZ, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

\% Face 5
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, Z, %g \n', Lz + 0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, X, %g \n', -0.5 \pm \text{ESize}, Lx + 0.5 - \text{ESize}});
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UY \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UZ, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

\% Face 6
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, Z, %g \n', -0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, X, %g \n', -0.5 \pm \text{ESize}, Lx + 0.5 - \text{ESize}});
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UY \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UZ, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

end \% along y direction

case 7
\[ dU = du \times (Ly + 1); \]
\% Face 3 is being pulled ux
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, Y, %g \n', Ly + 0.5});
\text{fprintf}(\text{fid}, \text{'D, ALL, UY, 0, \ldots, UZ \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, %g \n', du});
\]

\% Face 4 is totally constrained
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, Y, %g \n', -0.5});
\text{fprintf}(\text{fid}, \text{'D, ALL, ALL, 0 \n'});
\]

\% Face 1, 2, 5, 6 have the same stretching varying the position X
\% \[ dU(Y) = (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1) \]

for \( Y = -0.5 \pm \text{ESize} : \text{ESize} : Ly + 0.5 - \text{ESize} \)

\% Face 1
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, X, %g \n', Lx + 0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UY \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UZ, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

\% Face 2
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, X, %g \n', -0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UZ \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UY, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

\% Face 5
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, Z, %g \n', Lz + 0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, X, %g \n', -0.5 \pm \text{ESize}, Lx + 0.5 - \text{ESize}});
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UZ \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UY, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

\% Face 6
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, Z, %g \n', -0.5});
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, Y, %g \n'}, Y);
\text{fprintf}(\text{fid}, \text{'NSEL, R, LOC, X, %g \n', -0.5 \pm \text{ESize}, Lx + 0.5 - \text{ESize}});
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UZ \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UY, %g \n', (Y - (-0.5)) \times du \times (Ly + 1)/(Ly + 1)});
\]

end \% along y direction

case 8
\[ dU = du \times (Lx + 1); \]
\% Face 1 is being pulled uy
\[
\text{fprintf}(\text{fid}, \text{'NSEL, S, LOC, X, %g \n', Lx + 0.5});
\text{fprintf}(\text{fid}, \text{'D, ALL, UX, 0, \ldots, UZ \n'});
\text{fprintf}(\text{fid}, \text{'D, ALL, UY, %g \n', du});
\]
% Face 2 is totally constrained
fprintf(fid,'NSEL,S,LOC,X,%g \n',-0.5);
fprintf(fid,'D,ALL,ALL,0 \n');
% Face 3, 4, 5, 6 have the same stretching varying the position X
% dU(X)=(X-(-0.5))*du*(Lx+1)/(Lx+1)
for X=-0.5+ESize:ESize:Lx+0.5-ESize
  % Face 3
  fprintf(fid,'NSEL,S,LOC,Y,Xg \n',Ly+0.5);
  fprintf(fid,'NSEL,R,LOC,Xg \n',X);
  fprintf(fid,'D,ALL, UX,0,,,,UZ \n');
  fprintf(fid,'D,ALL, UY,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
  % Face 4
  fprintf(fid,'NSEL,S,LOC,Y,Xg \n',-0.5);
  fprintf(fid,'NSEL,R,LOC,Xg \n',X);
  fprintf(fid,'D,ALL, UX,0,,,,UY \n');
  fprintf(fid,'D,ALL, UZ,%g 
',dU);
  % Face 5
  fprintf(fid,'NSEL,S,LOC,Z,Xg \n',Lz+0.5);
  fprintf(fid,'NSEL,R,LOC,Xg \n',X);
  fprintf(fid,'NSEL,R,LOC,Y,Xg \n',-0.5+ESize,Ly+0.5-ESize);
  fprintf(fid,'D,ALL, UX,0,,,,UY \n');
  fprintf(fid,'D,ALL, UZ,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
  % Face 6
  fprintf(fid,'NSEL,S,LOC,Z,Xg \n',-0.5);
  fprintf(fid,'NSEL,R,LOC,Xg \n',X);
  fprintf(fid,'NSEL,R,LOC,Y,Xg \n',-0.5+ESize,Ly+0.5-ESize);
  fprintf(fid,'D,ALL, UX,0,,,,UY \n');
  fprintf(fid,'D,ALL, UZ,%g \n 
',(X-(-0.5))*du*(Lx+1)/(Lx+1));
end % along x direction

end % switch mode
fprintf(fid,'NSEL,ALL \n');
fprintf(fid,' 
');
%Solving
%fprint(fid,'PIVCHECK, OFF \n'); % For the case of
%underconstrained model
if constrain+1<=longconstrain
    if dumode(constrain+1,3)~=1
        % implement solve
        Numt=Numt+1;
        %fprintf(fid,'TIME,\d \n',Numt);
        %fprintf(fid,'TIME \n');
        fprintf(fid,'SOLVE \n');
    end
else
    % implement solve
    Numt=Numt+1;
    fprintf(fid,'TIME,\d \n',Numt);
    fprintf(fid,'SOLVE \n');
end %end solving and timing. No time increase is done if the constrains are added!!!
end %constraint for loop
%fprint(fid,'FINISH \n');

%Post Processing given that the solution was done previously
%fprint(fid,'*dim,USimulation,array,\d \n',Numt);
%fprint(fid,’/post1 
’);
%fprint(fid,’SET,FIRST \n’);
for constrain=1:Numt
    fprintf(fid,’ETABLE,UTOTENE,SEN
E, \n’);
    fprintf(fid,’SSUM,UTOTENE \n’);
    %fprintf(fid,’ETABLE,VOLUMEN,VOLU, \n’);
    %fprintf(fid,’SSUM,VOLUMEN \n’);
    fprintf(fid,’*GET,UTOT,SSUM,,ITEM,UTOTENE \n’);
    %fprintf(fid,’*GET,VTOT,SSUM,,ITEM,VOLUMEN \n’);
    fprintf(fid,’USimulation(\d)= UTOT \n’,constrain);
    fprintf(fid,’SET,NEXT \n’);
end %data retriving
%fprint(fid,’FINISH \n’);
end % IF fullyconnected
fclose(fid);
clear ConnectingMatrix Monomer
try
    eval(['!del ',dirpath,char(filesep),'solution.dat']);
end
eval(['!"',AnsysPath," -b -i ',dirpath,char(filesep),nameAnsys,' -o ',dirpath,char(filesep),nameout,' -dir ',dirpath,char(filesep)])
display('Running Ansys-Batch...Ready');

% Calculation of pinverse
dU=dUtot;
volume=LX*LY*LZ;
load([dirpath,char(filesep),'solution.dat']);
eval(['!del ',dirpath,char(filesep),'file.*']);
eval(['!del ',dirpath,char(filesep),'dataOut.dat']);
solution=1/volume*diag(solution);
L=1;

% Individual Cases
C(1,1)=solution(16,16)*2/(dU/L)^2;
C(2,2)=solution(11,11)*2/(dU/L)^2;
C(3,3)=solution(7,7)*2/(dU/L)^2;
C(4,4) = \text{solution}(1,1) \times 2 / (2dU/L)^2;
C(5,5) = \text{solution}(4,4) \times 2 / (2dU/L)^2;
C(6,6) = \text{solution}(2,2) \times 2 / (2dU/L)^2;

\begin{align*}
C(1,2) &= (\text{solution}(21,21) - 1/2C(1,1)*(dU/L)^2 - 1/2C(2,2)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(1,3) &= (\text{solution}(20,20) - 1/2C(1,1)*(dU/L)^2 - 1/2C(3,3)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(1,4) &= (\text{solution}(17,17) - 1/2C(1,1)*(dU/L)^2 - 1/2C(4,4)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(1,5) &= (\text{solution}(19,19) - 1/2C(1,1)*(dU/L)^2 - 1/2C(5,5)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(1,6) &= (\text{solution}(18,18) - 1/2C(1,1)*(dU/L)^2 - 1/2C(6,6)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\

C(2,3) &= (\text{solution}(15,15) - 1/2C(2,2)*(dU/L)^2 - 1/2C(3,3)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(2,4) &= (\text{solution}(12,12) - 1/2C(2,2)*(dU/L)^2 - 1/2C(4,4)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(2,5) &= (\text{solution}(14,14) - 1/2C(2,2)*(dU/L)^2 - 1/2C(5,5)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(2,6) &= (\text{solution}(13,13) - 1/2C(2,2)*(dU/L)^2 - 1/2C(6,6)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\

C(3,4) &= (\text{solution}(8,8) - 1/2C(3,3)*(dU/L)^2 - 1/2C(4,4)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(3,5) &= (\text{solution}(10,10) - 1/2C(3,3)*(dU/L)^2 - 1/2C(5,5)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(3,6) &= (\text{solution}(9,9) - 1/2C(3,3)*(dU/L)^2 - 1/2C(6,6)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\

C(4,5) &= (\text{solution}(5,5) - 1/2C(4,4)*(dU/L)^2 - 1/2C(5,5)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(4,6) &= (\text{solution}(3,3) - 1/2C(4,4)*(dU/L)^2 - 1/2C(6,6)*(dU/L)^2)) / ((dU/L)*(dU/L)); \\
C(5,6) &= (\text{solution}(6,6) - 1/2C(5,5)*(dU/L)^2 - 1/2C(6,6)*(dU/L)^2)) / ((dU/L)*(dU/L));
\end{align*}

\% Symmetry
for i=1:6
\quad for j=i+1:6
\quad \quad C(j,i) = C(i,j);
\quad end
end

\text{EffStiffness}(\text{iTest},\text{Node}+1) = C;
end \% Nodes
end \% iTests

\% =========== Additional Functions ============
function answer = isNotThroughFACE(Parti,Posj)
\% answer = true;
\% Parti=1;
\% Partj=A(Parti,Posj);
\% display(num2str(Parti));
\switch j
\case 1
\quad if (Monomer(Partj).PosY==1)
\quad \quad answer = false;
\quad end
\case 2
\quad if (Monomer(Parti).PosY==1)
\quad \quad answer = false;
\quad end
\case 3
\quad if (Monomer(Partj).PosZ==1)
\quad \quad answer = false;
\quad end
\case 4
\quad if (Monomer(Parti).PosZ==1)
\quad \quad answer = false;
\quad end
\case 5
\quad if (Monomer(Partj).PosX==1)
\quad \quad answer = false;
\quad end
\case 6
\quad if (Monomer(Parti).PosX==1)
\quad \quad answer = false;
\quad end
end % test of connection
end %isFACE

end %EffectiveStiffnessAlgorithm
A.7 AveIq in Matlab

function IqAve = AveIq(Iq)
% Created by Fritz A Campo © MAE@WVU 04/15/2011
% This function calculates the average Iq over the number of nodes

[NoTests, np] = size(Iq);

for iTest=1:NoTests
    tempIq = zeros(length(Iq{iTest,1}(:,2)),1);
    for Node=0:np-1
        tempIq = tempIq + Iq{iTest,Node + 1}(:,2);
    end
    IqAve{iTest} = [Iq{iTest,1}(:,1),tempIq / np];
end
A.8 AveStiffness in Matlab

function EffAveStiffness = AveStiffness(EffStiffness)

% Created by Fritz A Campo @ MAE@WVU 04/15/2011
% This function calculates the average stiffness over the number of nodes

[NoTests, np] = size(EffStiffness);

for iTest=1:NoTests
    tempStiffness = zeros(6,6);
    for Node=0:np-1
        tempStiffness = tempStiffness + EffStiffness{iTest,Node+1};
    end
    EffAveStiffness{iTest} = tempStiffness / np;
end
A.9 EffectStiffGenAlgorithm in Matlab

function [StiffGeneration] = EffectStiffGenAlgorithm(StructureIn, FEMParameters, Stiffness, Nohierarchies)

% This function calculates the effective properties of the material
% recursively using as properties of the primary particle the ones
% calculated at a previous iteration. To start, it uses Stiffness
% properties.

[NoTests, np] = size(StructureIn);

for iTest = 1:NoTests
    StiffGeneration{iTest,1} = Stiffness;
    for hierarchy = 2:Nohierarchies + 1
        EffStiffness = EffectiveStiffnessAlgorithm(StructureOut, FEMParameters, StiffGeneration{iTest, hierarchy - 1});
        StiffGeneration{iTest, hierarchy} = AveStiffness(EffStiffness);
    end
end
A.10 FindPolymer in Matlab

function polymer=FindPolymer(n1,ConnectingMatrix)
%Finds list of monomers conforming the Polymer

polymer=n1;
pos=1;
while(pos<length(polymer))
    newlist=[];
    testlist=find(ConnectingMatrix(:,polymer(pos))==1);
    for i=1:length(testlist);
        if isempty(find(polymer==testlist(i)))
            newlist=[newlist,testlist(i)];
        end
    end
    polymer=[polymer,newlist];
pos=pos+1;
end
A.11 ElementNodesP in Matlab

function [Element_Nodes,polymer]=ElementNodesP(Connection_Matrix,Particles)

% Returns a matrix Element_Nodes with a list of all the nodes that connect
% each element. Note that the elements have the order given by the
% function FindPolymer, i.e. the list polymer. The position of Element i,
% is determined by the position of particle polymer(i)

long=length(Particles);
%firstElement=1;
polymer=1:long;
%polymer=FindPolymer(firstElement,Connection_Matrix); % This goes against
%the possibility of a disconnected structure

Element_Nodes=zeros(long,8);
nodecounter=1;

for Elem=1:long
    %Element_Nodes=[Element_Nodes;[0 0 0 0 0 0 0 0]];
    PosXElem=Particles(Elem).PosX;
    PosYElem=Particles(Elem).PosY;
    PosZElem=Particles(Elem).PosZ;
    % First vertices
    for test=1:Elem-1 % Checking with the already numbered elements
        PosXtest=Particles(test).PosX;
        PosYtest=Particles(test).PosY;
        PosZtest=Particles(test).PosZ;
        %Faces
        ConnLength=ConnectionLength(Elem,test);
        if ConnLength==3 %Vertex
            if PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
                if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,7); end
            elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
                if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,8); end
            elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
                if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,4); end
            elseif PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem+1
                if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,3); end
            elseif PosXtest==PosXElem-1 && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
                if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,6); end
            elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
                if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,5); end
            elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
                if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,1); end
            elseif PosXtest==PosXElem-1 && PosYtest==PosYElem-1 && PosZtest==PosZElem+1
                if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,2); end
            else
                % nothing
            end
        end %if on the previous elements
    end %test on the previous elements

    % Then edges
    for test=1:Elem-1 % Checking with the already numbered elements
        PosXtest=Particles(test).PosX;
        PosYtest=Particles(test).PosY;
        PosZtest=Particles(test).PosZ;
        %Faces
        ConnLength=ConnectionLength(Elem,test);
        if ConnLength==2 %Edges
            if PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem
                if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,3); end
            elseif Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,7); end
            elseif Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,1); end
            elseif Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,2); end
        end %if on the previous elements
    end %test on the previous elements
end %for
APPENDIX A. PROGRAM CODES

176

if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,4); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,8); end
elseif PosXtest==PosXElem1 && PosYtest==PosYElem && PosZtest==PosZElem-1
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,8); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,1); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,4); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,3); end

elseif PosXtest==PosXElem1 && PosYtest==PosYElem & PosZtest==PosZElem+1
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,1); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,3); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,2); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,6); end

elseif PosXtest==PosXElem && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,5); end
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,6); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,8); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,2); end

elseif PosXtest==PosXElem && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,1); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,2); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,3); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,7); end

elseif PosXtest==PosXElem-1 && PosYtest==PosYElem && PosZtest==PosZElem-1
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,3); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,2); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,1); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,6); end

elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,3); end
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,4); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,8); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,5); end

elseif PosXtest==PosXElem-1 && PosYtest==PosYElem-1 && PosZtest==PosZElem
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,2); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,1); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,3); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,6); end

elseif PosXtest==PosXElem-1 && PosYtest==PosYElem && PosZtest==PosZElem+1
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,1); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,2); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,3); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,4); end

for test=1:Elem-1 % Checking with the already numbered elements
PosXtest=Particles(test).PosX;
PosYtest=Particles(test).PosY;
PosZtest=Particles(test).PosZ;
%Faces
ConnLength=ConnectionLength(Elem,test);
if ConnLength==1
if PosXtest==PosXElem1 && PosYtest==PosYElem & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,4); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,3); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,8); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,7); end

elseif PosXtest==PosXElem1 & PosYtest==PosYElem & PosZtest==PosZElem
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,1); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,2); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,3); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,6); end

elseif PosXtest==PosXElem1 & PosYtest==PosYElem & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,4); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,3); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,8); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,7); end
elseif PosXtest==PosXElem & PosYtest==PosYElem1 & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,2); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,1); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,6); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,5); end

elseif PosXtest==PosXElem & PosYtest==PosYElem & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,2); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,1); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,6); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,5); end
elseif PosXtest==PosXElem & PosYtest==PosYElem-1 & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,6); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,5); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,3); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,4); end

}
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,2); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,4); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,3); end
elseif PosXtest==PosXElem & PosYtest==PosYElem & PosZtest==PosZElem-1
    if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,5); end
    if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,6); end
    if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,8); end
    if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,7); end
end
end

% fill with consecutive numbers the remaining blanc nodes
for node=1:8
    if Element_Nodes(Elem,node)==0
        Element_Nodes(Elem,node)=nodecounter;
        nodecounter=nodecounter+1;
    end
end % filling empty nodes

% display(num2str(Elem));
end % over all elements in the (list) polymer

% for Elem=1:long
% Element_Nodes=[Element_Nodes;[0 0 0 0 0 0 0 0]];
% PosXElem=Particles(Elem).PosX;
% PosYElem=Particles(Elem).PosY;
% PosZElem=Particles(Elem).PosZ;
for test=1:Elem-1 % checking with the already numbered elements
    PosXtest=Particles(test).PosX;
    PosYtest=Particles(test).PosY;
    PosZtest=Particles(test).PosZ;
    ConnLength=ConnectionLength(Elem,test);
    if ConnLength==1
        if PosXtest==PosXElem+1 & PosYtest==PosYElem & PosZtest==PosZElem-1
            if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,5); end
            if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,6); end
            if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,8); end
            if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,7); end
        elseif PosXtest==PosXElem-1 & PosYtest==PosYElem & PosZtest==PosZElem+1
            if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,2); end
            if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,3); end
            if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,8); end
            if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,7); end
        elseif PosXtest==PosXElem & PosYtest==PosYElem+1 & PosZtest==PosZElem+1
            if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,3); end
            if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,1); end
            if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,4); end
            if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,6); end
        elseif PosXtest==PosXElem & PosYtest==PosYElem-1 & PosZtest==PosZElem+1
            if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,3); end
            if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,1); end
            if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,4); end
            if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,6); end
        else PosXtest==PosXElem & PosYtest==PosYElem & PosZtest==PosZElem-1
            if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,1); end
            if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,2); end
            if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,3); end
            if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,5); end
        end
    end
end
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,5); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,6); end
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,8); end
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,7); end
end
elseif ConnLength==2
%Edges
if PosXtest==PosXElem+1 & PosYtest==PosYElem-1 & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,3); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,7); end
elseif PosXtest==PosXElem+1 & PosYtest==PosYElem+1 & PosZtest==PosZElem
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,4); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,8); end
elseif PosXtest==PosXElem+1 & PosYtest==PosYElem+1 & PosZtest==PosZElem-1
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,8); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,7); end
elseif PosXtest==PosXElem+1 & PosYtest==PosYElem+1 & PosZtest==PosZElem+1
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,4); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,3); end
elseif PosXtest==PosXElem+1 & PosYtest==PosYElem+1 & PosZtest==PosZElem
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,6); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,2); end
elseif PosXtest==PosXElem & PosYtest==PosYElem+1 & PosZtest==PosZElem-1
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,2); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,3); end
elseif PosXtest==PosXElem & PosYtest==PosYElem-1 & PosZtest==PosZElem+1
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,2); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,5); end
elseif PosXtest==PosXElem & PosYtest==PosYElem & PosZtest==PosZElem
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,6); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,5); end
elseif ConnLength==3
%Vertex
if PosXtest==PosXElem+1 & PosYtest==PosYElem-1 & PosZtest==PosZElem
if Element_Nodes(Elem,1)==0 Element_Nodes(Elem,1)=Element_Nodes(test,7); end
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,8); end
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,6); end
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,5); end
elseif PosXtest==PosXElem+1 & PosYtest==PosYElem-1 & PosZtest==PosZElem
if Element_Nodes(Elem,2)==0 Element_Nodes(Elem,2)=Element_Nodes(test,8); end
if Element_Nodes(Elem,5)==0 Element_Nodes(Elem,5)=Element_Nodes(test,7); end
if Element_Nodes(Elem,6)==0 Element_Nodes(Elem,6)=Element_Nodes(test,1); end
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,5); end
elseif PosXtest==PosXElem & PosYtest==PosYElem-1 & PosZtest==PosZElem
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,6); end
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,5); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,2); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,1); end
elseif PosXtest==PosXElem & PosYtest==PosYElem & PosZtest==PosZElem
if Element_Nodes(Elem,3)==0 Element_Nodes(Elem,3)=Element_Nodes(test,5); end
if Element_Nodes(Elem,7)==0 Element_Nodes(Elem,7)=Element_Nodes(test,1); end
if Element_Nodes(Elem,4)==0 Element_Nodes(Elem,4)=Element_Nodes(test,6); end
if Element_Nodes(Elem,8)==0 Element_Nodes(Elem,8)=Element_Nodes(test,2); end
end
%else
% nothing
end %if
end %test on the previous elements

%fill with consecutive numbers the remaining blanc nodes
for node=1:8
    if Element_Nodes(Elem,node)==0
        Element_Nodes(Elem,node)=nodecounter;
        nodecounter=nodecounter+1;
    end
end % filling empty nodes

%display(num2str(Elem));
end %over all elements in the (list) polymer
%

function Lcon=ConnectionLength(testA,testB)
%
Lcon=0;
Condition=true;
lcounter=1;
polymerA=testA;
pos=1;
Posonetherarchy=2;
while(pos<=length(polymerA) && Condition)
    newlist=[];
testvalues=[];
testvalues=find(Connection_Matrix(:,polymerA(pos))==1)';
for ii=1:length(testvalues);
    if (isempty(find(polymerA==testvalues(ii))))
        newlist=[newlist,testvalues(ii)];
    end
end
if isempty(find(newlist==testB))
    polymerA=[polymerA,newlist];
pos=pos+1;
else
    Condition=false;
    Lcon=lcounter;
end
if pos==Posonetherarchy
    lcounter=lcounter+1;
    Posonetherarchy=length(polymerA)+1;
end
if lcounter>3
    Condition=false;
end
end
end
A.12 ElementNodes in Matlab

function [Element_Nodes,polymer]=ElementNodes(Connection_Matrix,Particles)
% Returns a matrix Element_Nodes with a list of all the nodes that connect
% each element. Note that the elements have the order given by the
% function FindPolymer, i.e. the list polymer. The position of Element i,
% is determined by the position of particle polymer(i)

long=length(Particles);
firstElement=1;
polymer=1:long;
%polymer=FindPolymer(firstElement,Connection_Matrix);
Element_Nodes=[];
nodecounter=1;
for Elem=1:long
    Element_Nodes=[Element_Nodes,[0 0 0 0 0 0 0 0]];
   PosXElem=Particles(polymer(Elem)).PosX;
   PosYElem=Particles(polymer(Elem)).PosY;
   PosZElem=Particles(polymer(Elem)).PosZ;
    for test=1:Elem-1
       PosXtest=Particles(polymer(test)).PosX;
       PosYtest=Particles(polymer(test)).PosY;
       PosZtest=Particles(polymer(test)).PosZ;
        %Faces
        if PosXtest==PosXElem+1 && PosYtest==PosYElem && PosZtest==PosZElem
            Element_Nodes(Elem,1)=Element_Nodes(test,4);
            Element_Nodes(Elem,2)=Element_Nodes(test,3);
            Element_Nodes(Elem,5)=Element_Nodes(test,8);
            Element_Nodes(Elem,6)=Element_Nodes(test,7);
        elseif PosXtest==PosXElem-1 && PosYtest==PosYElem && PosZtest==PosZElem
            Element_Nodes(Elem,4)=Element_Nodes(test,1);
            Element_Nodes(Elem,3)=Element_Nodes(test,2);
            Element_Nodes(Elem,8)=Element_Nodes(test,5);
            Element_Nodes(Elem,7)=Element_Nodes(test,6);
        elseif PosXtest==PosXElem && PosYtest==PosYElem+1 && PosZtest==PosZElem
            Element_Nodes(Elem,2)=Element_Nodes(test,1);
            Element_Nodes(Elem,3)=Element_Nodes(test,4);
            Element_Nodes(Elem,6)=Element_Nodes(test,5);
            Element_Nodes(Elem,7)=Element_Nodes(test,3);
        elseif PosXtest==PosXElem && PosYtest==PosYElem-1 && PosZtest==PosZElem
            Element_Nodes(Elem,1)=Element_Nodes(test,8);
            Element_Nodes(Elem,2)=Element_Nodes(test,7);
            Element_Nodes(Elem,5)=Element_Nodes(test,6);
            Element_Nodes(Elem,8)=Element_Nodes(test,7);
        elseif PosXtest==PosXElem && PosYtest==PosYElem && PosZtest==PosZElem+1
            Element_Nodes(Elem,5)=Element_Nodes(test,1);
            Element_Nodes(Elem,6)=Element_Nodes(test,2);
            Element_Nodes(Elem,8)=Element_Nodes(test,4);
            Element_Nodes(Elem,7)=Element_Nodes(test,3);
        elseif PosXtest==PosXElem && PosYtest==PosYElem && PosZtest==PosZElem-1
            Element_Nodes(Elem,1)=Element_Nodes(test,5);
            Element_Nodes(Elem,2)=Element_Nodes(test,6);
            Element_Nodes(Elem,4)=Element_Nodes(test,8);
            Element_Nodes(Elem,3)=Element_Nodes(test,7);
        elseif PosXtest==PosXElem+1 && PosYtest==PosYElem && PosZtest==PosZElem
            Element_Nodes(Elem,1)=Element_Nodes(test,3);
            Element_Nodes(Elem,5)=Element_Nodes(test,7);
        elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem
            Element_Nodes(Elem,2)=Element_Nodes(test,4);
            Element_Nodes(Elem,6)=Element_Nodes(test,8);
        elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem
            Element_Nodes(Elem,1)=Element_Nodes(test,8);
            Element_Nodes(Elem,2)=Element_Nodes(test,7);
            Element_Nodes(Elem,5)=Element_Nodes(test,6);
            Element_Nodes(Elem,8)=Element_Nodes(test,7);
        elseif PosXtest==PosXElem+2 && PosYtest==PosYElem && PosZtest==PosZElem
            Element_Nodes(Elem,5)=Element_Nodes(test,4);
        end
    end
end
Element_Nodes(Elem,6)=Element_Nodes(test,3);

elseif PosXtest==PosXElem && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,2)=Element_Nodes(test,5);
    Element_Nodes(Elem,3)=Element_Nodes(test,8);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,6)=Element_Nodes(test,1);
    Element_Nodes(Elem,7)=Element_Nodes(test,4);
elseif PosXtest==PosXElem && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,2)=Element_Nodes(test,2);
    Element_Nodes(Elem,8)=Element_Nodes(test,3);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,4)=Element_Nodes(test,6);
    Element_Nodes(Elem,8)=Element_Nodes(test,5);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,3)=Element_Nodes(test,5);
    Element_Nodes(Elem,4)=Element_Nodes(test,6);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,6)=Element_Nodes(test,4);
    Element_Nodes(Elem,7)=Element_Nodes(test,5);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,7)=Element_Nodes(test,5);
    Element_Nodes(Elem,8)=Element_Nodes(test,6);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,2)=Element_Nodes(test,1);
    Element_Nodes(Elem,8)=Element_Nodes(test,3);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,6)=Element_Nodes(test,4);
    Element_Nodes(Elem,7)=Element_Nodes(test,5);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,5)=Element_Nodes(test,3);
    Element_Nodes(Elem,8)=Element_Nodes(test,2);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem-1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,4)=Element_Nodes(test,6);
    Element_Nodes(Elem,8)=Element_Nodes(test,5);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,3)=Element_Nodes(test,5);
    Element_Nodes(Elem,8)=Element_Nodes(test,2);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,7)=Element_Nodes(test,5);
    Element_Nodes(Elem,8)=Element_Nodes(test,6);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,2)=Element_Nodes(test,1);
    Element_Nodes(Elem,8)=Element_Nodes(test,3);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,6)=Element_Nodes(test,4);
    Element_Nodes(Elem,7)=Element_Nodes(test,5);
elseif PosXtest==PosXElem+1 && PosYtest==PosYElem-1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,5)=Element_Nodes(test,3);
    Element_Nodes(Elem,8)=Element_Nodes(test,2);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem-1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,4)=Element_Nodes(test,6);
    Element_Nodes(Elem,8)=Element_Nodes(test,5);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem-1
    Element_Nodes(Elem,3)=Element_Nodes(test,5);
    Element_Nodes(Elem,8)=Element_Nodes(test,2);
elseif PosXtest==PosXElem-1 && PosYtest==PosYElem+1 && PosZtest==PosZElem+1
    Element_Nodes(Elem,7)=Element_Nodes(test,5);
    Element_Nodes(Elem,8)=Element_Nodes(test,2);
else
    null
end
end %test on the previous elements

%fill with consecutive numbers the remaining blanc nodes
for node=1:8
    if Element_Nodes(Elem,node)==0
        Element_Nodes(Elem,node)=nodecounter;
        nodecounter=nodecounter+1;
    end
end % filling empty nodes
end %over all elements in the (list) polymer
APPENDIX A. PROGRAM CODES

A.13 Generation of Hierarchical structure

This code is the implementation of the reconstruction of the hierarchical structure.

function Intensity = HierarchicalStructure(dd, Generationes, Structure, L, iTest)
    % created by Fritz Andres Campo MAE@WVU

    % Resolution
    ddx = dd.x;
    ddy = dd.y;
    ddz = dd.z;

    % setting up variables
    Intensity = zeros(ddx, ddy, ddz);
    Nx = 1;
    Ny = 1;
    Nz = 1;
    threshold = 0;
    for Gen = 1:Generaciones % Loop over the number of Generations
        % Loop over each of the cells at each generation
        for iix = 1:Nx
            for iiy = 1:Ny
                for iiz = 1:Nz
                    % Finding the position of each cell with the corresponding
                    % resolution
                    X = (iix-1)/Nx;
                    Y = (iiy-1)/Ny;
                    Z = (iiz-1)/Nz;

                    ppx = 1;
                    for i=1:ddx
                        if X > ppx*1/ddx, ppx = ppx + 1; end
                    end
                    ppy = 1;
                    for i=1:ddy
                        if Y > ppy*1/ddy, ppy = ppy + 1; end
                    end
                    ppz = 1;
                    for i=1:ddz
                        if Z > ppz*1/ddz, ppz = ppz + 1; end
                    end

                    % If the cell proves to have enough density at
                    % the scale in order to multiscale
                    if Intensity(ppx, ppy, ppz) == threshold
                        % Find the structure to copy at a smaller scale
                        Node = ceil(np(iTest)*rand());
                        [LS, di] = size(Structure(iTest, Node));
                        structure = Structure(iTest, Node);
                        for particle = 1:LS % Copying density at a
                            % smaller scale within the resolution
                            ix = (structure(particle,1)-1)/Nx/L;
                            iy = (structure(particle,2)-1)/Ny/L;
                            iz = (structure(particle,3)-1)/Nz/L;
                            PosX = (X + ix);
                            PosY = (Y + iy);
                            PosZ = (Z + iz);
                            Xmin = PosX - mod(PosX, 1 / Nx / L);
                            Ymin = PosY - mod(PosY, 1 / Ny / L);
                            Zmin = PosZ - mod(PosZ, 1 / Nz / L);
                            Xmax = PosX - mod(PosX, 1 / Nx / L) + 1/Nx/L;
                            Ymax = PosY - mod(PosY, 1 / Ny / L) + 1/Ny/L;
                            Zmax = PosZ - mod(PosZ, 1 / Nz / L) + 1/Nz/L;
                            imin = 1;
imax = ddx;
for i=1:ddx
    if Xmin > imin*1/ddx, imin = imin + 1; end
    if Xmax <= imax*1/ddx, imax = imax - 1; end
end
ixmin = imin;
ixmax = imax;

imin = 1;
imax = ddy;
for i=1:ddy
    if Ymin > imin*1/ddy, imin = imin + 1; end
    if Ymax <= imax*1/ddy, imax = imax - 1; end
end
iymin = imin;
iymax = imax;

imin = 1;
imax = ddz;
for i=1:ddz
    if Zmin > imin*1/ddz, imin = imin + 1; end
    if Zmax <= imax*1/ddz, imax = imax - 1; end
end
izmin = imin;
izmax = imax;

for ppx = ixmin:ixmax
    for ppy = iymin:iymax
        for ppz = izmin:izmax
            Intensity(ppx,ppy,ppz) = ...
            Intensity(ppx,ppy,ppz) + 1;
        end
    end
end
end
end

Nx = Nx*L;
Ny = Ny*L;
Nz = Nz*L;
display([' === Gen ===', num2str(Gen)]);
threshold = threshold + 1;
end
end % Function
Appendix B

Homogenization: Stresses Point of View

A mechanical property, as a constitutive law, correlates two or more mechanical variables, e.g. stress and strain. Based on [71][72], the homogenization of a mechanical property can be understood as the property that correlates averages over a volume of two or more mechanical variables.

For instance, the average of the stress $\bar{\sigma}_{ij}$ can be defined as

$$\bar{\sigma}_{ij} = \frac{1}{V} \int \sigma(x)_{ij} \, dV$$ (B.1)

where $V$ is the volume over which the stress field is averaged, for example a Representative Volume Element (RVE), and $x$ is the position on that volume.

Similarly, the average strain $\bar{\varepsilon}_{ij}$ can be defined as

$$\bar{\varepsilon}_{ij} = \frac{1}{V} \int \varepsilon(x)_{ij} \, dV$$ (B.2)

Since for linear materials, the Hooke’s law follows,

\[1\] The formalism can be extended to nonlinear materials starting from an energy point of view.
\[ \sigma_{ij} = C_{ijkl} \varepsilon_{kl} \quad (B.3) \]

where \( C_{ijkl} \) is the stiffness of the material, a homogenized stiffness contained in \( V \) can be calculated such that

\[ \bar{\sigma}_{ij} = \bar{C}_{ijkl} \bar{\varepsilon}_{kl} \quad (B.4) \]

Due to the volume dependence of the strain averages, the calculation of the homogenized stiffness seems elaborated. However, two steps allow an easy implementation: 1) Averaging of the strains using the Gauss theorem, 2) Application of uniform strains on the surface of the volume. The Gauss Theorem converts the volume integrals into surface integrals leading to easy definition of Boundary Conditions. Choosing uniform strains makes the calculation of the surface integral trivial leading to an easy evaluation of the homogenized stiffness.

The Gauss Theorem states,

\[ \int \frac{\partial f_i}{\partial x_j} dV = \int f_i n_j dS \quad (B.5) \]

Where \( f_i \) is a vectorial field, and \( n_j \) is the normal vector of the surface of size \( dS \) pointing outwards.

The uniform deformation \( u^0_i \) relative to a constant position vector \( x_{i0} \) can be defined as

\[ u^0_i = \varepsilon^0_{ij} (x_j - x_{j0}) \quad (B.6) \]

Note that the uniform strain \( \varepsilon^0_{ij} \) associated to the uniform deformation satisfies

\[ \varepsilon^0_{ij} = \frac{1}{2} \left( \frac{\partial u^0_i}{\partial x_j} + \frac{\partial u^0_j}{\partial x_i} \right) = \varepsilon^0_{ij} \quad (B.7) \]

which if imposed on the surface of the volume \( V \), using \( (B.5) \) to calculate the average
APPENDIX B. HOMOGENIZATION: STRESSES POINT OF VIEW

strain yields

\[
\bar{\varepsilon}_{ij} = \frac{1}{V} \int \varepsilon_{ij} \, dV = \frac{1}{2V} \int \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \, dV = \frac{1}{2V} \int u^0_i n_j \, dS + \frac{1}{2V} \int u^0_j n_i \, dS
\]

\[
= \frac{1}{2V} \int u^0_i n_j \, dS = \frac{1}{V} \int \varepsilon^0_{ik} (x_k - x_{k0}) n_j \, dS = \frac{1}{V} \int \frac{\partial u^0_k}{\partial x_j} \, dV = \varepsilon^0_{ik}
\]

(B.8)

(B.8) means that imposing a uniform strain at the surface leads to an average strain with the same value as the uniform strain!

Using the STANDARD tensor contraction \((ij) \rightarrow \beta\) defined in [72, 70],

\[
\beta(ij) = \begin{cases} 
  i & \text{when } i = j \\
  9 - (i + j) & \text{otherwise}
\end{cases}
\]

(B.9)

(B.4) for uniform strains yield

\[
\bar{\sigma}^0_{\alpha} = C_{\alpha\beta} \varepsilon^0_{\beta}
\]

(B.10)

Choosing a uniform strain \(k\) from the set of uniform strains

\[
\varepsilon^0_{\beta} \in \left\{ \varepsilon^0_{\beta} = \begin{cases} 
  1 & \text{if } \beta = k \\
  0 & \text{otherwise}
\end{cases} : k = \{1, 2, 3, 4, 5, 6\} \right\}
\]

(B.11)

each column of the stiffness \(C_{\alpha k}\) can be obtained calculated from the average stress \(\bar{\sigma}^0_{\alpha}(k)\)

using (B.1) from

\[
\bar{\sigma}^0_{\alpha} = C_{\alpha k}
\]

(B.12)
Finally, noting that (B.9) implies that

\[
\begin{align*}
\epsilon_1 &= \epsilon_{11} \\
\epsilon_2 &= \epsilon_{22} \\
\epsilon_3 &= \epsilon_{33} \\
\epsilon_4 &= \epsilon_{23} = \epsilon_{32} \\
\epsilon_5 &= \epsilon_{13} = \epsilon_{31} \\
\epsilon_6 &= \epsilon_{12} = \epsilon_{21}
\end{align*}
\]

(B.13)

The uniform deformations are calculated, over the whole surface \( S \) as

\[
\begin{align*}
u^0_1 &= \epsilon^0_1(x_1 - x_{10}) + \epsilon^0_6(x_2 - x_{20}) + \epsilon^0_5(x_3 - x_{30}) \\
u^0_2 &= \epsilon^0_6(x_1 - x_{10}) + \epsilon^0_2(x_2 - x_{20}) + \epsilon^0_4(x_3 - x_{30}) \\
u^0_3 &= \epsilon^0_5(x_1 - x_{10}) + \epsilon^0_4(x_2 - x_{20}) + \epsilon^0_3(x_3 - x_{30})
\end{align*}
\]

(B.14)  (B.15)  (B.16)

Using table B.1, \( C_{\alpha\beta} \) can be calculated. Note that the strains used are NOT engineering strains, which gives an homogenized stiffness which needs to be corrected with the Reuters matrix \( R \) \[70, 72\] to retrieve the engineering stiffness from

\[
C^{eng} = CR^{-1}
\]

(B.17)