universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"The nuclear magnetic resonance investigation of pore coupling effects in near-surface environments"

verfasst von / submitted by

Francisca Antonia Soto Bravo, B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 299
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Environmental Science
Betreut von / Supervisor:	Ass.-Prof. Dr. Chi Zhang

Acknowledgments

I would like to express my gratitude to my teacher and supervisor Ass.-Prof. Dr. Chi Zhang, for her mentoring, her dedicated guidance and unwavering encouragement through the process of writing this master's thesis and beyond. I am very thankful for the chance of joining the Environmental Geophysics working group and for the kindness, the learning opportunities and all the valuable advice I received.

I would also like to thank my family and my boyfriend for their patience and unconditional support.

Abstract

The characterisation of near-surface environments is of great importance for monitoring and managing underground water resources. These include fully saturated groundwater aquifers, water being transported or stored in the unsaturated vadose zone and even water trapped in weathered rocks as rock moisture. Nuclear magnetic resonance (NMR) as a geophysics tool was originally developed for the exploration of hydrocarbon resources but it is gaining popularity in the field of hydrogeophysics. By probing the magnetisation and relaxation behaviour of the spin magnetic moment of hydrogen atoms in external magnetic fields, NMR is uniquely sensitive to pore water in the subsurface and can help characterise the pore spaces in geologic material. As a technology, it is also non-invasive and cost-effective. However, the interpretation of NMR data is not always straightforward. In certain scenarios, with highly connected, complex, dual-porosity geometries, pore coupling effects can emerge. Pore-coupled systems show undesired magnetisation exchange between pore environments during an NMR measurement, which makes the characterisation of the individual pore environments difficult. Materials prone to pore coupling are carbonate rocks, shales and rocks with high clay contents. In this master's thesis, two main factors controlling pore coupling, surface geochemistry and network connectivity, are explored through Random Walk numerical simulations. Four values for the surface relaxivity are tested ($10\mu m/s$, $40\mu m/s$, $100\mu m/s$, and $250\mu m/s$), four throat widths ($0.1\mu m$, $0.5\mu m$, $1\mu m$, and $2\mu m$), and four throat lengths ($0.1\mu m$, $1\mu m$, $10\mu m$, and $100\mu m$) for two simple, synthetic, geometries with dual-porosity (one with similar pore sizes, one with largely dissimilar pore sizes). The resulting magnetisation decay curves and the inverted characteristic relaxation times were compared for the different scenarios. This comparison suggests that pore coupling can affect the two different pore environments differently depending on the driving factor of the coupling. The pore-size ratio of the geometry seems to determine the sensitivity of the system to any given control factor. For both pore-size ratios, pore throat length is identified as a key control factor. The high sensitivity of pore coupling to surface relaxivity highlights the importance of the correct value estimation even within small ranges. For NMR to reach its full potential as a hydrogeophysics tool, signals from pore coupling-prone materials must be better understood and better interpreted. Knowledge gaps are identified, in particular for unsaturated conditions, such as in the vadose zone.

Kurzfassung

Die Charakterisierung oberflächennaher Bereiche ist von großer Wichtigkeit für die Überwachung und Verwaltung unterirdischer Wasserressourcen. Zu diesen gehören vollständig gesättigte Grundwasserleiter, in der ungesättigten Zone transportiertes oder gespeichertes Wasser und sogar in verwittertem Gestein eingeschlossene „rock moisture“. Als Werkzeug in der Geophysik ursprünglich zur Erforschung von Kohlenwasserstoffressourcen entwickelt, gewinnt die Kernspinresonanz (engl. Nuclear magnetic resonance, NMR) zunehmend auch im Bereich der Hydrogeophysik an Bedeutung. Durch Messung der Magnetisierung und des Entspannungsverhaltens des Protonenspins von Wasserstoffatomen in externen Magnetfeldern ist NMR besonders sensitiv im Hinblick auf Porenwasser unter der Erdoberfläche und kann zur Charakterisierung der Porenstruktur im geologischen Material genutzt werden. Dabei ist NMR zusätzlich nichtinvasiv und kostengünstig. Die Interpretation mit dieser Methode gewonnenen Daten kann allerdings unter Umständen komplex sein. So können bei stark verbundenen und komplexen Geometrien mit zwei charakteristischen Porengrößen so genannte „pore coupling effects“ auftreten. Systeme mit verbundenen Poren zeigen während der NMR-Messung einen unerwünschten Magnetisierungsaustausch, was die Charakterisierung der einzelnen Poreneigenschaften erschwert. Materialien, die zu solchen Effekten neigen, sind Karbonatgesteine, Schiefer und Gesteine mit hohem Tonanteil. Diese Masterarbeit untersucht zwei Hauptfaktoren, die diese Pore-Coupling-Effekte beeinflussen: Die geochemischen Eigenschaften der Porenoberflächen und die Verbundenheit des Porennetzwerks. Diese werden mit Hilfe von numerischen Random-Walk-Simulationen untersucht. Getestet wurden vier Werte für die „surface relaxivity“ ($10\mu m/s$, $40\mu m/s$, $100\mu m/s$, and $250\mu m/s$), vier Porenhalsbreiten ($0.1\mu m$, $0.5\mu m$, $1\mu m$, and $2\mu m$) und vier Porenhalslängen ($0.1\mu m$, $1\mu m$, $10\mu m$, and $100\mu m$) bei zwei einfachen, synthetischen Geometrien mit dualer Porosität (eine mit ähnlich großen Poren und eine mit stark unterschiedlichen Porengrößen). Die verschiedenen Szenarien wurden sowohl im Hinblick auf die Abnahme der Magnetisierung und auf die invertierten charakteristischen Relaxionszeiten verglichen. Dieser Vergleich suggeriert, dass die unterschiedlichen Porenumgebungen unterschiedlich durch pore coupling beeinflusst werden können, je nachdem, welcher Faktor für die Kopplung maßgeblich ist. Das Porengrößenverhältnis der jeweiligen Geometrie scheint hierbei die Empfindlichkeit des Systems gegenüber seinen Kontrollfaktoren zu bestimmen. Bei beiden Geometrien wurde die Porenhalslänge als zentraler Kontrollfaktor identifiziert. Die Kopplung reagiert auch auf kleine Änderungen der „surface relaxivity“ empfindlich, was die Wichtigkeit genauer Parameterschätzungen hierfür unterstreicht. Damit NMR sein volles Potential als hydrogeophysisches Werkzeug entfalten kann, müssen NMR-Signale aus zu Porenkopplungen neigenden Materialien besser untersucht und interpretiert werden. Weitere Forschung, besonders in ungesättigten Regimen, ist hierfür vonnöten.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Identifying, quantifying and protecting groundwater resources is a major concern in a world with a warming climate and an expanding population. More frequent draughts (Mukherjee *et al.*, 2018), increasing demand of fresh water for human consumption or agricultural or industrial purposes (Sivakumar, 2011) and the anthropogenic contamination of flowing and standing water bodies above ground (Albert *et al.*, 2020) drive the exploitation of underground water resources. In order to avoid the contamination or the depletion of groundwater, it is key to have adequate technical capacities that allow for the constant monitoring of the aquifers, as well as to have a clear understanding of the transport processes of water from the Earth's surface through the soil column.

Hydrogeophysics, as a discipline, provides methods and tools to explore Earth's subsurface and characterise groundwater aquifers. Some of these methods include ground penetrating radar, DC resistivity or induced polarization (Binley *et al.*, 2015). A novel, non-invasive technique, Nuclear Magnetic Resonance (NMR) has been gaining popularity as an hydrogeophysical tool for its unique direct sensitivity to water molecules.

With NMR it is possible to remotely detect the hydrogen atoms forming the water molecules that saturate the pore spaces inside of porous geologic media, making use of the magnetic spin properties characteristic of hydrogen atoms. Using NMR technology in the near-surface environment it is possible to make estimations of the water content in porous media or determine important hydrological parameters, such as porosity or hydraulic conductivity, in fully saturated zones (Müller-Petke *et al.*, 2015). In the partially saturated vadose zone, NMR has been used to study processes related to dynamic storage (Schmidt & Rempe, 2020), drying (Tian & Wei, 2020) or the distribution of rock moisture (Schmidt & Rempe, 2020; Zhang & Zhang, 2021). NMR measurements can be performed from the Earth's surface (sNMR), in the laboratory (lab-NMR) or in-situ using boreholes (bNMR). This versatility, along with its non-invasive nature, its cost-efficiency and its direct sensitivity to water molecules, makes NMR a very promising tool to explore groundwater resources (Binley *et al.*, 2015) and even to study other environmental processes of concern, such as permafrost thawing (Parsekian *et al.*, 2013; Kass *et al.*, 2017) or frost damage (Ding *et al.*, 2020).

The key assumption behind the conventional interpretation of NMR data is that, while water molecules are always moving within the pore spaces, one specific hydrogen atom is only able to sample one specific pore space during the NMR measurement (Brownstein & Tarr, 1979). While this holds true for many geologic materials and in many scenarios, this assumption can

be broken, making the interpretation of NMR data to estimate hydrological parameters rather complicated (McCall *et al.*, 1991). Some complex systems with dual porosity present high connectivity between micro- and macropores that allow for hydrogen atoms to diffuse from one pore space to another during an NMR measurement, effectively probing pores of different sizes instead of being limited to one single pore (Ramakrishnan *et al.*, 1999). This effect is called diffusional coupling or pore coupling and it is known to happen, for example, in some carbonate rocks, shales or generally in systems with high clay content. In order to correctly interpret NMR data collected in all possible scenarios and all possible geologic materials, it is important to understand and quantify the pore coupling effects and the factors that drive their occurrence and severity.

Pore coupling effects can be studied experimentally, manipulating samples at a pore-scale in a laboratory setting, or through numerical simulations, that allow for the targeted examination of the role specific sample characteristics may have on the degree of pore coupling taking place. Numerical simulations allow, as well, for the systematic creation of exemplary synthetic geometries and the study of very specific scenarios that would be hard to construct experimentally.

A careful literature review lead to the identification of two main factors controlling pore coupling: pore surface geochemistry, related to the chemical properties of the pore wall (see Section 5.2), and network connectivity, related to the physical characteristics of the throats connecting the pore spaces (see Section 5.3). This master's thesis will use a well-established Random Walk NMR numerical simulation algorithm with the objective to test the magnetisation decay response and the pore coupling effects under different scenarios, examining the influence of changes in the geometry of the pore networks, in the geochemistry of the material, and in the network connectivity.

This manuscript is organized into seven chapters: in Chapter 2, Theoretical Background, the key concepts related to NMR are detailed and explained and the state-of-knowledge on pore coupling, its effects and the factors controlling it, are presented, along with the approaches, experimental and numerical, that can be used for its study. The equations needed for the data inversion algorithm used in this study and for the analytical solution are, lastly, shown and developed. In Chapter 3, Methods, a detailed explanation of the geometries chosen for this study and their realisation as input files for the Random Walk numerical simulation code are presented. This C++ Random Walk algorithm is then introduced, including all of the input and output files associated to it, a careful revision of the simulation parameters that must be defined by the user and a step-by-step description of the compilation process, the running of the code, the inversion step and the data processing procedure. In Chapter 4, Validation of Simulation and Inversion Algorithms, the simulation results for simple single-pore geometries are tested and compared to analytical solutions in order to validate the Random Walk and the inversion step. In Chapter 5, Results, the simulation results for all of the examined scenarios are presented in three subsections: first, a comparison between the geometries of the dual porosity systems, then, surface relaxivity as a pore coupling control factor, and lastly, network connectivity as a pore coupling control factor, with throat width and throat length as agents. The conclusions and discussion can be found in Chapter 6 followed by an outlook in Chapter 7, mostly focusing on pressing open questions for the unsaturated vadose zone. Samples of the input files used for the simulations can be found in the Appendices.

Chapter 2

Theoretical Background

2.1 Basic concepts of NMR

Nuclear magnetic resonance (NMR) is a non-invasive method with a wide range of applications in fields as different as geophysics and human medicine. The core physical phenomenon in which NMR tools are based, is the behaviour of protons (specifically their magnetic spins) inside of changing magnetic fields, as they become excited and relax.

Protons, for example in hydrogen atoms forming water molecules, have magnetic spins "pointing" in randomly distributed directions. If a constant external magnetic field (B_0) is applied, these spins will align and reach an equilibrium state of precession at they inherent Larmor frequency (f) defined by Equation 2.1, where γ is the gyromagnetic ratio.

$$f = \frac{\gamma}{2\pi} B_0 \tag{2.1}$$

Applying a second electromagnetic signal (B_1) will perturb this alignment by exciting the spins. The magnetisation in the system will increase. Consequently removing B_1 will allow for the spins to slowly return to equilibrium (i.e., for the magnetisation of the system to decrease again) in a process called relaxation (see Figure 2.1). It is possible to measure the exponential decay of magnetisation in the system and, after an inversion step, the time it takes for the spins to fully relax in each pore environment. As a result of the xy-plane rotation of the magnetic moment due to the achieved resonance condition through B_1, the time necessary until full relaxation is achieved will be different for the xy- and the z-components of the spins. The xy-components will relax with a "transverse relaxation time" (T_2), while the z-components will relax with a "longitudinal relaxation time" (T_1).

The key equations linking the relaxation times T_1 and T_2 to the evolution of the magnetization in the system $M(t)$ are called Bloch's equations and have the following solutions (Costabel & Yaramanci, 2011):

$$M_z(t) = M_0 \Big[1 - \exp\Big(-\frac{t}{T_1} \Big) \Big] \tag{2.2}$$

$$M_{xy}(t) = M_0 \exp\Big(-\frac{t}{T_2} \Big) \tag{2.3}$$

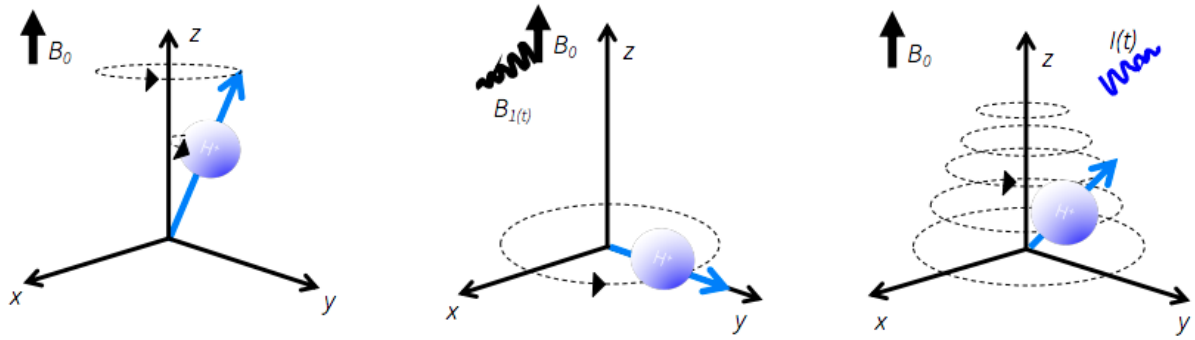


Figure 2.1: From left to right: a static magnetic field (B_0) along the z-axis aligns the rotating magnetic spin of the protons within it, the addition of a second magnetic field $B_{1(t)}$ perpendicular to B_0 flips the magnetic spins so they are now rotating along the xy-plane. Finally, removing $B_{1(t)}$, but keeping B_0, will allow the protons to relax back to equilibrium. Source: Zhang 2021, personal communication.

In the geophysical context, using NMR to probe the water saturating the pore spaces of a porous geological material, it is also possible to link relaxation times to the sizes of the water-containing pores, following, for example, Equation 2.4, where T_{relax} stands for the measured relaxation time, ρ for surface relaxivity (a parameter characterising the surface geochemistry of the pore walls) and r for the pore's characteristic size. α is the shape factor describing the geometry of the pore ($1, 2$ or 3 for planar, cylindrical and spherical pores respectively), and S_{por} is the surface area-to-volume ratio of the pore, ultimately used to derive the pore size distribution (PSD).

$$\frac{1}{T_{\text{relax}}} = \rho\frac{\alpha}{r} = \rho S_{\text{por}} \tag{2.4}$$

The key assumption behind this equation is that each proton will only sample one pore space. If this is the case, the exponential decay of magnetisation measured from that pore will be characteristic to its pore size. Probing a geological media containing pores of different sizes will lead to a multi-exponential magnetisation decay curve, made up from the addition of mono-exponential decay curves. This will result on a relaxation time distribution containing multiple peaks, as many as pore sizes in the porous media. This process can be visualised for a bi-porous media in Figure 2.2.

2.2 Modes of relaxation and diffusion regimes

The relaxation of excited spins back to their equilibrium state can take place through two or even three mechanisms: surface relaxation and bulk relaxation in the z-plane (T_1), or surface, bulk and diffusive relaxation in the xy-plane (T_2) (see Equations 2.5).

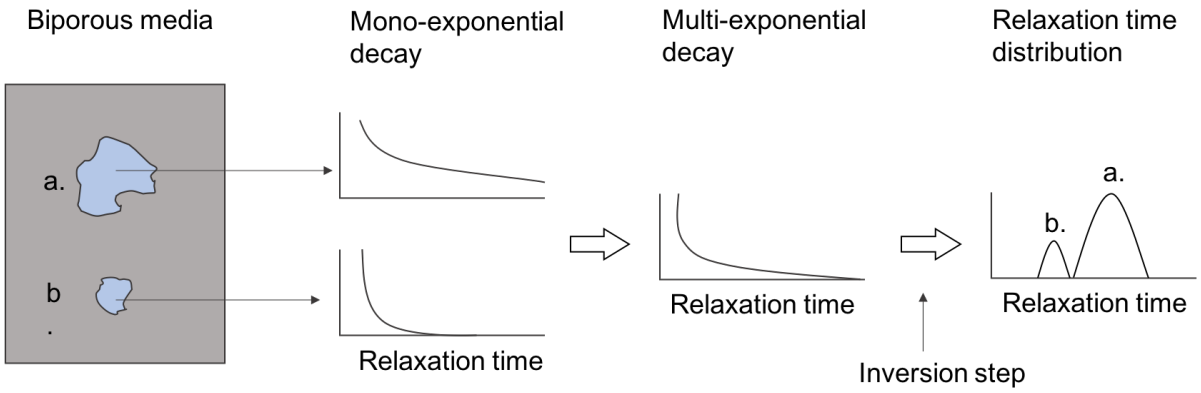


Figure 2.2: Depiction of the traditional interpretation process of NMR data, without the presence of pore coupling effects. Here, the pore spaces are unconnected from each other and associated to a mono-exponential decay of the magnetization signal. As they are measured together, they result on a multi-exponential decay curve. An inversion step translates these curves to relaxation times, where each peak is associated to a pore environment (i.e., a pore size).

$$\frac{1}{T_1} = \frac{1}{T_{1B}} + \frac{1}{T_{1S}} \tag{2.5}$$

$$\frac{1}{T_2} = \frac{1}{T_{2B}} + \frac{1}{T_{2S}} + \frac{1}{T_{2D}} \tag{2.6}$$

Surface relaxation tends to be the fastest and most important form and, being the only mechanism that requires interaction between the pore walls and the diffusing spins, it is the only mode that provides information regarding the pores' geometry. The mechanism driving this form of relaxation is the collision of spins, as they diffuse freely inside the pore space, with the solid pore walls. It is therefore controlled by the surface relaxivity (ρ), an inherent property of the material dependent on its geochemical composition (Bryar *et al.*, 2000), the pore surface-to-volume ratio (A_s/V) and the molecular diffusion. These last two will affect how much time it will take for any given spin to reach the pore all. Surface relaxivity and molecular diffusion both depend on the temperature of the system and on the fluid saturating the pore (Anand *et al.*, 2008). The relationship between pore surface-to-volume ratio and surface relaxation rates is precisely why NMR relaxation times can be used to characterise the geometry of pore spaces. Systems in which surface relaxivity is the main relaxation mechanism are said to be in the "fast diffusion regime" following (Brownstein & Tarr, 1979) terminology. In the fast diffusion regime, mono-exponential NMR signals are expected from each pore, as each spin relaxes promptly after probing a whole single pore space.

Bulk relaxation does not take place at the pore wall but it happens spontaneously anywhere within the pore space as the spins diffuses. Although it is often seen as negligible due to its slowness (e.g., Anand & Hirasaki, 2007), bulk relaxation can become important in certain scenarios, such as in materials with extremely large pore spaces or in pore spaces where the contact between the fluid and the solid pore walls is inhibited (e.g., water filling a pore that had been previously filled with oil). Systems in which this form of relaxation cannot be neglected, are

considered outside of the fast-diffusion regime (either in the slow or the intermediate regime). Outside of the fast-diffusion regime, spins are no longer able to probe the whole pore space before they relax, either because the pore space is too large or because their relaxation takes place too quickly. The difference between the diffusion regimes is illustrated in Figure 2.3, taken from Müller-Petke *et al.* (2015).

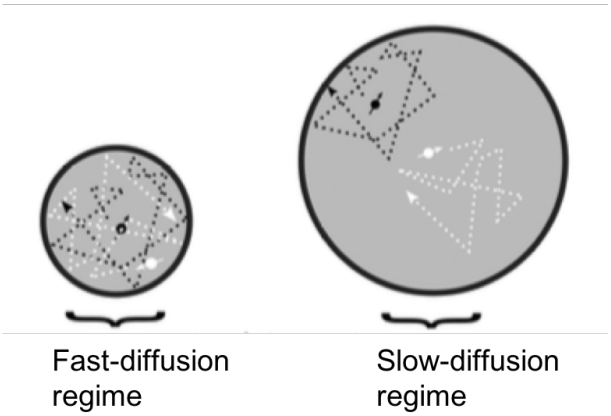


Figure 2.3: Illustration of different paths that two exemplary diffusing spins can take inside pores in the fast- (left) and the slow- (right) diffusion regime. In the fast-diffusion regime the spins move through the whole pore space before they relax, while in the slow-diffusion only a small portion of the pore can be sampled. This example is controlled by the pores' geometry but a similar relaxation behaviour is possible in pores with different surface relaxivities and the same size. Source: Müller-Petke *et al.* (2015).

Diffusive relaxation only takes place in magnetic fields with significant inhomogeneities, where the fluid can self-diffuse. These magnetic gradients are present, for example, in materials containing significant amounts of iron-bearing minerals. If at all, it can only affect T_2 relaxation but not T_1 and it can, in most scenarios, be neglected completely.

Since pore coupling is related to surface relaxation, its effects can be seen equally in the xy- and the z-plane, meaning that either T_1 or T_2 distributions can be used for its study. It can also take place inside and outside of the fast diffusion regime (Anand & Hirasaki, 2007).

2.3 The effect of pore coupling

While for many geological media it is safe to assume that each pore space relaxes independently from each other, each spin probing only one pore space, some materials have a more complex geochemistry or geometry, with better connection between pore spaces, and this assumption does not hold true anymore. If magnetisation can be transferred between pore spaces in the time scale of one NMR measurement, the system is said to be pore-coupled and the determination of pore sizes from the magnetisation decay curves or the T_1- or T_2-distributions is no longer straightforward. This is particularly problematic in systems with both macro- and micropores (dual porosity), such as carbonate rocks, shales and rocks with a high clay content.

Figure 2.4 shows the change in behaviour of diffusing spins in a bi-porous media for an uncoupled and a coupled scenario and how that leads to a loss of information in their corresponding T_1 or T_2 distribution curves: instead of showing two clear peaks, characteristic to two pore sizes,

one large peak would be observed and it would be impossible to accurately determine the two original pore sizes.

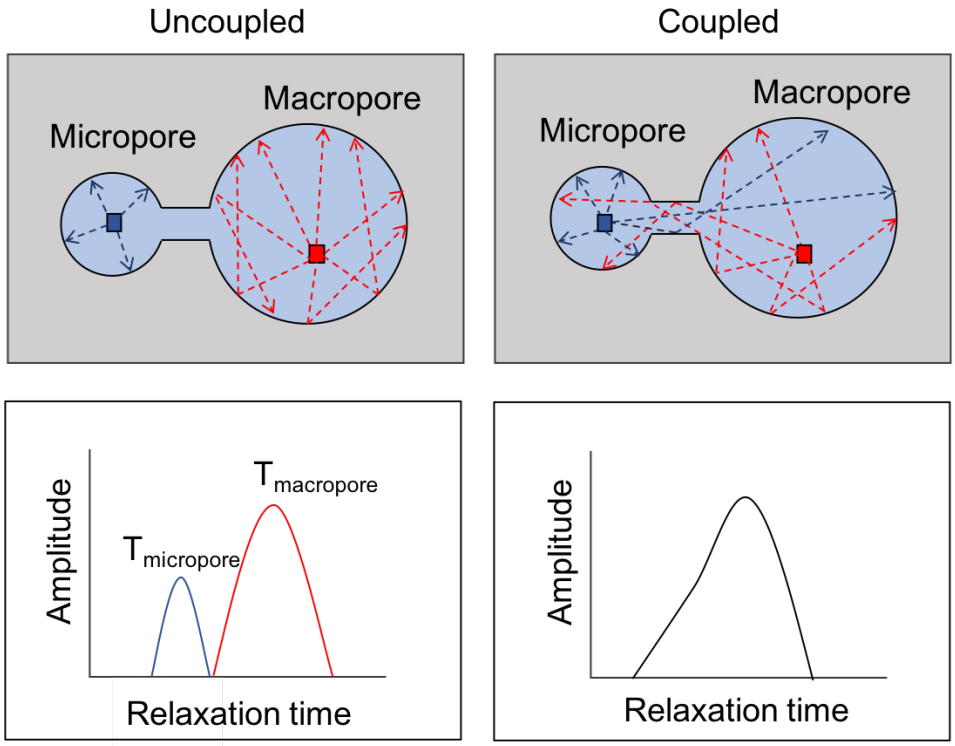


Figure 2.4: Representation of uncoupled (upper-left) and coupled (upper-right) scenarios with their corresponding relaxation time distributions (bottom right and left). The possible trajectories of an exemplary diffusing spin (squares) are marked as dotted arrows in red and blue.

Being able to predict which materials will be pore-coupled in which scenarios and to what extent pore coupling will have an effect on NMR measurements, is very important to be able to use NMR technology to explore complex geological materials, such as carbonate rocks, or complex processes, such as transport in the unsaturated zone.

2.4 Factors controlling pore coupling

There are two main type of controlling factors influencing pore coupling identified in literature: surface geochemistry and network connectivity. In this section, the mechanisms with which these factors control the degree of pore coupling will be explored along a review of available literature on this matter.

2.4.1 Surface geochemistry

Under surface geochemistry, two factors controlling pore coupling in a system are identified: temperature and, more importantly, surface relaxivity (ρ).

Surface relaxivity is a measurement of the ability of a material's pore surface to enhance relaxation (Grunewald & Knight, 2009). The surface relaxivity in a system can be influenced by the

presence of paramagnetic material, e.g., iron(III), which would lead to higher ρ values (Bryar *et al.*, 2000), by the type of fluid saturating the pore space, e.g., ρ values being over four times higher for water than for hexane with all other parameters kept equal (Anand & Hirasaki, 2007), and by temperature (see Anand *et al.*, 2008).

Surface relaxivity is an empirical factor that must be determined from a sample or estimated according to the geological material under study. Examples of studies on the determination of surface relaxivity are Anand & Hirasaki (2007), from pore size distributions measured with mercury porosimetry, or Fleury & Soualem (2009), from specific surface area measurements obtained from nitrogen adsorption methods.

In order to explore the effects of surface relaxivity on pore coupling experimentally, two approaches are prominent in literature: altering the amount of paramagnetic material in the pore walls using iron coatings (e.g. Grunewald & Knight, 2009) and trying different fluids to saturate the pore spaces, such as hexane or water and glycerol mixes (e.g., Anand & Hirasaki, 2007 or Fleury & Soualem, 2009). Surface relaxivity is always an input parameter that can be changed in any code used for the exploration of pore coupling effects from a numerical simulation perspective (see Section 2.5.2).

The current state of knowledge regarding pore coupling and surface relaxivity, suggests that systems with higher surface relaxivity values will present weaker pore coupling. This is because a higher surface relaxivity will enhance relaxation in the pore wall, thus reducing the amount of time a proton is able to travel before relaxing.

2.4.2 Network connectivity

Under "network connectivity" we understand all the physical characteristics of the throats connecting the pore spaces, in particular for this study, the throat widths and throat lengths.

The simplest model to visualise connected pore spaces, in terms of pores and pore throats is portrayed in Figure 2.5. It is very often used to imagine geological material such as shales and for the numerical simulation of NMR relaxation, specially using the Random Walk method.

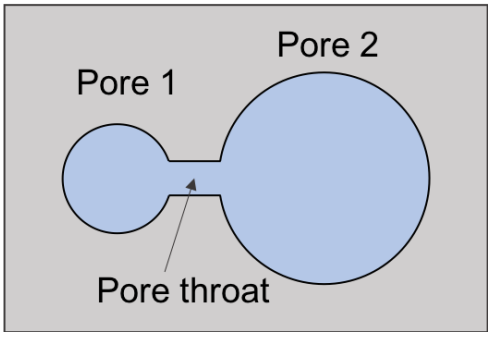


Figure 2.5: Simple two-pore system connected through a pore throat.

Other models include the micrograin consolidation model (e.g., Carneiro *et al.*, 2014), depicted in Figure 2.7, or a model of a macropore, where no relaxation takes place, lined with clay flakes, where the spins relax (e.g., Anand & Hirasaki, 2007). This is depicted in Figure 2.6.

In this master's thesis, restricted by the limitations of the Random Walk approach, only the simple model of pore spaces connected through pore throats will be studied. Nevertheless,

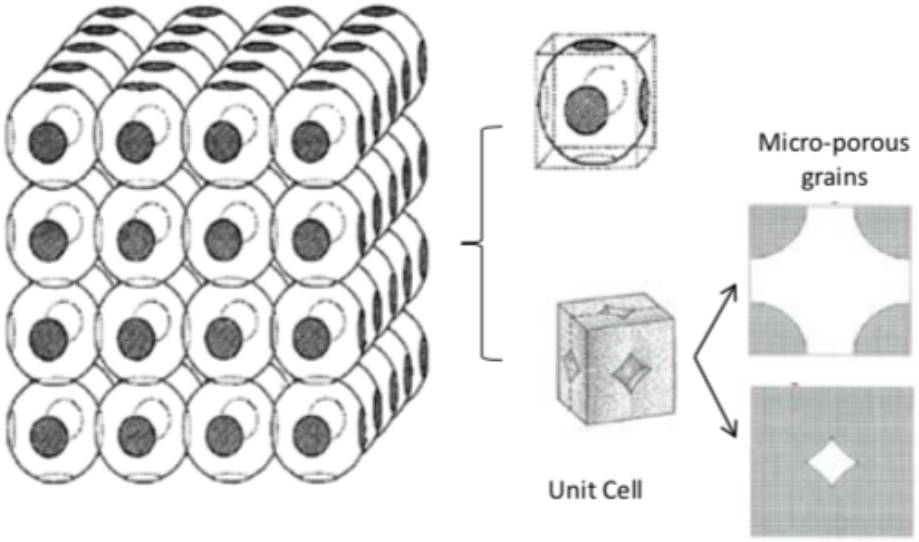


Figure 2.6: Representation of the three-dimensional packing of porous grains in the Micrograin Consolidation Model. Source: Carneiro *et al.* (2013).

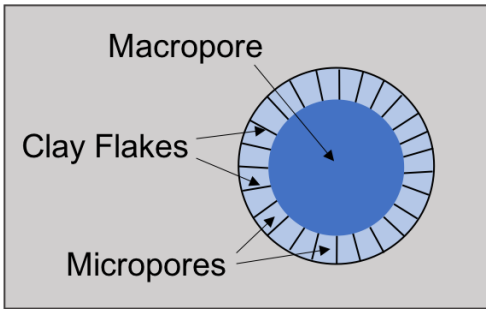


Figure 2.7: Representation of a large macropore lined with clay flakes where spin relaxation takes place. Modified from Anand & Hirasaki (2007).

two different geometries will be considered to cover different pore-size-ratios: networks of pores of similar pore sizes and networks with pores of largely dissimilar pore sizes. While the effect on pore coupling of changes in the pore-size-ratios has been studied for the micrograin consolidation model (e.g., Anand & Hirasaki, 2007, Mitchell *et al.*, 2019), it has not, to the best of our knowledge, been examined for this simplest geometrical model. The factors that have been tested in past studies are pore throat width and pore throat length.

Mohnke & Klitzsch (2010) considered a simple geometry of one pore, $20\mu m$ in diameter, connected to four smaller pores, of $10\mu m$ in diameter, through throats of fixed widths (of $0.1\mu m$) and varying lengths (in the $0.1\mu m$ to $80\mu m$ range) at a fixed surface relaxivity of $10\mu m/s$. They used the finite elements method in the T_1 eigenvalue domain to determine how the size of the throat affected pore coupling. They concluded that large throat lengths were equivalent to unconnected scenarios, presenting two T_1 peaks corresponding to the two pore sizes in the network, while, by smaller throat lengths, pore coupling increased to the extent of only presenting one T_1 peak instead of two.

Ghomeshi *et al.* (2018) considered an even simpler geometry of only two pores with a pore size ratio of 0.25 and performed theoretical calculations to compute the eigenvalues of differ-

ent coupled scenarios and their characteristic relaxation times. In particular, they probed the effect of changing the pore throat diameter in the range of 0 to $5mm$ at a surface relaxivity of $10\mu m/s$. The results of their calculations pointed to an increase in connectivity and in pore coupling strength with increasing throat width values. These theoretical efforts were followed by the numerical simulation of the NMR response using the finite element method on a system representing porous grains. They used a set of concentric spheres with radii between $5\mu m$ and $11\mu m$, at a fixed surface relaxivity of $5\mu m/s$, with 10 connecting throats of changing diameter in the range of $0.1\mu m$ and $1.2\mu m$. In their model only the large pore supplied magnetisation flow towards the smaller pores and not the other way around, meaning that only the slow relaxation T_2 peaks shifted toward faster relaxation times as the system became more coupled with increasing throat diameters.

While these two studies provide insight on how network connectivity in general, and throat geometry in particular, influence the strength of the pore coupling effect, they explore the two main characteristic of the throats, length and width, separately, using different geometrical models. They focus only on very low surface relaxivities ($\rho = 5\mu m$ and $10\mu m$) and work with pores rather similar in size.

2.5 Approaches to study pore coupling

Pore coupling can be studied using different approaches depending on the geological material or the controlling factor of interest. The main division is between experimental and numerical approaches, each one containing multiple methodologies. It is precisely this abundance of methodologies, approaches and even geometrical models that have been used to study pore coupling, what makes it harder to compare the conclusions reached by different studies and form a complete image of pore coupling effects in NMR measurements.

Analytical approaches (i.e., analytically solving the equations governing NMR relaxation inside pore spaces) are limited to single pore geometries, as will be shown later in Section 2.7.

Here, both experimental and numerical approaches to study pore coupling will be introduced, with a special emphasis on numerical approaches as it is the methodology chosen for this study.

It is noteworthy that pore coupling can be studied in 1D and in 2D. 1D approaches, as it has been mentioned until now, deal with T_1 or T_2 distributions of relaxation times, while 2D approaches, originally proposed by Washburn & Callaghan (2006) and Monteilhet *et al.* (2007), involve T_2-store-T_2 measurements and the construction of T_2-T_2 or T_1-T_2 maps. T_2-store-T_2 measurements require a different pulse sequence during measurement to shift magnetisation between the planes and allow for a "mixing" time. T_2-T_2 maps will only present peaks along the diagonal axis if the system is uncoupled but will include off-diagonal peaks if magnetisation exchange between pore spaces is taking place. This is visualised in Figure 2.8.

While it is possible to detect and quantify pore coupling through 2D approaches, it requires the application of more complex techniques than for 1D approaches (Fleury & Soualem, 2009) with evidence signaling that no new information is gained that cannot be obtained through 1D T_1- or T_2-distributions (Johnson & Schwartz, 2014). For this reason, this study will take a 1D approach. In fact, since this study focuses on artificial geometries made out of a very limited number of precisely defined network elements, instead of T_1- or T_2-distributions with Gaussian-looking peaks, characteristic relaxation times (i.e., single point signals) will be use to

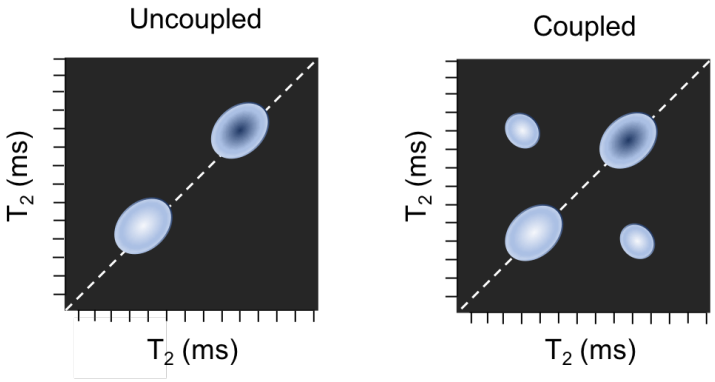


Figure 2.8: Constructed T_2-T_2 maps to visualise the difference between a coupled (right) and an uncoupled (left) scenario. Here, off-diagonal peaks signalize magnetisation exchange between pore environments.

characterise each pore environment.

2.5.1 Experimental approaches

A common method identified in literature to study pore coupling in the laboratory, is performing NMR measurements on fully saturated bi-porous materials (with both macro- and micropores), centrifuging the sample so as to fully drain the macropores and keep the micropores fully saturated, and repeat the NMR measurements. The second measurement will lead to the identification of a fast-relaxation signal, T_2 values, corresponding to the microporosity peak. This peak can be then compared to the T_2-distributions obtained during the first measurement on a fully saturated media: if the system is uncoupled, two peaks should be visible for the fully-saturated scenario and one of them should be identical to the microporosity peak identified. In a coupled system such two peaks will not appear. Figure 2.9, taken from Grunewald & Knight (2009) shows the result of such an experiment on a pore-coupled silica gel sample, where saturated and centrifuged relaxation time distributions are plotted for samples with different surface geochemistry and different degrees of pore coupling. Other studies using this methodology are, for example, Toumelin *et al.* (2003), Anand & Hirasaki (2007) and Anand *et al.* (2008).

2.5.2 Numerical approaches

The use of numerical models and simulations allow for the recreation of very specific scenarios and the targeted study of parameters with a level of control that is not possible to achieve experimentally. This makes simulations very good tools to study the effects of pore coupling. Nevertheless, their construction is time consuming and they require either analytical or experimental verification.

Numerical simulations, regardless of their specific method, all share certain elements in common, in particular: a geometry must be chosen and carefully described by the user, signaling the size and location of pore spaces and pore throats, and a range of simulation parameters must be given as input, usually factors such as surface relaxivity values and the fluid diffusion constant. The capabilities of the computers in which the simulations will be run constrain their complexity and scope and the interpretation of the results must always consider the limitations and biases associated to the simulation method and the chosen geometry.

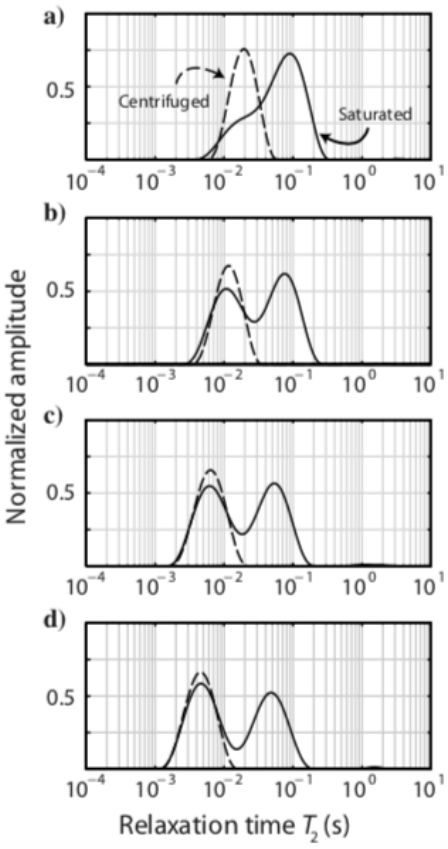


Figure 2.9: Experimentally obtained NMR T_2-distributions for four samples of iron-coated silica gel: (a) 0% Fe^{3+}, (b) 0.002% Fe^{3+}, (c) 0.004% Fe^{3+} and (d) 0.010% Fe^{3+}. In each panel the centrifuged measurement (in dotted lines) and the saturated measurement are plotted. The decreasing difference between the centrifuged peak and the fast saturated peak indicate a decrease in the degree of pore coupling as the iron content of the samples increase. Source: Grunewald & Knight (2009)

Two main methods can be identified in literature to study pore coupling numerically: Random Walk and Finite Elements approaches. Random Walk algorithms, largely used to simulate NMR relaxation in porous media beyond the exploration of pore coupling effects, represent diffusing protons (e.g., hydrogen in water molecules) as "walkers". "Walkers" are small particles that are placed arbitrarily within the spaces designed as pores and are left to move user-defined distances, ("steps", of size ε) in random directions (e.g., using a Monte Carlo algorithm). When a "walker" touches a pore wall, it has a certain percent chance (based on the user-defined ρ) to relax and become "eliminated" or to "survive" and bounce back to keep moving in the pore space. The ratio of "surviving" to "eliminated" "walkers" gets recorded in histograms at a certain time interval and represents the evolution of the magnetisation in the system. Particularly efficient for the study of pore coupled system is the so-called "First Arrival" Random Walk (e.g., Ramakrishnan *et al.*, 1999, Toumelin *et al.*, 2003 or Carneiro *et al.*, 2013), where the step size ε starts as variable and becomes fixed, at a very small value, when the "walker" finds itself very close to the pore wall or a pore throat. The variable size of the step ε will be chosen as the ratio of the largest sphere that is possible to be draw around the "walker" that reaches a pore wall, as visualized in Figure 2.10.

Random Walk is a very well established method that has been used for years and in many studies. It is possible to find tried-and-tested open-source codes capable of running on personal

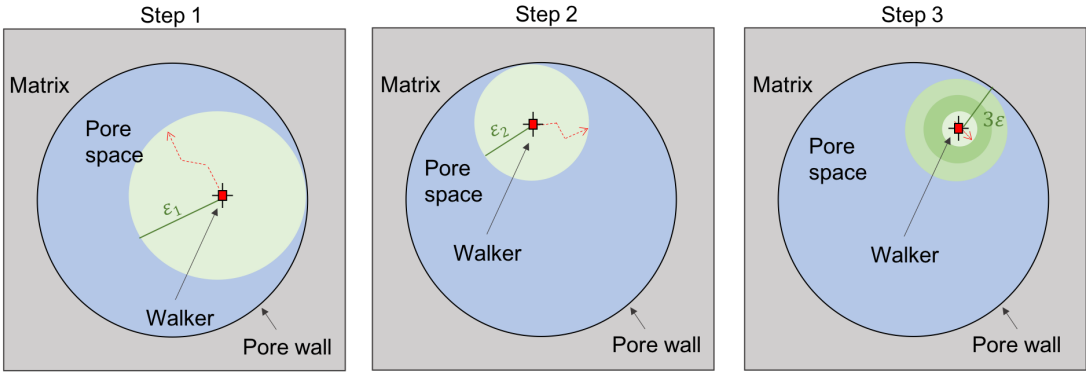


Figure 2.10: Visualization of the First Arrival Random Walk method. The variable step distance ε changes according to the proximity of each "walker" to the pore wall until a certain distance of 3ε is reached.

computers, such as the one used in this master's thesis which was developed by Talabi (2008) and made available by the London Imperial College at no costs.

Finite Element Approaches (e.g., Mitchell *et al.*, 2019), along with others such as Finite Volume Approaches or Lattice Boltzman methods (e.g., Mohnke *et al.*, 2014), divide the chosen geometry into smaller sections (elements, or volumes) and solve the diffusion equations numerically and continuously for each section. This is possible to do in the eigenvalue domain to skip the data inversion step. These approaches can reduce the computational cost of modeling complex scenarios and provide more versatility when it comes to pore geometries (e.g., Mitchell *et al.*, 2019) or allow for the joint simulation of NMR relaxation and other properties (e.g., Mohnke *et al.*, 2014). Nonetheless, the private software often used for these approaches, such as COMSOL Multiphysics, can be extremely pricey and offer limited customisation options, though they can be rather straightforward to use. To date, no code for an open-source software, such as OpenFoam, as been made available. The construction of such a code requires a high level of know-how and computational skill and is beyond the scope of this master's thesis.

2.6 Inversion step

The inversion of results is an important step to convert magnetisation decay curves into relaxation times. It is required both for experimental and numerical approaches, it can be done under different assumptions and with more or less accuracy. Many different software and codes are available with varying degrees of complexity and customisation options. For this study, a rather simple inversion algorithm, made available along the NMR simulation code, was used. The details of the inversion algorithm will be explained in Section 3.6, but the theory behind the inversion step, the matrices and equations involved, will be introduced here, following Talabi (2008).

Inputs for the inversion are normalised magnetisation decay values (the data gained either through measurements or simulations) and exponentially spaced possible T_1 or T_2 values (to which the data will be fitted). The goal will be a curvature smoothing regularization to minimize the equation:

$$\frac{M(t)}{M_0} = \sum_{i=1}^{N} a_i \, exp\left(\frac{-t}{T_{2i}}\right) \tag{2.7}$$

Where $\frac{M(t)}{M_0}$ is the normalized magnetization decay at a time t and a_i is the volume fraction of pores of size i decaying with a relaxation time T_{2i}.

In the matrix notation 2.7 can be written as:

$$\overline{\mathbf{M}} = \overline{\mathbf{K}} \cdot \overline{\mathbf{A}} \tag{2.8}$$

The dimension of $\overline{\mathbf{M}}$ will coincide with the amount of data points being used for the inversion (M) as $[M \times 1]$. $\overline{\mathbf{K}}$ will have the dimension $[M \times N]$ where N is the number of exponentially spaced T_2 values to which the data is being fitted and will be constructed with the equation:

$$K_{i,\,j} = exp\left(\frac{-i\,t}{T_{2j}}\right) \tag{2.9}$$

$\overline{\mathbf{A}}$ has the dimension $[N \times 1]$ and its elements corresponding to a_i, is the end-product of the inversion.

In order to solve for $\overline{\mathbf{A}}$ and since the matrix $\overline{\mathbf{K}}$ is not square the following expresion is derived:

$$\overline{\mathbf{K}}^{\mathrm{T}}\,\overline{\mathbf{M}} = \overline{\mathbf{K}}^{\mathrm{T}}\,\overline{\mathbf{K}}\,\overline{\mathbf{A}} \tag{2.10}$$

$$\overline{\mathbf{A}} = \left(\overline{\mathbf{K}}^{\mathrm{T}}\,\overline{\mathbf{K}}\right)^{-1}\overline{\mathbf{K}}^{\mathrm{T}}\,\overline{\mathbf{M}} \tag{2.11}$$

$$\tag{2.12}$$

For the curvature smoothing, fourth order derivative weighted matrix $\overline{\mathbf{W_m}}$ of dimension $[N \times N]$ and a regularisation parameter λ are needed and determined from the expressions:

$$\overline{\mathbf{W_m}} = \begin{bmatrix} 5 & -4 & 1 & & & & & \\ -4 & 6 & -4 & 1 & & 0 & 0 & 0 \\ 1 & -4 & 6 & -4 & \mathrm{O} & & & \\ & 1 & -4 & 6 & \mathrm{O} & 1 & & \\ & & 1 & -4 & \mathrm{O} & -4 & 1 & \\ & & & 1 & \mathrm{O} & 6 & -4 & 1 \\ & 0 & 0 & 0 & \mathrm{O} & -4 & 6 & -4 \\ & & & & \mathrm{O} & 1 & -4 & 5 \end{bmatrix}_{[N \times N]} \tag{2.13}$$

And

$$\lambda = \frac{\left(\overline{\mathbf{M}} - \overline{\mathbf{K}} \cdot \overline{\mathbf{M_1}}\right)^{\mathbf{T}} \cdot \left(\overline{\mathbf{M}} - \overline{\mathbf{K}} \cdot \overline{\mathbf{M_1}}\right)}{\overline{\mathbf{A_1}}^{\mathbf{T}} \cdot \overline{\mathbf{A_1}}} \tag{2.14}$$

Where $\overline{\mathbf{A_1}}$ corresponds to equation 2.16 with a normalization parameter equal to 1

$$\overline{\mathbf{A_1}} = \left(\overline{\mathbf{K}}^{\mathbf{T}} \cdot \overline{\mathbf{K}} + \overline{\mathbf{W_m}}\right)^{-1} \cdot \overline{\mathbf{K}}^{\mathbf{T}} \cdot \overline{\mathbf{M}} \tag{2.15}$$

$\overline{\mathbf{A}}$ is then determined with the equation:

$$\overline{\mathbf{A}} = \left(\overline{\mathbf{K}}^{\mathbf{T}} \cdot \overline{\mathbf{K}} + \lambda^2 \overline{\mathbf{W_m}}\right)^{-1} \cdot \overline{\mathbf{K}}^{\mathbf{T}} \cdot \overline{\mathbf{M}} \tag{2.16}$$

That can be rewritten to:

$$\overline{\mathbf{A}} = \left(\overline{\mathbf{KK}}^{ex} + \lambda^2 \overline{\mathbf{W_m}}\right)^{-1} \cdot \overline{\mathbf{KM}}^{ex} \tag{2.17}$$

After defining

$$\overline{\mathbf{KK}}^{ex} = \overline{\mathbf{K}}^{\mathbf{T}} \cdot \overline{\mathbf{K}} \tag{2.18}$$

And

$$\overline{\mathbf{KM}}^{ex} = \cdot \overline{\mathbf{K}}^{T} \cdot \overline{\mathbf{M}} \tag{2.19}$$

After solving Equation 2.17, the elements with negative values are set to zero and their indices are deleted from the matrices $\overline{\mathbf{KK}}^{ex}$, $\overline{\mathbf{KM}}^{ex}$ and $\overline{\mathbf{W_m}}$. Using these new matrices, $\overline{\mathbf{A}}$ is recalculated. When all of the elements of $\overline{\mathbf{A}}$ are positive, the inversion process is finished.

2.7 Analytical solution for single pores

Numerical simulations always need to be verified with experimental data or, at the very least, validated through a comparison with analytical solutions. In the case of NMR magnetisation decay, analytical solutions are only available for single spherical pores (Brownstein & Tarr, 1979). These are shown in the following equations. The results obtained from computing the analytical solutions and the subsequent comparison with simulated data for single spherical pores are presented in Chapter 4.

In Equation 2.20, M stands for the total magnetisation in the system, M_0 for the initial magnetisation and I_n is the corresponding intensity function for each relaxation time T_n. I_n can be derived from Equation 2.21, where ε_n are the positive roots of the equation 2.23, and T_n can be derived from Equation 2.22, where D is the diffusion coefficient, R_s the radius of the pore, T_{2B} is the bulk relaxation time and ε_n are the positive roots of the equation 2.23, where ρ_2 stands for the surface relaxivity, R_s the radius of the pore and D is the diffusion coefficient.

$$M = M_0 \sum_{l}^{n} I_n \exp\left(\frac{-t}{T_n}\right) \tag{2.20}$$

$$I_n = \frac{12(sin\varepsilon_n - \varepsilon_n cos\varepsilon_n)^2}{\varepsilon_n^2(2\varepsilon_n - sin(2\varepsilon_n))} \tag{2.21}$$

$$\frac{1}{T_n} = \frac{D\varepsilon_n^2}{R_s^2} + \frac{1}{T_{2B}} \tag{2.22}$$

$$1 - \varepsilon_n \, cot\varepsilon_n = \frac{\rho_2 R_s}{D} \tag{2.23}$$

Chapter 3

Methods

As discussed in Section 2.5.2, numerical simulations are a great approach to investigate pore coupling effects and the factors controlling them. They allow for the construction of very specific scenarios and the targeted alteration of parameters in a way that experimental approaches do not. For this study, the Random Walk method has been chosen over the Finite Elements method because of its reputation as a well-established approach and its accessibility, in terms of software requirements and code availability.

The Random Walk NMR simulation code used for this study was the C++ "Single-phase NMR network simulation code" developed by Talabi (2008) at the London Imperial College. In this code, two types of network elements are defined by the user: "nodes", representing the pore spaces and "links" for the pore throats. A user-defined amount of "walkers", representing hydrogen atoms, are placed inside of the network elements in random locations, but at equal volume densities, and are left to move at a user-defined unit-distance within the elements. When a walker touches a solid network element (representing a pore or throat wall) it has a certain probability to relax and leave the simulation. This probability will depend on the surface relaxivity value chosen by the user. If a walker "survives" the collision, it will be bounced back to a random location inside the network element. If it does not "survive", it will relax and effectively reduce the total magnetisation of the system. If the pores are not connected through a throat, the walkers are unable to move from one pore to the other, but if a throat is present and their diameter is bigger than the unit-distance at which the walkers move, diffusion between pores is possible. The normalised magnetisation in the system is recorded at user-determined time intervals, therefore documenting the loss of magnetisation from surface relaxation. It also calculates an estimation of the magnetisation loss to bulk relaxation.

The outputs of the simulations are time-series of the decay of magnetisation in the system, as well as files that can be used to invert the results into T_2 relaxation times with the help of a provided MATLAB inversion code (see Section 3.6). Both the magnetisation decay time series and the T_2 relaxation times can be studied and compared to each other to visualise and quantify the changes taking place in each scenario as the geometries become pore-coupled.

As a first step for the numerical study of pore coupling, different geometries to be tried were chosen and described in a format compatible with the NMR network simulation code (Statoil format). This process is explained in Section 3.1 and Section 3.2. In Section 3.3, the input and output files of the Random Walk simulation are explained. The customisation of the input simulation parameters is described in Section 3.4. A description of the specific commands used

to compile and run the code are given in Section 3.5, while Section 3.6 and Section 3.7 handle the inversion step and the data analysis process.

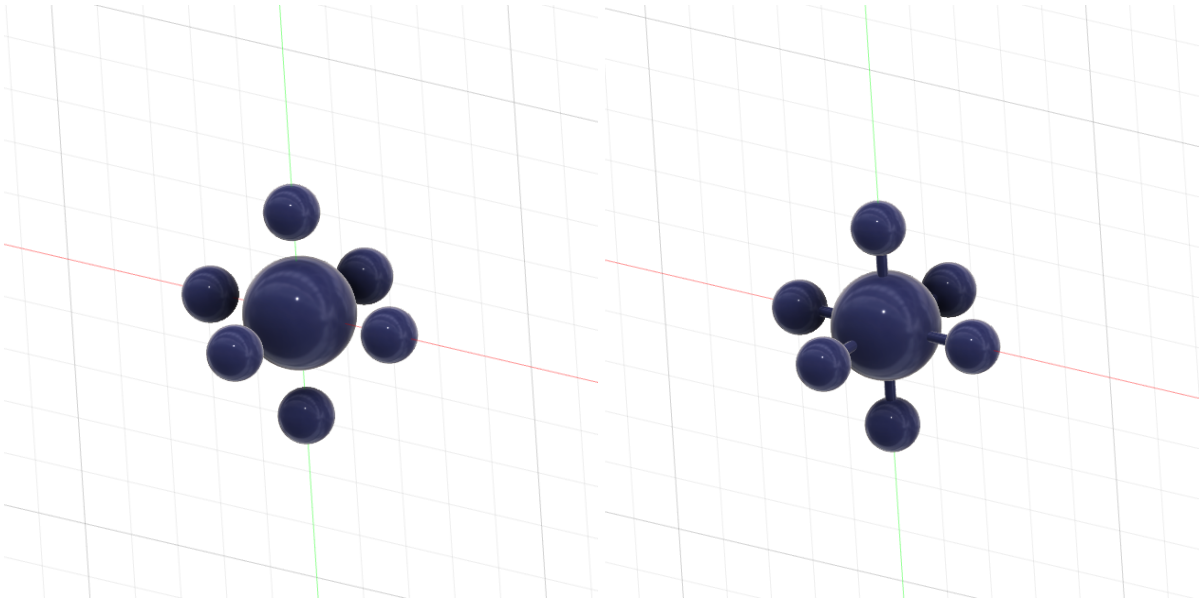
3.1 Geometry design

For the realisation of this master's thesis, multiple scenarios within three different geometries were tested. In this section the geometries chosen will be illustrated and explained, followed by a detailed description of how these geometries were translated into the four ASCII input files required for the C++ NMR code.

The first and simplest geometry tested, Geometry 0, was that of a single spherical pore using different radius ($1\mu m$, $10\mu m$, $20\mu m$ and $100\mu m$). The magnetisation decay data obtained from the Random Walk simulation were compared to the analytical solutions, which are available only for relaxation in single pores (see Chapter 4). This exercise served as a way to verify the Random Walk algorithm.

For Geometry 1 and 2, representing systems with similar and dissimilar pore sizes, a simple approach was desired, with few pores in the network. A reduced number of pore spaces kept the computational power required and the simulation times manageable. Nevertheless, constructing the geometries with only two pore spaces was also avoided for two reasons: first, and particularly for the geometry with pore sizes of dissimilar diameters, the Random Walk code, assigning "walkers" according to the volume proportion of a network element, would largely initiate "walkers" on the larger pore space. This would lead to the under-representation of the smaller pore in the magnetisation decay signal. This can be avoided, by designing a geometry that contains more smaller pores than larger pores. Secondly, when considering the pore surface-area-to-throat-width ratio, since the width of the throat connecting a larger and a smaller pore is the same on both ends, it is clear that for a "walker" initiated in the smaller pore it will be easier to "find" a throat than for a "walker" initiated in the bigger pore. In a bigger pore, a "walker" has a much larger chance to touch a pore wall than enter the throat opening solely for the fact that there is more pore wall available. This means that diffusion from the smaller pore to the larger pore will be largely over represented in comparison to diffusion from the larger to the smaller pore, specially for Geometry 2, which has a larger pore size difference. With these considerations in mind, Geometries 1 and 2 were designed as follows:

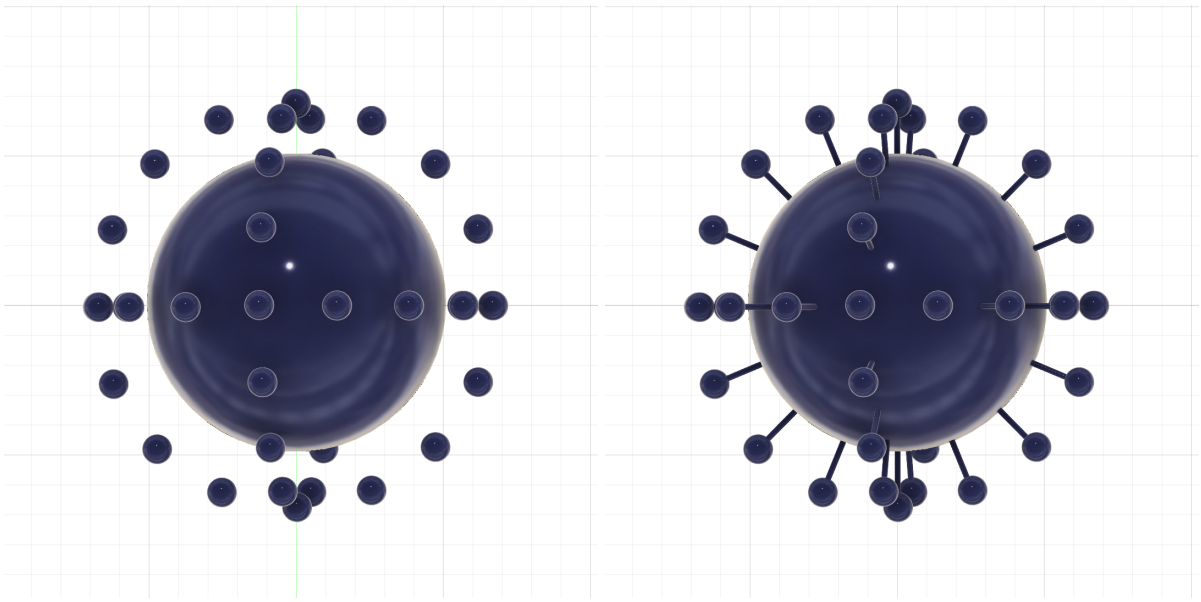
Geometry 1 consists of one larger pore ($20\mu m$ in diameter) surrounded by six smaller pores ($10\mu m$ in diameter) in the configuration shown in Figure 3.1. The smaller pores always remain unconnected from one another, but are connected to the larger pore to test the pore coupling effect (connected scenario). Geometry 1 is supposed to represent a system with two similar pore sizes.



(a) Unconnected scenario (b) Connected scenario

Figure 3.1: Representation of Geometry 1, a system with two similar pore sizes. The spherical pores are at scale, one larger pore surrounded by six smaller ones, with diameters of 20μm and 10μm. Panel (a) shows the unconnected scenario, while Panel (b) shows the connected scenario to probe the factors controlling pore coupling. These figures are for illustrative purposes only and were not used as input files.

Geometry 2 consists of one larger pore (100μm in diameter) surrounded by 42 smaller pores (each 10μm in diameter) placed around the larger pore in rings at 90° to each other, in the configuration shown in Figure 3.2. The smaller pores always remain unconnected from one another, but are eventually connected to the larger pore to test the pore coupling effect. Geometry 2 is supposed to represent a system with two dissimilar pore sizes.



(a) Unconnected scenario (b) Connected scenario

Figure 3.2: Representation of Geometry 2, a system with two dissimilar pore sizes. The spherical pores are at scale: one large pore surrounded by 42 small ones, with diameters of $100\mu m$ and $10\mu m$ respectively. Panel (a) shows the unconnected scenario, while Panel (b) shows the connected scenario to probe the factors controlling pore coupling. These figures are for illustrative purposes only and were not used as input files.

3.2 Geometry network files

These distinctly different geometries must be translated into network input files that the C++ Random Walk NMR code can read. The network input files are four ASCII files (link1.dat, link2.dat, node1.dat and node2.dat) following the Statoil format. The two main concepts introduced are "nodes", referring to the pore spaces, and "links", referring to the throats (although, technically, links connect pores from their centres, so their total length is greater than that of the throats).

For large and complex geometries, encoded in traditional network files, there are codes available that will create these ASCII files as outputs (see, e.g., Dong & Blunt, 2009). Nevertheless since the geometries used for this study were designed to be simple, the ASCII files were created manually by filling each data entry with the appropriate information.

The four ASCII files will now be described in detail. More explanations regarding the ASCII geometry files can be found in Talabi (2008) or Torland (2018). Examples for each of the four network files can be found in Appendix A for the different geometries used in this master's thesis.

File link1.dat

The link1.dat file contains information regarding the throats connecting the pores in the system. The first line of the file will have, as a single entry, the total numbers of throats. Each throat will have its own line in the file, containing the following six data entries:

1. Throat index: a number identifying each respective throat.
2. Pore 1 index: the index identifying one of the pores the throat is connecting. Which pore is labeled as "1" or "2" is irrelevant as long as the nomenclature remains consistent.
3. Pore 2 index
4. Throat radius, in meters.
5. Shape factor (G) of the throat: a numerical value to describe the geometrical shape of the pore throat that can be calculated with Equation 3.1 (Mason & Morrow, 1991).

$$G = \frac{r^2}{4A} \tag{3.1}$$

   where r is the inscribed ratio and A is the cross-sectional area. For spherical pores and cylindrical pore throats the shape factor is the same (equal to $\frac{1}{4\pi}$) regardless of the sizes of the elements.
6. Total length of link, in meters, as measured from the center of pore 1 to the center of pore 2.

File link2.dat

The link2.dat file also concerns the throats in the system. Each throat has its own line in the file, each line containing eight data entries. The first three are repeated from the link1.dat file:

1. Throat index.
2. Pore 1 index.
3. Pore 2 index.
4. Length of pore 1, in meters, understanding "length" as the radius of the pore.
5. Length of pore 2, in meters, understanding "length" as the radius of the pore.
6. Length of throat, in meters, understood as the total length of the link minus the lengths of pores 1 and 2.
7. Throat volume, in cubic meters, calculated from the radius and the length of the throat depending on its geometry. This entry is very important, as it will determine how many "walkers" are initially placed inside the throat. The algorithm will not double-check this value or give out any warning if the wrong number is set by the user.
8. Throat clay volume, in cubic meters. The volume of clay present in the throat can be estimated based on the material being simulated or, for example, through the analysis of 2D thin sections, when real geological samples are being considered. For this study it was kept as zero.

File node1.dat

The node1.dat file describes the location of the pore spaces within the system and their degree of connection to other pores. The first line of the file contains only four data entries: the total number of pores and the length in meters (x-coordinate), width in meters (y-coordinate) and height in meters (z-coordinate) of the network. Each pore in the network is then described in its own line through the following data entries:

1. Pore index.
2. Pore x-coordinate.
3. Pore y-coordinate.
4. Pore z-coordinate.
5. Pore connection number, signaling how many other pores this particular pore is connected to.
6. Multiple entries concerning the pore connection: for a connection number i, there will be 2(i+1) entries:
    - the first i entries are the indices of the connected pores.
    - the (i+1) and (i+2) entries are the pore inlet and outlet status (with a value 0 for false and 1 for true). This entry is relevant for studies involving the flow of liquids through pore networks, where certain pores can be identified as the point where the liquid enters or leaves the system. This is not relevant for the simulation of NMR signals in porous media. For this study both these entries were set to 0.
    - the final i entries are the corresponding connecting throat indices. Here it is important to notice how, if "Pore 1" is connected to "Pore 2" through "Throat 1", "Pore 2", in the next line, will also be connected to "Pore 1" through "Throat 1" and not through "Throat 2".

File node2.dat

The node2.dat file provides information about the geometrical properties of the pore spaces. It contains as many lines as the network contains pores, each line having the following five data entries:

1. Pore index.
2. Pore volume, in cubic meters.
3. Pore radius, in meters.
4. Shape factor (G) of the pore (see shape factor of the throat).
5. Pore clay volume, which was set to zero for this study.

3.3 Random Walk simulation files

Input files

The C++ Single-phase NMR network simulation code developed by Talabi (2008) and made available by the Imperial College of London on their website comes within a folder that contains 12 files. Five of them are ".dat" input files, while the other seven are either ".cpp" or ".h" files and do not require adjustments by the user. These files will be listed and described below. They are can also be found, unabridged, in the Appendix B.2.

- "link1.dat", a geometry file described in Section 3.2
- "link2.dat", a geometry file described in Section 3.2
- "node1.dat", a geometry file described in Section 3.2
- "node2.dat", a geometry file described in Section 3.2
- "input.h", listing, storing and defining the variables that will be used in "input.cpp".
- "MersenneTwister.h", an algorithm that generates the random numbers to be used for the Random Walk.
- "NMRsim.h", listing, storing and defining the variables that will be used in "NMRsim.cpp".
- "threeSome.h", defining storing classes for two, three, four and five elements, necessary for the process of reading the input files in "Input.cpp" and "NMRsim.cpp".
- "Input.cpp", reads the input files provided by the user and assign them variable names that will be use in "NMRsim.cpp".
- "NMRsim.cpp", constructs the simulation of the magnetisation decay of the system by the relaxation of walkers as they encounter solid network elements.
- "Main.cpp", brings all of the elements together, initiates the random walk and records the output files.

The steps required to compile and run the code will be described in Section 3.5.

Output files

As output files, the algorithm will create and save inside the folder six ".csv" files. Two of them contain time series data regarding the magnetisation decay in the system:

- "_100.csv", containing 100 data-points of normalised magnetisation recordings equally spaced in time (column A being time in seconds and column B normalised magnetisation).
- " _nmr.csv ", containing all of the data gained through the simulation: Column A and B show the time (s) and normalised magnetisation solely through the surface relaxation mechanism, Column D and E for both surface and bulk relaxation, while column G and H

show the normalised magnetisation through surface, bulk and diffusion mechanism due to inhomogeinities in the magnetic field. These last two columns have the same values as columns D and E, as the keyword controlling relaxation due to gradients in the magnetic field (Magnet_gradient, see Section 3.4) was kept as zero. For this study, the values from column D and E were used to characterise the evolution in time of the magnetisation in the systems.

The other four ".csv" files ("K.csv", "M.csv", "T2.csv" and "Wm.csv"), are related to the inversion of the results to get T_2 relaxation times. Along with the C++ NMR simulation file, Talabi (2008) also developed a MATLAB code for the T_2-inversion step which requires these four files as inputs. The theory behind the inversion step is explained in Section 2.6, while the MATLAB code used, and the data entries of these .csv files, are explained in Section 3.6.

Each time after running the simulation, all of the output file were copied and saved at a different folder for storage and further analysis.

3.4 Random Walk simulation parameters

Beyond the four already described geometry files, the user must provide the Random Walk algorithm with 12 simulation parameters (keywords) through the input file "default.dat". In this section, each keyword and parameter will be described along with the considerations taken when choosing each value. A similar explanation of the keywords can be found on the manual accompanying the C++ code and also in Talabi (2008). An exemplary input file can be found in Appendix B.1.

TITLE

This is a user-defined title for the project that will be used at the beginning of the name of each output file. It can be chosen freely using letters, numbers, or a combination of both, without spaces.

WALKERS

This is the amount of "walkers" that will be placed within the pore spaces and throats of the network at the beginning of the simulation.

The placement of the "walkers" will be determined randomly but according to the volume of each network element (i.e., more "walkers" will be initiated inside larger pores than inside smaller pores). If the number chosen here is too small, the final magnetisation decay curve will be staircase-shaped and might not be reproducible. On the other hand, each extra walker placed in the network will increase the computer-power needed and the running time of the simulation, so a balance must be found. For this study, and following the findings of Toumelin *et al.* (2007), 2000 walkers were used for each simulation. A larger number, such as 10,000 walkers, common in NMR random walk literature, coupled to the low surface relaxation values chosen for this study, leads to too-long running times without an improvement of the signal.

TIME_STEP

This is the time, in seconds, required by a "walker" to diffuse a certain unit distance. The unit distance must be chosen by the user so as to be much smaller than the resolution of the network. The time step (Δt) is then calculated from that unit distance following Equation 3.2, where Δt is the time step in seconds, s is the unit distance in meters, and D is the diffusion constant of the liquid in m^2/s. (Talabi, 2008)

$$\Delta t = \frac{s^2}{6D} \tag{3.2}$$

For this study, a time step of $1\mu s$ was chosen, corresponding to a unit distance of $0.12\mu m$. Faster time steps will lead to faster simulation run times, and slower time steps to slower simulation runs. Nevertheless, when modeling connected networks, relatively slow time steps are necessary for the diffusion of walkers through the thin pore throat to take place.

DIFF_CONSTANT

This keyword stands for diffusion coefficient, a property inherent of the liquid diffusing in the network. In this case, the value for water ($2.9nm^2/s$), was chosen for all scenarios.

RELAXIVITY

This is the surface relaxivity ρ (see Section 5.2) in m/s. For this study, different surface relaxivity values were tried, including $10\mu m/s$, as an upper estimate for carbonate rocks (Talabi, 2008) and $40\mu m/s$ as an upper value for sandstones (Roberts *et al.*, 1995).

BULK_RELAXIVITY

This is the bulk relaxivity value, in seconds, necessary for the calculation of the bulk relaxation. For this study it was set to $2.5s$.

REPORT_TIME

This integer will be multiplied with the time-step to determine the time interval at which the algorithm will report and record the magnetisation decay. For this study a report time of 1 was chosen.

NETWORK

This is the name found at the beginning of the network files (e.g., for the network file "title_-link1.dat", this keyword would be "title"). All of the ASCII network files must begin with the same name and be stored in the same folder as the NMR simulation code.

NUM_TIME_DATA

This parameter is related to the data inversion MATLAB code. It is the number of data points, selected with an equal spacing from the magnetisation decay spectrum, that will be used for the T_2 inversion. It will correspond with the number of data entries in the output file M.csv (see Section 3.3). For this study it was kept at 100.

DECAY_CUT_OFF

This is the time, in seconds, after which the simulation should be terminated regardless of whether the magnetisation decay has reached zero or not (the common condition to stop the simulation). For this study, this keyword was set to 10, as all of the scenarios were expected to reach zero magnetisation before the 10 seconds.

INTER_ECHO_TIME

This keyword is related to the inter-echo times used for experimental NMR measurements. Since this study was purely numerical and analytical, with no experimental measurements to compare, this keyword was set to zero.

MAGNET_GRADIENT

This keyword is also necessary for the comparison of simulated results with experimental NMR measurements for scenarios presenting magnetic gradients. For this study, this keyword was set to zero.

3.5 Compiling and running the code

The C++ Single-phase NMR network simulation code was downloaded from the Imperial College of London website (https://imperialcollegelondon.app.box.com/v/NMR-single-phase-network-model) and, after the customisation of the input files, it was compiled and ran on a (early 2011) MacBook Pro with the macOS High Sierra operating system, a processor of 2.3 GHz Intel Core i5 and 8GB memory. The commands used for compiling and running the code from the terminal were as follows:

"g++ *.cpp -o NAME", to compile the C++ program, after opening the folder where all of the files are located into the terminal, and "./NAME", for running it.

The program initiates as a window pop-up, where the passage of time inside the simulation (left column) and the magnetisation in the system (right column) is shown in real time. Should there be a mistake in either of the input files, an error warning would specify the error taking place.

3.6 Inversion step

The theory behind the inversion algorithm has been described in Section 2.6. Here, only the chosen parameters and configurations of the MATLAB code provided by Talabi (2008) will be presented (MATLAB license number 338656). The code itself can be seen in Appendix C.

It is important to note that the inversion parameters cannot be changed a posteriori, as they are already embedded on the output files of the random walk simulation. They must be chosen before running the NMR simulation and changed within the input files of the NMR simulation. This means that one same run of an experiment cannot be inverted using different inversion parameters with this code. Likewise, a correct simulation of the magnetisation decay is required for a correct estimation of the T_2 values for the system: if, for example, a too small number of "walkers" was chosen and the magnetisation decay curves resulting from two simulation runs with the same parameters are substantially different, so will be the T_2 values after inversion.

The predetermined configuration provides 100 exponentially spaced possible T2-values in the range of 0.1ms to 10,000ms. This configuration is set in line 638 and 642 of the input file "NMRsim.cpp" (see Section 3.3) and can be customised in terms of the amount of possible T2-values and their range. For this study, the predetermined configuration was kept

On the MATLAB code itself ("T2_inversion.m"), it is important to customise line 106 and 128 by changing the predetermined value '50' to the value given as "NUM_TIME_DATA". In the case of this study, this value is 100. In order to run the code, the values from the Random Walk output files "Title_K" and "Title_M" must be manually accessed, copied and pasted to the respective ".csv" files already existing on the code's folder. This step is necessary due to changes on file formatting since 2009.

The output file of the MATLAB inversion code is a ".cvs" file titled "T2-Distribution", where the first column corresponds to the vector N (100 exponentially spaced possible T_2-values) and the second column to the vector A (volume fraction of pores decaying at a given T_2, the output of the inversion algorithm), following the notation of Section 2.6. After every inversion, these ".cvs" files were stored at a separate folder and then read and analysed using RStudio, as will be described in the next section.

3.7 Data processing and analysis

In order to read and analyse the magnetisation decay data gained as output files of the simulation code, the software RStudio was used. This was also the chosen software to handle the characteristic relaxation time (T_2) data files obtained from the inversion step. All of the plots presented in Chapter 4 and Chapter 5 were created using the R package ggplot.

Since the geometries used in this study were simple and had only two different pore sizes, the results of the inversion step were not continuous, gaussian-shaped, typical T_2-distribution curves, but rather characteristic relaxation times with only two peaks (or three, for some of the connected scenarios with signals emitted in the pore throats). In some scenarios, the T_2-relaxation files had only two signals that had a frequency different than zero. In these cases, these signals were directly assigned as characteristic for the smaller pore (fastest signal) or the larger pore (slower signal). Other scenarios presented two small peaks, with multiple low frequencies associated to similar relaxation times. In these cases, the weighted mean relaxation

time (with frequencies as weight) for each peak was calculated and used as the characteristic relaxation time for each pore environment. The frequencies used as weight were added together for each peak and used as the characteristic frequency to accompany the characteristic relaxation time.

Magnetisation decay curves and characteristic relaxation times were plotted for each scenario modelled. In the case of the single-pore geometries, the magnetisation decay curves were compared to the results from the analytical solutions provided by Ramakrishnan *et al.* (1999). While analytical solutions are not available for multi-pore system, these comparisons for single-pore systems was of great advantage both to verify the simulation code and to compare the multi-pore scenarios with the corresponding single-pore scenarios. This was helpful to understand the assignation and movement of "walkers" in the multi-pores system.

Chapter 4

Validation of Simulation and Inversion Algorithms

In this section, the results from the simulation and the inversion algorithm will be assessed and validated by comparing them to analytical solutions available for the magnetisation decay curve and the characteristic relaxation times. This is only appropriate for the unconnected systems, where no pore coupling effects are taking place.

4.1 Random Walk code validation

The simplest geometry to test is a single spherical pore fully saturated with water. For this geometry, analytical solutions developed by Ramakrishnan *et al.* (1999) are available. Comparing the magnetisation decay curves obtained through the Random Walk algorithm to the analytical solutions, calculated using the same values for the key parameters, is a great way to validate the simulation and to test the user's ability to run it. Three pore diameters were tested: $10\mu m$, $20\mu m$ and $100\mu m$. These three sizes are the building blocks of the bi-porous geometries studied. Each geometry was tested for two surface relaxivity scenarios: $10\mu m/s$, upper range for carbonate rocks, and $40\mu m/s$, upper range for sandstones (Chi & Heidari, 2015).

Table 4.1 shows the user-defined parameters that were used for all the simulations presented in this section. The fourth parameter is the pore's radius which will change in each of the following subsections.

Table 4.1: Parameters for the simulation and analytical calculation of the magnetization decay of a spherical pore. D stands for diffusion constant, ρ_2 for surface relaxivity and T_{2B} for bulk relaxivity.

D (nm^2/s)	ρ_2 (μm/s)	T_{2B}(s)
2.5	10 and 40	2.5

Diameter = 10 μm

Figure 4.1 shows the comparison between the magnetisation decay in the simulated system ("Data points", in black) and the analytical solution (in red) for a single spherical pore of radius

$R_s = 5\mu m$ and surface relaxivity $\rho = 10\mu m/s$. In Figure 4.2, the surface relaxivity has been increased to $\rho = 40\mu m/s$. Figure 4.3 compares the magnetisation decay curves obtained through the Random Walk simulation at the two different surface relaxivity values.

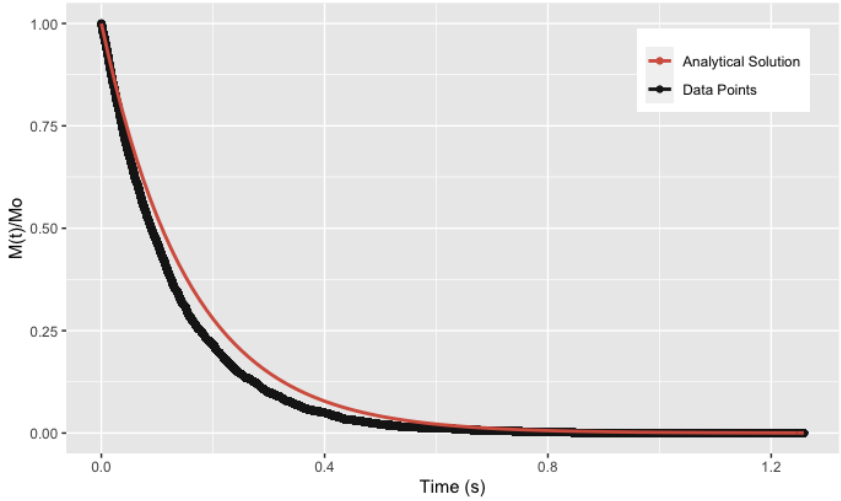


Figure 4.1: Comparison of the analytical solution and the simulated magnetization decay of a single pore of radius $R_s = 5\mu m$ and surface relaxivity $\rho = 10\mu m/s$.

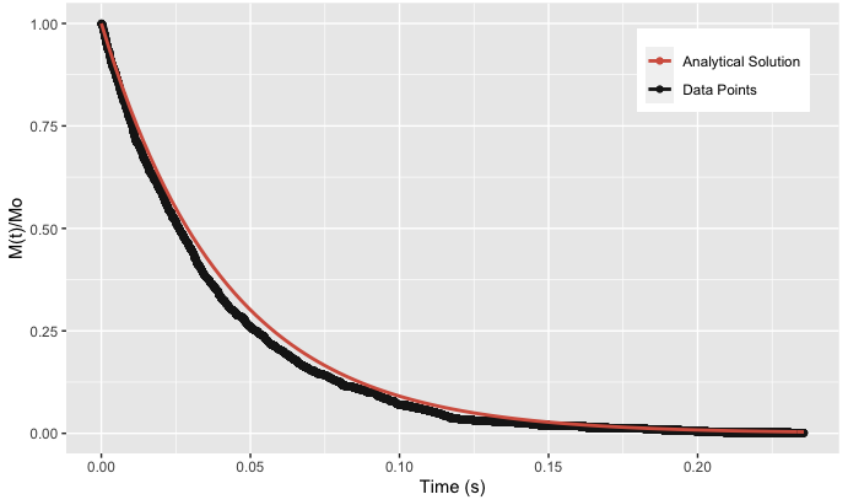


Figure 4.2: Comparison of the analytical solution and the simulated magnetization decay of a single pore of radius $R_s = 5\mu m$ and surface relaxivity $\rho = 40\mu m/s$.

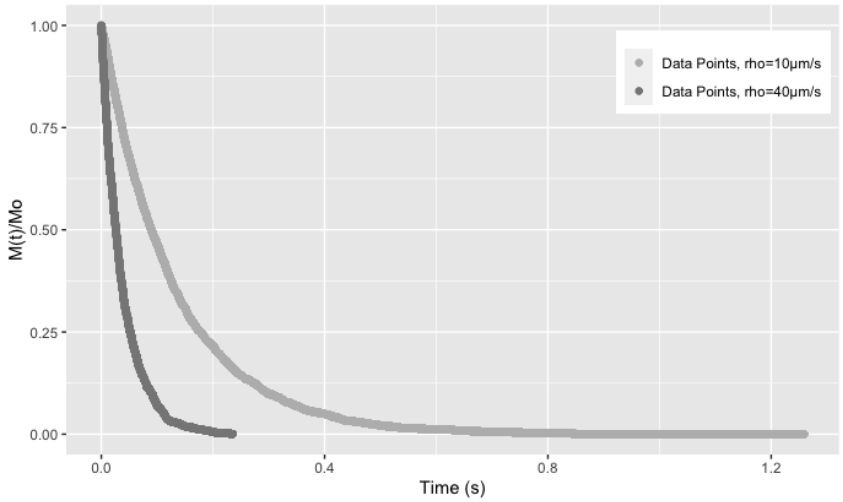


Figure 4.3: Comparison of the simulated magnetization decay of a single pore of radius $R_s = 5\mu m$ and surface relaxivity $\rho = 10\mu m/s$ and $\rho = 40\mu m/s$.

Diameter = 20 μm

Figure 4.4 shows the comparison between the magnetisation decay in the simulated system ("Data points", in black) and the analytical solution (in red) for a single spherical pore of radius $R_s = 10\mu m$ and surface relaxivity $\rho = 10\mu m/s$. In Figure 4.5, the surface relaxivity has been increased to $\rho = 40\mu m/s$. Figure 4.6 compares the magnetisation decay obtained through the random walk simulation at the two different surface relaxivity values.

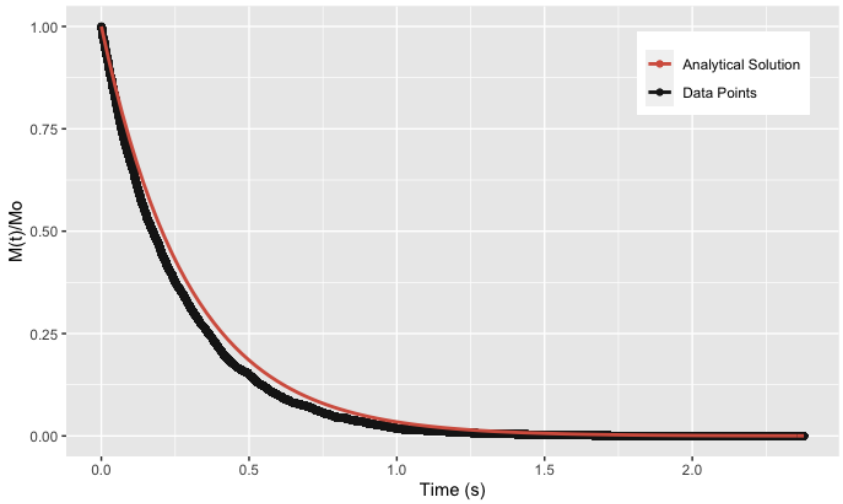


Figure 4.4: Comparison of the analytical solution and the simulated magnetization decay of a single pore of radius $R_s = 10\mu m$ and surface relaxivity $\rho = 10\mu m/s$.

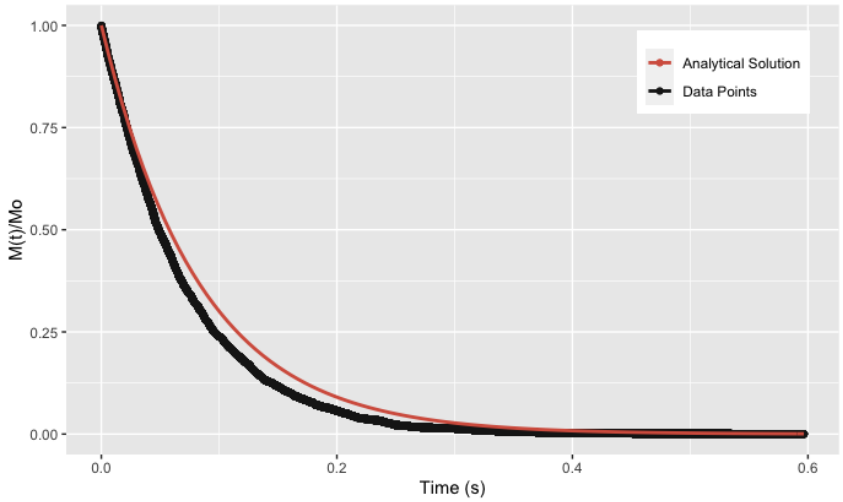


Figure 4.5: Comparison of the analytical solution and the simulated magnetization decay of a single pore of radius $R_s = 10\mu m$ and surface relaxivity $\rho = 40\mu m/s$.

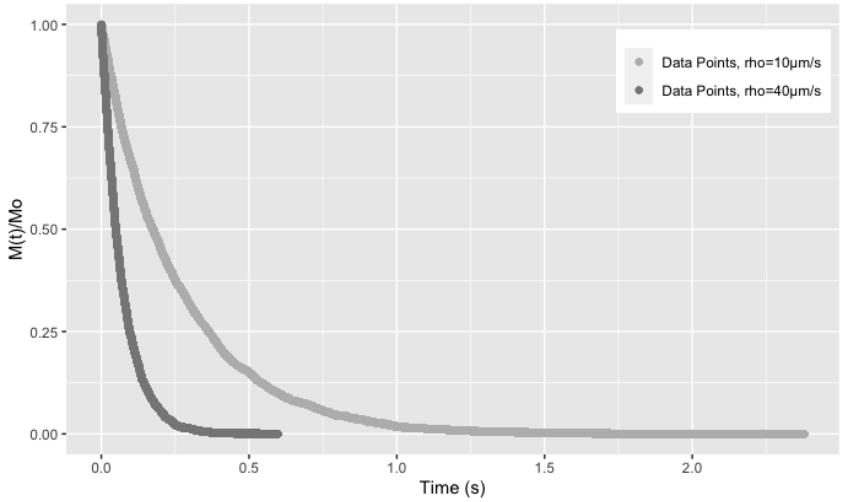


Figure 4.6: Comparison of the simulated magnetization decay of a single pore of radius $R_s = 10\mu m$ and surface relaxivity $\rho = 10\mu m/s$ and $\rho = 40\mu m/s$.

Diameter = 100 μm

Figure 4.7 shows the comparison between the magnetisation decay in the simulated system ("Data points", in black) and the analytical solution (in red) for a single spherical pore of radius $R_s = 50\mu m$ and surface relaxivity $\rho = 10\mu m/s$. In Figure 4.8, the surface relaxivity has been increased to $\rho = 40\mu m/s$. Figure 4.9 compares the magnetisation decay obtained through the random walk simulation at the two different surface relaxivity values.

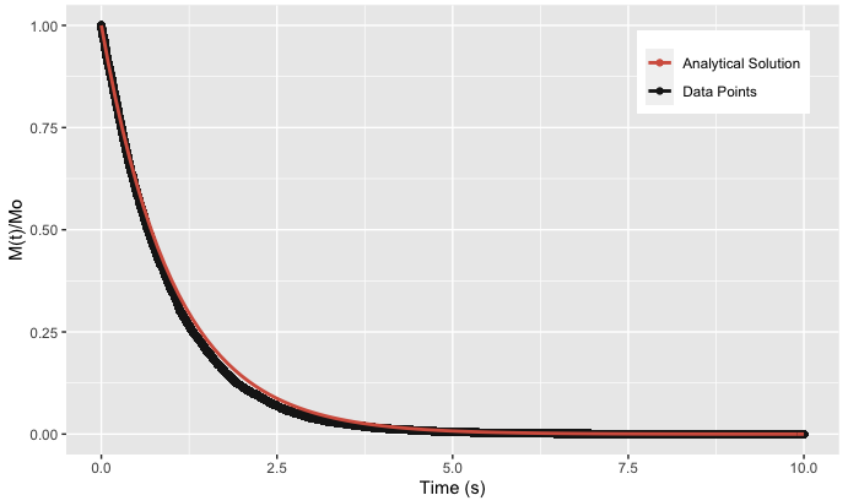


Figure 4.7: Comparison of the analytical solution and the simulated magnetization decay of a single pore of radius $R_s = 50\mu m$ and surface relaxivity $\rho = 10\mu m/s$.

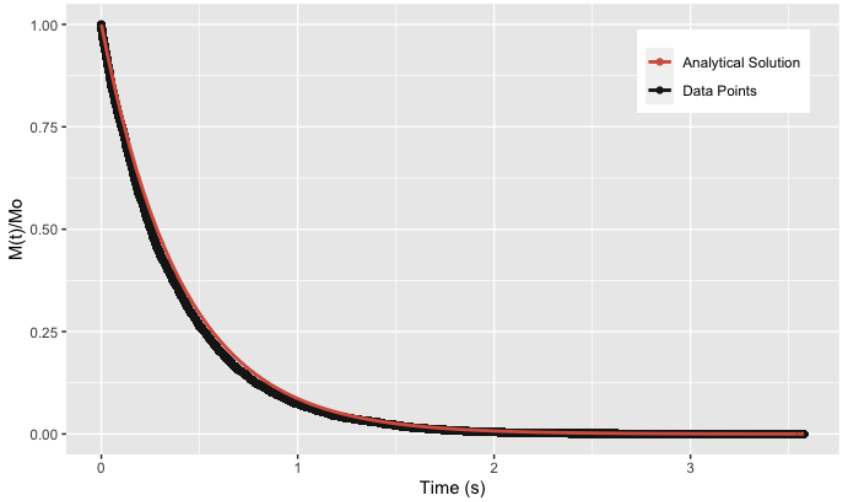


Figure 4.8: Comparison of the analytical solution and the simulated magnetization decay of a single pore of radius $R_s = 50\mu m$ and surface relaxivity $\rho = 40\mu m/s$.

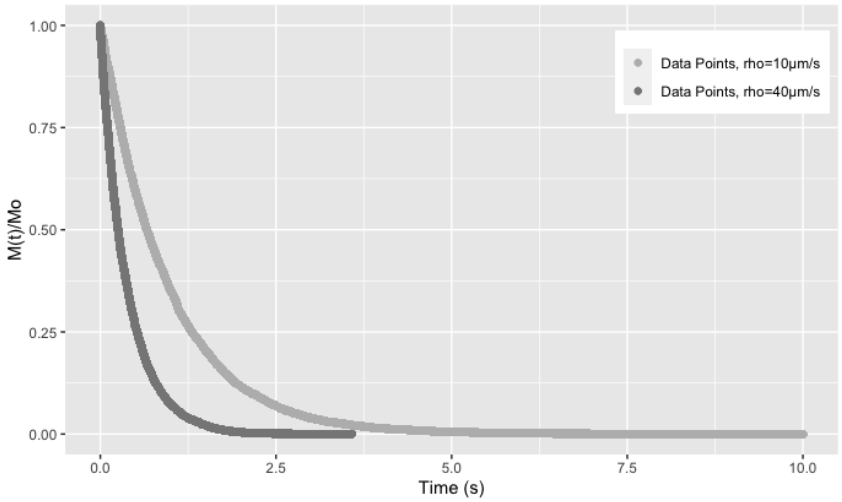


Figure 4.9: Comparison of the simulated magnetization decay of a single pore of radius $R_s = 50\mu m$ and surface relaxivity $\rho = 10\mu m/s$ and $\rho = 40\mu m/s$.

Discussion

The data points obtained from the numerical simulation correspond nicely to the results calculated from the analytical solution, which validates the model for further use. Comparing the magnetisation decay at different surface relaxivities also confirms the correct working of the random walk code: higher surface relaxivities will enhance relaxation, speeding the magnetisation decay in the system.

4.2 Validation of the inversion algorithm

After validating the random walk simulation by comparing the resulting magnetization decay curves to the available analytical solution for single pores, the inversion step remains to be tested. The characteristic relaxation times obtained after inverting the simulated results were compared to the T_2 values obtained from an "envelope calculation" following Equation 4.1. This "envelope calculations" only provides an approximation of the order of magnitude and general values that can be expected for the relaxation times given the parameters chosen for the single-pore simulations. They do not consider pore coupling effects so they must be compared with unconnected systems.

$$\frac{1}{T_2} = \rho \frac{A_s}{V} \tag{4.1}$$

where ρ is the surface relaxivity (chosen to be $40\mu m/s$), A_s is the surface area of the pore wall and V is the pore volume.

Table 4.2 shows the computed and simulated values for two pores with $R = 5\mu m$ and $R = 10\mu m$ respectively.

Table 4.2: Comparison between the simulated and computed T_2 values for two pores of 5 μm and 10 μm radius at $\rho = 40\mu m/s$ to validate the inversion algorithm.

	R = 5 μm	R = 10 μm	R = 50 μm
$T_{2simulated}$ (ms)	36	70	383
$T_{2computed}$ (ms)	42	83	417

The similarity between the computed and the simulated values is a good indicator of the effectiveness of the random walk simulation and the inversion algorithm.

Chapter 5

Results

The simulation results obtained for dual-porosity systems will be presented in this section. First, a general overview of the magnetisation decay in the two dual-porosity geometries will be provided, followed by the numerical exploration of both the surface relaxivity and network connectivity as control factors for pore coupling. Here, both magnetisation decay curves and characteristic relaxation times will be provided for each scenario along a short interpretation of the results. The magnetisation decay curves are plotted with the raw data obtained directly as output files from the Random Walk simulation, while the characteristic relaxation times are obtained after an inversion step using a second algorithm.

Two important parameters can be read from the characteristic relaxation time graphs: frequency, on the y-axis, and time, on the x-axis. The frequency can be interpreted as the fraction of the total water in the system that is measured at each pore environment (i.e., how the water is divided between the pores of different sizes, or how many pores of each size are present in each system). The characteristic relaxation time is linked to the sizes characteristic to each pore environment (i.e., the measurement of X different characteristic relaxation times signals the presence of pores of X different sizes in the sample).

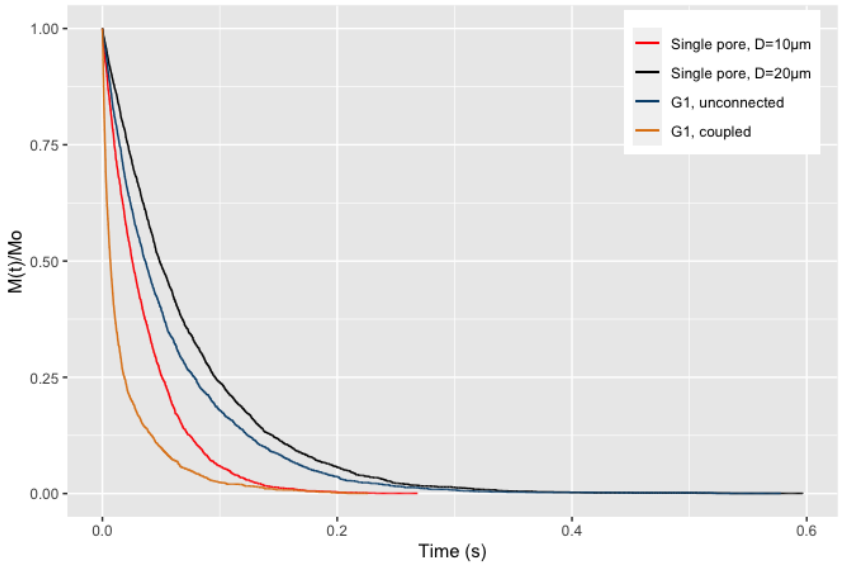
5.1 Dual porosity systems

The dual porosity geometries considered for this study represent pore networks with two distinct pore sizes: Geometry 1 is characterised by two pore diameters that are rather similar in size ($10\mu m$ and $20\mu m$), while Geometry 2 has two pore diameters largely different in size ($10\mu m$ and $100\mu m$). Figure 5.1 and Figure 5.2, respectively, show the magnetisation decay curves resulting from the NMR simulations in each geometry, in panels (a), with their corresponding characteristic relaxation times, in panels (b). Two scenarios are considered: unconnected pores, in blue, without any pore throats in the system, and connected pores, in yellow, where throats of $1\mu m$ in length and $2\mu m$ in diameter allow for magnetisation exchange between pores of different sizes. These two scenarios are, furthermore, compared to the relaxation behaviour of single pores of the diameters characterising each geometry: red for a single pore of diameter $10\mu m$ and black for a single pore of diameter $20\mu m$, for Geometry 1, and $100\mu m$, for Geometry 2.

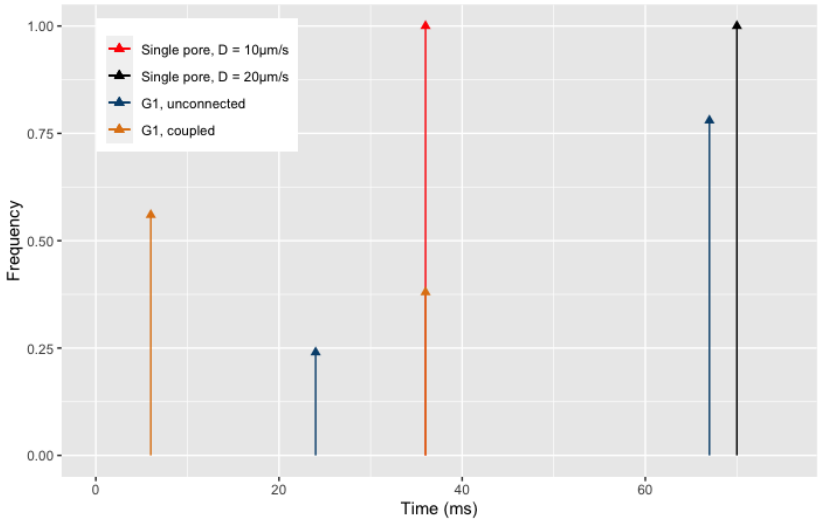
Panel (a) of Figure 5.1 and panel (a) of Figure 5.2 both show the same two major characteris-

tics: first, the magnetisation decay curves for the unconnected scenarios (in blue) are located in between the curves for the single pore geometries (in red and black), an indication that the Random Walk algorithm is successfully detecting both pore sizes within each geometry. Secondly, the magnetisation decay curves for the connected scenarios (in yellow) are significantly distinct from the curves for the unconnected scenarios (in blue), showing that the behaviour of the magnetisation decay is substantially different in the connected scenarios, which are pore-coupled. The coupled scenarios show, in both geometries, an accelerated magnetisation decay in comparison with the unconnected scenarios, hinting at the behaviour of the characteristic relaxation times.

Panel (b) of Figure 5.1 and panel (b) of Figure 5.2 show the characteristic relaxation times for Geometry 1 and Geometry 2. They show agreement with the findings of the magnetisation decay curves in panels (a) in respect to the differences between the unconnected scenarios and the single pore geometries and the differences between the unconnected and the pore-coupled scenarios. For the unconnected scenarios in both geometries, it would be expected for the slow relaxation peak to coincide with the single-pore peak of the bigger pore and for the fast relaxation peak to coincide with the smaller single-pore peak. This is loosely the case for the slow relaxation peaks, but, in both geometries, a bigger disagreement can be seen for the fast relaxation peaks, possibly indicating a better simulation resolution for bigger pores. In relation to the frequencies observed, both unconnected geometries show higher frequencies for the slow relaxation peak than for the fast one (i.e., a larger amount of water detected in the larger pore). In the case of Geometry 1 (Figure 5.1b), the frequency of the slow characteristic relaxation time decreases for the connected scenario in comparison to the unconnected scenario, as the frequency for the fast relaxation time increases. This signals a magnetisation exchange taking place predominantly from the bigger pore towards the smaller pores. The contrary can be seen for Geometry 2 (Figure 5.2b), where the predominant direction of magnetisation exchange is from the smaller pores towards the larger pore. This might be the case because, in Geometry 2, it is easier for the "walkers" initiated in the smaller pores to "find" the throat apertures than for the "walkers" initiated in the larger pore.

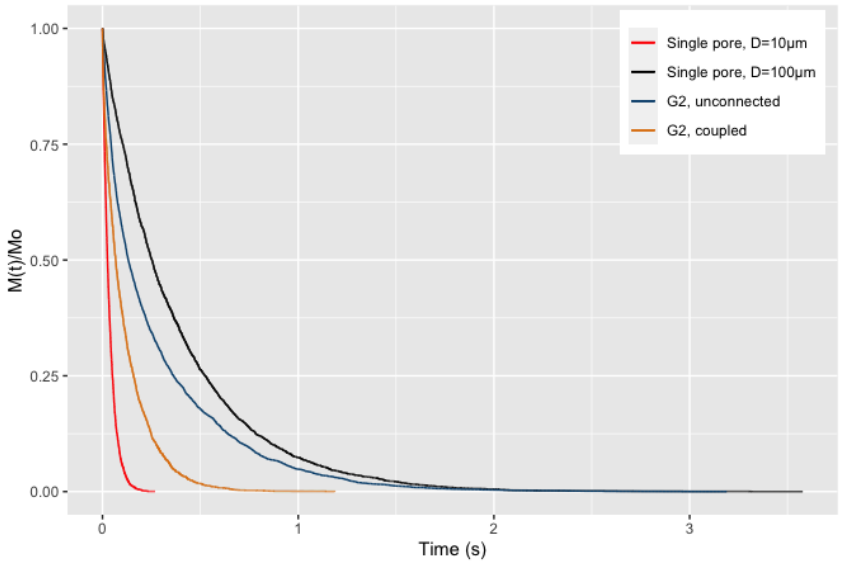


(a) Magnetisation decay curves.

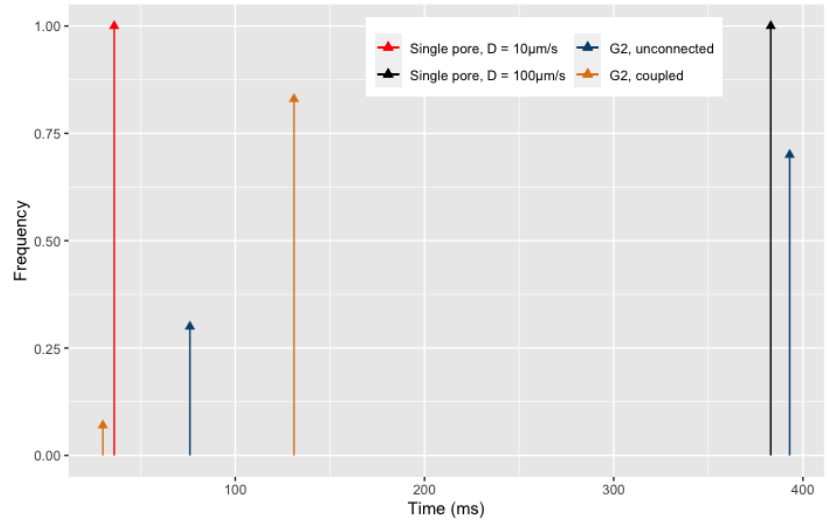


(b) Characteristic relaxation times.

Figure 5.1: Comparison of the simulated magnetization decay curves (Panel (a)) and the characteristic relaxation times (Panel (b)) for single pores of diameter $D = 10\mu m$ (in red) and $D = 20\mu m$ (in black) and an unconnected (in blue) and coupled (in yellow) scenarios for Geometry 1, a dual porosity system with pores of diameter $D = 10\mu m$ and $D = 20\mu m$. All scenarios at a surface relaxivity $\rho = 40\mu m/s$.



(a) Magnetisation decay curves.



(b) Characteristic relaxation times.

Figure 5.2: Comparison of the simulated magnetization decay curves (Panel (a)) and the characteristic relaxation times (Panel (b)) for single pores of diameter $D = 10\mu m$ (in red) and $D = 100\mu m$ (in black) and an unconnected (in blue) and coupled (in yellow) scenarios for Geometry 2, a dual porosity system with pores of diameter $D = 10\mu m$ and $D = 100\mu m$. All scenarios at a surface relaxivity $\rho = 40\mu m/s$.

In this section only one connected scenario was presented for each geometry, with a single degree of pore coupling. Different levels of pore coupling, as a function of the controlling factors, will be explored in the following sections.

5.2 Surface relaxivity as a pore coupling control factor

The first factor controlling pore coupling that will be studied, is the surface geochemistry of the geological material, understood as the surface relaxivity (ρ) of the pore walls. Surface

relaxivity influences the relaxation behaviour in systems that are not pore coupled, as can be seen in Figure 5.3a, for Geometry 1, and Figure 5.7a, for Geometry 2. It is aimed to probe how it also influences the degree of pore coupling in connected systems. To do so, four values for surface relaxivity are tested: $10\mu m/s$ and $40\mu m/s$, as values typically found in carbonate rocks and sandstones, respectively, and $100\mu m/s$ and $250\mu m/s$, for a comparison with considerably faster ρs. Due to the significance and our particular interest on the surface relaxivity values $10\mu m/s$ and $40\mu m/s$, they will, for both geometries, first be compared only with each other.

In this section, both magnetisation decay curves (raw data) and characteristic relaxation times (after data inversion) under different surface relaxivity values will be presented for Geometry 1 and for Geometry 2. The upper panels (a) of every figure will show the results for the unconnected geometries, while the lower panels (b) will show the results for the connected geometries. The scale of the x-axis (time in seconds) will be maintained for both panels of each figure to facilitate the comparison between the scenarios. At the end, results from both geometries will be compared to each other.

As expected, the magnetisation decay curves show, for both geometries, similar trends: on the one hand, higher surface relaxivity values lead to faster decays of the magnetisation in the systems and, on the other, Geometry 2, with the largest pore size, present slower decays in magnetisation in comparison to Geometry 1. This is because higher surface relaxivities enhance relaxation and because diffusing spins (represented by the "walkers") require longer times to reach the pore walls inside larger pores than inside smaller pores. These trends can be seen in Figures 5.3 and 5.5, for Geometry 1, and in Figure 5.7 and 5.9, for Geometry 2.

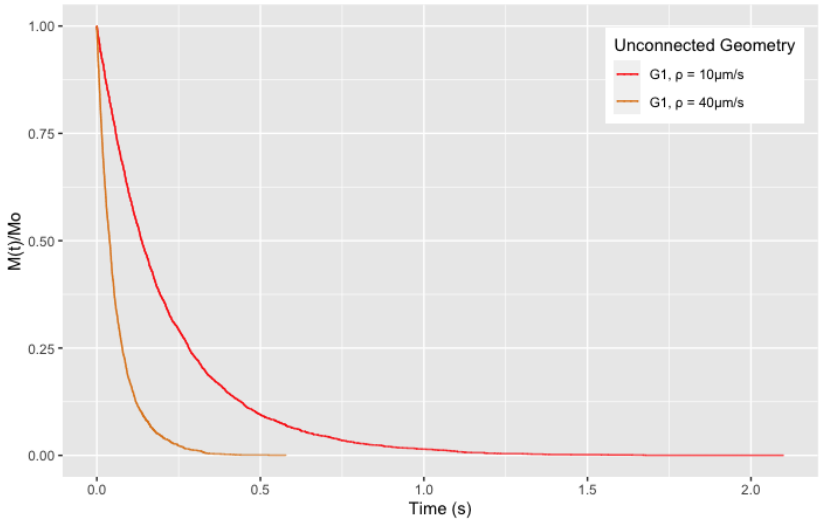
Characteristic relaxation times, after the inversion of the magnetisation decay results, are presented in Figure 5.4 and Figure 5.6, for Geometry 1, and Figure 5.8 and Figure 5.10, for Geometry 2. For each surface relaxivity value, two characteristic relaxation times can be distinguished, corresponding to the two different pore sizes in the geometries. The faster characteristic relaxation time signals correspond to the relaxation taking place in the smaller pores, while the slower characteristic relaxation time signals correspond to the larger pore.

For both geometries, the unconnected scenarios show how an increase in surface relaxivity leads to faster relaxation times. In the lower ranges ($10\mu m/s$ and $40\mu m/s$) this increase is much more pronounced than in the scenarios with higher surface relaxivity ($100\mu m/s$ and $250\mu m/s$).

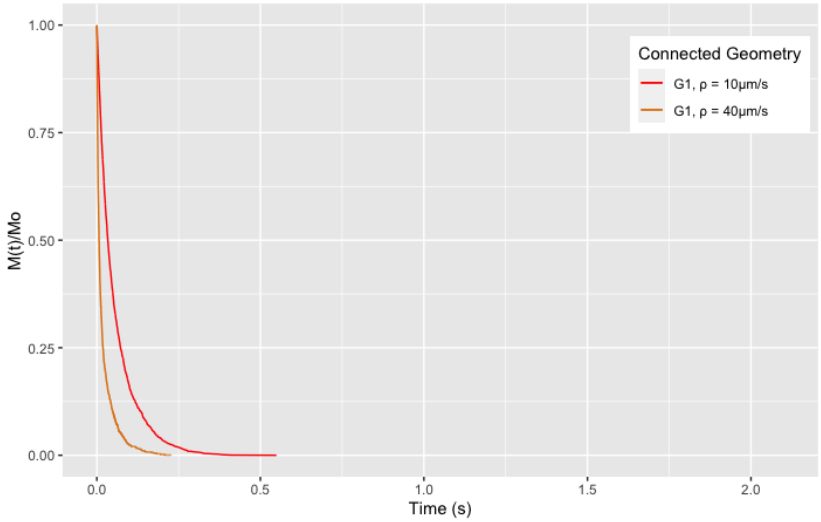
By comparing the characteristic relaxation times for the unconnected and the connected scenarios in both geometries, the effects of pore coupling can be visualised: in both geometries, connected systems always show faster characteristic relaxation times than the unconnected systems with the same surface relaxivity. This is true for both the fast and the slow relaxation peaks. The results also show that systems with lower surface relaxivities will be affected more strongly by pore coupling than systems with higher surface relaxivities.

An important difference between the relaxation behaviour in Geometry 1 and Geometry 2 are the shifts in frequencies associated to each characteristic relaxation time: for connected systems, the fast relaxation peaks in Geometry 2 are always much smaller than the slow relaxation peaks, with frequencies below 0.25 (see Figure 5.6b). This type of signal could lead to the interpretation of Geometry 2, as a network with only one predominant pore size. The connected systems of Geometry 1 do not show this feature: the fast relaxation peaks are mostly larger than the slow relaxation peaks and the slow relaxation peaks still mostly present large frequencies, well above 0.25, still suggesting the presence of two characteristic pore sizes (see Figure 5.4b).

Geometry 1

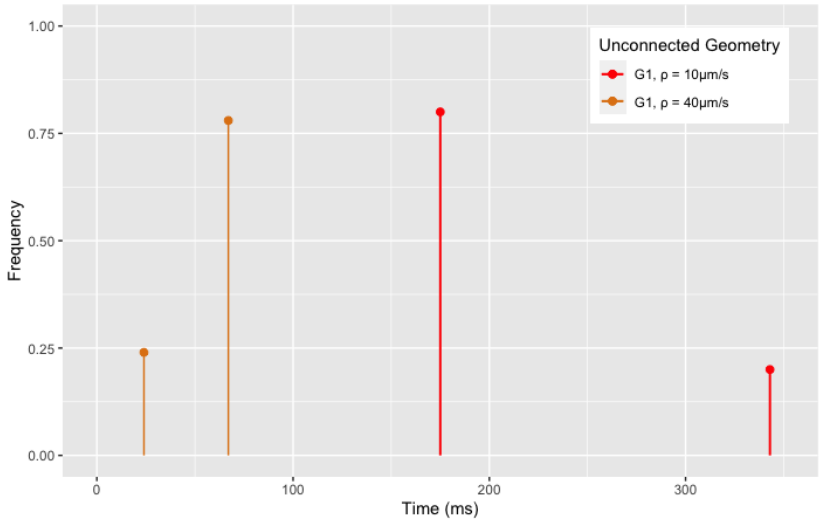


(a) Magnetisation decay curves for unconnected pores of two similar diameters at $\rho = 10\mu m/s$ and $40\mu m/s$.

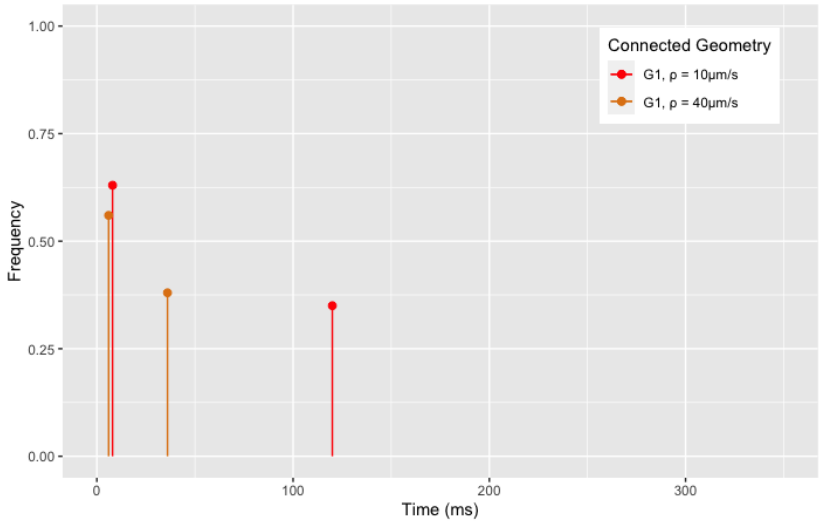


(b) Magnetisation decay curves for connected pores of two similar diameters at $\rho = 10\mu m/s$ and $40\mu m/s$.

Figure 5.3: Comparison of the magnetisation decay curves for a system of unconnected (panel (a)) and connected (panel (b)) pores of two similar pore diameters (10 and $20\mu m$) at surface relaxivity values of 10 and $40\mu m/s$.

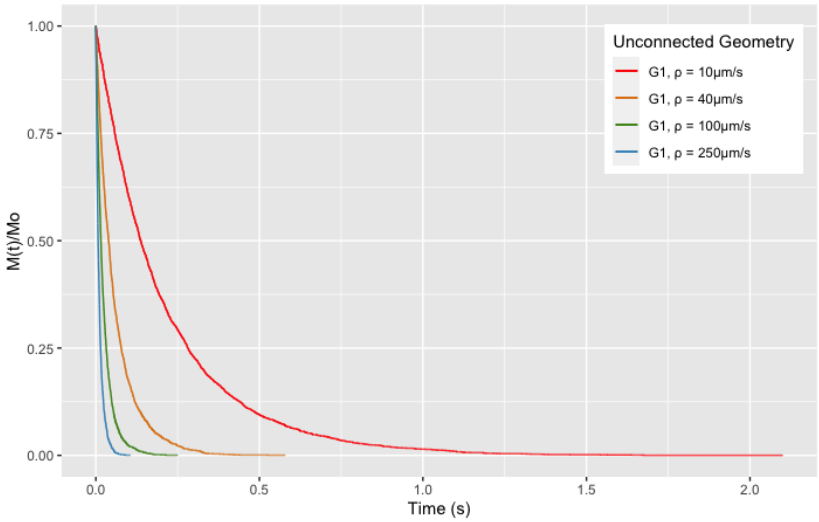


(a) Characteristic relaxation times for unconnected pores of two similar diameters at $\rho = 10\mu m/s$ and $40\mu m/s$.

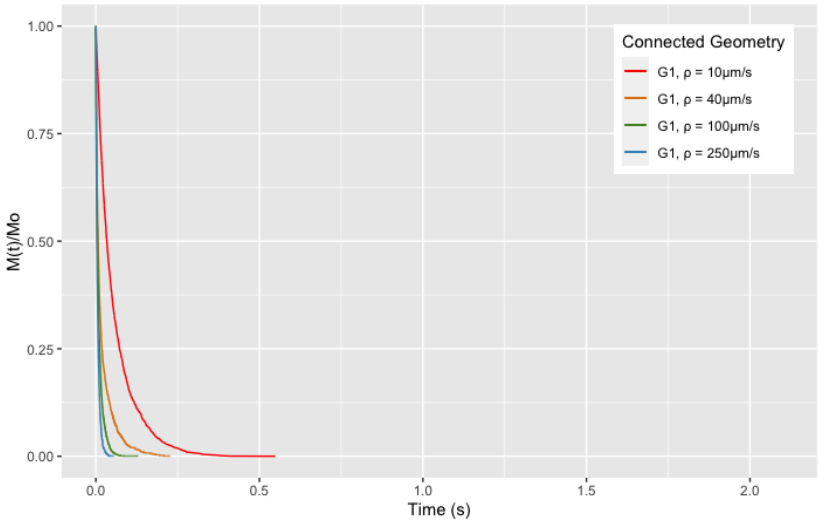


(b) Characteristic relaxation times for connected pores of two similar diameters at $\rho = 10\mu m/s$ and $40\mu m/s$.

Figure 5.4: Comparison of the characteristic relaxation times for a system of unconnected (panels (a)) and connected (panel (b)) pores of two similar pore diameters (10 and $20\mu m$) at surface relaxivity values of 10 and $40\mu m/s$.

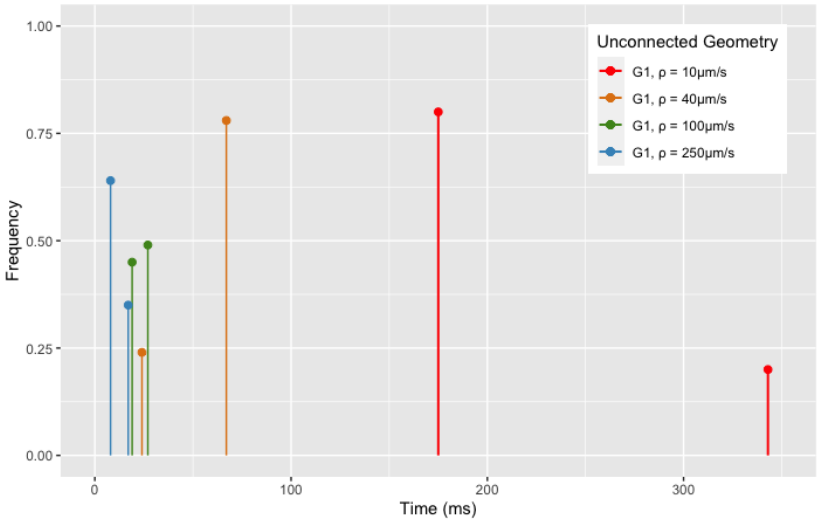


(a) Magnetisation decay curves for unconnected pores of two similar diameters at $\rho = 10\mu m/s$, $40\mu m/s$, $100\mu m/s$ and $250\mu m/s$.

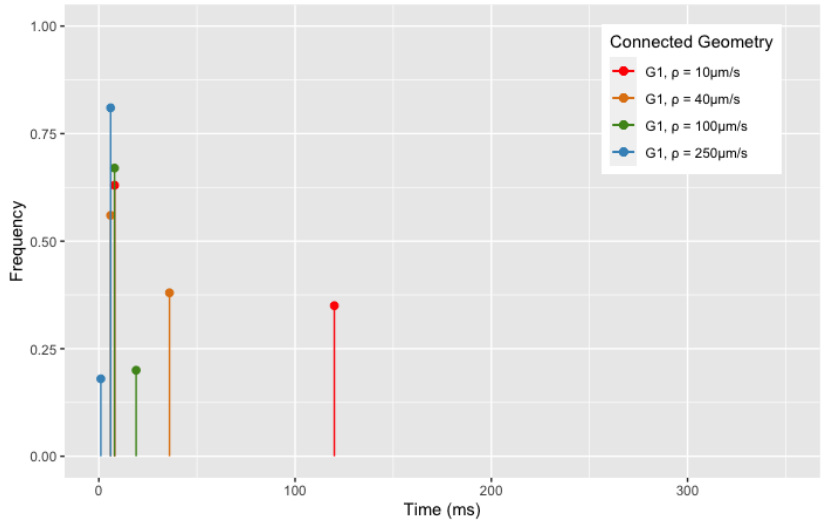


(b) Magnetisation decay curves for connected pores of two similar diameters at $\rho = 10\mu m/s$, $40\mu m/s$, $100\mu m/s$ and $250\mu m/s$.

Figure 5.5: Comparison of the magnetisation decay curves for a system of unconnected (panels (a)) and connected (panel (b)) pores of two similar pore diameters (10 and $20\mu m$) at surface relaxivity values of 10, 40, 100 and $250\mu m/s$.



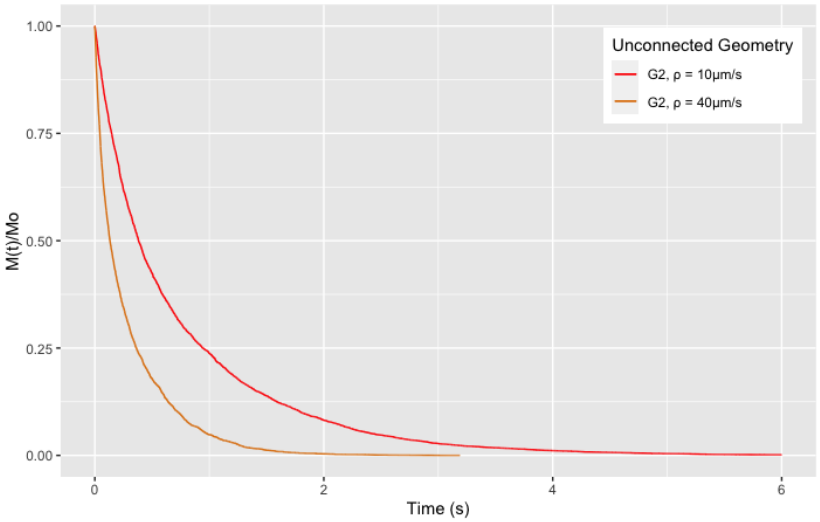
(a) Characteristic relaxation times for unconnected pores of two similar diameters at ρ = $10\mu m/s$, $40\mu m/s$, $100\mu m/s$ and $250\mu m/s$.



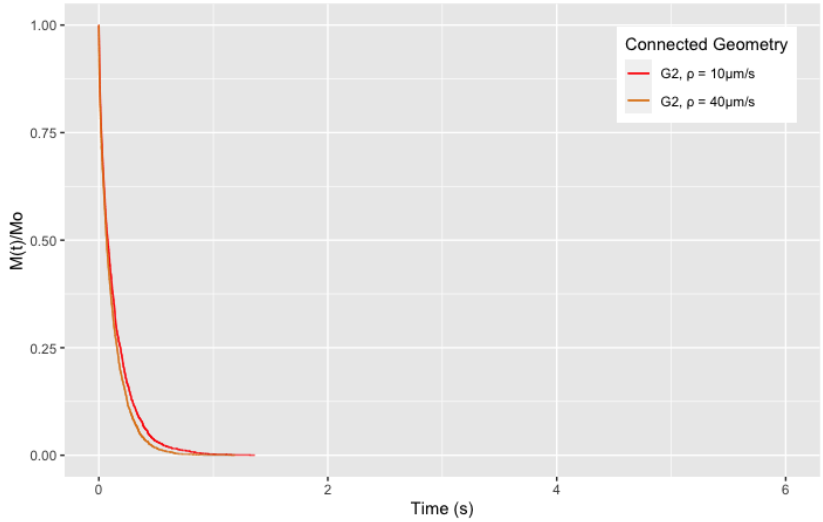
(b) Characteristic relaxation times for connected pores of two similar diameters at ρ = $10\mu m/s$, $40\mu m/s$, $100\mu m/s$ and $250\mu m/s$.

Figure 5.6: Comparison of the characteristic relaxation times for a system of unconnected (panels (a)) and connected (panel (b)) pores of two similar pore diameters (10 and $20\mu m$) at surface relaxivity values of 10, 40, 100 and $250\mu m/s$. $G1$ stands for Geometry 1 and ρ for surface relaxivity.

Geometry 2

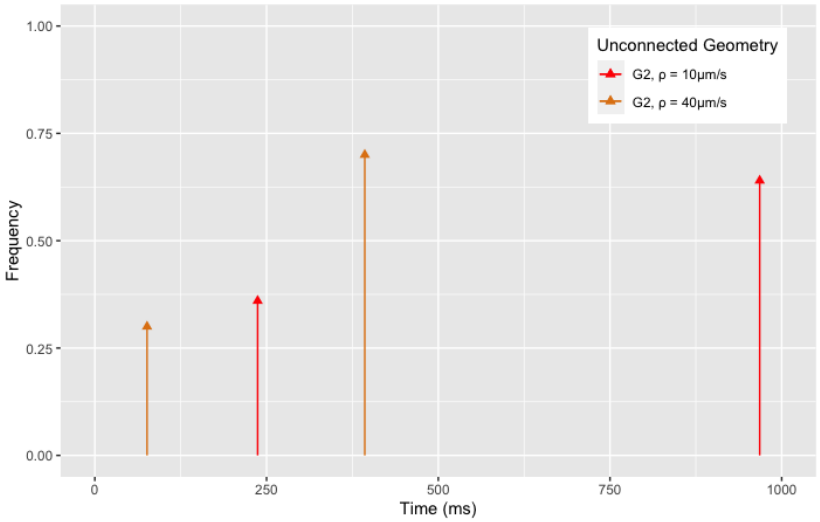


(a) Magnetisation decay curves for unconnected pores of two dissimilar sizes diameters at $\rho = 10\mu m/s$ and $40\mu m/s$.

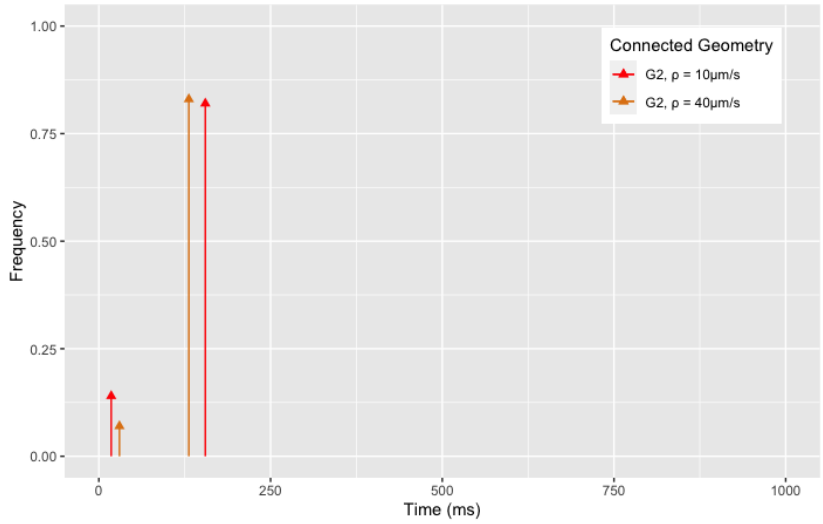


(b) Magnetisation decay curves for connected pores of two dissimilar sizes at $\rho = 10\mu m/s$ and $40\mu m/s$.

Figure 5.7: Comparison of the magnetisation decay curves for a system of unconnected (panels (a)) and connected (panel (b)) pores of two dissimilar pore sizes (10 and $100\mu m$ in diameter) at surface relaxivity values of 10 and $40\mu m/s$.

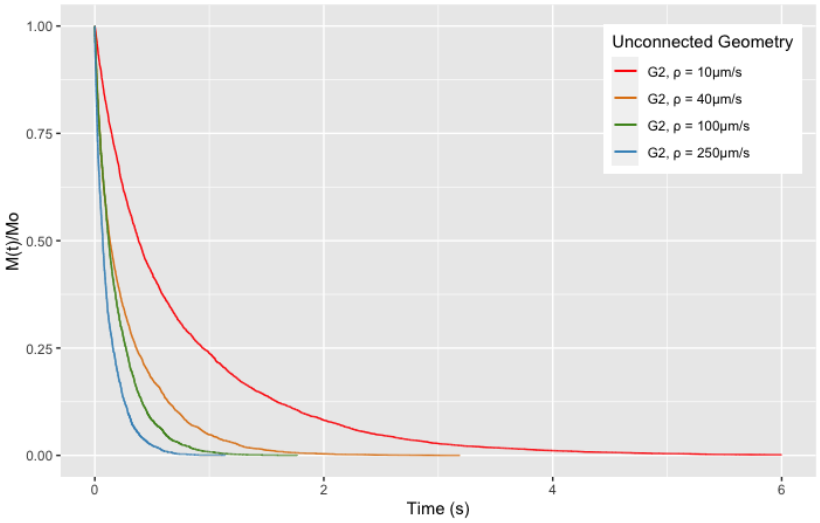


(a) Characteristic relaxation times for unconnected pores of two dissimilar sizes at $\rho = 10\mu m/s$ and $40\mu m/s$.

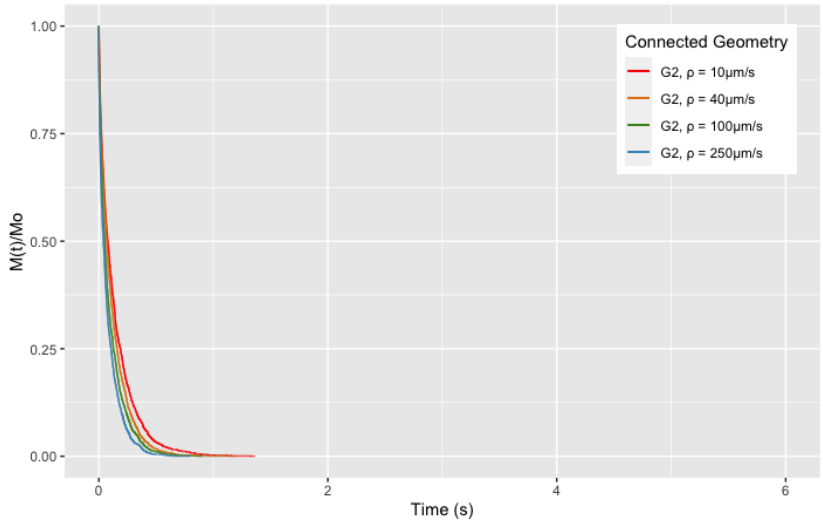


(b) Characteristic relaxation times for connected pores of two dissimilar sizes at $\rho = 10\mu m/s$ and $40\mu m/s$.

Figure 5.8: Comparison of the characteristic relaxation times for a system of unconnected (panels (a)) and connected (panel (b)) pores of two dissimilar pore sizes (10 and $100\mu m$ in diameter) at surface relaxivity values of 10 and $40\mu m/s$. $G1$ stands for Geometry 1 and ρ for surface relaxivity.

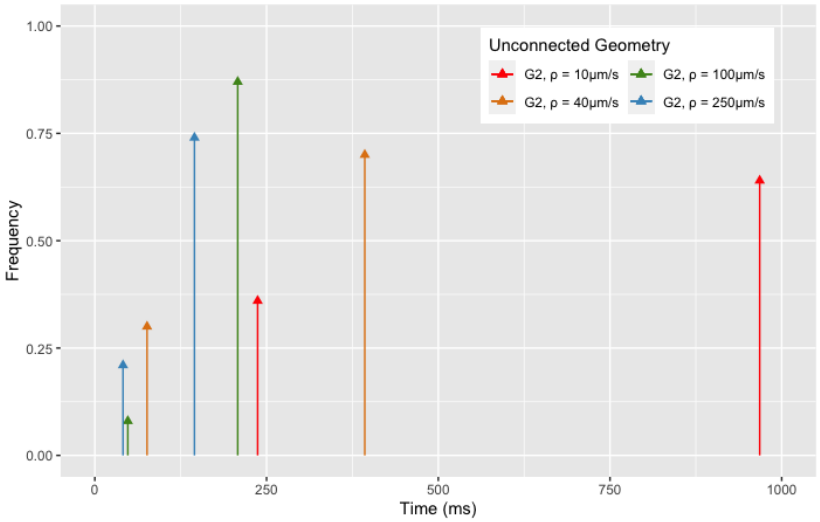


(a) Magnetisation decay curves for unconnected pores of two dissimilar sizes diameters at $\rho = 10$, 40, 100 and $250 \mu m/s$.

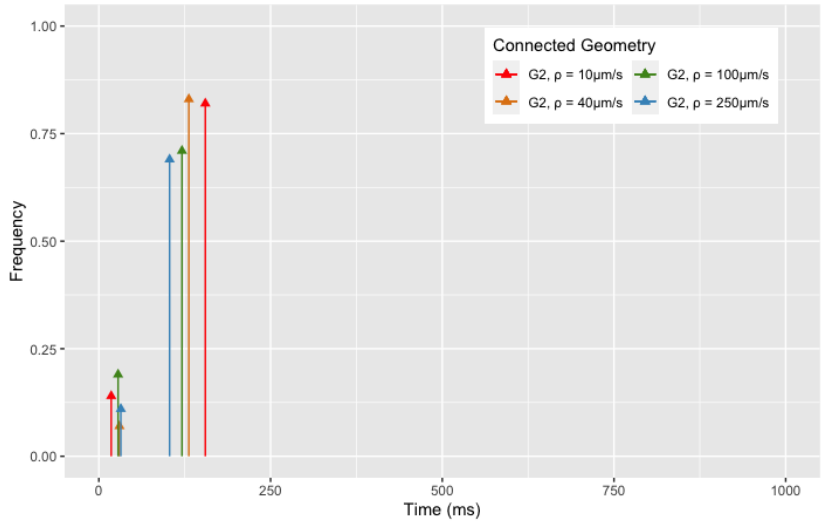


(b) Magnetisation decay curves for connected pores of two dissimilar sizes at $\rho = 10$, 40, 100 and $250 \mu m/s$.

Figure 5.9: Comparison of the magnetisation decay curves for a system of unconnected (panels (a)) and connected (panel (b)) pores of two dissimilar pore sizes (10 and $100 \mu m$ in diameter) at surface relaxivity values of 10, 40, 100 and $250 \mu m/s$. $G2$ stands for Geometry 2 and ρ for surface relaxivity.



(a) Characteristic relaxation times for unconnected pores of two dissimilar sizes at $\rho = 10$, 40, 100 and $250 \mu m/s$.



(b) Characteristic relaxation times for unconnected pores of two dissimilar sizes at $\rho = 10 \mu m/s$, $40 \mu m/s$, $100 \mu m/s$ and $250 \mu m/s$.

Figure 5.10: Comparison of the characteristic relaxation times for a system of unconnected (panels (a)) and connected (panel (b)) pores of two dissimilar pore sizes (10 and $100 \mu m$ in diameter) at surface relaxivity values of 10, 40, 100 and $250 \mu m/s$. *G*2 stands for Geometry 2 and ρ for surface relaxivity.

Comparison between geometries

Surface relaxivity, as a pore coupling control factor, seems to affect connected systems differently depending on the pore-size-ratio of the network (i.e., in Geometry 1 and Geometry 2). To further highlight these differences, the characteristic relaxation times for four scenarios with connected pore networks are plotted together and compared: Geometry 1 at $\rho = 10 \mu m/s$ and at $\rho = 40 \mu m/s$ and Geometry 2 at $\rho = 10 \mu m/s$ and at $\rho = 40 \mu m/s$. This is visualised in Figure 5.11.

As mentioned before, a main difference between the pore-coupled scenarios in Geometry 1

and Geometry 2 are the rather low-frequency fast relaxation peaks seen for Geometry 2 in comparison to Geometry 1: Geometry 2, unlike Geometry 1, might be misinterpreted as a system with only one predominant pore size. Nevertheless, when observing the location of the peaks, the actual characteristic relaxation times, it would seem that surface relaxivity is a stronger control factor for Geometry 1 than for Geometry 2. See, for example, the slow relaxation peaks on the farthest right of Figure 5.11: for Geometry 1, the slow relaxation time shifts drastically from $\approx 120ms$ to $\leq 30ms$, while for Geometry 2, the shift is less pronounced from $\approx 155ms$ to $\approx 130ms$.

Furthermore, changes in surface relaxivity in pore coupled network, will affect the size characterisation of the larger pores more strongly than of the smaller pores, regardless of the pore-size-ratio of the geometry. Again, this is true for the determination of the size of the pores but not for the estimation of the amount of pores of each size present in the network (i.e., the partition of water within the system).

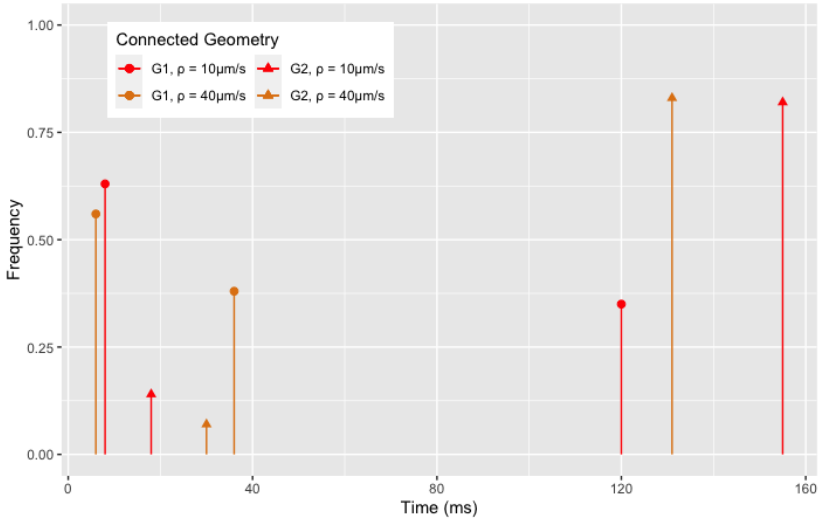


Figure 5.11: Comparison between the characteristic relaxation times of systems of two pores of similar (Geometry 1, G1) and dissimilar (Geometry 2, G2) diameters at surface relaxivities of $10\mu m/s$ and $40\mu m/s$.

5.3 Network connectivity as a pore coupling control factor

The second factor controlling pore coupling that will be studied is the network connectivity, influenced by the geometry of the throats connecting the pores. Both the effects of changing throat widths (TW) and changing throat lengths (TL) will be considered for Geometry 1 and Geometry 2 following a similar structure as in the previous section.

Changing pore throat widths

To study the influence of throat width on pore coupling, four different values for throat widths will be examined: $0.1\mu m, 0.5\mu m, 1\mu m$ and $2\mu m$. Surface relaxivity and throat length will be kept constant at $40\mu m/s$ and $1\mu m$, respectively. The magnetisation decay curves and their corresponding characteristic relaxation times obtained for each scenario will be compared to each

other and to the unconnected geometry. Figure 5.12 shows the results for Geometry 1 and Figure 5.13 for Geometry 2, while in Figure 5.14 a comparison between geometries is illustrated.

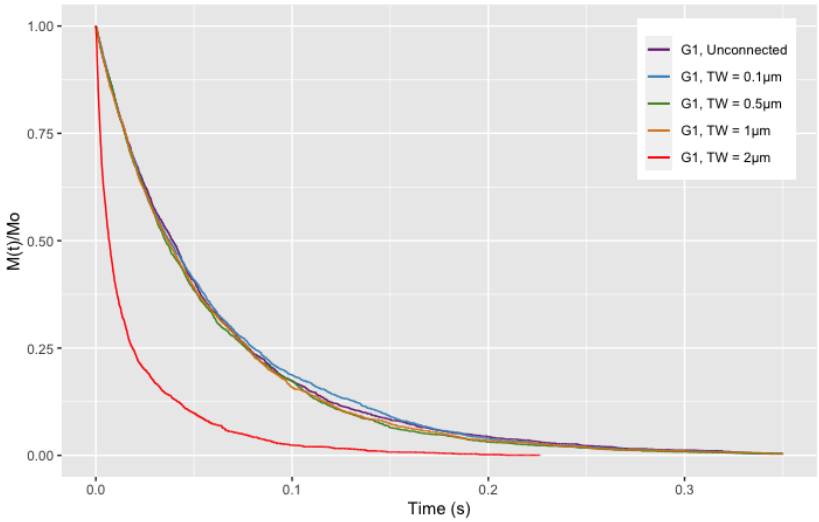
Unlike for changing surface relaxivity values, changing pore throat widths lead to more subtle differences in the magnetisation decay curves, making the characteristic relaxation times graphs the more valuable tools to interpret the changes occurring in the systems. Magnetisation decay curves are, nevertheless, still included in this manuscript as they represent the raw data obtained from the Random Walk simulation.

In both geometries, thin pore throats ($TW = 0.1\mu m$) will difficult the exchange of magnetisation between pore types. This can be seen particularly clearly for the slow characteristic relaxation time peak in Figure 5.12b, where the unconnected scenario (in purple) and the $TW = 0.1\mu m$ scenario (in blue) share the same slow characteristic relaxation time with a similar frequency. For both geometries, increasing the throat widths will increase the magnetisation exchange, as well as the effect of pore coupling. The systems are very well coupled for $TW = 2\mu m$ (in red), with the characteristic relaxation times being considerably distinct from the ones associated to an unconnected geometry (see Figure 5.12b and 5.13b).

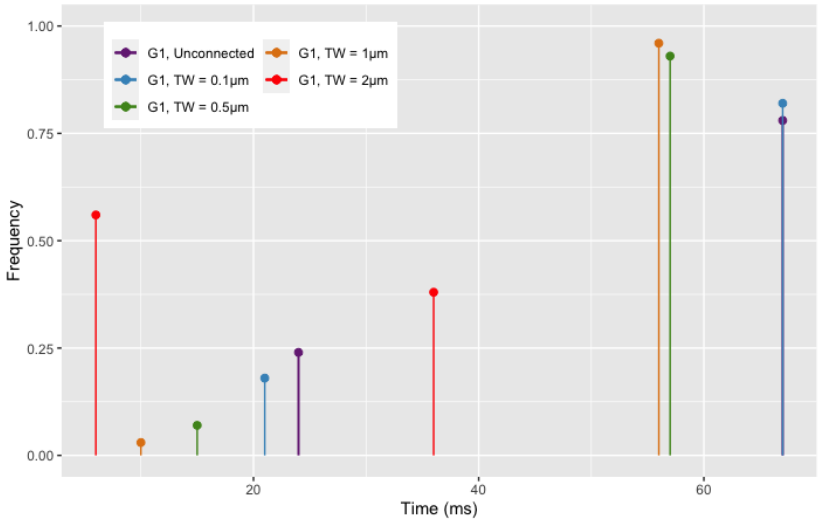
In both geometries, the fast characteristic relaxation times are more strongly affected by changes in the throat widths than the slow relaxation times; both in terms of the relaxation times and of the frequencies of the peaks in comparison to the unconnected scenarios. This is particularly relevant for Geometry 2 (Figure 5.13b). Here, the fast relaxation frequencies are below 0.12 for all tested throat widths, incorrectly suggesting a geometry with only one dominant pore size. This behavior is also present but not so pronounced in Geometry 1.

The slow relaxation peaks, on the other hand, present a threshold-like behaviour in relation to changes in throat widths: relaxation times and frequencies remain somewhat similar for the unconnected scenarios and the connected scenarios with throat widths in the range of $0.1\mu m - 1\mu m$ but show a drastic change for the throat width $TW = 2\mu m$. This is also particularly noticeable in Geometry 2.

Geometry 1



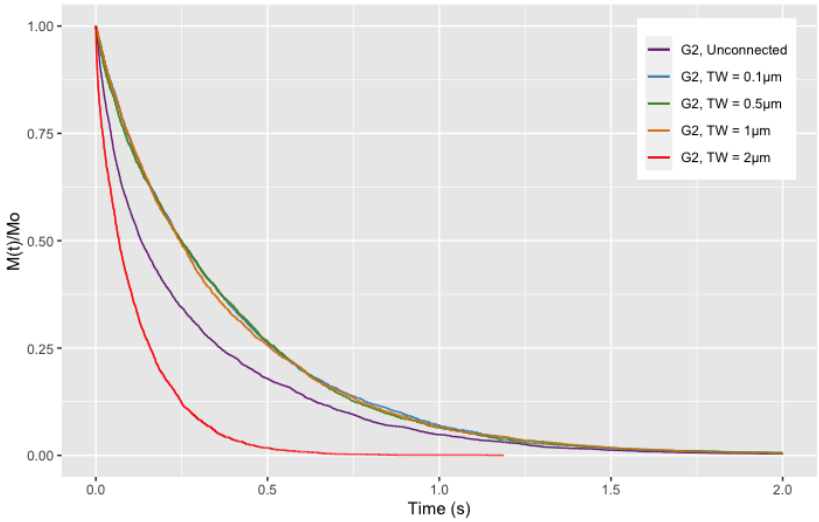
(a) Magnetisation decay curves.



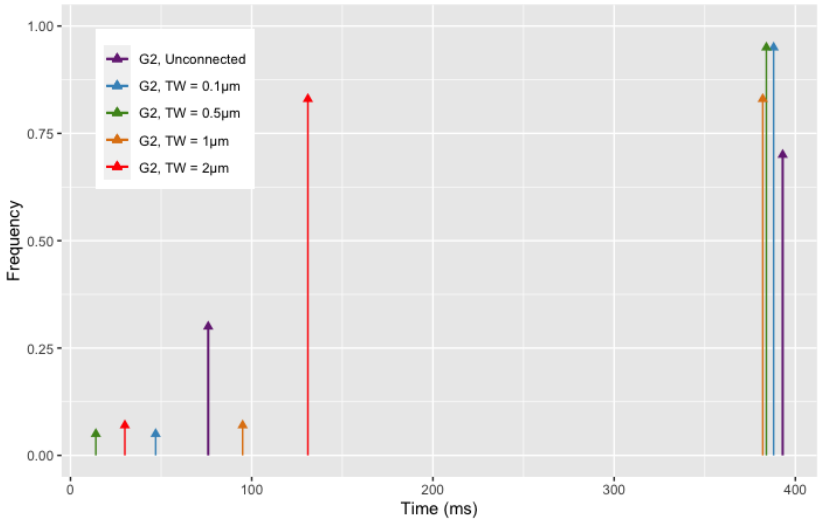
(b) Characteristic relaxation times.

Figure 5.12: Magnetisation decay curves (panel (a)) and characteristic relaxation times (panel (b)) for networks of pores of two similar diameters connected through pore throats of fixed length $TL = 1\mu m$ and varying throat widths at a fixed surface relaxivity $\rho = 40\mu m$. $G1$ stands for Geometry 1 and TW for throat width.

Geometry 2



(a) Magnetisation decay curves.



(b) Characteristic relaxation times.

Figure 5.13: Magnetisation decay curves (panel (a)) and characteristic relaxation times (panel (b)) for networks of pores of two dissimilar diameters connected through pore throats of fixed length $TL = 1\mu m$ and varying throat widths at a fixed surface relaxivity $\rho = 40\mu m$. *G2* stands for Geometry 2 and *TW* for throat width.

Comparison between geometries

To better illustrate the differences in relaxation behavior between the connected Geometries 1 and 2 with different throat widths, Figure 5.14 compares the characteristic relaxation times for Geometry 1, throat widths $0.5\mu m$ (green dots) and $2\mu m$ (red dots), and Geometry 2, throat widths $0.5\mu m$ (green triangles) and $2\mu m$ (red triangles).

Looking at the strongest coupling studied, Geometry 2 shows a bigger sensitivity to changes in throat width than Geometry 1 for the size characterisation of the larger pore. Geometry 1, on the other hand, presents a greater discordance on the frequencies of both peaks, which would

lead to a overestimation of the amount of smaller pores.

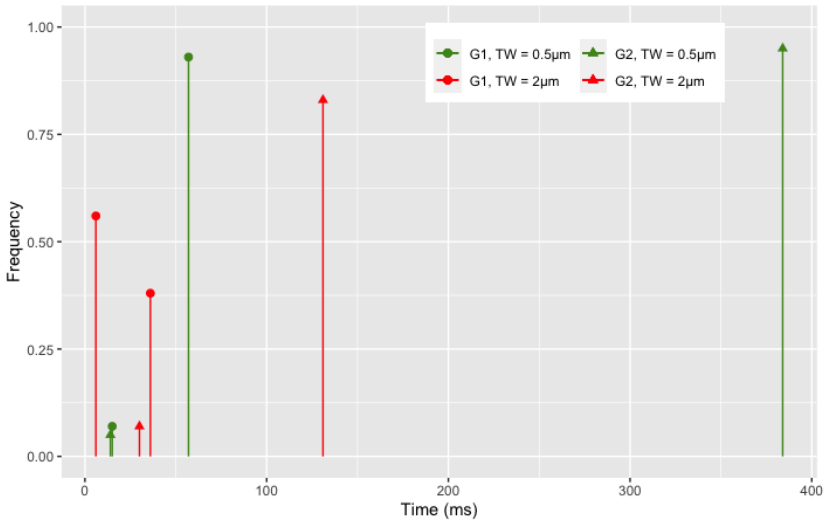


Figure 5.14: Comparison between the characteristic relaxation times for systems of two pores of similar (G1) and dissimilar (G2) diameters connected through pore throats of fixed throat lengths $TL = 1\mu m$ and throat widths of $TW = 0.5$ and $2\mu m$ at a fixed surface relaxivity $\rho = 40\mu m$.

Changing pore throat lengths

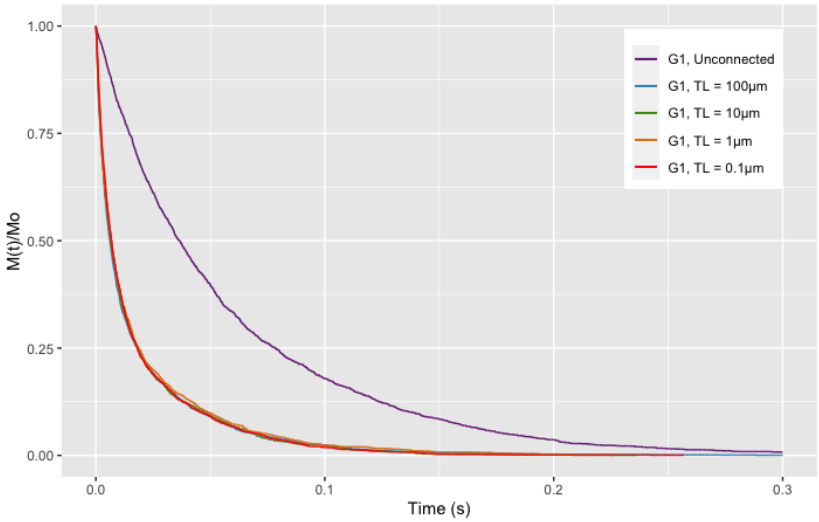
To study the influence of pore throat length on pore coupling, four different throat lengths are examined: $100\mu m, 10\mu m, 1\mu m$ and $0.1\mu m$. Surface relaxivity and throat width are kept constant at $40\mu m/s$ and $2\mu m$, respectively. The magnetisation decay curves and their corresponding characteristic relaxation times obtained for each scenario are compared to each other and to the unconnected geometry. Figure 5.12 shows the results for Geometry 1 and Figure 5.13 for Geometry 2, while in Figure 5.14 a comparison between geometries is illustrated.

As in the section for changing throat widths, the differences in the magnetisation decay curves for each scenario are very subtle so the characteristic relaxation times will be the preferred tool for the visualisation and interpretation of the results.

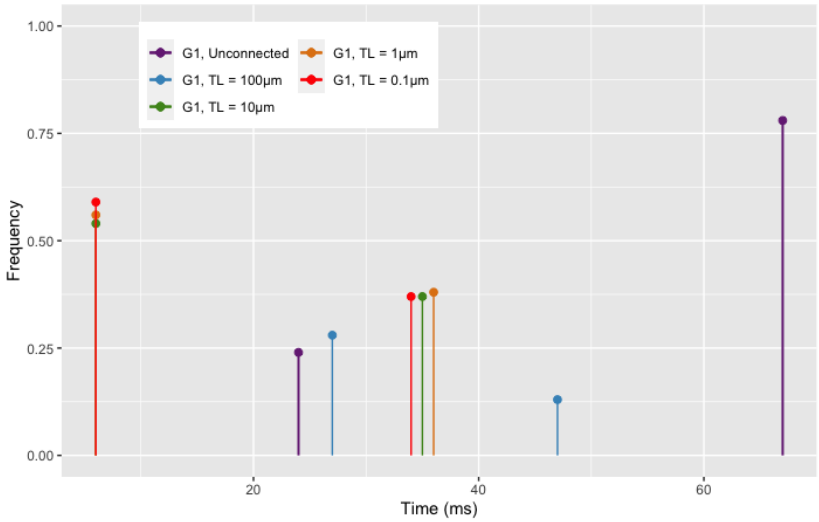
For both geometries, even large throat lengths lead to pore-coupled systems. However, for Geometry 1 (see Figure 5.15b), the degree of pore coupling was not very sensitive to changes in the throat length within the range of $10\mu m - 0.1\mu m$: both the relaxation times and the frequencies were similar for these three scenarios (in green, yellow and red). The frequencies of the fast relaxation peaks were all above 0.50, at least double the value of the fast relaxation peak frequency for the unconnected scenario. This suggests magnetisation exchange from the larger pore towards the smaller pore. This relaxation behaviour was largely different in Geometry 2, where changes in throat lengths always lead to changes in the degree of pore coupling (see Figure 5.16b). Here, good connected systems ($TL = 1\mu m$) will even only present one large (slow) characteristic relaxation peak instead of two. Furthermore, in the range of $10\mu m - 0.1\mu m$, the fast peaks always decreased in frequency with decreasing throat length.

Although in different ways for each geometry, changes in throat lengths will affect the characterisation of both large and small pores, both in terms of their size and their frequency in the network.

Geometry 1



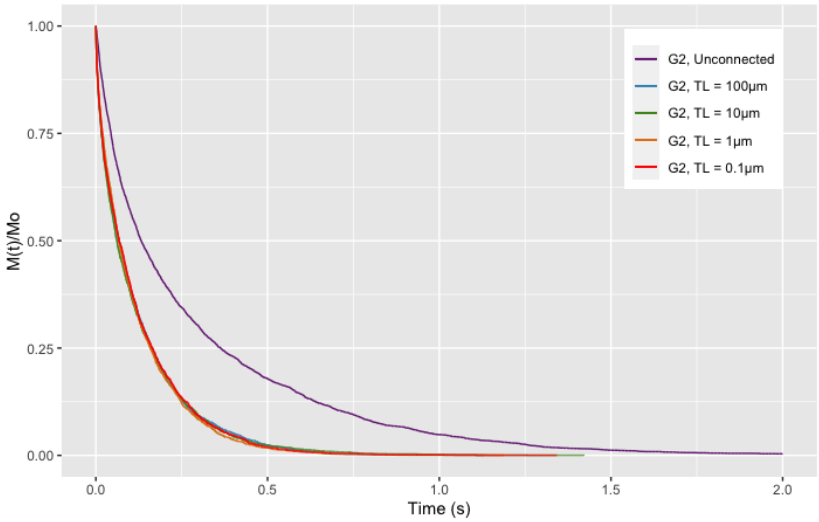
(a) Magnetisation decay curves.



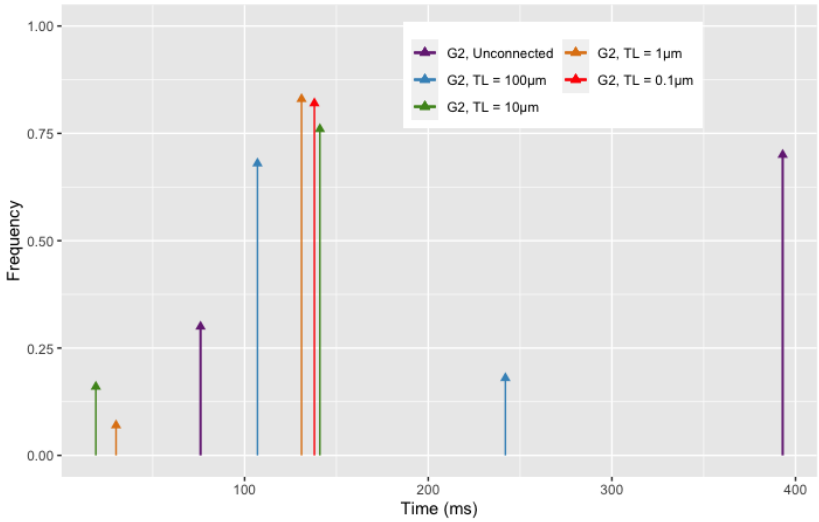
(b) Characteristic relaxation times.

Figure 5.15: Magnetisation decay curves (panel (a)) and characteristic relaxation times (panel (b)) for networks of pores of two similar diameters connected through pore throats of fixed width $TW = 2\mu m$ and varying throat lengths at a fixed surface relaxivity $\rho = 40\mu m$. $G1$ stands for Geometry 1 and TL for throat length.

Geometry 2



(a) Magnetisation decay curves.



(b) Characteristic relaxation times.

Figure 5.16: Magnetisation decay curves (panel (a)) and characteristic relaxation times (panel (b)) for networks of pores of two dissimilar diameters connected through pore throats of fixed widths $TW = 2\mu m$ and varying throat lengths at a fixed surface relaxivity $\rho = 40\mu m$. $G2$ stands for Geometry 2 and TL for throat length.

Comparison between geometries

To better illustrate the differences in relaxation behavior between the connected Geometries 1 and 2 with different throat lengths, Figure 5.17 compares the characteristic relaxation times for Geometry 1, throat lengths $1\mu m$ (yellow dots) and $10\mu m$ (green dots), and Geometry 2, throat lengths $1\mu m$ (yellow triangles) and $10\mu m$ (green triangles).

Here, the stronger sensitivity of Geometry 2 to pore coupling driven by changes in throat lengths can again be recognised: the fast and the slow peaks show greater differences in position and frequency for Geometry 2 than for Geometry 1.

Also, as already mentioned above, the inverted direction of magnetisation exchange between the geometries is visualised: in Geometry 2 magnetisation diffuses from the smaller pores to the bigger pore, showing a decrease in frequency for the fast relaxation peaks, while in Geometry 1, the magnetisation exchange is from the bigger pore towards the smaller ones.

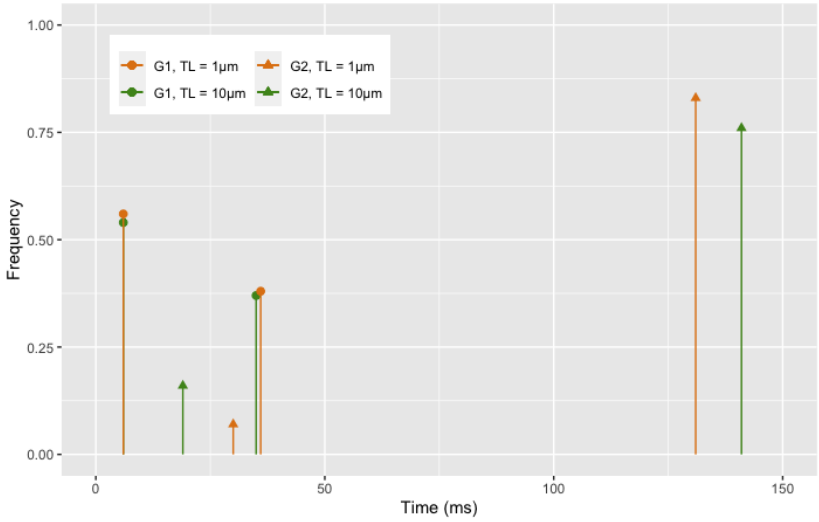


Figure 5.17: Comparison between the characteristic relaxation times for systems of two pores of similar (G1) and dissimilar (G2) diameters connected through pore throats of fixed throat widths $TW = 2\mu m$ and throat lengths of $TW = 1$ and $10\mu m$ at a fixed surface relaxivity $\rho = 40\mu m$.

Further discussions and conclusions drawn from these results along with comparisons to results from previous studies can be found in Chapter 6.

Chapter 6

Discussion and Conclusions

The aim of this master's thesis was to explore the factors controlling pore coupling in fully saturated environments through numerical simulations. To do so, a tried-and-used NMR Random Walk code was deployed on manually designed simple geometries meant to represent two types of dual-porosity environments: networks with pores of different but similar sizes and networks with pores of largely dissimilar sizes. Within these two geometries, different factors identified as controlling pore coupling in available literature, were manipulated to test their influence on the pore coupling strength in different scenarios. These factors were: surface relaxivity, signaling a change in the surface geochemistry of the porous media, and network connectivity, controlled by the length and the width of the throats connecting the pores. While these factors have been tried in previous studies, never had they been explored together using the same methodology nor had such precise scenarios been directly compared to each other within one framework.

When it comes to the methodology used for this master's thesis, it is important to consider certain limitations, particularly on three fronts: the geometries tested, the chosen simulation technique and the T_2-inversion algorithm applied. These will be addressed in the following paragraphs.

The geometries studied were designed to be simple models of pore networks at a very small scale. While designing them, keeping the running-time of the simulations and the computer-power required low was considered of high importance. The simplicity of the models helped with the interpretation of the NMR signals and allowed for precise manual alterations, while providing a baseline understanding of the factors controlling pore coupling. Nevertheless, real geological material have a much more complex network of pores in terms of pore sizes (a wider range), connectivity between the pores (pores being connected to multiple pores at once), and pore and throat shape (spheres and cylinders being a basic simplification). The geometries were not designed to represent any particular sample or rock morphology and are meant to only provide general directions and visualisations of how NMR signals may look on pore coupled material.

The chosen methodology to study pore coupling effects were numerical simulations. While analytical solutions for single pore geometries were used to verify the simulation and the inversion algorithms, any experimental verification of the simulated results was considered outside of the scope of this project. Numerical simulations are a great tool to model the behaviour of complex systems and allow for great customisation, but are never a perfect representation of the natural

world. Further experimental work would always be required to solidify the conclusions drawn from simulated experiments.

Since the programming of a complex NMR relaxation algorithm from scratch was beyond the aims of this study, an already-existing and publicly available simulation code needed to be chosen. The Random Walk is a commonly used approach to model NMR relaxation and has been used in the past to study pore coupling effects. Still, the available NMR code had a fixed step-size for the diffusion of the "walkers" instead of a variable one, typical of the "First-Arrival" Random Walks. Fixed step-sizes mean that, in connected systems, a very small value must be chosen and kept all throughout the simulation time, which is very computationally expensive. Variable step-sizes save computer-power that can be utilised, for example, for the consideration of more complex systems. Newer approaches, such as Finite Elements or Finite Volumes, might also reduce the computational load required to model difficult scenarios and simplify the process of making changes in the geometries studied. They should be considered for projects with wider scopes.

Finally, the inversion of the magnetisation decay data into relaxation times is a very important step for the interpretation of NMR data. For this study, a straightforward inversion algorithm, linked to the NMR simulation itself and not allowing for much customisation, was used. Considering the scope of the simulation and the illustrative goal of the characteristic relaxation times obtained, this code performed appropriately and allowed for the comparison of multiple scenarios. Especially considering that no experimental data was available for the calibration of a more complex inversion process. Nevertheless, it is important to keep into consideration the existence of more complex inversion algorithms providing the necessary customisation and calibration capacities needed to gain T_2 values for purposes beyond illustrating NMR behaviour in simplified geometries.

Nonetheless, and despite the acknowledged limitations of the methodology, interesting and consistent results were obtained through the simulations, which allowed for the following conclusions to be drawn. These conclusions are largely in agreement with findings from previous studies as will be discussed in this section. The comparisons of the results gained through the Random Walk simulations in this master's thesis with the results available in literature will mostly be qualitative. Due to the large range of methods and geometries that have been used in the past to study pore coupling effects, which are often very different to the approaches taken in this work, it is difficult to perform sensible quantitatively comparisons .

Pore coupling in materials with two very different pore sizes, as in Geometry 2, has been shown in available literature to merge two T_2-distributions peaks into one peak (e.g., Toumelin *et al.*, 2003; Anand & Hirasaki, 2007 or Grunewald & Knight, 2009, among others). The results obtained for pore coupled scenarios of Geometry 2, with ever-decreasing frequencies for the fast relaxation peaks, show agreement with this pattern. This is not the case for Geometry 1, with pores of two very similar sizes, where the two characteristic relaxation times feature was mostly maintained at all degrees of pore coupling, even if heavily distorted.

Pore coupling is often considered as a problem affecting materials with two pore environments of largely different sizes, but the simulations ran as part of this master's thesis suggest that pore coupling might also show significant effects in dual-porosity materials with a less drastic size difference, even if the effects are not quite the same (e.g., no loss of the two-peak characteristic feature) or not quite as intense.

In relation to the surface relaxivity as a controlling factor for pore coupling, it was found that

systems with smaller ρ values are clearly more strongly affected by pore coupling than systems with higher ρ values. This in line with findings from other experimental (e.g., Anand & Hirasaki, 2007 or Fleury & Soualem, 2009) and numerical (e.g., Mitchell *et al.*, 2019) studies. Nevertheless, systems presenting higher values (even as high as $100\mu m/s$ or $250\mu m/s$, and definitively for ρs about $40\mu m/s$) are also affected by pore coupling, regardless of the pore-size-ratio of the geometry.

The results of the Random Walk simulations in this master's thesis, suggest that surface relaxivity is a stronger control factor in dual porosity systems with pores similar in size than in systems with two largely different pore diameters. Also, looking at the characteristic relaxation times, it would seem that the larger pores (i.e., the slow relaxation peaks) are more strongly affected by surface relaxivity-driven pore coupling than the smaller pores, which could lead to the erroneous characterisation of the larger pore sizes in the network. Nevertheless, the frequencies associated to each pore environment were heavily affected for both pore sizes. This was true for both geometries observed.

Looking at network connectivity as a control factor for pore coupling, it must be noted that characteristic relaxation times obtained from the inversion step are more valuable for the visualisation of the changes in the relaxation behavior than the magnetisation decay curves. This is not surprising, given by how T_2-distributions are the main visualisation tool used in literature dealing with pore coupling effects in NMR measurements. Other studies, such as Mohnke & Klitzsch (2010) and Ghomeshi *et al.* (2018), also, like in this master's thesis, visualise their results in the form of characteristic relaxation times instead of gaussian-looking T_2-distributions.

Regarding the connectivity between the pores, it was possible to test extreme throat widths that were "too thin", for any change in the system to be appreciated in comparison to unconnected scenarios. This was the case for a throat width of $0.1\mu m$. Mohnke & Klitzsch (2010) also used this value for the width of the throats connecting pores of $20\mu m$ and $10\mu m$ in diameter and was successful in observing magnetisation exchange through a finite elements simulation. The fact that such behavior could not be observed using the Random Walk code for this master's thesis, points at the limitations posed by this simulation method. For scenarios with changing throat lengths, no such a thing as a "too long" throat was observed: even very large throats ($100\mu m$) affected the relaxation behavior of the system. This does not necessarily means, however, that magnetisation exchange was truly taking place between the pores: "walkers" could be able to enter the throats and relax within them before reaching the other pore, leading to distorted relaxation signals. It would be necessary to deploy other simulation approaches to test whether this behaviour is inherent of the simulation mechanism used for this study or a characteristic of pore-coupled or pore-throat-coupled systems.

In the case of throat width as a control factor of pore coupling, the results of the Random Walk simulation suggested that throat width-driven pore coupling affected the characterisation of the smaller pores more strongly. Furthermore, in systems with two dissimilar pore sizes, a threshold-like behaviour could be seen for the larger pore: the changes in the relaxation behaviour were very mild for a wider range of pore widths, but the highly connected system ($TW = 2\mu m$) showed a very large distortion for the slower relaxation signal. This might be connected to the pore-surface-area-to-throat-cross-sectional-area ratio issue discussed for Geometry 2. Nevertheless, this behavior was not observed by Ghomeshi *et al.* (2018) on their analytical, eigenvalue, exploration of characteristic relaxation times in two-pores system with different throat diameters. However, the throat diameters tested in that study were significantly larger than the ones tested in this master's thesis, with their smaller diameter being $20\mu m$ (versus

the $4\mu m$ corresponding to the largest diameter in this study).

Regarding throat length as a factor, in agreement with the results from the finite elements simulations by Mohnke & Klitzsch (2010), the characterisation of both the smaller and the larger pore environments was heavily affected for all of the lengths tried in both geometries. Geometry 2, however, showed a much stronger sensitivity to throat length-driven pore coupling. The adequate characterisation of the lengths of the throats is essential to estimate the strength of the pore coupling effects in coupling-prone systems.

The analysis carried out in this master's thesis provide further evidence to the importance of considering pore coupling effects when taking NMR measurements in certain type of materials and under certain conditions. Multiple scenarios for two different geometries were tested to provide new insight on the circumstances under which pore coupling effects might be of greater relevance and should not be ignored. It is known from previous studies, that pore coupling predominantly affects systems with two very different pore sizes and slow surface relaxivity values. Data obtained in this study advises for the consideration of pore coupling effects also in systems with two similar pore sizes and already at surface relaxivities of $40\mu m/s$.

Furthermore, the results from these NMR Random Walk experiments, suggest that pore coupling does not necessarily affect both pore environments equally and that identifying the factor driving pore coupling might be helpful to recognize what effects can be expected in the data and which pore environment will be more likely to be wrongly characterised.

Surface relaxivity is an empirical factor that must be determined for any given sample or roughly estimated from the type of material studied. In pore-coupled systems, the correct estimation of surface relaxavity is of great importance, as the relaxation behaviour under pore coupling changes heavily under different surface relaxivity values, even within the rather small range of $10\mu m/s$ to $40\mu m/s$.

Pore coupling effects can be visualised and quantified already in very simple geometries and small scales, like in this master's thesis, but our understanding of the phenomena needs to be develop in the direction of more realistic geological geometries through the study of real samples. It remains an open question, very much worthy of exploration but beyond the scope of this master's thesis, to consider the relaxation behaviour and the effects of pore coupling in systems presenting a wider range of pore sizes (e.g., $10\mu m$, $20\mu m$ and also $100\mu m$ in the same geometry).

NMR is a very promising tool with the potential to help in the investigation and the monitoring of key near-surface processes. This makes the study of the pore coupling effects, as a mean to widen the spectrum of possible NMR applications, a vastly significant endeavor. Open questions, knowledge gaps and further research opportunities were discovered in the development of this master's thesis and will be highlighted in the next chapter. The focus of this outlook will be the unsaturated vadose zone and karst aquifers.

Chapter 7

Outlook

This study has focused on the numerical simulation of the NMR response of well connected porous networks in order to better understand the factors controlling pore coupling. NMR is a promising emerging technology in the field of hydrogeophysics and in the exploration of water resources in the subsurface. Pore coupling effects difficult the interpretation of NMR data in certain scenarios but can also provide information about the connectivity of a system.

After a careful literature review on pore coupling effects and a numerical exploration of the factors controlling it using simple, fully saturated, geometries, I would like to draw attention to two main areas where the further study of pore coupling might prove beneficial: the unsaturated vadose zone, and karst aquifers. I choose to highlight these two topics because of their importance, looking at the terrestrial system as a whole, because of the great potential NMR studies have in these environments, and because of the multiple open questions and opportunities for further NMR pore coupling research I can identify.

Pore coupling in the vadose zone

The effects of pore coupling on the relaxation behaviour of complex geological media has been observed and studied for fully saturated environments (i.e., for networks of pores that are completely filled with water). However, there are still multiple open questions related to the NMR relaxation and pore coupling effects in systems that are only partially saturated, such as the vadose zone.

Considering the important ecosystem functions of the vadose zone and its retention and buffering roles, there are multiple environmental questions that could be approached using NMR technology in general, and pore coupling effects in particular. Understanding how pore coupling affects the NMR signal in unsaturated conditions will improve our interpretation of NMR data taken from bi-porous, pore coupling-prone systems, such as carbonate rocks, and lead to more accurate predictions, for example, of water contents.

Particularly in arid regions and areas where climate change has increased the severity and frequency of draughts, an accurate quantification of water resources available to plants through the vadose zone is vital to understand ecosystem functioning and resilience capabilities. While this field of research has been widely explored for soils (Schmidt & Rempe, 2020), only recently has the quantification of moisture trapped in unsaturated fractured bedrock, the so-called “rock

moisture", been attempted (e.g., Rempe & Dietrich, 2018; Schmidt & Rempe, 2020; Zhang & Zhang, 2021). This moisture trapped in weathered rock, located below the soil column but above the groundwater table, has been found to be a stable source of water to vegetation during dry periods that needs to be quantified and considered in hydrologic models (Rempe & Dietrich, 2018). An accurate measurement of water content, necessary for systems where only small quantities are expected, is only possible using NMR if the signals are well understood and possible distortions related to pore coupling effects are accounted for.

When considering a contaminated site, pore coupling strength could also be used to determine the degree of connectivity between the pores (Carneiro *et al.*, 2014; Zhi, 2021), which could, in turn, provide information regarding the flow speed of contaminants and the extent of their spread within the vadose zone. This knowledge is helpful to plan and execute successful remediation strategies, making the quantification of pore coupling strength in unsaturated environments of relevance for practical environmental applications.

While the Random Walk method is not adequate for the simulation of unsaturated media and the study of the effects controlling pore coupling in the vadose zone are beyond the scope of this investigation, this section will focus on important considerations needed for the undertaking of such as a study.

Due to the inherent characteristics of the unsaturated vadose zone, the numerical modelling of NMR measurements increase in complexity. This, of course, makes it more difficult to study the effects of pore coupling. From the available literature, three new parameters that control NMR relaxation must be considered in unsaturated conditions: the level of saturation, the pore-filling mechanism and the shape of the pores in the network (Costabel & Yaramanci, 2011).

The level of saturation must be chosen depending on the scenario or the process intended to be modelled. The choice of pore-filling, or pore-draining mechanism is between two main models currently considered in literature: one where the pore walls are coated with a thinning layer of water as they drain, and another where that is not the case (e.g., Costabel & Müller-Petke, 2014; Mohnke & Klitzsch, 2010). These two models are depicted in Figure 7.1.

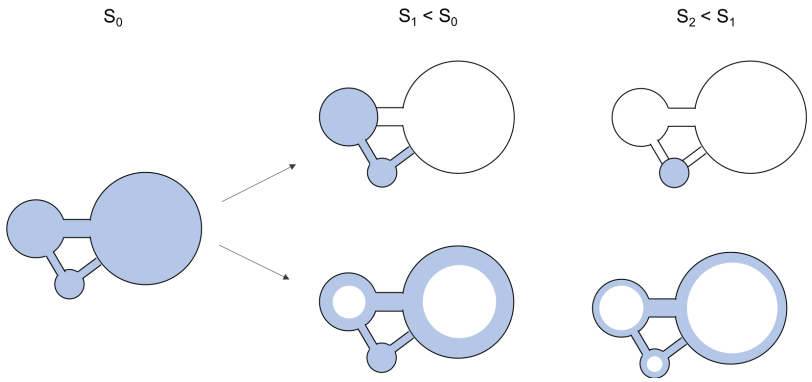


Figure 7.1: Two possible pore-draining mechanism. Modified from Costabel & Müller-Petke (2014).

The shape of the pores becomes important in unsaturated environments because of the capability of sharp angles to withhold small quantities of water during and after the draining process. This residual water will emit NMR relaxation signals and affect the magnetisation decay in the system. The importance of this parameter is discussed at length e.g., in Costabel & Yaramanci

(2011) and in Mohnke *et al.* (2015). The draining process for a triangular pore under this model can be visualised in Figure 7.2.

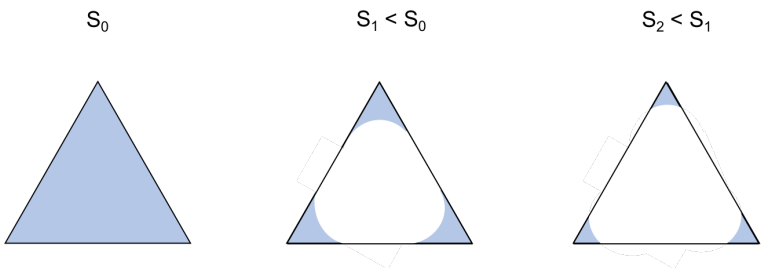


Figure 7.2: Model of the draining process for a single triangular pore. Modified from Costabel & Hiller (2021).

More specifically, for the study of pore coupling effects in unsaturated media, the connection between the degree of saturation, the diffusion regime and the degree of pore coupling needs to be explored. Costabel (2011) suggests, through analytical models, that with lowering degree of fluid saturation, systems will transition from the slow into the fast diffusion regime. Whether this faster relaxation behavior will translate to a reduction on the pore coupling effects remains unexplored.

Pore coupling in karst aquifers

Karst aquifers have complex heterogeneous pore systems formed through the chemical weathering of soluble rocks. The most important and widespread type of karstifiable rocks are carbonate rocks (limestones and dolomites). Carbonate karst aquifers result from the chemical dissolution along the largest fractures in carbonate rocks. This, after coming in contact with infiltrating water rich in CO_2 from the atmosphere or the soil (Goldscheider *et al.*, 2020). This process produces highly connected pore networks with a wide range of pore sizes including nanopores, micropores, meso and macropores as well as fractures, fissures and conduits.

Goldscheider *et al.* (2020) estimates that carbonate rocks can be found in 15.2% of the world's land surface, in mountains, hills and plains, and in all continents and climatic zones, while Stevanovic (2019) estimates that 9.2% of the global population consumes water extracted from karst aquifers. The high connectivity found in karst systems makes them both highly prolific in terms of aquifer capability, but also highly vulnerable to contamination and sensitive to hydrometeorological changes (Hartmann *et al.*, 2014; Goldscheider *et al.*, 2020). For these reasons, improved methods to characterise, monitor and manage these systems are very well needed.

Due to their inter- and intragranular porosity and their low- to mid surface relaxivity values, carbonate systems are highly prone to pore coupling (Ramakrishnan *et al.*, 1999). They are also often outside the fast diffusion regime, generating the need to consider bulk relaxation processes as well as surface relaxation. For these reasons, characterising carbonate karst systems using NMR is not a straightforward endeavour and requires careful consideration.

One of the main open questions that can be highlighted to better understand pore coupling in karst systems, is the relaxation behaviour of mesopores, or pores in a wider range of sizes, under pore coupling. Since multi-porosity is a common attribute in karst systems, it might be

advisable to explore further options to achieve a more realistic model. A popular approach to simulate NMR relaxation using actual, geological geometries involves the creation of digital porous media from micro-CT images (in the context of diffusional coupling, see e.g., Chi & Heidari, 2015). Nevertheless, for carbonate rocks, it is not easy to define the right threshold to differentiate between pore spaces (voids) and rock matrix (solids) from micro-CT images and it becomes necessary to include a further image processing step to reduce noise and improve the identification of the pore spaces (Lucas-Oliveira *et al.*, 2018).

Improving our ability to model magnetization decay and pore coupling in realistic karstic geometries will improve our understanding of the NMR signal and our ability to use NMR technology in the field and the laboratory to explore geometrical structures, water availability and flow dynamics in carbonate aquifers. Karst systems play important roles within both the carbon and the water cycle and their characterization is, therefore, of great importance to understand big scales systems, such as the terrestrial water cycle.

Closing remarks

To summarise and conclude, NMR is an emerging technology in the field of hydrogeophysics with a great potential for practical applications in near-surface environments. Much could be gained by improving and spreading the use of NMR, with its non-invasive, direct sensitivity to water molecules. To do so, a better understanding of NMR signals in complex scenarios is required. This includes, though it is not limited to, understanding pore coupling effects in fully and partially saturated environments. This master's thesis attempted to provide further insight on the mechanisms driving pore coupling and the effects they produce on the NMR signals by numerically modelling the relaxation behavior in different scenarios for simple geometries. The simulation method used, Talabi (2008)'s Random Walk, proved to be sufficient for the scope of this study, albeit with technical limitations on the complexity of the models that could be easily overcome using other simulation techniques, such as finite elements or finite volumes.

Ignoring possible pore coupling effects can lead to substantial misinterpretation of distorted NMR data both related to the pore sizes estimated for a material and the abundance of pores in each pore environment. Multiple methodologies, both numerical and experimental, are available to practitioners and researchers to explore pore coupling and consider their possible effects when studying dual porosity materials at low surface relaxivities. Moreover, multiple knowledge gaps are still open for further investigation, as NMR continues to gain popularity for hydrological applications in near-surface environments.

Bibliography

Albert, James, Destouni, Georgia, Duke-Sylvester, Scott, Magurran, Anne, Oberdorff, Thierry, Reis, Roberto, Winemiller, Kirk, & Ripple, William. 2020. Scientists' warning to humanity on the freshwater biodiversity crisis. *Ambio*, **50**(02).

Anand, Vivek, & Hirasaki, G. 2007. Diffusional Coupling Between Micro And Macroporosity For NMR Relaxation In Sandstones And Grainstones1. *Petrophysics*, **48**(Aug.), 289–307.

Anand, Vivek, Hirasaki, G., & Fleury, Marc. 2008. NMR Diffusional Coupling: Effects of Temperature And Clay Distribution1. *Petrophysics*, **49**(Aug.).

Binley, Andrew, Hubbard, Susan S., Huisman, Johan A., Revil, André, Robinson, David A., Singha, Kamini, & Slater, Lee D. 2015. The emergence of hydrogeophysics for improved understanding of subsurface processes over multiple scales. *Water Resources Research*, **51**(6), 3837–3866.

Brownstein, K. R., & Tarr, C. E. 1979. Importance of classical diffusion in NMR studies of water in biological cells. *Physical Review A*, **19**(6), 2446–2453.

Bryar, Traci R, Daughney, Christopher J, & Knight, Rosemary J. 2000. Paramagnetic Effects of Iron(III) Species on Nuclear Magnetic Relaxation of Fluid Protons in Porous Media. *Journal of Magnetic Resonance*, **142**(1), 74–85.

Carneiro, Giovanna, Souza, Andre, Boyd, Austin, Schwartz, Lawrence, Coutinho, Bernardo, Trevizan, Willian, Rios, Edmilson, & Machado, Vinicius. 2013. *Image Analysis and NMR modeling of Sedimentary Rocks*.

Carneiro, Giovanna, Souza, Andre, Boyd, Austin, Schwartz, Lawrence, Song, Yi-Qiao, Trevizan, Willian, Coutinho, Bernardo, Rios, Edmilson, & Machado, Vinicius. 2014 (05). Evaluating pore space connectivity by NMR diffusive coupling.

Chi, Lu, & Heidari, Zoya. 2015. Diffusional coupling between microfractures and pore structure and its impact on nuclear magnetic resonance measurements in multiple-porosity systems. *Geophysics*, **80**(1), D31–D42.

Costabel, S., & Müller-Petke, M. 2014. *NMR-Relaxation bei Teilsättigung unter Berücksichtigung von fast- und slow diffusion.*

Costabel, Stephan. 2011 (Dec.). *Nuclear magnetic resonance on laboratory and field scale for estimating hydraulic parameters in the vadose zone*. Ph.D. thesis.

Costabel, Stephan, & Hiller, Thomas. 2021. Soil hydraulic interpretation of nuclear magnetic resonance measurements based on circular and triangular capillary models. *Vadose Zone Journal*, **20**(Jan.).

Costabel, Stephan, & Yaramanci, Ugur. 2011. Relative hydraulic conductivity and effective saturation from Earth's field nuclear magnetic resonance – a method for assessing the vadose zone. *Near Surface Geophysics*, **9**(2), 155–167.

Ding, Shun, Jia, Hailiang, Zi, Fan, Dong, Yuanhong, & Yao, Yuan. 2020. Frost Damage in Tight Sandstone: Experimental Evaluation and Interpretation of Damage Mechanisms. *Materials*, **13**(20), 4617.

Dong, Hu, & Blunt, Martin J. 2009. Pore-network extraction from micro-computerized-tomography images. *Phys. Rev. E*, **80**(Sep), 036307.

Fleury, Marc, & Soualem, Jawed. 2009. Quantitative analysis of diffusional pore coupling from T2-store-T2 NMR experiments. *Journal of Colloid and Interface Science*, **336**(1), 250–259.

Ghomeshi, Shahin, Kryuchkov, Sergey, & Kantzas, Apostolos. 2018. An investigation into the effects of pore connectivity on T2 NMR relaxation. *Journal of Magnetic Resonance*, **289**(Apr.), 79–91.

Goldscheider, Nico, Chen, Zhao, Auler, Augusto, Bakalowicz, Michel, Broda, Stefan, Drew, David, Hartmann, Jens, Jiang, Guanghui, Moosdorf, Nils, Stevanovic, Zoran, & Veni, George. 2020. Global distribution of carbonate rocks and karst water resources. *Hydrogeology Journal*, **28**(04), 1–17.

Grunewald, Elliot, & Knight, Rosemary. 2009. A laboratory study of NMR relaxation times and pore coupling in heterogeneous media. *Geophysics*, **74**(6), E215–E221.

Hartmann, A., Goldscheider, N., Wagener, T., Lange, J., & Weiler, M. 2014. Karst water resources in a changing world: Review of hydrological modeling approaches. *Reviews of Geophysics*, **52**(3), 218–242.

Johnson, David Linton, & Schwartz, Lawrence M. 2014. Analytic theory of two-dimensional NMR in systems with coupled macro- and micropores. *Physical Review E*, **90**(3), 032407.

Kass, M. Andy, Irons, Trevor P., Minsley, Burke J., Pastick, Neal J., Brown, Dana R. N., & Wylie, Bruce K. 2017. In situ nuclear magnetic resonance response of permafrost and active layer soil in boreal and tundra ecosystems. *The Cryosphere*, **11**(6), 2943–2955.

Lucas-Oliveira, Everton, Araujo-Ferreira, A., Trevizan, Willian, Fortulan, Carlos, & Bonagamba, Tito. 2018. Computational approach to integrate 3D X-ray microtomography and NMR data. *Journal of Magnetic Resonance*, **292**(05).

Mason, Geoffrey, & Morrow, Norman R. 1991. Capillary behavior of a perfectly wetting liquid in irregular triangular tubes. *Journal of Colloid and Interface Science*, **141**(1), 262–274.

McCall, K. R., Johnson, D. L., & Guyer, R. A. 1991. Magnetization evolution in connected pore systems. *Phys. Rev. B*, **44**(Oct), 7344–7355.

Mitchell, Jonathan, Souza, Andre, Fordham, Edmund, & Boyd, Austin. 2019. A finite element approach to forward modeling of nuclear magnetic resonance measurements in coupled pore systems. *The Journal of Chemical Physics*, **150**(15), 154708.

Mohnke, O., & Klitzsch, N. 2010. Microscale simulations of NMR relaxation in porous media considering internal field gradients. *Vadose Zone Journal*, **9**(4), 846–857.

Mohnke, O., Stiebler, M., & Klitzsch, N. 2014. Joint numerical microscale simulations of multiphase flow and NMR relaxation behavior in porous media using Lattice Boltzmann methods. *Water Resources Research*, **50**(9), 7378–7393.

Mohnke, O., Jorand, R., Nordlund, C., & Klitzsch, N. 2015. Understanding NMR relaxometry of partially water-saturated rocks. *Hydrology and Earth System Sciences*, **19**(6), 2763–2773.

Monteilhet, L, Korb, jean-pierre, Mitchell, Jonathan, & McDonald, P. 2007. Observation of exchange of micropore water in cement pastes by two-dimensional T 2 T 2 nuclear magnetic resonance relaxometry. *Physical review. E, Statistical, nonlinear, and soft matter physics*, **74**(Jan.), 061404.

Mukherjee, Sourav, Mishra, Ashok, & Trenberth, Kevin E. 2018. Climate change and drought: a perspective on drought indices. *Current Climate Change Reports*, **4**(2), 145–163.

Müller-Petke, Mike, Dlugosch, Raphael, Lehmann-Horn, Jochen, & Ronczka, Mathias. 2015. Nuclear magnetic resonance average pore-size estimations outside the fast-diffusion regime. *Geophysics*, **80**(3), D195–D206.

Parsekian, Andrew D., Grosse, Guido, Walbrecker, Jan O., Müller-Petke, Mike, Keating, Kristina, Liu, Lin, Jones, Benjamin M., & Knight, Rosemary. 2013. Detecting unfrozen sediments below thermokarst lakes with surface nuclear magnetic resonance. *Geophysical Research Letters*, **40**(3), 535–540.

Ramakrishnan, T. S., Schwartz, L. M., Fordham, E. J., Kenyon, W. E., & Wilkinson, D. J. 1999. *Forward models for nuclear magnetic resonance in carbonate rocks: The Log Analyst, 40, 260–270*.

Rempe, Daniella M., & Dietrich, William E. 2018. Direct observations of rock moisture, a hidden component of the hydrologic cycle. *Proceedings of the National Academy of Sciences*, **115**(11), 2664–2669.

Roberts, S.P., Mcdonald, P.J., & Pritchard, T. 1995. A Bulk and Spatially Resolved NMR Relaxation Study of Sandstone Rock Plugs. *Journal of Magnetic Resonance, Series A*, **116**(2), 189–195.

Schmidt, Logan, & Rempe, Daniella. 2020. Quantifying Dynamic Water Storage in Unsaturated Bedrock with Borehole Nuclear Magnetic Resonance. *Geophysical Research Letters*, **47**(22), e2020GL089600.

Sivakumar, B. 2011. Global climate change and its impacts on water resources planning and management: Assessment and challenges. *Stochastic Environmental Research and Risk Assessment*, **25**(05), 583–600.

Stevanovic, Zoran. 2019. Karst waters in potable water supply: a global scale overview. *Environmental Earth Sciences*, **78**(11).

Talabi, Olumide Adegbenga. 2008. *Pore-scale simulation of NMR response in porous media.* Ph.D. thesis.

Tian, H. H., & Wei, C. F. 2020. Characterization and Quantification of Pore Water in Clays During Drying Process With Low-Field NMR. *Water Resources Research*, **56**(10).

Torland, Anders. 2018. *A Pore Network Investigation of Factors Influencing the Residual Oil Saturation.* M.Phil. thesis.

Toumelin, E., Torres-Verdín, C., Chen3, S., & Fischer, D.M. 2003. Reconciling NMR Measurements and Numerical Simulations: Assessment of Temperature and Diffusive Coupling Effects on Two-Phase Carbonate Samples. *Petrophysics - The SPWLA Journal of Formation Evaluation and Reservoir Description*, **44**(02).

Toumelin, Emmanuel, Torres-Verdín, Carlos, Sun, Boqin, & Dunn, Keh-Jim. 2007. Random-walk technique for simulating NMR measurements and 2D NMR maps of porous media with relaxing and permeable boundaries. *Journal of Magnetic Resonance*, **188**(1), 83–96.

Washburn, K. E., & Callaghan, P. T. 2006. Tracking pore to pore exchange using relaxation exchange spectroscopy. *Physical Review Letters*, **97**(17), 175502.

Zhang, Fan, & Zhang, Chi. 2021. Probing water partitioning in unsaturated weathered rock using nuclear magnetic resonance. *Geophysics*, **86**(5), WB131–WB147.

Zhi, TIAN. 2021. NMR diffusional coupling of multiple-scale porous rock and its detection. *Chinese Journal of Geophysics*, **64**(3), 1119–1130.

Annexes

Appendix A

Geometry network files

In this Appendix, exemplary input geometry network files will be presented exactly as they were used for the simulation of the different scenarios. Each line of code is numerated on the left side to facilitate comprehension. A detailed explaination of each column of the files can be found in Section 3.2, as well as in Talabi (2008) or Torland (2018).

A.1 Single pores

In this section, the network files for the three single pores geometries are included. They were used to simulate the magnetisation decay in simple geometries for which analytical solutions are available (see Chapter 4).

A.1.1 Single pore of diameter 10μm

link1.dat

```
0
```

link2.dat

```
0
```

node1.dat

```
1     150.000e-06       150.000e-06       150.000e-06
1      50.000e-06        50.000e-06        50.000e-06   0  0  0
```

node2.dat

```
1          5.235988e-16          5.0000e-06           0.07957          0
```

A.1.2 Single pore of diameter 20μm

link1.dat

```
0
```

link2.dat

```
0
```

node1.dat

```
1      150.000e-06       150.000e-06          150.000e-06
1       50.000e-06        50.000e-06           50.000e-06     0      0     0
```

node2.dat

```
1          4.18879e-15          10.0000e-06           7.9580e-02           0
```

A.1.3 Single pore of diameter 100μm

link1.dat

```
0
```

link2.dat

```
0
```

node1.dat

```
1      150.000e-06       150.000e-06          150.000e-06
1       50.000e-06        50.000e-06           50.000e-06     0      0     0
```

node2.dat

```
1           5.235988e-13           50.0000e-06           7.9580e-02           0
```

A.2 Geometry 1

In this section, exemplary files are presented for Geometry 1, with two similar pore sizes. For the connected geometry, a scenario with a throat width of $2\mu m$ and a throat length of $1\mu m$ is depicted. These values were adjusted accordingly for scenarios with different throat widths or lengths.

A.2.1 Unconnected geometry

link1.dat

```
0
```

link2.dat

```
0
```

node1.dat

```
7     60.000e-06       60.000e-06     60.000e-06

1     31.000e-06       31.000e-06     31.000e-06        0      0     0
2     15.000e-06       31.000e-06     31.000e-06        0      0     0
3     47.000e-06       31.000e-06     31.000e-06        0      0     0
4     31.000e-06       15.000e-06     31.000e-06        0      0     0
5     31.000e-06       47.000e-06     31.000e-06        0      0     0
6     31.000e-06       31.000e-06     15.000e-06        0      0     0
7     31.000e-06       31.000e-06     47.000e-06        0      0     0
```

node2.dat

```
1     4.18879e-15       10.000e-06     7.9570e-02     0.000
2     5.23599e-16        5.000e-06     7.9570e-02     0.000
3     5.23599e-16        5.000e-06     7.9570e-02     0.000
4     5.23599e-16        5.000e-06     7.9570e-02     0.000
5     5.23599e-16        5.000e-06     7.9570e-02     0.000
6     5.23599e-16        5.000e-06     7.9570e-02     0.000
7     5.23599e-16        5.000e-06     7.9570e-02     0.000
```

A.2.2 Connected geometry

link1.dat

```
1  1    2   1.00000001E-06    0.07957  16E-06
2  1    3   1.00000001E-06    0.07957  16E-06
3  1    4   1.00000001E-06    0.07957  16E-06
4  1    5   1.00000001E-06    0.07957  16E-06
5  1    6   1.00000001E-06    0.07957  16E-06
6  1    7   1.00000001E-06    0.07957  16E-06
```

link2.dat

```
1    1    2   10E-06    5.0E-06    1E-06    3.141593e-18    0
2    1    3   10E-06    5.0E-06    1E-06    3.141593e-18    0
3    1    4   10E-06    5.0E-06    1E-06    3.141593e-18    0
4    1    5   10E-06    5.0E-06    1E-06    3.141593e-18    0
5    1    6   10E-06    5.0E-06    1E-06    3.141593e-18    0
6    1    7   10E-06    5.0E-06    1E-06    3.141593e-18    0
```

node1.dat

```
7    55.00e-06    55.00e-06    55.00e-06

1    31.00e-06    31.00e-06    31.00e-06    6    1 2 3 4 5 6   0  0   1 2
    3 4 5 6

2    15.00e-06    31.00e-06    31.00e-06    1    1    0 0      1
3    47.00e-06    31.00e-06    31.00e-06    1    1    0 0      2
4    31.00e-06    15.00e-06    31.00e-06    1    1    0 0      3
5    31.00e-06    47.00e-06    31.00e-06    1    1    0 0      4
6    31.00e-06    31.00e-06    15.00e-06    1    1    0 0      5
7    31.00e-06    31.00e-06    47.00e-06    1    1    0 0      6
```

node2.dat

```
1    4.18879e-15    10.000e-06    0.07957    0.000
2    5.23599e-16    5.0000e-06    0.07957    0.000
3    5.23599e-16    5.0000e-06    0.07957    0.000
4    5.23599e-16    5.0000e-06    0.07957    0.000
5    5.23599e-16    5.0000e-06    0.07957    0.000
6    5.23599e-16    5.0000e-06    0.07957    0.000
7    5.23599e-16    5.0000e-06    0.07957    0.000
```

A.3 Geometry 2

In this section, exemplary files are presented for Geometry 2, with two dissimilar pore sizes. For the connected geometry, a scenario with a throat width of $2\mu m$ and a throat length of $1\mu m$ is depicted. These values were adjusted accordingly for scenarios with different throat widths or lengths.

A.3.1 Unconnected geometry

link1.dat

```
0
```

link2.dat

```
0
```

node1.dat

```
43   150.0000e-06    150.0000e-06    150.0000e-06

1    80.000e-06    80.000e-06    80.00e-06    0     0     0

2    135.10e-06    80.000e-06    80.00e-06    0     0     0
3    130.91e-06    101.09e-06    80.00e-06    0     0     0
4    118.96e-06    118.96e-06    80.00e-06    0     0     0
5    101.09e-06    130.91e-06    80.00e-06    0     0     0
6    80e-06        135.10e-06    80.00e-06    0     0     0
7    58.91e-06     130.91e-06    80.00e-06    0     0     0
8    41.04e-06     118.96e-06    80.00e-06    0     0     0
9    29.09e-06     101.09e-06    80.00e-06    0     0     0
10    24.9e-06      80e-06       80.00e-06    0     0     0
11    29.09e-06    58.91e-06     80.00e-06    0     0     0
12    41.04e-06    41.03e-06     80.00e-06    0     0     0
13    58.91e-06    29.09e-06     80.00e-06    0     0     0
14    80e-06        24.9e-06     80.00e-06    0     0     0
15    101.09e-06    29.09e-06    80.00e-06    0     0     0
16    118.96e-06    41.04e-06    80.00e-06    0     0     0
17    130.91e-06    58.91e-06    80.00e-06    0     0     0

18    80.00e-06    130.910e-06  101.09e-06    0     0     0
19    80.00e-06     118.96e-06  118.96e-06    0     0     0
20    80.00e-06    101.090e-06  130.91e-06    0     0     0
21    80.00e-06    80e-06        135.1e-06    0     0     0
22    80.00e-06    58.91e-06    130.91e-06    0     0     0
23    80.00e-06    41.04e-06    118.96e-06    0     0     0
24    80.00e-06    29.09e-06    101.09e-06    0     0     0
25    80.00e-06    29.09e-06    58.91e-06     0     0     0
26    80.00e-06    41.04e-06     41.03e-06    0     0     0
27    80.00e-06    58.91e-06     29.09e-06    0     0     0
28    80.00e-06    80e-06        24.9e-06     0     0     0
29    80.00e-06    101.09e-06    29.09e-06    0     0     0
30    80.00e-06    118.96e-06    41.04e-06    0     0     0
```

```
31    80.00e-06    130.91e-06    58.91e-06    0    0    0

32    130.91e-06    80.00e-06    101.09e-06    0    0    0
33    118.96e-06    80.00e-06    118.96e-06    0    0    0
34    101.09e-06    80.00e-06    130.91e-06    0    0    0
35    58.91e-06     80.00e-06    130.91e-06    0    0    0
36    41.04e-06     80.00e-06    118.96e-06    0    0    0
37    29.09e-06     80.00e-06    101.09e-06    0    0    0
38    29.09e-06     80.00e-06    58.91e-06     0    0    0
39    41.04e-06     80.00e-06    41.03e-06     0    0    0
40    58.91e-06     80.00e-06    29.09e-06     0    0    0
41    101.09e-06    80.00e-06    41.04e-06     0    0    0
43    130.91e-06    80.00e-06    58.91e-06     0    0    0
```

node2.dat

```
1     5.235988e-13    50.00e-06    7.9570e-02    0.000

2     5.235988e-16    5.00e-06    7.9570e-02    0.000
3     5.235988e-16    5.00e-06    7.9570e-02    0.000
4     5.235988e-16    5.00e-06    7.9570e-02    0.000
5     5.235988e-16    5.00e-06    7.9570e-02    0.000
6     5.235988e-16    5.00e-06    7.9570e-02    0.000
7     5.235988e-16    5.00e-06    7.9570e-02    0.000
8     5.235988e-16    5.00e-06    7.9570e-02    0.000
9     5.235988e-16    5.00e-06    7.9570e-02    0.000
10    5.235988e-16    5.00e-06    7.9570e-02    0.000
11    5.235988e-16    5.00e-06    7.9570e-02    0.000
12    5.235988e-16    5.00e-06    7.9570e-02    0.000
13    5.235988e-16    5.00e-06    7.9570e-02    0.000
14    5.235988e-16    5.00e-06    7.9570e-02    0.000
15    5.235988e-16    5.00e-06    7.9570e-02    0.000
16    5.235988e-16    5.00e-06    7.9570e-02    0.000
17    5.235988e-16    5.00e-06    7.9570e-02    0.000
18    5.235988e-16    5.00e-06    7.9570e-02    0.000
19    5.235988e-16    5.00e-06    7.9570e-02    0.000
20    5.235988e-16    5.00e-06    7.9570e-02    0.000
21    5.235988e-16    5.00e-06    7.9570e-02    0.000
22    5.235988e-16    5.00e-06    7.9570e-02    0.000
23    5.235988e-16    5.00e-06    7.9570e-02    0.000
24    5.235988e-16    5.00e-06    7.9570e-02    0.000
25    5.235988e-16    5.00e-06    7.9570e-02    0.000
26    5.235988e-16    5.00e-06    7.9570e-02    0.000
27    5.235988e-16    5.00e-06    7.9570e-02    0.000
28    5.235988e-16    5.00e-06    7.9570e-02    0.000
29    5.235988e-16    5.00e-06    7.9570e-02    0.000
30    5.235988e-16    5.00e-06    7.9570e-02    0.000
31    5.235988e-16    5.00e-06    7.9570e-02    0.000
32    5.235988e-16    5.00e-06    7.9570e-02    0.000
33    5.235988e-16    5.00e-06    7.9570e-02    0.000
35    5.235988e-16    5.00e-06    7.9570e-02    0.000
36    5.235988e-16    5.00e-06    7.9570e-02    0.000
37    5.235988e-16    5.00e-06    7.9570e-02    0.000
38    5.235988e-16    5.00e-06    7.9570e-02    0.000
39    5.235988e-16    5.00e-06    7.9570e-02    0.000
40    5.235988e-16    5.00e-06    7.9570e-02    0.000
41    5.235988e-16    5.00e-06    7.9570e-02    0.000
```

```
42      5.235988e-16     5.00e-06     7.9570e-02     0.000
43      5.235988e-16     5.00e-06     7.9570e-02     0.000
```

A.3.2 Connected geometry

link1.dat

```
1     1     2     1.00000000001E-06     0.07957     56E-06
2     1     3     1.00000000001E-06     0.07957     56E-06
3     1     4     1.00000000001E-06     0.07957     56E-06
4     1     5     1.00000000001E-06     0.07957     56E-06
5     1     6     1.00000000001E-06     0.07957     56E-06
6     1     7     1.00000000001E-06     0.07957     56E-06
7     1     8     1.00000000001E-06     0.07957     56E-06
8     1     9     1.00000000001E-06     0.07957     56E-06
9     1     10    1.00000000001E-06     0.07957     56E-06
10    1     11    1.00000000001E-06     0.07957     56E-06
11    1     12    1.00000000001E-06     0.07957     56E-06
12    1     13    1.00000000001E-06     0.07957     56E-06
13    1     14    1.00000000001E-06     0.07957     56E-06
14    1     15    1.00000000001E-06     0.07957     56E-06
14    1     16    1.00000000001E-06     0.07957     56E-06
16    1     17    1.00000000001E-06     0.07957     56E-06
17    1     18    1.00000000001E-06     0.07957     56E-06
18    1     19    1.00000000001E-06     0.07957     56E-06
19    1     20    1.00000000001E-06     0.07957     56E-06
20    1     21    1.00000000001E-06     0.07957     56E-06
21    1     22    1.00000000001E-06     0.07957     56E-06
22    1     23    1.00000000001E-06     0.07957     56E-06
23    1     24    1.00000000001E-06     0.07957     56E-06
24    1     25    1.00000000001E-06     0.07957     56E-06
25    1     26    1.00000000001E-06     0.07957     56E-06
26    1     27    1.00000000001E-06     0.07957     56E-06
27    1     28    1.00000000001E-06     0.07957     56E-06
28    1     29    1.00000000001E-06     0.07957     56E-06
29    1     30    1.00000000001E-06     0.07957     56E-06
30    1     31    1.00000000001E-06     0.07957     56E-06
31    1     32    1.00000000001E-06     0.07957     56E-06
32    1     33    1.00000000001E-06     0.07957     56E-06
33    1     34    1.00000000001E-06     0.07957     56E-06
34    1     35    1.00000000001E-06     0.07957     56E-06
35    1     36    1.00000000001E-06     0.07957     56E-06
36    1     37    1.00000000001E-06     0.07957     56E-06
37    1     38    1.00000000001E-06     0.07957     56E-06
38    1     39    1.00000000001E-06     0.07957     56E-06
39    1     40    1.00000000001E-06     0.07957     56E-06
40    1     41    1.00000000001E-06     0.07957     56E-06
41    1     42    1.00000000001E-06     0.07957     56E-06
42    1     43    1.00000000001E-06     0.07957     56E-06
```

link2.dat

```
1     1     2    50E-06     5.0E-06    1E-06     3.141593e-18    0
2     1     3    50E-06     5.0E-06    1E-06     3.141593e-18    0
3     1     4    50E-06     5.0E-06    1E-06     3.141593e-18    0
4     1     5    50E-06     5.0E-06    1E-06     3.141593e-18    0
5     1     6    50E-06     5.0E-06    1E-06     3.141593e-18    0
6     1     7    50E-06     5.0E-06    1E-06     3.141593e-18    0
7     1     8    50E-06     5.0E-06    1E-06     3.141593e-18    0
```

```
8  1    9    50E-06    5.0E-06   1E-06    3.141593e-18   0
9  1   10    50E-06    5.0E-06   1E-06    3.141593e-18   0
10 1   11    50E-06    5.0E-06   1E-06    3.141593e-18   0
11 1   12    50E-06    5.0E-06   1E-06    3.141593e-18   0
12 1   13    50E-06    5.0E-06   1E-06    3.141593e-18   0
13 1   14    50E-06    5.0E-06   1E-06    3.141593e-18   0
14 1   15    50E-06    5.0E-06   1E-06    3.141593e-18   0
15 1   16    50E-06    5.0E-06   1E-06    3.141593e-18   0
16 1   17    50E-06    5.0E-06   1E-06    3.141593e-18   0
17 1   18    50E-06    5.0E-06   1E-06    3.141593e-18   0
18 1   19    50E-06    5.0E-06   1E-06    3.141593e-18   0
19 1   20    50E-06    5.0E-06   1E-06    3.141593e-18   0
20 1   21    50E-06    5.0E-06   1E-06    3.141593e-18   0
21 1   22    50E-06    5.0E-06   1E-06    3.141593e-18   0
22 1   23    50E-06    5.0E-06   1E-06    3.141593e-18   0
23 1   24    50E-06    5.0E-06   1E-06    3.141593e-18   0
24 1   25    50E-06    5.0E-06   1E-06    3.141593e-18   0
25 1   26    50E-06    5.0E-06   1E-06    3.141593e-18   0
26 1   27    50E-06    5.0E-06   1E-06    3.141593e-18   0
27 1   28    50E-06    5.0E-06   1E-06    3.141593e-18   0
28 1   29    50E-06    5.0E-06   1E-06    3.141593e-18   0
29 1   30    50E-06    5.0E-06   1E-06    3.141593e-18   0
30 1   31    50E-06    5.0E-06   1E-06    3.141593e-18   0
31 1   32    50E-06    5.0E-06   1E-06    3.141593e-18   0
32 1   33    50E-06    5.0E-06   1E-06    3.141593e-18   0
33 1   34    50E-06    5.0E-06   1E-06    3.141593e-18   0
34 1   35    50E-06    5.0E-06   1E-06    3.141593e-18   0
35 1   36    50E-06    5.0E-06   1E-06    3.141593e-18   0
36 1   37    50E-06    5.0E-06   1E-06    3.141593e-18   0
37 1   38    50E-06    5.0E-06   1E-06    3.141593e-18   0
38 1   39    50E-06    5.0E-06   1E-06    3.141593e-18   0
39 1   40    50E-06    5.0E-06   1E-06    3.141593e-18   0
40 1   41    50E-06    5.0E-06   1E-06    3.141593e-18   0
41 1   42    50E-06    5.0E-06   1E-06    3.141593e-18   0
42 1   43    50E-06    5.0E-06   1E-06    3.141593e-18   0
```

node1.dat

```
1  43     200.0000e-06        200.0000e-06        200.0000e-06
2
3  1     80.000e-06          80.000e-06          80.00e-06                  42
       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27
       28 29 30 31 32 33 34 35 36 37 38 39 40 41 42   0    0   1 2 3 4 5 6 7 8
       9 10 11 12 13 14 15 16 17 18 19 20 21 21 23 24 25 26 27 28 29 30 31 32
      33 34 35 36 37 38 39 40 41 42
4
5  2    136.00e-06      80.000e-06       80.00e-06      1     1    0   0    1
6  3    131.7373e-06    101.4303e-06     80.00e-06      1     1    0   0    2
7  4    119.598e-06     119.598e-06      80.00e-06      1     1    0   0    3
8  5    101.4303e-06    131.7373e-06     80.00e-06      1     1    0   0    4
9  6    80e-06          136e-06          80.00e-06      1     1    0   0    5
10 7    58.56973e-06    131.7373e-06     80.00e-06      1     1    0   0    6
11 8    40.40202e-06    119.598e-06      80.00e-06      1     1    0   0    7
12 9    28.26275e-06    101.4303e-06     80.00e-06      1     1    0   0    8
13 10   24e-06          80e-06           80.00e-06      1     1    0   0    9
14 11   28.26275e-06    58.56973e-06     80.00e-06      1     1    0   0    10
15 12   40.40202e-06    40.40202e-06     80.00e-06      1     1    0   0    11
```

```
13   58.56973e-06    28.26275e-06    80.00e-06       1    1    0    0    12
14   80e-06          24e-06          80.00e-06       1    1    0    0    13
15   101.4303e-06    28.26275e-06    80.00e-06       1    1    0    0    14
16   119.598e-06     40.40202e-06    80.00e-06       1    1    0    0    15
17   131.7373e-06    58.56973e-06    80.00e-06       1    1    0    0    16

18   80.00e-06       131.7373e-06    101.4303e-06    1    1    0    0    17
19   80.00e-06       119.598e-06     119.598e-06     1    1    0    0    18
20   80.00e-06       101.4303e-06    131.7373e-06    1    1    0    0    19
21   80.00e-06       80e-06          136e-06         1    1    0    0    20
22   80.00e-06       58.56973e-06    131.7373e-06    1    1    0    0    21
23   80.00e-06       40.40202e-06    119.598e-06     1    1    0    0    22
24   80.00e-06       28.26275e-06    101.4303e-06    1    1    0    0    23
25   80.00e-06       28.26275e-06    58.56973e-06    1    1    0    0    24
26   80.00e-06       40.40202e-06    40.40202e-06    1    1    0    0    25
27   80.00e-06       58.56973e-06    28.26275e-06    1    1    0    0    26
28   80.00e-06       80e-06          24e-06          1    1    0    0    27
29   80.00e-06       101.4303e-06    28.26275e-06    1    1    0    0    28
30   80.00e-06       119.598e-06     40.40202e-06    1    1    0    0    29
31   80.00e-06       131.7373e-06    58.56973e-06    1    1    0    0    30

32   131.7373e-06    80.00e-06       101.4303e-06    1    1    0    0    31
33   119.598e-06     80.00e-06       119.598e-06     1    1    0    0    32
34   101.4303e-06    80.00e-06       131.7373e-06    1    1    0    0    33
35   58.56973e-06    80.00e-06       131.7373e-06    1    1    0    0    34
36   40.40202e-06    80.00e-06       119.598e-06     1    1    0    0    35
37   28.26275e-06    80.00e-06       101.4303e-06    1    1    0    0    36
38   28.26275e-06    80.00e-06       58.56973e-06    1    1    0    0    37
39   40.40202e-06    80.00e-06       40.40202e-06    1    1    0    0    38
40   58.56973e-06    80.00e-06       28.26275e-06    1    1    0    0    39
41   101.4303e-06    80.00e-06       28.26275e-06    1    1    0    0    40
42   119.598e-06     80.00e-06       40.40202e-06    1    1    0    0    41
43   131.7373e-06    80.00e-06       58.56973e-06    1    1    0    0    42
```

node2.dat

```
1           5.235988e-13          50.000e-06          0.07957
    0.000

2            5.235988e-16         5.0000e-06          0.07957
    0.000
3            5.235988e-16         5.0000e-06          0.07957
    0.000
4            5.235988e-16         5.0000e-06          0.07957
    0.000
5            5.235988e-16         5.0000e-06          0.07957
    0.000
6            5.235988e-16         5.0000e-06          0.07957
    0.000
7            5.235988e-16         5.0000e-06          0.07957
    0.000
8            5.235988e-16         5.0000e-06          0.07957
    0.000
9            5.235988e-16         5.0000e-06          0.07957
    0.000
10           5.235988e-16         5.0000e-06          0.07957
    0.000
```

```
11          5.235988e-16          5.0000e-06          0.07957
    0.000
12          5.235988e-16          5.0000e-06          0.07957
    0.000
13          5.235988e-16          5.0000e-06          0.07957
    0.000
14          5.235988e-16          5.0000e-06          0.07957
    0.000
15          5.235988e-16          5.0000e-06          0.07957
    0.000
16          5.235988e-16          5.0000e-06          0.07957
    0.000
17          5.235988e-16          5.0000e-06          0.07957
    0.000
18          5.235988e-16          5.0000e-06          0.07957
    0.000
19          5.235988e-16          5.0000e-06          0.07957
    0.000
20          5.235988e-16          5.0000e-06          0.07957
    0.000
21          5.235988e-16          5.0000e-06          0.07957
    0.000
22          5.235988e-16          5.0000e-06          0.07957
    0.000
23          5.235988e-16          5.0000e-06          0.07957
    0.000
24          5.235988e-16          5.0000e-06          0.07957
    0.000
25          5.235988e-16          5.0000e-06          0.07957
    0.000
26          5.235988e-16          5.0000e-06          0.07957
    0.000
27          5.235988e-16          5.0000e-06          0.07957
    0.000
28          5.235988e-16          5.0000e-06          0.07957
    0.000
29          5.235988e-16          5.0000e-06          0.07957
    0.000
30          5.235988e-16          5.0000e-06          0.07957
    0.000
31          5.235988e-16          5.0000e-06          0.07957
    0.000
32          5.235988e-16          5.0000e-06          0.07957
    0.000
33          5.235988e-16          5.0000e-06          0.07957
    0.000
34          5.235988e-16          5.0000e-06          0.07957
    0.000
35          5.235988e-16          5.0000e-06          0.07957
    0.000
36          5.235988e-16          5.0000e-06          0.07957
    0.000
37          5.235988e-16          5.0000e-06          0.07957
    0.000
38          5.235988e-16          5.0000e-06          0.07957
    0.000
39          5.235988e-16          5.0000e-06          0.07957
    0.000
```

```
40              5.235988e-16          5.0000e-06          0.07957
    0.000
41              5.235988e-16          5.0000e-06          0.07957
    0.000
42              5.235988e-16          5.0000e-06          0.07957
    0.000
43              5.235988e-16          5.0000e-06          0.07957
    0.000
```

Appendix B

Single-phase NMR network simulation code

B.1 Input file

In this section, an exemplary input file for the single-phase NMR network simulation code is presented, including the 12 Keywords described in detail in Section 3.4.

```
TITLE
2Pores
#

WALKERS
2000
#

TIME_STEP
0.01E-4
#

DIFF_CONSTANT
2.5E-9
#

RELAXIVITY
40.0E-6
#

BULK_RELAXIVITY
2.5
#

REPORT_TIME
1
#

NETWORK
F   2Pores
#

NUM_TIME_DATA
```

```
100
#

DECAY_CUT_OFF
8.0
#

INTER_ECHO_TIME
00.0E-6
#

MAGNET_GRADIENT
0.0
#
```

B.2 Simulation files

In this Appendix, the C++ "Single-phase NMR network simulation code" developed by Talabi (2008) and available at the Imperial College of London's website (https://imperialcollegelondon.app.box.com/v/NMR-single-phase-network-model) is included. For this master's thesis, no alterations were made to the codes.

As previously described in Section 2.5.2 and in Section 3, Talabi (2008)'s code is a C++ pack composed of 12 files, all necessary to run the Random Walk simulation. In this code, the water-forming hydrogen atoms are represented as "walkers" that are initiated and allowed to diffuse within the user-delimited pores and throats elements. The direction of the "walkers" movement is random (hence the name Random Walk) but the distance taken in each time-step is defined by the user. After touching a pore or throat wall, a "walker" will either "relax" and leave the simulation or "survive" and bounce back into the network element. The fate of the "walker" will depend on the user-defined surface relaxivity value. The ratio between surviving and relaxed "walkers" is registered as the magnetisation of the system. Further explanation on the Random Walk methodology can be found in Section 2.5.2 and in Section 3. A description of each file within the C++ code is given in Section 3.3.

input.h

```
#ifndef INPUT_H
#define INPUT_H

class Input
{
public:

    Input(){}
    Input(const string&);

    void title(string& baseFileName);
    void randSeed(int& seedNum);
    void walkers(double& numWalkers);
  void timeStep(double& tStep);
  void diffConst(double& dConstant);
  void timeData (int& timedt);
  void decayCutOff (double& dCutOff);
  void relaxivity(double& relax);
    void bulkRelax (double& bulkRelax1);
  void reportTime(int& rTime);
    void interEchoTime(double& dEchoTime);
    void magnetGradient(double& dMagGradient);


private:

    inline void errorMsg(const string& keyword) const;
  inline void missingDataErr(const string& keyword) const;
    inline bool getData(istringstream& data, const string& keyword);
    inline void errorInDataCheck(istringstream& data, const string& keyword
    ) const;
    void removeComments(string& data) const;
  bool terminatorFound(string& data) const;
```

```
33
34      vector< pair< string, string > >                        m_parsedData;
35      string                                                  m_baseFileName;
36
37  };
38
39  ////////////////////////////////////////////////////////////////////////
40  // Returns error message if error occurs during reading of data string
41  ////////////////////////////////////////////////////////////////////////
42
43  inline void Input::errorMsg(const string& keyword) const
44  {
45      cerr << endl
46          << "================================"   << endl
47          << "Error while reading input file. "   << endl
48          << "Keyword: " << keyword               << endl
49          << "================================"   << endl;
50      exit(-1);
51  }
52
53  ////////////////////////////////////////////////////////////
54  // Returns error message when required keyword is missing
55  ////////////////////////////////////////////////////////////
56  inline void Input::missingDataErr(const string& keyword) const
57  {
58      cerr << endl
59          << "============================================"   << endl
60          << "Error: " << keyword << " is a required keyword."  << endl
61          << "============================================"   << endl;
62      exit(-1);
63  }
64
65  ////////////////////////////////////////////////////////////////////////
66  // Retrieves data sting based on supplied keyword, and connects a
67  // string stream to it. Data is removed from storage after having been
68  // retrived
69  ////////////////////////////////////////////////////////////////////////
70  inline bool Input::getData(istringstream& data, const string& keyword)
71  {
72      for(unsigned i = 0; i < m_parsedData.size(); ++i)
73      {
74          if(m_parsedData[i].first == keyword)
75          {
76              data.str(m_parsedData[i].second);
77              return true;
78          }
79      }
80  //    cout <<m_parsedData.size()<<endl;
81         return false;
82
83  }
84
85  inline void Input::errorInDataCheck(istringstream& data, const string&
        keyword) const
86  {
87      char leftOvers;
88      data >> leftOvers;
89      if(data)
```

```
90      {
91          cerr << endl
92              << "======================================"   << endl
93              << "Error: Too much data read for keyword: " << keyword
                                << endl
94              << "Data read: " << data.str()
                                << endl
95              << "======================================"  << endl;
96          exit(-1);
97      }
98  }
99
100
101
102 #endif
```

MersenneTwister.h

```
// MersenneTwister.h
// Mersenne Twister random number generator -- a C++ class MTRand
// Based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn Cokus
// Richard J. Wagner  v1.0  15 May 2003  rjwagner@writeme.com

// The Mersenne Twister is an algorithm for generating random numbers.  It
// was designed with consideration of the flaws in various other generators.
// The period, 2^19937-1, and the order of equidistribution, 623 dimensions,
// are far greater.  The generator is also fast; it avoids multiplication and
// division, and it benefits from caches and pipelines.  For more information
// see the inventors' web page at http://www.math.keio.ac.jp/~matumoto/emt.html

// Reference
// M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally
// Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on
// Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3-30.

// Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
// Copyright (C) 2000 - 2003, Richard J. Wagner
// All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
//   1. Redistributions of source code must retain the above copyright
//      notice, this list of conditions and the following disclaimer.
//
//   2. Redistributions in binary form must reproduce the above copyright
//      notice, this list of conditions and the following disclaimer in the
//      documentation and/or other materials provided with the distribution.
//
//   3. The names of its contributors may not be used to endorse or promote
//      products derived from this software without specific prior written
//      permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```

```
// The original code included the following notice:
//
//     When you use this, send an email to: matumoto@math.keio.ac.jp
//     with an appropriate reference to your work.
//
// It would be nice to CC: rjwagner@writeme.com and Cokus@math.washington.edu
// when you write.

#ifndef MERSENNETWISTER_H
#define MERSENNETWISTER_H

// Not thread safe (unless auto-initialization is avoided and each thread has
// its own MTRand object)

#include <iostream>
#include <limits.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

class MTRand {
// Data
public:
  typedef unsigned long uint32;  // unsigned integer type, at least 32 bits

  enum { N = 624 };       // length of state vector
  enum { SAVE = N + 1 };  // length of array for save()

protected:
  enum { M = 397 };  // period parameter

  uint32 state[N];   // internal state
  uint32 *pNext;     // next value to get from state
  int left;          // number of values left before reload needed


//Methods
public:
  MTRand( const uint32& oneSeed );  // initialize with a simple uint32
  MTRand( uint32 *const bigSeed, uint32 const seedLength = N );  // or an array
  MTRand();  // auto-initialize with /dev/urandom or time() and clock()

  // Do NOT use for CRYPTOGRAPHY without securely hashing several returned
  // values together, otherwise the generator state can be learned after
  // reading 624 consecutive values.

  // Access to 32-bit random numbers
  double rand();                          // real number in [0,1]
  double rand( const double& n );         // real number in [0,n]
  double randExc();                       // real number in [0,1)
  double randExc( const double& n );      // real number in [0,n)
  double randDblExc();                    // real number in (0,1)
  double randDblExc( const double& n );   // real number in (0,n)
  uint32 randInt();                       // integer in [0,2^32-1]
  uint32 randInt( const uint32& n );      // integer in [0,n] for n < 2^32
```

```
104   double operator()() { return rand(); }  // same as rand()
105
106   // Access to 53-bit random numbers (capacity of IEEE double precision)
107   double rand53();  // real number in [0,1)
108
109   // Access to nonuniform random number distributions
110   double randNorm( const double& mean = 0.0, const double& variance = 0.0 )
       ;
111
112   // Re-seeding functions with same behavior as initializers
113   void seed( const uint32 oneSeed );
114   void seed( uint32 *const bigSeed, const uint32 seedLength = N );
115   void seed();
116
117   // Saving and loading generator state
118   void save( uint32* saveArray ) const;  // to array of size SAVE
119   void load( uint32 *const loadArray );  // from such array
120   friend std::ostream& operator<<( std::ostream& os, const MTRand& mtrand )
       ;
121   friend std::istream& operator>>( std::istream& is, MTRand& mtrand );
122
123 protected:
124   void initialize( const uint32 oneSeed );
125   void reload();
126   uint32 hiBit( const uint32& u ) const { return u & 0x80000000UL; }
127   uint32 loBit( const uint32& u ) const { return u & 0x00000001UL; }
128   uint32 loBits( const uint32& u ) const { return u & 0x7fffffffUL; }
129   uint32 mixBits( const uint32& u, const uint32& v ) const
130     { return hiBit(u) | loBits(v); }
131   uint32 twist( const uint32& m, const uint32& s0, const uint32& s1 ) const
132     { return m ^ (mixBits(s0,s1)>>1) ^ (-loBit(s1) & 0x9908b0dfUL); }
133   static uint32 hash( time_t t, clock_t c );
134 };
135
136
137 inline MTRand::MTRand( const uint32& oneSeed )
138   { seed(oneSeed); }
139
140 inline MTRand::MTRand( uint32 *const bigSeed, const uint32 seedLength )
141   { seed(bigSeed,seedLength); }
142
143 inline MTRand::MTRand()
144   { seed(); }
145
146 inline double MTRand::rand()
147   { return double(randInt()) * (1.0/4294967295.0); }
148
149 inline double MTRand::rand( const double& n )
150   { return rand() * n; }
151
152 inline double MTRand::randExc()
153   { return double(randInt()) * (1.0/4294967296.0); }
154
155 inline double MTRand::randExc( const double& n )
156   { return randExc() * n; }
157
158 inline double MTRand::randDblExc()
159   { return ( double(randInt()) + 0.5 ) * (1.0/4294967296.0); }
```

```
160
161 inline double MTRand::randDblExc( const double& n )
162   { return randDblExc() * n; }
163
164 inline double MTRand::rand53()
165 {
166   uint32 a = randInt() >> 5, b = randInt() >> 6;
167   return ( a * 67108864.0 + b ) * (1.0/9007199254740992.0);  // by Isaku
      Wada
168 }
169
170 inline double MTRand::randNorm( const double& mean, const double& variance
      )
171 {
172   // Return a real number from a normal (Gaussian) distribution with given
173   // mean and variance by Box-Muller method
174   double r = sqrt( -2.0 * log( 1.0-randDblExc()) ) * variance;
175   double phi = 2.0 * 3.14159265358979323846264338328 * randExc();
176   return mean + r * cos(phi);
177 }
178
179 inline MTRand::uint32 MTRand::randInt()
180 {
181   // Pull a 32-bit integer from the generator state
182   // Every other access function simply transforms the numbers extracted
      here
183
184   if( left == 0 ) reload();
185   --left;
186
187   register uint32 s1;
188   s1 = *pNext++;
189   s1 ^= (s1 >> 11);
190   s1 ^= (s1 <<  7) & 0x9d2c5680UL;
191   s1 ^= (s1 << 15) & 0xefc60000UL;
192   return ( s1 ^ (s1 >> 18) );
193 }
194
195 inline MTRand::uint32 MTRand::randInt( const uint32& n )
196 {
197   // Find which bits are used in n
198   // Optimized by Magnus Jonsson (magnus@smartelectronix.com)
199   uint32 used = n;
200   used |= used >> 1;
201   used |= used >> 2;
202   used |= used >> 4;
203   used |= used >> 8;
204   used |= used >> 16;
205
206   // Draw numbers until one is found in [0,n]
207   uint32 i;
208   do
209     i = randInt() & used;  // toss unused bits to shorten search
210   while( i > n );
211   return i;
212 }
213
214
```

```
inline void MTRand::seed( const uint32 oneSeed )
{
  // Seed the generator with a simple uint32
  initialize(oneSeed);
  reload();
}


inline void MTRand::seed( uint32 *const bigSeed, const uint32 seedLength )
{
  // Seed the generator with an array of uint32's
  // There are 2^19937-1 possible initial states.  This function allows
  // all of those to be accessed by providing at least 19937 bits (with a
  // default seed length of N = 624 uint32's).  Any bits above the lower 32
  // in each element are discarded.
  // Just call seed() if you want to get array from /dev/urandom
  initialize(19650218UL);
  register int i = 1;
  register uint32 j = 0;
  register int k = ( N > seedLength ? N : seedLength );
  for( ; k; --k )
  {
    state[i] =
      state[i] ^ ( (state[i-1] ^ (state[i-1] >> 30)) * 1664525UL );
    state[i] += ( bigSeed[j] & 0xffffffffUL ) + j;
    state[i] &= 0xffffffffUL;
    ++i;  ++j;
    if( i >= N ) { state[0] = state[N-1];  i = 1; }
    if( j >= seedLength ) j = 0;
  }
  for( k = N - 1; k; --k )
  {
    state[i] =
      state[i] ^ ( (state[i-1] ^ (state[i-1] >> 30)) * 1566083941UL );
    state[i] -= i;
    state[i] &= 0xffffffffUL;
    ++i;
    if( i >= N ) { state[0] = state[N-1];  i = 1; }
  }
  state[0] = 0x80000000UL;  // MSB is 1, assuring non-zero initial array
  reload();
}


inline void MTRand::seed()
{
  // Seed the generator with an array from /dev/urandom if available
  // Otherwise use a hash of time() and clock() values

  // First try getting an array from /dev/urandom
  FILE* urandom = fopen( "/dev/urandom", "rb" );
  if( urandom )
  {
    uint32 bigSeed[N];
    register uint32 *s = bigSeed;
    register int i = N;
    register bool success = true;
    while( success && i-- )
```

```
273       success = fread( s++, sizeof(uint32), 1, urandom );
274     fclose(urandom);
275     if( success ) { seed( bigSeed, N );  return; }
276   }
277
278   // Was not successful, so use time() and clock() instead
279   seed( hash( time(NULL), clock() ) );
280 }
281
282
283 inline void MTRand::initialize( const uint32 seed )
284 {
285   // Initialize generator state with seed
286   // See Knuth TAOCP Vol 2, 3rd Ed, p.106 for multiplier.
287   // In previous versions, most significant bits (MSBs) of the seed affect
288   // only MSBs of the state array.  Modified 9 Jan 2002 by Makoto Matsumoto
      .
289   register uint32 *s = state;
290   register uint32 *r = state;
291   register int i = 1;
292   *s++ = seed & 0xffffffffUL;
293   for( ; i < N; ++i )
294   {
295     *s++ = ( 1812433253UL * ( *r ^ (*r >> 30) ) + i ) & 0xffffffffUL;
296     r++;
297   }
298 }
299
300
301 inline void MTRand::reload()
302 {
303   // Generate N new values in state
304   // Made clearer and faster by Matthew Bellew (matthew.bellew@home.com)
305   register uint32 *p = state;
306   register int i;
307   for( i = N - M; i--; ++p )
308     *p = twist( p[M], p[0], p[1] );
309   for( i = M; --i; ++p )
310     *p = twist( p[M-N], p[0], p[1] );
311   *p = twist( p[M-N], p[0], state[0] );
312
313   left = N, pNext = state;
314 }
315
316
317 inline MTRand::uint32 MTRand::hash( time_t t, clock_t c )
318 {
319   // Get a uint32 from t and c
320   // Better than uint32(x) in case x is floating point in [0,1]
321   // Based on code by Lawrence Kirby (fred@genesis.demon.co.uk)
322
323   static uint32 differ = 0;  // guarantee time-based seeds will change
324
325   uint32 h1 = 0;
326   unsigned char *p = (unsigned char *) &t;
327   for( size_t i = 0; i < sizeof(t); ++i )
328   {
329     h1 *= UCHAR_MAX + 2U;
```

```
    h1 += p[i];
  }
  uint32 h2 = 0;
  p = (unsigned char *) &c;
  for( size_t j = 0; j < sizeof(c); ++j )
  {
    h2 *= UCHAR_MAX + 2U;
    h2 += p[j];
  }
  return ( h1 + differ++ ) ^ h2;
}


inline void MTRand::save( uint32* saveArray ) const
{
  register uint32 *sa = saveArray;
  register const uint32 *s = state;
  register int i = N;
  for( ; i--; *sa++ = *s++ ) {}
  *sa = left;
}


inline void MTRand::load( uint32 *const loadArray )
{
  register uint32 *s = state;
  register uint32 *la = loadArray;
  register int i = N;
  for( ; i--; *s++ = *la++ ) {}
  left = *la;
  pNext = &state[N-left];
}


inline std::ostream& operator<<( std::ostream& os, const MTRand& mtrand )
{
  register const MTRand::uint32 *s = mtrand.state;
  register int i = mtrand.N;
  for( ; i--; os << *s++ << "\t" ) {}
  return os << mtrand.left;
}


inline std::istream& operator>>( std::istream& is, MTRand& mtrand )
{
  register MTRand::uint32 *s = mtrand.state;
  register int i = mtrand.N;
  for( ; i--; is >> *s++ ) {}
  is >> mtrand.left;
  mtrand.pNext = &mtrand.state[mtrand.N-mtrand.left];
  return is;
}

#endif  // MERSENNETWISTER_H

// Change log:
//
// v0.1 - First release on 15 May 2000
```

```
388 //      - Based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn
    Cokus
389 //      - Translated from C to C++
390 //      - Made completely ANSI compliant
391 //      - Designed convenient interface for initialization, seeding, and
392 //        obtaining numbers in default or user-defined ranges
393 //      - Added automatic seeding from /dev/urandom or time() and clock()
394 //      - Provided functions for saving and loading generator state
395 //
396 // v0.2 - Fixed bug which reloaded generator one step too late
397 //
398 // v0.3 - Switched to clearer, faster reload() code from Matthew Bellew
399 //
400 // v0.4 - Removed trailing newline in saved generator format to be
    consistent
401 //        with output format of built-in types
402 //
403 // v0.5 - Improved portability by replacing static const int's with enum's
    and
404 //        clarifying return values in seed(); suggested by Eric Heimburg
405 //      - Removed MAXINT constant; use 0xffffffffUL instead
406 //
407 // v0.6 - Eliminated seed overflow when uint32 is larger than 32 bits
408 //      - Changed integer [0,n] generator to give better uniformity
409 //
410 // v0.7 - Fixed operator precedence ambiguity in reload()
411 //      - Added access for real numbers in (0,1) and (0,n)
412 //
413 // v0.8 - Included time.h header to properly support time_t and clock_t
414 //
415 // v1.0 - Revised seeding to match 26 Jan 2002 update of Nishimura and
    Matsumoto
416 //      - Allowed for seeding with arrays of any length
417 //      - Added access for real numbers in [0,1) with 53-bit resolution
418 //      - Added access for real numbers from normal (Gaussian)
    distributions
419 //      - Increased overall speed by optimizing twist()
420 //      - Doubled speed of integer [0,n] generation
421 //      - Fixed out-of-range number generation on 64-bit machines
422 //      - Improved portability by substituting literal constants for long
    enum's
423 //      - Changed license from GNU LGPL to BSD
```

NMRsim.h

```
#ifndef NMRSIM_H
#define NMRSIM_H

typedef struct
{
  int          index;
    double       x;
    double       y;
    double       z;
    int          connNum;
    double       volume;
    double       radius;
    double       shapeFact;
    double       clayVol;

} PoreStruct;

typedef struct
{
    int          index;
    double       radius;
    double       length;
    double       volume;
  double       shapeFact;
    int          connNum;

} PoreStruct1;

typedef struct
{
  int      index;
    int      poreOne;
    int      poreTwo;
    double   radius;
    double   shapeFact;
    double   lenPoreOne;
    double   lenPoreTwo;
    double   lenThroat;
    double   lenTot;
    double   volume;
    double   clayVol;

} ThroatStruct;

typedef struct
{
    int      index;
    int      poreOne;
    int      poreTwo;
    double   radius;
    double   length;
    double   volume;
  double   shapeFact;

} ThroatStruct1;

```

```
class NMRsim
{
public:

    NMRsim(unsigned seed);
    NMRsim(const NMRsim& net);
    void init(Input& input);
    void network(int& numPores, int& numThroats, double& xDim, double& yDim
   , double& zDim);
    void randomMotion();
  void readData(const string&);
  void createFile();
  void writeDecay();
    void dataFileOut();
  void weightedMatrix();
  void kernelMatrix();

private:


    unsigned                                    m_randSeed;
    void loadPoreData();
    void loadThroatData();
  void initWalkers();
  double calcPorosity();
  double calcAvrgCordNum();

    inline void errorMsg(const string& keyword) const;
    inline void missingDataErr(const string& keyword) const;
    inline bool getData(istringstream& data, const string& keyword);
    inline void errorInDataCheck(istringstream& data, const string& keyword
   ) const;
    void removeComments(string& data) const;
    bool terminatorFound(string& data) const;


    vector< pair< string, string > >            m_parsedData;
    string                                      m_baseFileName;
    string                                      m_simFileName;

    int                                         m_numPores;
    int                                         m_numThroats;
    int                                         m_origNumPores;
    int                                         m_origNumThroats;
    int                                         m_reportTime;
  int                                         m_numInletThroats;
    int                                         m_numOutletThroats;
    int                                         m_InOutThroats;
  int                                         m_numTimeData;

  bool                                        m_binaryFiles;
    bool                                        m_nmr;
  bool                                        m_initPos;
  bool                                        m_resultSum;
  bool                                        m_walkData;
  double                                      m_numWalkers;
    double                                      m_xSize;
    double                                      m_ySize;
```

```
    double                                          m_zSize;
  double                                           m_avrgConnNum;
    double                                          m_origXDim;
    double                                          m_origYDim;
    double                                          m_origZDim;
  double                                           m_walkVolume;
  double                                           m_timeStep;
  double                                           m_scaleFactor;
    double                                          m_diffConstant;
  double                                           m_relaxivity;
  double                                           m_bulkRelaxivity;
    double                                          m_bulkVolume;
  double                                           m_porosity;
    double                                          m_decayCutOff;

  double                                           m_areaFract1T;
    double                                          m_areaFract2T;
    double                                          m_areaFract3T;
    double                                          m_areaFractSqT;

  double                                           m_area1T;
    double                                          m_area2T;
    double                                          m_area3T;
    double                                          m_areaSqT;

    double                                          m_fractHit1T;
    double                                          m_fractHit2T;
    double                                          m_fractHit3T;
    double                                          m_fractHitSqT;

    double                                          m_interEchoTime;
  double                                           m_magneticGradient;




    ifstream                                        m_poreConn;
    ifstream                                        m_poreProp;
    ifstream                                        m_throatConn;
    ifstream                                        m_throatProp;
  ofstream                                         m_nmrResponse;
  ofstream                                         m_initialPos;
  ofstream                                         m_resultSumm;
  ofstream                                         m_walkout;
    ofstream                                        m_resultPrt;
    ofstream                                        out;
    ofstream                                        m_resultM;
    ofstream                                        m_resultK;



    vector <double> m_timeEcho;

    vector< ThreeSome< PoreStruct*, int*, int* > >  m_poreData1;
    vector< ThroatStruct* >                         m_throatData1;

    vector< TwoSome< PoreStruct1*, int* > >         m_poreData;
  vector< ThroatStruct1* >                          m_throatData;
```

```
171     vector< ThroatStruct1* >                          m_throatData22;
172
173     vector< ThreeSome <int, int, int> >               throatHold;
174
175   vector< FiveSome <char, int, double, double, double> >   m_initPosition;
176     vector< FiveSome <char, int, double, double, double> >
      m_finalPosition;
177     vector< FiveSome <char, int, double, double, double> >
      m_initPosition1;
178     vector< FiveSome <char, int, double, double, double> >   m_elemGeometry
      ;
179     vector< TwoSome <double, double      > >                 amplitudeDecay
      ;
180
181       static const double                               PI;
182       static const double                               SIGN;
183       static const double                               DUMMY;
184 };
185
186 ////////////////////////////////////////////////////////////////////////
187 // Returns error message is error occurs during reading of data string
188 ////////////////////////////////////////////////////////////////////////
189 inline void NMRsim::errorMsg(const string& keyword) const
190 {
191     cerr << endl
192         << "================================"   << endl
193         << "Error while reading input file. "   << endl
194         << "Keyword: " << keyword               << endl
195         << "================================"   << endl;
196     exit(-1);
197 }
198
199 ////////////////////////////////////////////////////////////
200 // Returns error message when required keyword is missing
201 ////////////////////////////////////////////////////////////
202 inline void NMRsim::missingDataErr(const string& keyword) const
203 {
204     cerr << endl
205         << "============================================="   << endl
206         << "Error: " << keyword << " is a required keyword."  << endl
207         << "============================================="   << endl;
208     exit(-1);
209 }
210
211 ////////////////////////////////////////////////////////////////////////
212 // Retrieves data sting based on supplied keyword, and connects a
213 // string stream to it. Data is removed from storage after having been
214 // retrived
215 ////////////////////////////////////////////////////////////////////////
216 inline bool NMRsim::getData(istringstream& data, const string& keyword)
217 {
218     for(unsigned i = 0; i < m_parsedData.size(); ++i)
219     {
220         if(m_parsedData[i].first == keyword)
221         {
222             data.str(m_parsedData[i].second);
223             return true;
224         }
```

```
225     }
226     cout <<m_parsedData.size()<<endl;
227        return false;
228 
229 }
230 
231 inline void NMRsim::errorInDataCheck(istringstream& data, const string&
    keyword) const
232 {
233     char leftOvers;
234     data >> leftOvers;
235     if(data)
236     {
237         cerr << endl
238             << "=====================================================" <<
    endl
239             << "Error: Too much data read for keyword: " << keyword     <<
    endl
240             << "Data read: " << data.str()                              <<
    endl
241             << "=====================================================" <<
    endl;
242         exit(-1);
243     }
244 }
245 
246 #endif
```

threeSome.h

```
#ifndef THREESOME_H
#define THREESOME_H

////////////////////////////////////////////////////////////////////////
// A little storage class for two elements
////////////////////////////////////////////////////////////////////////
template<typename TOne, typename TTwo>
class TwoSome
{
public:

    TwoSome() {}
    TwoSome(TOne one, TTwo two) : m_first(one), m_second(two) {}

    const TOne& first() const {return m_first;}
    const TTwo& second() const {return m_second;}
    void first(TOne entry) {m_first = entry;}
    void second(TTwo entry) {m_second = entry;}

 private:
    TOne                m_first;
    TTwo                m_second;


};




////////////////////////////////////////////////////////////////////////
// A little storage class for three eleemnts
////////////////////////////////////////////////////////////////////////
template<typename TOne, typename TTwo, typename TThree>
class ThreeSome
{
public:

    ThreeSome() {}
    ThreeSome(TOne one, TTwo two, TThree three) : m_first(one), m_second(
   two), m_third(three) {}

    const TOne& first() const {return m_first;}
    const TTwo& second() const {return m_second;}
    const TThree& third() const {return m_third;}
    void first(TOne entry) {m_first = entry;}
    void second(TTwo entry) {m_second = entry;}
    void third(TThree entry) {m_third = entry;}

private:

    TOne                m_first;
    TTwo                m_second;
    TThree              m_third;

};

```

```
56 ////////////////////////////////////////////////////////////////////////
57 // A little storage class for four eleemnts
58 ////////////////////////////////////////////////////////////////////////
59 template<typename TOne, typename TTwo, typename TThree, typename TFour>
60 class FourSome
61 {
62 public:
63
64     FourSome() {}
65     FourSome(TOne one, TTwo two, TThree three, TFour four) : m_first(one),
       m_second(two), m_third(three), m_fourth(four) {}
66
67     const TOne& first() const {return m_first;}
68     const TTwo& second() const {return m_second;}
69     const TThree& third() const {return m_third;}
70     const TFour& fourth() const {return m_fourth;}
71     void first(TOne entry) {m_first = entry;}
72     void second(TTwo entry) {m_second = entry;}
73     void third(TThree entry) {m_third = entry;}
74     void fourth(TFour entry) {m_fourth = entry;}
75
76 private:
77
78     TOne                m_first;
79     TTwo                m_second;
80     TThree              m_third;
81     TFour               m_fourth;
82
83 };
84
85 ////////////////////////////////////////////////////////////////////////
86 // A little storage class for five elements
87 ////////////////////////////////////////////////////////////////////////
88
89 template<typename TOne, typename TTwo, typename TThree, typename TFour,
       typename TFive>
90 class FiveSome
91 {
92 public:
93
94     FiveSome() {}
95     FiveSome(TOne one, TTwo two, TThree three, TFour four, TFive five) :
       m_first(one),
96     m_second(two), m_third(three), m_fourth(four), m_fifth (five) {}
97
98     const TOne& first() const {return m_first;}
99     const TTwo& second() const {return m_second;}
100    const TThree& third() const {return m_third;}
101    const TFour& fourth() const {return m_fourth;}
102    const TFive& fifth() const {return m_fifth;}
103
104    void first(TOne entry) {m_first = entry;}
105    void second(TTwo entry) {m_second = entry;}
106    void third(TThree entry) {m_third = entry;}
107    void fourth(TFour entry) {m_fourth = entry;}
108    void fifth(TFive entry) {m_fifth = entry;}
109
110 private:
```

```
111
112     TOne                  m_first;
113     TTwo                  m_second;
114     TThree                m_third;
115     TFour                 m_fourth;
116   TFive                 m_fifth;
117
118 };
119 #endif
```

Input.cpp

```
1
2  #include <iostream>
3  #include <fstream>
4  #include <sstream>
5  #include <iomanip>
6  #include <utility>
7  #include <vector>
8  #include <cmath>
9  #include <ctime>
10 #include <string>
11 #include <cassert>
12 #include <algorithm>
13 #include <map>
14 #include <set>
15 using namespace std;
16
17 #include "threeSome.h"
18 #include "Input.h"
19
20
21 Input::Input(const string& inputFileName)
22 {
23     string keyword, previousKeyword("NO_KEYWORD_READ");
24     char buffer[512];
25     ifstream in(inputFileName.c_str());
26     if (!in)
27     {
28         cerr << "================================================"      <<
    endl
29              << "Error: Unable to open input file " << inputFileName    <<
    endl
30              << "================================================"      <<
    endl;
31         exit( -1 );
32     }
33
34     while(in >> keyword)
35     {
36         if(keyword[0] == '%')
37         {
38             in.ignore(512, '\n');
39         }
40         else if(keyword.size() < 5 || keyword.size() > 15)
41         {
42             cerr << endl
43                  << "======================================="   << endl
44                  << "Data file contains errors after keyword:"   << endl
45                  << "======================================="   << endl;
46             exit( -1 );
47         }
48         else
49         {
50             in.ignore(512, '\n');
51             string dataString;
52             while(true)
53             {
```

```
                in.getline(buffer, 512, '\n');
                string bufferStr(buffer);
                removeComments(bufferStr);
                dataString += bufferStr + "  ";
                if(terminatorFound(dataString)) break;
            }
            pair< string, string > dataEntry(keyword, dataString);
            m_parsedData.push_back(dataEntry);
            previousKeyword = keyword;
        }
    }

    in.close();
    m_baseFileName = inputFileName.substr(0, inputFileName.find('.'));
}

void Input::removeComments(string& data) const
{
    string::iterator itr;
    itr = find(data.begin(), data.end(), '%');
    if(itr != data.end()) data.erase(itr, data.end());
}

bool Input::terminatorFound(string& data) const
{
    string::iterator itr;
    itr = find(data.begin(), data.end(), '#');
    if(itr != data.end())
    {
        data.erase(itr, data.end());
        return true;
    }
    return false;
}


////////////////////////////////////////////////////////////////////////
// All basic keywords are retived from the parsed storage. If not present
   the
// default values are used.
////////////////////////////////////////////////////////////////////////

void Input::title(string& baseFileName)
{
    istringstream data;
    string keyword("TITLE");

    if(getData(data, keyword))
    {
    cout <<"      "<<"OUTPUT "<<keyword <<":                          ";
    data >> baseFileName;
        if(!data) errorMsg(keyword);
        errorInDataCheck(data, keyword);
    }
    else
        baseFileName = m_baseFileName;
}

```

```
111 void Input::randSeed(int& seedNum)
112 {
113     istringstream data;
114     string keyword("RAND_SEED");
115 
116     if(getData(data, keyword))
117     {
118      // cout <<"      "<< "Reading " << keyword << endl;
119         data >> seedNum;
120         if(!data) errorMsg(keyword);
121         errorInDataCheck(data, keyword);
122     }
123     else
124         seedNum = (unsigned)time( NULL );
125 }
126 
127 void Input::walkers(double& numWalkers)
128 {
129     istringstream data;
130     string keyword("WALKERS");
131 
132     if(getData(data, keyword))
133     {
134         cout <<"      "<<"NUMBER OF "<<keyword <<":                 ";
135     data >> numWalkers;
136         if(!data) errorMsg(keyword);
137         errorInDataCheck(data, keyword);
138 
139     }
140     else
141         numWalkers = 50;
142 
143 }
144 
145 void Input::timeStep(double& tStep)
146 {
147     istringstream data;
148     string keyword("TIME_STEP");
149 
150     if(getData(data, keyword))
151     {
152         data >> tStep;
153         cout <<"      "<<"UNIT "<<keyword <<":                    ";
154         if(!data) errorMsg(keyword);
155         errorInDataCheck(data, keyword);
156 
157     }
158     else
159         tStep = 3.0E-5;
160 
161 }
162 
163 void Input::diffConst(double& dConstant)
164 {
165     istringstream data;
166     string keyword("DIFF_CONSTANT");
167 
168     if(getData(data, keyword))
```

```
169     {
170         cout <<"      "<<"DIFF. COEFFICIENT "<<":                ";
171     data >> dConstant;
172         if(!data) errorMsg(keyword);
173         errorInDataCheck(data, keyword);
174 
175     }
176     else
177         dConstant = 2.5E-9;
178 
179 }
180 
181 void Input::timeData (int& timedt)
182 {
183     istringstream data;
184     string keyword("NUM_TIME_DATA");
185 
186     if(getData(data, keyword))
187     {
188         cout <<"      "<<"DATA POINTS FOR T2-INVERSION"<<":      ";
189     data >> timedt;
190         if(!data) errorMsg(keyword);
191         errorInDataCheck(data, keyword);
192 
193     }
194     else
195         timedt = 50;
196 
197 }
198 
199 void Input::decayCutOff (double& dCutOff)
200 {
201     istringstream data;
202     string keyword("DECAY_CUT_OFF");
203 
204     if(getData(data, keyword))
205     {
206         cout <<"      "<<"NMR CUT OFF - T2 INVERSION"<<":        ";
207     data >> dCutOff;
208         if(!data) errorMsg(keyword);
209         errorInDataCheck(data, keyword);
210 
211     }
212     else
213         dCutOff = 0.0001;
214 
215 }
216 
217 void Input::relaxivity(double& relax)
218 {
219     istringstream data;
220     string keyword("RELAXIVITY");
221 
222     if(getData(data, keyword))
223     {
224         cout <<"      "<<"SURFACE RELAXIVITY "<<":               ";
225     data >> relax;
226         if(!data) errorMsg(keyword);
```

```
227         errorInDataCheck(data, keyword);
228 
229     }
230     else
231         relax = 20.0E-6;
232 
233 }
234 
235 void Input::bulkRelax(double& bulkRelax1)
236 {
237     istringstream data;
238     string keyword("BULK_RELAXIVITY");
239 
240     if(getData(data, keyword))
241     {
242         cout <<"      "<<"BULK RELAXIVITY "<<":                  ";
243     data >> bulkRelax1;
244         if(!data) errorMsg(keyword);
245         errorInDataCheck(data, keyword);
246 
247     }
248     else
249         bulkRelax1 = 2.5;
250 
251 }
252 
253 
254 void Input::reportTime(int& rTime)
255 {
256     istringstream data;
257     string keyword("REPORT_TIME");
258 
259     if(getData(data, keyword))
260     {
261         cout <<"      "<<"NMR RESULTS TIME INTERVAL"<<":         ";
262     data >> rTime;
263         if(!data) errorMsg(keyword);
264         errorInDataCheck(data, keyword);
265 
266     }
267     else
268         rTime = 2.0E-4;
269 
270 }
271 
272 void Input::interEchoTime(double& dEchoTime)
273 {
274     istringstream data;
275     string keyword("INTER_ECHO_TIME");
276 
277     if(getData(data, keyword))
278     {
279         cout <<"      "<<"INTER ECHO TIME"<<":                   ";
280     data >> dEchoTime;
281         if(!data) errorMsg(keyword);
282         errorInDataCheck(data, keyword);
283 
284     }
```

```
285     else
286         dEchoTime = 250.0E-6;
287 
288 }
289 
290 
291 void Input::magnetGradient(double& dMagGradient)
292 {
293     istringstream data;
294     string keyword("MAGNET_GRADIENT");
295 
296     if(getData(data, keyword))
297     {
298         cout <<"      "<<"MAGNETIC GRADIENT"<<":                ";
299     data >> dMagGradient;
300         if(!data) errorMsg(keyword);
301         errorInDataCheck(data, keyword);
302 
303     }
304     else
305         dMagGradient = 25.0;
306 
307 }
```

NMRsim.cpp

```

#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <cassert>
#include <ctime>
#include <set>
#include <vector>
#include <algorithm>
#include <functional>
#include <map>
using namespace std;

#include "threeSome.h"
#include "Input.h"
#include "NMRsim.h"
#include "MersenneTwister.h"

const double    NMRsim::PI           =       acos(-1.0);
const double    NMRsim::SIGN         =       -10.00;
const double    NMRsim::DUMMY         =       1.0E-12;

//////////////////////////
// NMRsim constructor
//////////////////////////

NMRsim::NMRsim(unsigned seed) : m_randSeed(seed)
{
    srand(m_randSeed);
}

//  NMRsim::NMRsim(const NMRsim& NMRsim) : m_randSeed(NMRsim.m_randSeed)
NMRsim::NMRsim(const NMRsim& NMRsim) : m_randSeed(NMRsim.m_randSeed)
{
    m_numPores = NMRsim.m_numPores;
    m_numThroats = NMRsim.m_numThroats;
    m_randSeed = 11222;
    m_numInletThroats = 0;
    m_numOutletThroats = 0;

    m_xSize = NMRsim.m_xSize;
    m_ySize = NMRsim.m_ySize;
    m_zSize = NMRsim.m_zSize;

}

void NMRsim::readData(const string& inputFileName)
{
    string keyword, previousKeyword("NO_KEYWORD_READ");
    char buffer[512];
    ifstream in(inputFileName.c_str());
    if (!in)
    {
```

```
57          cerr << "================================================"       <<
    endl
58               << "Error: Unable to open input file " << inputFileName     <<
    endl
59               << "================================================"       <<
    endl;
60          exit( -1 );
61      }
62
63      while(in >> keyword)
64      {
65          if(keyword[0] == '%')
66          {
67              in.ignore(512, '\n');
68          }
69          else if(keyword.size() < 5 || keyword.size() > 15)
70          {
71              cerr << endl
72                   << "========================================"    << endl
73                   << "Data file contains errors after keyword:"    << endl
74                   << "========================================"    << endl;
75              exit( -1 );
76          }
77          else
78          {
79              in.ignore(512, '\n');
80              string dataString;
81              while(true)
82              {
83                  in.getline(buffer, 512, '\n');
84                  string bufferStr(buffer);
85                  removeComments(bufferStr);
86                  dataString += bufferStr + "  ";
87                  if(terminatorFound(dataString)) break;
88              }
89              pair< string, string > dataEntry(keyword, dataString);
90              m_parsedData.push_back(dataEntry);
91              previousKeyword = keyword;
92          }
93      }
94
95      in.close();
96      m_baseFileName = inputFileName.substr(0, inputFileName.find('.'));
97  }
98
99  void NMRsim::removeComments(string& data) const
100 {
101     string::iterator itr;
102     itr = find(data.begin(), data.end(), '%');
103     if(itr != data.end()) data.erase(itr, data.end());
104 }
105
106 bool NMRsim::terminatorFound(string& data) const
107 {
108     string::iterator itr;
109     itr = find(data.begin(), data.end(), '#');
110     if(itr != data.end())
111     {
```

```
112         data.erase(itr, data.end());
113         return true;
114     }
115     return false;
116 }
117
118 void NMRsim::init(Input& input)
119 {
120     input.title(m_simFileName);
121     cout<<m_simFileName<<endl;
122   cout<<endl;
123
124   string space, resSum;
125   resSum = m_simFileName+"_Results.prt";
126   m_resultPrt.open(resSum.c_str());
127
128   m_resultPrt<<"      "<<"
      ==========================================================================="<<endl
      ;
129   m_resultPrt<<endl;
130     m_resultPrt<<"      "<<" NMR RESPONSE SIMULATION CODE FOR SINGLE-PHASE
      FLUID IN NETWORKS "<<endl;
131     m_resultPrt<<endl;
132     m_resultPrt<<"      "<<"
      ==========================================================================="<<endl
      ;
133     m_resultPrt<<endl;
134   m_resultPrt<<endl;
135
136   m_resultPrt<<"      "<<"OUTPUT TITLE"<<":                          ";
137   m_resultPrt<<m_simFileName<<endl;
138   m_resultPrt<<endl;
139
140   input.walkers(m_numWalkers);
141   cout<<m_numWalkers<<endl;
142   cout<<endl;
143     m_resultPrt<<"      "<<"NUMBER OF WALKERS"<<":                  ";
144   m_resultPrt<<m_numWalkers<<endl;
145   m_resultPrt<<endl;
146
147     input.timeStep(m_timeStep);
148   cout<<m_timeStep<<"s"<<endl;
149   cout<<endl;
150     m_resultPrt<<"      "<<"UNIT TIME STEP"<<":                       ";
151   m_resultPrt<<m_timeStep<<"s"<<endl;
152   m_resultPrt<<endl;
153
154   input.diffConst(m_diffConstant);
155   cout<<m_diffConstant<<"m^2/s"<<endl;
156   cout<<endl;
157     m_resultPrt<<"      "<<"DIFF. COEFFICIENT "<<":                 ";
158   m_resultPrt<<m_diffConstant<<"m^2/s"<<endl;
159   m_resultPrt<<endl;
160
161   input.relaxivity(m_relaxivity);
162   cout<<m_relaxivity<<"m/s"<<endl;
163   cout<<endl;
164     m_resultPrt<<"      "<<"SURFACE RELAXIVITY "<<":                ";
```

```
  m_resultPrt<<m_relaxivity<<"m/s"<<endl;
  m_resultPrt<<endl;

  input.bulkRelax (m_bulkRelaxivity);
  cout<<m_bulkRelaxivity<<"s"<<endl;
  cout<<endl;
    m_resultPrt<<"      "<<"BULK RELAXIVITY "<<":                  ";
  m_resultPrt<<m_bulkRelaxivity<<"s"<<endl;
  m_resultPrt<<endl;

  input.timeData (m_numTimeData);
  cout<<m_numTimeData<<endl;
  cout<<endl;
    m_resultPrt<<"      "<<"DATA POINTS FOR T2-INVERSION"<<":     ";
  m_resultPrt<<m_numTimeData<<endl;
  m_resultPrt<<endl;

  input.decayCutOff(m_decayCutOff);
  cout<<m_decayCutOff<<"s"<<endl;
  cout<<endl;
    m_resultPrt<<"      "<<"NMR CUT OFF - T2 INVERSION"<<":       ";
  m_resultPrt<<m_decayCutOff<<"s"<<endl;
  m_resultPrt<<endl;


  input.interEchoTime(m_interEchoTime);
    cout<<m_interEchoTime<<"s"<<endl;
  cout<<endl;
    m_resultPrt<<"      "<<"INTER ECHO TIME"<<":                   ";
  m_resultPrt<<m_interEchoTime<<"s"<<endl;
  m_resultPrt<<endl;


    input.magnetGradient(m_magneticGradient);
    cout<<m_magneticGradient<<" Gauss/cm"<<endl;
  cout<<endl;
    m_resultPrt<<"      "<<"MAGNETIC GRADIENT"<<":                  ";
  m_resultPrt<<m_magneticGradient<<" Gauss/cm"<<endl;
  m_resultPrt<<endl;


  input.reportTime(m_reportTime);
  cout<<(m_reportTime*m_timeStep)<<"s"<<endl;
  cout<<endl;
  m_resultPrt<<"      "<<"NMR RESULTS TIME INTERVAL"<<":        ";
  m_resultPrt<<(m_reportTime*m_timeStep)<<"s"<<endl;
  m_resultPrt<<endl;


}

void NMRsim::network (int& numPores, int& numThroats, double& xDim, double&
    yDim, double& zDim)
{
    istringstream data;
    string netFileBase;
    string keyword("NETWORK");

```

```
222     char binFile;
223     if(getData(data, keyword))
224     {
225    //    cout <<"      "<< "Reading " << keyword << endl;
226          cout <<"      "<<"LOADED "<< keyword<<":                    ";
227     data >> binFile >> netFileBase;
228     //  data >> netFileBase;
229       cout<<netFileBase<<endl;
230     cout<<endl;
231
232          m_resultPrt<<"      "<<"LOADED "<< keyword<<":                    ";
233       m_resultPrt<<netFileBase<<endl;
234       m_resultPrt<<endl;
235
236
237
238
239     m_binaryFiles = (binFile == 'T' || binFile == 't');
240         if(!data) errorMsg(keyword);
241         errorInDataCheck(data, keyword);
242     }
243     else
244         missingDataErr(keyword);
245
246     if(m_binaryFiles)
247     {
248         string porePropFile(netFileBase + "_node.bin");
249         m_poreProp.open(porePropFile.c_str(), ios::binary);
250
251         string throatPropFile(netFileBase + "_link.bin");
252         m_throatProp.open(throatPropFile.c_str(), ios::binary);
253
254         m_poreProp.read((char *)(&numPores), sizeof(int));
255         m_poreProp.read((char *)(&xDim), sizeof(double));
256         m_poreProp.read((char *)(&yDim), sizeof(double));
257         m_poreProp.read((char *)(&zDim), sizeof(double));
258         m_throatProp.read((char *)(&numThroats), sizeof(int));
259     }
260     else
261     {
262           string poreConnFile(netFileBase + "_node1.dat");          // Open
    file containing pore connection data
263           m_poreConn.open(poreConnFile.c_str());
264
265           string porePropFile(netFileBase + "_node2.dat");          // Open
    file containing pore geometry data
266           m_poreProp.open(porePropFile.c_str());
267
268           string throatConnFile(netFileBase + "_link1.dat");        // Open
    file containing throat connection data
269           m_throatConn.open(throatConnFile.c_str());
270
271           string throatPropFile(netFileBase + "_link2.dat");        // Open
    file containing throat geometry data
272           m_throatProp.open(throatPropFile.c_str());
273
274           m_poreConn >> numPores >> xDim >> yDim >> zDim;
275           m_throatConn >> numThroats;
```

```
276
277     }
278           m_origNumPores = numPores;
279           m_origNumThroats = numThroats;
280           m_origXDim = xDim;
281           m_origYDim = yDim;
282           m_origZDim = zDim;
283
284
285    if (!m_poreProp || !m_throatProp || (!m_binaryFiles && (!m_poreConn || !
       m_throatConn)))
286     {
287         cerr << "=======================================" << endl
288             << "Error: Unable to open network data files" << endl
289             << "=======================================" << endl;
290         exit( -1 );
291     }
292
293
294         loadPoreData();
295           loadThroatData();
296
297 }
298
299 void NMRsim::loadPoreData()
300 {
301   m_poreData1.resize(m_origNumPores);
302     for(int i = 0; i < m_origNumPores; ++i)
303     {
304         if ((!m_binaryFiles && !m_poreConn) || !m_poreProp)
305         {
306             cerr << "================================" << endl
307                 << "Error while reading network data." << endl
308                 << "================================" << endl;
309             exit( -1 );
310         }
311
312         PoreStruct *poreProp = new PoreStruct;
313         int *connThroats, *connPores;
314
315         if(m_binaryFiles)
316         {
317             m_poreProp.read((char *)(poreProp), sizeof(*poreProp));
318
319             connThroats = new int[poreProp->connNum];
320             connPores = new int[poreProp->connNum];
321             m_poreProp.read((char *)(connPores), poreProp->connNum*sizeof(
       int));
322             m_poreProp.read((char *)(connThroats), poreProp->connNum*sizeof
       (int));
323         }
324         else
325         {
326             int idx;
327             bool isAtInletRes, isAtOutletRes;
328
329             m_poreProp >> poreProp->index
330                 >> poreProp->volume
```

```
                >> poreProp->radius
                >> poreProp->shapeFact
                >> poreProp->clayVol;

            m_poreConn >> idx
                >> poreProp->x
                >> poreProp->y
                >> poreProp->z
                >> poreProp->connNum;

            assert(idx == poreProp->index);

            connThroats = new int[poreProp->connNum];
            connPores = new int[poreProp->connNum];

            for(int k = 0; k < poreProp->connNum; ++k)
                m_poreConn >> connPores[k];

            m_poreConn >> isAtInletRes >> isAtOutletRes;

            for(int j = 0; j < poreProp->connNum; ++j)
                m_poreConn >> connThroats[j];
        }
        ThreeSome< PoreStruct*, int*, int* > elem(poreProp, connPores,
    connThroats);
        m_poreData1[i] = elem;

    }

    m_poreProp.close();
    m_poreConn.close();
}

void NMRsim::loadThroatData()
{

    m_throatData1.resize(m_origNumThroats);
    for(int i = 0; i < m_origNumThroats; ++i)
    {
        if ((!m_binaryFiles && !m_throatConn) || !m_throatProp)
        {
            cerr << "================================" << endl
                << "Error while reading network data." << endl
                << "================================" << endl;
            exit( -1 );
        }

        ThroatStruct *throatProp = new ThroatStruct;

        if(m_binaryFiles)
        {
            m_throatProp.read((char *)(throatProp), sizeof(*throatProp));
        }
        else
        {
            int idx, tmp;

            m_throatConn >> throatProp->index
```

```
                >> throatProp->poreOne
                >> throatProp->poreTwo
                >> throatProp->radius
                >> throatProp->shapeFact
                >> throatProp->lenTot;

            m_throatProp >> idx
                >> tmp
                >> tmp
                >> throatProp->lenPoreOne
                >> throatProp->lenPoreTwo
                >> throatProp->lenThroat
                >> throatProp->volume
                >> throatProp->clayVol;

            assert(idx == throatProp->index);

            if(throatProp->poreOne == -1 || throatProp->poreTwo == -1) ++
    m_numInletThroats;
            if(throatProp->poreOne == 0 || throatProp->poreTwo == 0) ++
    m_numOutletThroats;
        }
        m_throatData1[i] = throatProp;

    }

  m_InOutThroats = m_numInletThroats + m_numOutletThroats;
    m_throatConn.close();
    m_throatProp.close();

}


void NMRsim::createFile()
{   string space;
  space = ",";

  double lengthh, radiuss, volumee, shapeFactt;
  int pore1, pore2, counter(0), thrtIdx, pOne, pTwo, cordN, later;

  later = m_origNumThroats-m_InOutThroats;

//    m_throatData.resize(later);
  for(int i = 0; i < m_origNumThroats; ++i)
  { if (m_throatData1[i]->poreOne!=0 && m_throatData1[i]->poreOne!=-1
       && m_throatData1[i]->poreTwo!=0 && m_throatData1[i]->poreTwo!=-1)
  { counter+=1;
   int j = counter - 1;
   radiuss = m_throatData1[i]->radius;
     volumee = m_throatData1[i]->volume;
     shapeFactt = m_throatData1[i]->shapeFact;
   lengthh = (4.0*volumee*shapeFactt)/(radiuss*radiuss);
   thrtIdx = counter;
   pOne = m_throatData1[i]->poreOne;
   pTwo = m_throatData1[i]->poreTwo;

    {ThreeSome<int, int, int > TNT(thrtIdx, pOne, pTwo);
     throatHold.push_back(TNT);}
```

```
444
445    if (shapeFactt>(sqrt(3.0)/36)&& shapeFactt<0.0553)
446    shapeFactt = (sqrt(3.0)/36);
447    else if (shapeFactt>=0.0553)
448    shapeFactt = 0.0625;
449
450      ThroatStruct1 *throatProp1 = new ThroatStruct1;
451
452    throatProp1->index = counter;
453      throatProp1->poreOne = m_throatData1[i]->poreOne;
454      throatProp1->poreTwo = m_throatData1[i]->poreTwo;
455      throatProp1->radius = radiuss;
456    throatProp1->length = lengthh;
457      throatProp1->volume = volumee;
458      throatProp1->shapeFact = shapeFactt;
459
460    m_throatData.push_back(throatProp1);
461
462  //     m_throatData[i-m_InOutThroats] = throatProp1;
463   }
464   }
465
466
467 //   for(int i = 0; i<throatHold.size(); ++i)
468 //   cout<<throatHold[i].first()<<space<<throatHold[i].second()<<space<<
469 //     throatHold[i].third()<<endl;
470
471     m_poreData.resize(m_origNumPores);
472
473   for(int i = 0; i < m_origNumPores; ++i)
474 {
475     vector <double> poreList;
476     vector <double> throatList;
477
478   counter = m_poreData1[i].first()->index;
479   radiuss = m_poreData1[i].first()->radius;
480   volumee = m_poreData1[i].first()->volume;
481   shapeFactt = m_poreData1[i].first()->shapeFact;
482     lengthh = (4.0*volumee*shapeFactt)/(radiuss*radiuss);
483
484   if (shapeFactt>(sqrt(3.0)/36)&& shapeFactt<0.0553)
485   shapeFactt = (sqrt(3.0)/36);
486   else if (shapeFactt>=0.0553)
487   shapeFactt = 0.0625;
488
489     for(int j = 0; j<throatHold.size(); ++j)
490  {
491   if (counter == throatHold[j].second())
492   { poreList.push_back(throatHold[j].third());
493     throatList.push_back(throatHold[j].first()); }
494
495   if (counter == throatHold[j].third())
496   { poreList.push_back(throatHold[j].second());
497       throatList.push_back(throatHold[j].first()); }
498
499   }
500
501
```

```
502     PoreStruct1 *poreProp1 = new PoreStruct1;
503     int *connThroats;
504 
505     poreProp1->index = counter;
506     poreProp1->radius = radiuss;
507     poreProp1->length = lengthh;
508     poreProp1->volume = volumee;
509     poreProp1->shapeFact = shapeFactt;
510     poreProp1->connNum = throatList.size();
511 
512     connThroats = new int[throatList.size()];
513 
514     for(int m=0; m<throatList.size() ; ++m)
515     connThroats[m] = throatList[m];
516 
517     TwoSome< PoreStruct1*, int* > elem(poreProp1, connThroats);
518     m_poreData[i] = elem;
519 
520     throatList.clear();
521     poreList.clear();
522 
523 }
524 
525 
526 }
527 
528 
529 void NMRsim::dataFileOut()
530 
531 {
532     ofstream outPore;
533     outPore.open("outputPore.csv");
534     ofstream outThroat;
535     outThroat.open("outputThroat.csv");
536 
537     string space;
538   space = ",";
539 
540   for (int i=0; i<m_throatData.size(); ++i)
541   outThroat<<m_throatData[i]->index<<space<<m_throatData[i]->poreOne<<space
      <<
542     m_throatData[i]->poreTwo<<space<<m_throatData[i]->radius<<space<<
      m_throatData[i]->length<<
543   space<<m_throatData[i]->volume<<space<<m_throatData[i]->shapeFact<<endl;
544 
545     for (int i=0; i<m_poreData.size(); ++i)
546 { outPore<<m_poreData[i].first()->index<<space<<m_poreData[i].first()->
      radius<<space<<
547     m_poreData[i].first()->length<<space<<m_poreData[i].first()->volume<<
      space<<
548     m_poreData[i].first()->shapeFact<<space<<m_poreData[i].first()->connNum
      ;
549     for (int j=0; j<m_poreData[i].first()->connNum; ++j)
550   outPore<<space<<m_poreData[i].second()[j];
551     outPore<<endl;   }
552 
553 }
554 
```

```
555 void NMRsim::writeDecay()
556
557 {
558     ofstream outDecay;
559     outDecay.open("Magnet_100.csv");
560
561 //  ofstream outMag;
562 //  outMag.open("M.csv");
563
564   string outFileM;
565
566     outFileM = m_simFileName+"_M.csv";
567     m_resultM.open(outFileM.c_str());
568
569
570
571   string space;
572   space = ",";
573   int compData, counter(0);
574   double compDt;
575   compDt = amplitudeDecay.size()/m_numTimeData;
576   compData = static_cast < double > (compDt);
577   vector <double> magnetDecay;
578
579
580   for (int i=0; i<amplitudeDecay.size() ; ++i)
581    {if (i%compData==0)
582    { counter+=1;
583   if (counter<=m_numTimeData)
584    {magnetDecay.push_back(amplitudeDecay[i].second());
585   m_timeEcho.push_back(amplitudeDecay[i].first()*1000);       // echo Time
     in milliseconds (ms)
586   m_resultM <<amplitudeDecay[i].second()<<endl;
587   outDecay <<amplitudeDecay[i].first()<<space<<amplitudeDecay[i].second()<<
     endl;
588     m_resultPrt<<"      "<<fixed<<setprecision(5)<<amplitudeDecay[i].first
     ()
589   <<"                 "<<amplitudeDecay[i].second()<<endl; }}}
590
591 //  outMag.close();
592
593     m_resultPrt<<endl;
594    m_resultPrt<<endl;
595
596     m_resultPrt<<"      TOTAL AREA SIDE 1:                    "<<scientific
    <<m_area1T<<" m^2"<<endl;
597     m_resultPrt<<"      TOTAL AREA SIDE 2                     "<<m_area2T<<
    " m^2"<<endl;
598    m_resultPrt<<"      TOTAL AREA SIDE 3                    "<<m_area3T<<"
    m^2"<<endl;
599    m_resultPrt<<"      TOTAL SQUARE AREA                    "<<m_areaSqT<<"
     m^2"<<endl;
600
601     m_resultPrt<<endl;
602    m_resultPrt<<endl;
603
604     m_resultPrt<<"      TOTAL FRACTION SIDE 1:                "<<fixed<<
    setprecision(4)<<m_areaFract1T<<endl;
```

```
605    m_resultPrt<<"      TOTAL FRACTION SIDE 2:                "<<
       m_areaFract2T<<endl;
606    m_resultPrt<<"      TOTAL FRACTION SIDE 3:                "<<
       m_areaFract3T<<endl;
607    m_resultPrt<<"      TOTAL FRACTION SQUARE AREA:           "<<
       m_areaFractSqT<<endl;
608
609      m_resultPrt<<endl;
610    m_resultPrt<<endl;
611
612    m_resultPrt<<"      FRACTION WALKERS HITTING SIDE 1:      "<<
       m_fractHit1T<<endl;
613    m_resultPrt<<"      FRACTION WALKERS HITTING SIDE 2:      "<<
       m_fractHit2T<<endl;
614    m_resultPrt<<"      FRACTION WALKERS HITTING SIDE 3:      "<<
       m_fractHit3T<<endl;
615    m_resultPrt<<"      FRACTION WALKERS HITTING SQUARE:      "<<
       m_fractHitSqT<<endl;
616
617      m_resultM.close();
618    outDecay.close();
619
620 }
621
622 void NMRsim::kernelMatrix()
623 {
624     ofstream outKM;
625     outKM.open("T2.csv");
626
627  // ofstream outKK;
628  // outKK.open("K.csv");
629
630   string outFileK;
631
632     outFileK = m_simFileName+"_K.csv";
633     m_resultK.open(outFileK.c_str());
634
635
636   string space;
637   space = ",";
638   double initTime (-2.41887), timeInc(0.1162886), value;
639
640   vector <double> exponentialTime;
641
642   for (int i=0; i<100; ++i)
643    {initTime+=timeInc;
644   exponentialTime.push_back(exp(initTime));}
645
646     for (int i=0; i<100; ++i)
647   outKM << exponentialTime[i]<<endl;
648
649     for (int i=0; i<m_timeEcho.size() ; ++i)
650    {for (int k=0; k<exponentialTime.size(); ++k)
651    {value = exp(-(m_timeEcho[i]/exponentialTime[k]));
652   if (value<1.0E-300)
653   value = 0.0;
654     m_resultK<<value<<space;}
655     m_resultK<<endl;}
```

```
656
657     m_resultK.close();
658     outKM.close();
659
660 }
661
662 void NMRsim::weightedMatrix()
663 {
664     ofstream outWeight;
665     outWeight.open("Wm.csv");
666
667   string space;
668   space = ",";
669
670   int WmSize(100);
671   int marker(-1), count(0), count1(0);
672
673   vector<int> first;
674     vector<int> second;
675     vector<int> third;
676     vector<int> write;
677     vector<int> first1;
678     vector<int> second1;
679
680   first.push_back(5);
681     first.push_back(-4);
682     first.push_back(1);
683
684     second.push_back(-4);
685     second.push_back(6);
686     second.push_back(-4);
687     second.push_back(1);
688
689     third.push_back(1);
690   third.push_back(-4);
691   third.push_back(6);
692   third.push_back(-4);
693   third.push_back(1);
694
695   for (int i=0; i<WmSize; ++i)
696     write.push_back(0);
697
698   for (int i=0; i<first.size(); ++i)
699     first1.push_back(first[first.size()-i-1]);
700
701     for (int i=0; i<second.size(); ++i)
702     second1.push_back(second[second.size()-i-1]);
703
704    {for (int j=0; j<first.size(); ++j)
705     outWeight<<first[j]<<space;
706     for (int j=0; j<(WmSize-first.size()); ++j)
707   outWeight<<0<<space;}
708     outWeight<<endl;
709
710    {for (int j=0; j<second.size(); ++j)
711     outWeight<<second[j]<<space;
712     for (int j=0; j<(WmSize-second.size()); ++j)
713   outWeight<<0<<space;}
```

```
714     outWeight<<endl;
715
716    {for (int j=0; j<third.size(); ++j)
717     outWeight<<third[j]<<space;
718     for (int j=0; j<(WmSize-third.size()); ++j)
719   outWeight<<0<<space;}
720     outWeight<<endl;
721
722     while (count<(WmSize-5))
723    {for (int j=-1; j<count; ++j)
724     outWeight<<0<<space;
725     for (int j=0; j<third.size(); ++j)
726     outWeight<<third[j]<<space;
727     for (int j=0; j<(WmSize-count-third.size()-1); ++j)
728     outWeight<<0<<space;
729   outWeight<<endl;
730   count+=1;}
731
732    {for (int j=0; j<(WmSize-second.size()); ++j)
733     outWeight<<0<<space;
734     for (int j=0; j<second1.size(); ++j)
735     outWeight<<second1[j]<<space;}
736     outWeight<<endl;
737
738    {for (int j=0; j<(WmSize-first.size()); ++j)
739     outWeight<<0<<space;
740     for (int j=0; j<first1.size(); ++j)
741     outWeight<<first1[j]<<space;}
742
743 }
744
745 double NMRsim::calcPorosity()
746 {
747   double fluidVolume(0), porosity(0);
748   m_bulkVolume = m_origXDim*m_origYDim*m_origZDim;
749   for(int i=0; i<m_origNumPores; ++i)
750     fluidVolume+=(m_poreData[i].first()->radius*m_poreData[i].first()->
     radius*m_poreData[i].first()->length)/
751   (4*m_poreData[i].first()->shapeFact);
752   for(int i=0; i<m_throatData.size(); ++i)
753   fluidVolume+=(m_throatData[i]->radius*m_throatData[i]->radius*
     m_throatData[i]->length)/
754   (4*m_throatData[i]->shapeFact);
755   porosity = fluidVolume / m_bulkVolume;
756     m_walkVolume = fluidVolume / m_numWalkers;
757   return porosity;
758
759 }
760
761 double NMRsim::calcAvrgCordNum()
762 {
763   double cordNumVal(0.0), cordNumAvrg(0.0);
764   for(int i=0; i<m_origNumPores; ++i)
765     cordNumVal+= m_poreData[i].first()->connNum;
766   cordNumAvrg = cordNumVal/m_origNumPores;
767   return cordNumAvrg;
768 }
769
```

```
770
771 void NMRsim::initWalkers()
772 {
773   double beta2Min, beta2Max, beta1, beta2, beta3, height1, m_shapeFact,
      diffHeight;
774   double base1, base2, base, B1, B2, B3, height2, height, area, m_radius,
      areaSq, areaSqT(0);
775   int m_index, walkersPore, walkersThroat;
776     double xPos, yPos, zPos, xPosOpp, BASE1, BASE2, BASE, area1(0), area2
      (0), area3(0);
777   double area1T(0), area2T(0), area3T(0),xBdry, yBdry, zBdry, areaEnd(0),
      areaE(0);
778
779   //ofstream out;
780     out.open("output.dat");
781
782     m_porosity = calcPorosity();
783
784   MTRand mtrand1;
785     string space, resWalk, resSum;
786   space = ",";
787 //  m_poreData.resize(m_origNumPores);
788 //  m_throatData.resize(m_origNumThroats);
789
790
791     for(int i = 0; i < m_origNumPores; ++i)
792 {   int count(0);
793     m_shapeFact = m_poreData[i].first()->shapeFact;
794     if (m_shapeFact > 0.0481)
795   m_shapeFact = 0.0625;
796
797
798   m_index = m_poreData[i].first()->index;
799   m_radius = m_poreData[i].first()->radius;
800     char poreSign('P');
801
802   if (m_shapeFact<0.0625)
803 {   beta2Min = atan((2.0/sqrt(3.0))*cos(acos(-12.0*sqrt(3.0)*m_shapeFact)
      /3.0+4.0*PI/3.0));
804     beta2Max = atan((2.0/sqrt(3.0))*cos(acos(-12.0*sqrt(3.0)*m_shapeFact)
      /3.0));
805
806   beta2 = beta2Min + (beta2Max - beta2Min)*mtrand1();
807 //  beta2 = beta2Min + (beta2Max - beta2Min)*0.001;     // Used for
      debugging purposes
808     beta1 = -0.5*beta2 + 0.5*asin((tan(beta2)+4.0*m_shapeFact)*sin(beta2)/(
      tan(beta2)-4.0*m_shapeFact));
809     beta3 = PI/2.0 - beta2 - beta1;
810
811     base1 = m_radius/tan(beta1);
812   base2 = m_radius/tan(beta2);
813   base = base1+base2;
814
815   B3 = base;
816   B2 = B3*sin(beta2*2)/sin(beta3*2);
817   B1 = B3*sin(beta1*2)/sin(beta3*2);
818   height1 = B2*sin(beta1*2);
819     height2 = B1*sin(beta2*2);
```

```
820
821  assert(abs(height1-height2)<DUMMY);
822
823  height = (height1+height2)/2;
824  area = (base*height)/2;
825
826  BASE1 = height/tan(beta1*2);
827  BASE2 = height/tan(beta2*2);
828  BASE = BASE1 + BASE2;
829
830  area1 = B1*m_poreData[i].first()->length;
831  area2 = B2*m_poreData[i].first()->length;
832  area3 = B3*m_poreData[i].first()->length;
833
834  if (m_poreData[i].first()->connNum==0)
835   {areaE = base*height;
836  areaEnd+=areaE;}
837
838  area1T+=area1;
839  area2T+=area2;
840  area3T+=area3;
841
842    assert(abs(base-BASE)<DUMMY);
843
844   { FiveSome<char,int,double,double,double> geometry(poreSign, m_index,(
     beta1*2),(beta2*2),height );
845   m_elemGeometry.push_back(geometry);}
846
847   walkersPore = static_cast < double > (m_radius*m_radius*m_poreData[i].
     first()->length*1  // olumide
848    /(4*m_shapeFact*m_walkVolume));  // olumide
849
850
851    if (walkersPore==0)
852    walkersPore = 1;
853
854    if (m_walkData)
855    m_walkout<<m_poreData[i].first()->index<<space<<walkersPore<<endl;
856
857  while (count<walkersPore)
858  {  xPos =  mtrand1()*base;
859     yPos =  mtrand1()*height;
860       zPos =  mtrand1()*m_poreData[i].first()->length;
861     assert (m_poreData[i].first()->index == m_index);
862
863      if (xPos<BASE1)
864     {if (atan(yPos/xPos)<(beta1*2))
865    {FiveSome<char, int, double, double, double>initPos(poreSign, m_index,
     xPos, yPos, zPos);
866     m_initPosition1.push_back(initPos);
867     count+=1;}}
868    else
869       {xPosOpp = base - xPos;
870       if (atan(yPos/xPosOpp)<(beta2*2))
871         {FiveSome<char, int, double, double, double>initPos(poreSign,
     m_index, xPos, yPos, zPos);
872     m_initPosition1.push_back(initPos);
873     count+=1;}}
```

```
874   } }
875   else
876    {
877      xBdry = 2*m_radius;
878    yBdry = 2*m_radius;
879    zBdry = m_poreData[i].first()->length;
880    areaSq = 2*(xBdry+yBdry)*zBdry;
881    areaSqT+=areaSq;
882    {FiveSome<char,int,double,double,double> geometry(poreSign, m_index,
       m_shapeFact, yBdry, zBdry);
883       m_elemGeometry.push_back(geometry);}
884
885    walkersPore = static_cast < double > (m_radius*m_radius*m_poreData[i].
       first()->length*1  // olumide
886     /(4*m_shapeFact*m_walkVolume));  // olumide
887
888     if (walkersPore==0)
889     walkersPore = 1;
890
891     while (count<walkersPore)
892   {  xPos =  mtrand1()*xBdry;
893      yPos =  mtrand1()*yBdry;
894        zPos =  mtrand1()*zBdry;
895      assert (m_poreData[i].first()->index == m_index);
896       {FiveSome<char, int, double, double, double>initPos(poreSign, m_index
       , xPos, yPos, zPos);
897      m_initPosition1.push_back(initPos);
898      count+=1;}
899 }}}
900
901
902     for(int i = 0; i < m_throatData.size(); ++i)
903 {    int count(0);
904   m_shapeFact = m_throatData[i]->shapeFact;
905     if (m_shapeFact > 0.0481)
906   m_shapeFact = 0.0625;
907 //  else if (m_shapeFact < 0.001)
908 //  m_shapeFact = 0.001;
909
910   m_index = m_throatData[i]->index;
911   m_radius = m_throatData[i]->radius;
912     char throatSign('T');
913
914     if (m_shapeFact<0.0625)
915   { beta2Min = atan((2.0/sqrt(3.0))*cos(acos(-12.0*sqrt(3.0)*m_shapeFact)
       /3.0+4.0*PI/3.0));
916     beta2Max = atan((2.0/sqrt(3.0))*cos(acos(-12.0*sqrt(3.0)*m_shapeFact)
       /3.0));
917
918   beta2 = beta2Min + (beta2Max - beta2Min)*mtrand1();
919 //  beta2 = beta2Min + (beta2Max - beta2Min)*0.001;   // Used for debugging
       purposes
920     beta1 = -0.5*beta2 + 0.5*asin((tan(beta2)+4.0*m_shapeFact)*sin(beta2)/(
       tan(beta2)-4.0*m_shapeFact));
921     beta3 = PI/2.0 - beta2 - beta1;
922
923     base1 = m_radius/tan(beta1);
924   base2 = m_radius/tan(beta2);
```

```
925 base = base1+base2;
926
927 B3 = base;
928 B2 = B3*sin(beta2*2)/sin(beta3*2);
929 B1 = B3*sin(beta1*2)/sin(beta3*2);
930 height1 = B2*sin(beta1*2);
931   height2 = B1*sin(beta2*2);
932 diffHeight = abs (height1-height2)/height1*100;
933 height = (height1+height2)/2;
934 area = (base*height)/2;
935
936   BASE1 = height/tan(beta1*2);
937 BASE2 = height/tan(beta2*2);
938
939   area1 = B1*m_throatData[i]->length;
940 area2 = B2*m_throatData[i]->length;
941 area3 = B3*m_throatData[i]->length;
942
943 area1T+=area1;
944 area2T+=area2;
945 area3T+=area3;
946
947
948 { FiveSome<char,int,double,double,double> geometry(throatSign, m_index,(
    beta1*2),(beta2*2),height );
949 m_elemGeometry.push_back(geometry);}
950
951   walkersThroat = static_cast < double > (m_radius*m_radius*m_throatData[
    i]->length*1   // Olumide
952 /(4*m_shapeFact*m_walkVolume));   // olumide
953
954 if (walkersThroat==0)
955 walkersThroat = 1;
956
957 if (m_walkData)
958 m_walkout<<m_throatData[i]->index<<space<<walkersThroat<<endl;
959
960      while (count<walkersThroat)
961
962     { xPos =  mtrand1()*base;
963   yPos =  mtrand1()*height;
964       zPos =  mtrand1()*m_throatData[i]->length;
965       assert (m_throatData[i]->index == m_index);
966
967        if (xPos<BASE1)
968       {if (atan(yPos/xPos)<(beta1*2))
969   {FiveSome<char, int, double, double, double>initPos(throatSign, m_index
    , xPos, yPos, zPos);
970    m_initPosition1.push_back(initPos);
971    count+=1;}}
972   else
973     {xPosOpp = base - xPos;
974     if (atan(yPos/xPosOpp)<(beta2*2))
975       {FiveSome<char, int, double, double, double>initPos(throatSign,
    m_index, xPos, yPos, zPos);
976    m_initPosition1.push_back(initPos);
977    count+=1;}}}}
978
```

```
  else
    {
     xBdry = 2*m_radius;
   yBdry = 2*m_radius;
   zBdry = m_throatData[i]->length;
     areaSq = 2*(xBdry+yBdry)*zBdry;
   areaSqT+=areaSq;

   {FiveSome<char,int,double,double,double> geometry(throatSign, m_index,
   m_shapeFact, yBdry, zBdry);
      m_elemGeometry.push_back(geometry);}
      walkersThroat = static_cast < double > (m_radius*m_radius*
   m_throatData[i]->length*1   // Olumide
  /(4*m_shapeFact*m_walkVolume));   // olumide

      if (walkersThroat==0)
      walkersThroat = 1;

    while (count<walkersThroat)
  {  xPos =  mtrand1()*xBdry;
     yPos =  mtrand1()*yBdry;
       zPos =  mtrand1()*zBdry;
     assert (m_throatData[i]->index == m_index);
      {FiveSome<char, int, double, double, double>initPos(throatSign,
   m_index, xPos, yPos, zPos);
     m_initPosition1.push_back(initPos);
     count+=1;}
}}

}

    m_areaFract1T = (area1T+areaEnd)/(area1T+area2T+area3T+areaSqT+areaEnd)
   ;
    m_areaFract2T = area2T/(area1T+area2T+area3T+areaSqT+areaEnd);
    m_areaFract3T = area3T/(area1T+area2T+area3T+areaSqT+areaEnd);
    m_areaFractSqT = areaSqT/(area1T+area2T+area3T+areaSqT+areaEnd);

    m_area1T = area1T+areaEnd;
    m_area2T = area2T;
    m_area3T = area3T;
    m_areaSqT = areaSqT;




    out<<"Surface Area   1      "<<area1T+areaEnd<<endl;
    out<<"Surface Area   2      "<<area2T<<endl;
    out<<"Surface Area   3      "<<area3T<<endl;
    out<<"Surface Area   SQ     "<<areaSqT<<endl;
    out<<"TOTAL SURFACE AREA    "<<area1T+area2T+area3T+areaEnd<<endl;
    out<<"FRACTION AREA  1      "<<area1T+areaEnd/(area1T+area2T+area3T+
   areaSqT+areaEnd)<<endl;
    out<<"FRACTION AREA  2      "<<area2T/(area1T+area2T+area3T+areaSqT+
   areaEnd)<<endl;
    out<<"FRACTION AREA  3      "<<area3T/(area1T+area2T+area3T+areaSqT+
   areaEnd)<<endl;
    out<<"FRACTION AREA  SQ     "<<areaSqT/(area1T+area2T+area3T+areaSqT+
   areaEnd)<<endl;
```

```
1029
1030
1031
1032 }
1033
1034 void NMRsim::randomMotion()
1035 {
1036     srand(time(0));
1037     MTRand mtrand1;
1038
1039   double distanceDr, randPhi, randTheta, angleTheta, anglePhi, base, BASE1,
        BASE2, hit1(0);
1040   double xPos, yPos, zPos, walkersAlive, setProb, beta1, beta2, height,
       hit2(0), hit3(0) ;
1041     double bulkFact(0),cosPhi, init, xBound, yBound, zBound, xPosOpp, hitSq
       (0), diffusionDecay;;
1042   int probab1 (1), probab2, deathProb, countThroat2Pore(0),
       countPore2Throat(0), elemIdxNumNew;
1043   int elemIdxNum, countX(0), countY(0), countZ(0), countZpos(0), countZneg
       (0);
1044   int walkersTot, total(0), cordNum, cordNum1, cordNum2, randCord1,
       randCord2;
1045   int countNew1 (0), countNew2 (0), timer(0), setElemIdxNum, hit4(0),
       refPropIdx1 ;
1046   int timeCounter(0), positive(0), negative(0), refGeomIdx, refPropIdx;
1047   int randThrtEnd1, randThrtEnd2;
1048   char elementID;
1049
1050
1051   string space, resNmr, resInitPos, resSum;
1052   space = ",";
1053     resNmr = m_simFileName+"_nmr.csv";
1054   m_nmrResponse.open(resNmr.c_str());
1055
1056     initWalkers();
1057   walkersTot = m_initPosition1.size();
1058     init = static_cast< int > (m_initPosition1.size());
1059
1060     distanceDr = sqrt(6.0*m_diffConstant*m_timeStep);
1061     setProb = (2.0*m_relaxivity*distanceDr)/(3.0*m_diffConstant);
1062     deathProb = static_cast< double > (1 / setProb);
1063
1064     diffusionDecay = pow((2*PI*4258), 2)*m_magneticGradient*
       m_magneticGradient*10000*
1065                 m_interEchoTime*m_interEchoTime*m_diffConstant; // from
       (2*PI*gyro)^2*Do*G^2*EchoT^2
1066
1067     m_porosity = calcPorosity();
1068     m_avrgConnNum = calcAvrgCordNum();
1069
1070
1071   cout<<"      "<<"NETWORK POROSITY:                "<<m_porosity*100<<"%"
       <<endl;
1072   cout<<endl;
1073     cout<<"      "<<"ASSIGNED WALKERS:                  "<<init<<endl;
1074     cout<<endl;
1075   m_resultPrt<<"      "<<"NETWORK POROSITY:                "<<m_porosity
       *100<<"%"<<endl;
```

```
1076     m_resultPrt<<endl;
1077     m_resultPrt<<"      "<<"ASSIGNED WALKERS:                  "<<init<<endl
         ;
1078     m_resultPrt<<endl;
1079
1080   cout<<endl;
1081   cout<<endl;
1082     cout<<"      **********************"<<endl;
1083   cout<<"            SINGLE - PHASE   "<<endl;
1084     cout<<"      **********************"<<endl;
1085     cout<<endl;
1086     cout<<endl;
1087   cout<<endl;
1088     m_resultPrt<<endl;
1089     m_resultPrt<<endl;
1090     m_resultPrt<<endl;
1091     m_resultPrt<<endl;
1092
1093     m_resultPrt<<"      **********************"<<endl;
1094     m_resultPrt<<"           SINGLE - PHASE    "<<endl;
1095     m_resultPrt<<"      **********************"<<endl;
1096     m_resultPrt<<endl;
1097
1098
1099   cout<<"      ================================="<<endl;
1100     cout<<"      "<<" TIME (s)    "<<"          [Mt/Mo] "<<endl;
1101   cout<<"      ================================="<<endl;
1102
1103     cout<<"      "<<fixed<<setprecision(5)<<0.000<<"                  "
       <<1.000<<endl;
1104
1105   m_resultPrt<<"      ================================="<<endl;
1106     m_resultPrt<<"      "<<" TIME (s)    "<<"            [Mt/Mo] "<<endl;
1107   m_resultPrt<<"      ================================="<<endl;
1108
1109     m_resultPrt<<"      "<<fixed<<setprecision(5)<<0.000<<"
       "<<1.000<<endl;
1110
1111
1112
1113
1114 {  TwoSome< double, double > decay(0.0, 1.0);
1115    amplitudeDecay.push_back(decay); }
1116
1117
1118    m_nmrResponse<<"SURFACE"<<space<<space<<space<<"SUR+BULK"<<space<<space
       <<space<<"SUR+BULK+MAG"<<endl;
1119    m_nmrResponse<<"TIME"<<space<<"M(t)/Mo"<<space<<space<<"TIME"<<space<<"M
       (t)/Mo"<<space<<space<<"TIME"<<space<<"M(t)/Mo"<<endl;
1120    m_nmrResponse<<0<<space<<1<<space<<space<<0<<space<<1<<space<<space<<0<<
       space<<1<<endl;
1121
1122   while (!m_initPosition1.empty()&& (timer*m_timeStep)<m_decayCutOff)
1123
1124 {
1125   timer+=1;
1126
1127     for (int i=0; i<m_initPosition1.size(); ++i)
```

```
                        /// STEP 2
1128
1129      {
1130         randPhi =  mtrand1();
1131       randTheta =  mtrand1();
1132     cosPhi = 1 - 2*randPhi;
1133     anglePhi = acos(cosPhi);
1134       angleTheta = randTheta*2.0*PI;
1135
1136
1137         xPos = m_initPosition1[i].third() + (distanceDr*cos(angleTheta)*
    sin(anglePhi));          /// STEP 1
1138       yPos = m_initPosition1[i].fourth() + (distanceDr*sin(angleTheta)*sin
    (anglePhi));
1139         zPos = m_initPosition1[i].fifth() + (distanceDr*cosPhi);
1140
1141       elementID = m_initPosition1[i].first();
1142       elemIdxNum = m_initPosition1[i].second();
1143         refPropIdx = elemIdxNum - 1;
1144     refPropIdx1 = elemIdxNum - 1;
1145
1146     if (elementID == 'P')
1147     refGeomIdx = elemIdxNum - 1;
1148     else if (elementID == 'T')
1149     refGeomIdx = elemIdxNum + m_origNumPores - 1;
1150
1151
1152          if (elementID=='P' && m_elemGeometry[refGeomIdx].first()=='P')
                        /// STEP 4
1153
1154       {      if (m_elemGeometry[refGeomIdx].third()!= 0.0625)
1155
1156    {     assert (m_elemGeometry[refGeomIdx].second() == (refGeomIdx+1));
1157         BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].third());
1158       BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1159       base = BASE1+BASE2;
1160
1161         xBound =  BASE1+BASE2;
1162       yBound =  m_elemGeometry[refGeomIdx].fifth();
1163       zBound =  m_poreData[refPropIdx].first()->length;
1164
1165       beta1 = m_elemGeometry[refGeomIdx].third();
1166       beta2 = m_elemGeometry[refGeomIdx].fourth();
1167       height = m_elemGeometry[refGeomIdx].fifth();
1168
1169
1170       if (zPos>zBound ||zPos<0.0)
                          /// STEP 12
1171
1172        {if (m_poreData[refPropIdx].first()->connNum >1)
                          /// STEP 16
1173      {assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1174       cordNum = m_poreData[refPropIdx].first()->connNum;
1175       vector<int> connThroats;
1176       vector<double> connThrtRadii;
1177       vector<int>otherThrts;
```

```
1178        bool signalThrt1(false), signalThrt2(false);
1179        for (int ai = 0; ai < cordNum; ++ai)
                        /// STEP 21
1180       {connThroats.push_back(m_poreData[refPropIdx].second()[ai]);
1181        otherThrts.push_back(m_poreData[refPropIdx].second()[ai]);
1182        connThrtRadii.push_back(m_throatData[m_poreData[refPropIdx].second()
    [ai]-1]
1183        ->radius);}
1184        sort (connThrtRadii.begin(), connThrtRadii.end());
1185              double cordNumRad1 = connThrtRadii.back();
1186              connThrtRadii.pop_back();
1187        double cordNumRad2 = connThrtRadii.back();
1188
1189        for (int bi = 0; bi < cordNum; ++bi)
1190        {if (m_throatData[connThroats[bi]-1]->radius==cordNumRad1 && !
    signalThrt1)
1191        {randCord1 = connThroats[bi];
1192         signalThrt1 = true;}
1193         else if (m_throatData[connThroats[bi]-1]->radius==cordNumRad2 && !
    signalThrt2)
1194        {randCord2 = connThroats[bi];
1195               signalThrt2 = true;}
1196        //      else if ((m_throatData[connThroats[bi]-1]->radius!=
    cordNumRad1)
1197          //   && (m_throatData[connThroats[bi]-1]->radius!=cordNumRad2)&&
    signalThrt1 && signalThrt2)
1198        //      otherThrts.push_back(connThroats[bi]);
1199          }
1200
1201              int XcordNum1;
1202        XcordNum1 = static_cast< double > (otherThrts.size()/2);
1203        int XcordNum2 = otherThrts.size() - XcordNum1;
1204        int detThroat, XrandCord1, XrandCord2;
1205
1206
1207
1208       //       cout<<"WWWWWWWWWWWWWWWWWW                    "<<cordNumRad1
    <<endl;
1209
1210
1211        if (zPos<0.0)
                          /// STEP 23
1212         {detThroat = mtrand1.randInt(1);
1213          if (detThroat == 0 || cordNum<3)
1214              elemIdxNumNew = randCord1-1;
1215        else if (detThroat>0 && cordNum>=3)
1216        { XrandCord1  = mtrand1.randInt(XcordNum1-1);
1217  //      cout<<"GGGGGGG  "<<otherThrts.size()<<"        "<<cordNum<<"
      "<<XcordNum1<<endl;
1218        elemIdxNumNew = otherThrts[XrandCord1]-1;}
1219              else
1220        cout<<"this is ERRORRRRRRRRRR                "<<endl;
1221
1222          elementID = 'T';
1223        refGeomIdx = elemIdxNumNew + m_origNumPores;
1224              elemIdxNum = elemIdxNumNew + 1;
1225              refPropIdx = elemIdxNum - 1;
1226
```

```
1227      if (m_elemGeometry[refGeomIdx].third()!= 0.0625)  // diffusion
    Triangular-Triangular      /// STEP 27
1228
1229      {assert (m_elemGeometry[refGeomIdx].first()=='T');
                        /// STEP 29
1230            assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1231            BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].third());
1232       BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1233       beta1 = m_elemGeometry[refGeomIdx].third();
1234       beta2 = m_elemGeometry[refGeomIdx].fourth();
1235            height = m_elemGeometry[refGeomIdx].fifth();
1236       base = BASE1 + BASE2;
1237
1238      int acct(0);
1239      while (acct<1)
1240      {xPos =  mtrand1()*base;
1241         yPos =  mtrand1()*height;
1242       zPos = m_throatData[refPropIdx]->length - abs(zPos);
1243            if (xPos<BASE1)
1244            {if (atan(yPos/xPos)<beta1)
1245       acct+=1;}
1246      else
1247      {xPosOpp = base - xPos;
1248            if (atan(yPos/xPosOpp)<beta2)
1249      acct+=1;}}
1250          countNew1 = countNew1 + 1;}
1251
1252      else                  // diffusion from Triangle to Square
                            /// STEP 28
1253      {
1254        assert (m_elemGeometry[refGeomIdx].first()=='T');
1255            assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1256            xBound =  m_elemGeometry[refGeomIdx].fourth();
1257      yBound =  m_elemGeometry[refGeomIdx].fourth();
1258
1259      xPos =  mtrand1()*xBound;
1260        yPos =  mtrand1()*yBound;
1261      zPos = m_throatData[refPropIdx]->length - abs(zPos);
1262        assert (m_throatData[refPropIdx]->length==m_elemGeometry[refGeomIdx
    ].fifth());}
1263
1264      }
1265
1266         if (zPos>zBound)
                            /// STEP 23
1267             {detThroat = mtrand1.randInt(1);
1268         if (detThroat == 0 || cordNum<3)
1269             elemIdxNumNew = randCord2-1;
1270       else if (detThroat>0 && cordNum>=3)
1271       { XrandCord2  = mtrand1.randInt(XcordNum2-1);
1272       elemIdxNumNew = otherThrts[XrandCord2+XcordNum1]-1;}
1273       else
1274       cout<<"this is ERRORRRRRRRRRR                "<<endl;
1275
1276                                          /// STEP 24
1277        setElemIdxNum = elemIdxNum;
```

```
1278        elementID = 'T';
1279              elemIdxNum = elemIdxNumNew + 1;
1280              refGeomIdx = elemIdxNumNew + m_origNumPores;
1281              refPropIdx = elemIdxNum - 1;
1282
1283  //      cout<<"this is ERRORRRRRRRRRR               "<<detThroat<<endl;
1284
1285              if (m_elemGeometry[refGeomIdx].third()!= 0.0625) // diffusion
     Triangular-Triangular    /// STEP 27
1286     {    assert (m_elemGeometry[refGeomIdx].first()=='T');
                          /// STEP 29
1287              assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1288
1289              BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry
     [refGeomIdx].third());
1290         BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
     refGeomIdx].fourth());
1291         beta1 = m_elemGeometry[refGeomIdx].third();
1292         beta2 = m_elemGeometry[refGeomIdx].fourth();
1293              height = m_elemGeometry[refGeomIdx].fifth();
1294         base = BASE1 + BASE2;
1295
1296         int acct(0);
1297         while (acct<1)
1298        {xPos =  mtrand1()*base;
1299           yPos =  mtrand1()*height;
1300         zPos = zPos - zBound; ;
1301              if (xPos<BASE1)
1302              {if (atan(yPos/xPos)<beta1)
1303         acct+=1;}
1304         else
1305        {xPosOpp = base - xPos;
1306              if (atan(yPos/xPosOpp)<beta2)
1307        acct+=1;}}}
1308
1309      else          //           Diffusion Triangular to square
                          /// STEP 28
1310     {    assert (m_elemGeometry[refGeomIdx].first()=='T');
1311           assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1312           xBound =  m_elemGeometry[refGeomIdx].fourth();
1313      yBound =  m_elemGeometry[refGeomIdx].fourth();
1314
1315      xPos =  mtrand1()*xBound;
1316        yPos =  mtrand1()*yBound;
1317      zPos =  zPos - zBound;
1318          assert (m_throatData[refPropIdx]->length==m_elemGeometry[
     refGeomIdx].fifth());}
1319
1320      } }
1321
1322
1323     else if (m_poreData[refPropIdx].first()->connNum == 0)
                          /// STEP 17
1324         {probab2 = mtrand1.randInt(deathProb-1);
                             /// STEP 22
1325          if (probab1==probab2)
                        /// STEP 19
1326               {// hit3 = hit3 + 1;      added commented out
```

```
1327                xPos = SIGN;
                                /// STEP 25
1328            yPos = SIGN;
1329            zPos = SIGN;}
1330            else
1331          {xPos = m_initPosition1[i].third();
                        /// STEP 30
1332               yPos = m_initPosition1[i].fourth();
1333         zPos = m_initPosition1[i].fifth();} }
1334
1335
1336
1337     else    if (m_poreData[refPropIdx].first()->connNum == 1)
                           /// STEP 18
1338
1339       { vector<int> connThroats;
1340            connThroats.push_back(m_poreData[refPropIdx].second()[0]);
1341            setElemIdxNum = elemIdxNum;
1342       elemIdxNumNew = connThroats[0]-1;
1343            elemIdxNum = elemIdxNumNew + 1 ;
1344            elementID = 'T';
1345            refGeomIdx = elemIdxNumNew + m_origNumPores;
1346            refPropIdx = elemIdxNum - 1;
1347
1348            if (m_elemGeometry[refGeomIdx].third()!= 0.0625) // diffusion
    Triangular-Triangular    /// STEP 27
1349
1350     {  assert (m_elemGeometry[refGeomIdx].first()=='T');
                        /// STEP 29
1351            assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1352            BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].third());
1353            BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1354         beta1 = m_elemGeometry[refGeomIdx].third();
1355       beta2 = m_elemGeometry[refGeomIdx].fourth();
1356            height = m_elemGeometry[refGeomIdx].fifth();
1357            base = BASE1 + BASE2;
1358
1359       int acct(0);
1360       while (acct<1)
1361      {xPos =  mtrand1()*base;
1362         yPos =  mtrand1()*height;
1363       zPos = mtrand1()*m_throatData[refPropIdx]->length;
1364
1365            if (xPos<BASE1)
1366            {if (atan(yPos/xPos)<beta1)
1367      acct+=1;}
1368      else
1369      {xPosOpp = base - xPos;
1370            if (atan(yPos/xPosOpp)<beta2)
1371      acct+=1;}}}
1372
1373      else        //           Diffusion Triangular to square
                        /// STEP 28
1374
1375     {    assert (m_elemGeometry[refGeomIdx].first()=='T');
1376           assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
```

```
1377            xBound =  m_elemGeometry[refGeomIdx].fourth();
1378       yBound =  m_elemGeometry[refGeomIdx].fourth();
1379       zBound =  m_elemGeometry[refGeomIdx].fifth();
1380
1381       xPos =  mtrand1()*xBound;
1382         yPos =  mtrand1()*yBound;
1383       zPos =  mtrand1()*yBound;
1384           assert (m_throatData[refPropIdx]->length==m_elemGeometry[
     refGeomIdx].fifth());}
1385
1386        } }
1387
1388         else   if ( xPos>xBound || (xPos>0 && xPos<(height/tan(beta1)) &&
     /// SMALLEST SIDE      /// STEP 13
1389           yPos>0 && yPos<yBound && atan(yPos/xPos)>beta1))
1390            {probab2 = mtrand1.randInt(deathProb-1);
1391          if (probab1==probab2)
                        /// STEP 19
1392              {hit2 = hit2 + 1;
                              /// STEP 25
1393               xPos = SIGN;
1394            yPos = SIGN;
1395            zPos = SIGN;}
1396            else
1397         {xPos = m_initPosition1[i].third();
                        /// STEP 30
1398             yPos = m_initPosition1[i].fourth();
1399       zPos = m_initPosition1[i].fifth();} }
1400
1401
1402         else if (xPos<0 || (xPos>(height/tan(beta1)) && yPos>0 &&  // 2nd
     LONGEST SIDE           /// STEP 13
1403         yPos<yBound && atan(yPos/(base- xPos))>beta2 ))
1404            {probab2 = mtrand1.randInt(deathProb-1);
1405          if (probab1==probab2)
                        /// STEP 19
1406              {hit1 = hit1 + 1;
                              /// STEP 25
1407               xPos = SIGN;
1408            yPos = SIGN;
1409            zPos = SIGN;}
1410            else
1411         {xPos = m_initPosition1[i].third();
                        /// STEP 30
1412             yPos = m_initPosition1[i].fourth();
1413       zPos = m_initPosition1[i].fifth();} }
1414
1415
1416
1417     else   if ( yPos<0 ||yPos>yBound)  //    LONGEST SIDE
                        /// STEP 13
1418            {probab2 = mtrand1.randInt(deathProb-1);
1419          if (probab1==probab2)
                        /// STEP 19
1420              {hit3 = hit3 + 1;
                              /// STEP 25
1421               xPos = SIGN;
1422            yPos = SIGN;
```

```
1423            zPos = SIGN;}
1424            else
1425         {xPos = m_initPosition1[i].third();
                    /// STEP 30
1426             yPos = m_initPosition1[i].fourth();
1427         zPos = m_initPosition1[i].fifth();} }
1428
1429 }
1430
1431
1432
1433       else if (m_elemGeometry[refGeomIdx].third()== 0.0625)          //
    marker
1434
1435        {    assert (m_elemGeometry[refGeomIdx].second() == (refGeomIdx+1))
    ;
1436         xBound =  m_elemGeometry[refGeomIdx].fourth();
1437       yBound =  m_elemGeometry[refGeomIdx].fourth();
1438       zBound =  m_elemGeometry[refGeomIdx].fifth();
1439
1440
1441       if (zPos>zBound ||zPos<0.0)
                        /// STEP 12
1442
1443        {if (m_poreData[refPropIdx].first()->connNum >1)
                          /// STEP 16
1444      {assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1445       cordNum = m_poreData[refPropIdx].first()->connNum;
1446       vector<int> connThroats;
1447       vector<double> connThrtRadii;
1448       vector<int>otherThrts;
1449       bool signalThrt1(false), signalThrt2(false);
1450       for (int ai = 0; ai < cordNum; ++ai)
                      /// STEP 21
1451      {connThroats.push_back(m_poreData[refPropIdx].second()[ai]);
1452       otherThrts.push_back(m_poreData[refPropIdx].second()[ai]);
1453       connThrtRadii.push_back(m_throatData[m_poreData[refPropIdx].second()
    [ai]-1]
1454       ->radius);}
1455       sort (connThrtRadii.begin(), connThrtRadii.end());
1456             double cordNumRad1 = connThrtRadii.back();
1457             connThrtRadii.pop_back();
1458       double cordNumRad2 = connThrtRadii.back();
1459
1460       for (int bi = 0; bi < cordNum; ++bi)
1461       {if (m_throatData[connThroats[bi]-1]->radius==cordNumRad1 && !
    signalThrt1)
1462       {randCord1 = connThroats[bi];
1463        signalThrt1 = true;}
1464        else if (m_throatData[connThroats[bi]-1]->radius==cordNumRad2 && !
    signalThrt2)
1465       {randCord2 = connThroats[bi];
1466              signalThrt2 = true;}
1467       //     else if ((m_throatData[connThroats[bi]-1]->radius!=
    cordNumRad1)
1468     //   && (m_throatData[connThroats[bi]-1]->radius!=cordNumRad2)&&
    signalThrt1 && signalThrt2)
1469       //     otherThrts.push_back(connThroats[bi]);
```

```
1470          }
1471
1472               int XcordNum1;
1473        XcordNum1 = static_cast< double > (otherThrts.size()/2);
1474        int XcordNum2 = otherThrts.size() - XcordNum1;
1475        int detThroat, XrandCord1, XrandCord2;
1476
1477        if (zPos<0.0)
                          /// STEP 23
1478         {
1479        detThroat = mtrand1.randInt(1);
1480          if (detThroat == 0 || cordNum<3)
1481               elemIdxNumNew = randCord1-1;
1482        else if (detThroat>0 && cordNum>=3)
1483        { XrandCord1  = mtrand1.randInt(XcordNum1-1);
1484  //      cout<<"GGGGGGG  "<<otherThrts.size()<<"        "<<cordNum<<"
    "<<XcordNum1<<endl;
1485        elemIdxNumNew = otherThrts[XrandCord1]-1;}
1486               else
1487        cout<<"this is ERRORRRRRRRRRR               "<<endl;
1488
1489          elementID = 'T';
1490        refGeomIdx = elemIdxNumNew + m_origNumPores;
1491               elemIdxNum = elemIdxNumNew + 1;
1492               refPropIdx = elemIdxNum - 1;
1493
1494
1495           if (m_elemGeometry[refGeomIdx].third()!= 0.0625) // diffusion
    Square-Triangular          /// STEP 27
1496
1497           { assert (m_elemGeometry[refGeomIdx].first()=='T');
                           /// STEP 29
1498             assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1499             BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].third());
1500             BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1501          beta1 = m_elemGeometry[refGeomIdx].third();
1502        beta2 = m_elemGeometry[refGeomIdx].fourth();
1503               height = m_elemGeometry[refGeomIdx].fifth();
1504               base = BASE1 + BASE2;
1505
1506        int acct(0);
1507        while (acct<1)
1508       {xPos =  mtrand1()*base;
1509          yPos =  mtrand1()*height;
1510        zPos =  m_throatData[refPropIdx]->length - abs(zPos);
1511
1512               if (xPos<BASE1)
1513               {if (atan(yPos/xPos)<beta1)
1514        acct+=1;}
1515        else
1516        {xPosOpp = base - xPos;
1517               if (atan(yPos/xPosOpp)<beta2)
1518        acct+=1;}}}
1519
1520        else          // diffusion Square-Square
                          /// STEP 28
```

```
1521
1522    {   assert (m_elemGeometry[refGeomIdx].first()=='T');
1523            assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1524            xBound =  m_elemGeometry[refGeomIdx].fourth();
1525        yBound =  m_elemGeometry[refGeomIdx].fourth();
1526
1527      xPos =  mtrand1()*xBound;
1528        yPos =  mtrand1()*yBound;
1529      zPos =  m_throatData[refPropIdx]->length - abs(zPos);
1530        assert (m_throatData[refPropIdx]->length==m_elemGeometry[refGeomIdx
    ].fifth());}
1531
1532            }
1533
1534         if (zPos>zBound)
                              /// STEP 23
1535           {
1536       detThroat = mtrand1.randInt(1);
1537         if (detThroat == 0 || cordNum<3)
1538             elemIdxNumNew = randCord2-1;
1539       else if (detThroat>0 && cordNum>=3)
1540       { XrandCord2  = mtrand1.randInt(XcordNum2-1);
1541       elemIdxNumNew = otherThrts[XrandCord2+XcordNum1]-1;}
1542       else
1543       cout<<"this is ERRORRRRRRRRRR                "<<endl;
1544
1545                                              /// STEP 24
1546        setElemIdxNum = elemIdxNum;
1547        elementID = 'T';
1548              elemIdxNum = elemIdxNumNew + 1;
1549              refGeomIdx = elemIdxNumNew + m_origNumPores;
1550              refPropIdx = elemIdxNum - 1;
1551
1552
1553              if (m_elemGeometry[refGeomIdx].third()!= 0.0625) // diffusion
     Square-Triangular       /// STEP 27
1554
1555     {   assert (m_elemGeometry[refGeomIdx].first()=='T');
                         /// STEP 29
1556             assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1557             BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].third());
1558             BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1559          beta1 = m_elemGeometry[refGeomIdx].third();
1560       beta2 = m_elemGeometry[refGeomIdx].fourth();
1561             height = m_elemGeometry[refGeomIdx].fifth();
1562             base = BASE1 + BASE2;
1563
1564       int acct(0);
1565       while (acct<1)
1566      {xPos =  mtrand1()*base;
1567        yPos =  mtrand1()*height;
1568       zPos =  zPos - zBound;
1569
1570             if (xPos<BASE1)
1571             {if (atan(yPos/xPos)<beta1)
1572       acct+=1;}
```

```
1573        else
1574        {xPosOpp = base - xPos;
1575              if (atan(yPos/xPosOpp)<beta2)
1576        acct+=1;}}}
1577
1578       else                    // diffusion Square-Square
                                 /// STEP 28
1579
1580     {    assert (m_elemGeometry[refGeomIdx].first()=='T');
1581             assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1582             xBound =  m_elemGeometry[refGeomIdx].fourth();
1583          yBound =  m_elemGeometry[refGeomIdx].fourth();
1584
1585       xPos =  mtrand1()*xBound;
1586         yPos =  mtrand1()*yBound;
1587       zPos =  zPos - zBound;
1588         assert (m_throatData[refPropIdx]->length==m_elemGeometry[refGeomIdx
    ].fifth());}
1589
1590             } }
1591
1592
1593
1594          else if (m_poreData[refPropIdx].first()->connNum == 0)
                                 /// STEP 17
1595         {probab2 = mtrand1.randInt(deathProb-1);
                                 /// STEP 22
1596          if (probab1==probab2)
                               /// STEP 19
1597               {//hit3 = hit3 + 1;
1598                xPos = SIGN;
                                 /// STEP 25
1599            yPos = SIGN;
1600            zPos = SIGN;}
1601            else
1602         {xPos = m_initPosition1[i].third();
                              /// STEP 30
1603              yPos = m_initPosition1[i].fourth();
1604        zPos = m_initPosition1[i].fifth();} }
1605
1606
1607
1608     else    if (m_poreData[refPropIdx].first()->connNum == 1)
                               /// STEP 18
1609
1610          {vector<int> connThroats;
1611              connThroats.push_back(m_poreData[refPropIdx].second()[0]);
1612              setElemIdxNum = elemIdxNum;
1613       elemIdxNumNew = connThroats[0]-1;
1614              elemIdxNum = elemIdxNumNew + 1 ;
1615              elementID = 'T';
1616              refGeomIdx = elemIdxNumNew + m_origNumPores;
1617              refPropIdx = elemIdxNum - 1;
1618
1619              if (m_elemGeometry[refGeomIdx].third()!= 0.0625) // diffusion
    Square-Triangular        /// STEP 27
1620
1621          {    assert (m_elemGeometry[refGeomIdx].first()=='T');
```

```
                              /// STEP 29
1622            assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1623            BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].third());
1624            BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1625          beta1 = m_elemGeometry[refGeomIdx].third();
1626        beta2 = m_elemGeometry[refGeomIdx].fourth();
1627            height = m_elemGeometry[refGeomIdx].fifth();
1628            base = BASE1 + BASE2;
1629
1630        int acct(0);
1631        while (acct<1)
1632       {xPos =  mtrand1()*base;
1633          yPos =  mtrand1()*height;
1634        zPos = mtrand1()*m_throatData[refPropIdx]->length;
1635
1636            if (xPos<BASE1)
1637            {if (atan(yPos/xPos)<beta1)
1638        acct+=1;}
1639        else
1640        {xPosOpp = base - xPos;
1641            if (atan(yPos/xPosOpp)<beta2)
1642        acct+=1;}}}
1643
1644            else           // diffusion Square-Square
                              /// STEP 28
1645
1646         {   assert (m_elemGeometry[refGeomIdx].first()=='T');
1647            assert (m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1648            xBound =  m_elemGeometry[refGeomIdx].fourth();
1649         yBound =  m_elemGeometry[refGeomIdx].fourth();
1650
1651       xPos =  mtrand1()*xBound;
1652         yPos =  mtrand1()*yBound;
1653       zPos =  mtrand1()*m_throatData[refPropIdx]->length;
1654            assert (m_throatData[refPropIdx]->length==m_elemGeometry[
    refGeomIdx].fifth());
1655       }}
1656 }
1657
1658
1659     else   if (xPos>xBound||xPos<0.0||yPos>yBound||yPos<0)
                              /// STEP 13
1660              {probab2 = mtrand1.randInt(deathProb-1);
1661          if (probab1==probab2)
                             /// STEP 19
1662              {hitSq += 1;
1663               xPos = SIGN;
                              /// STEP 25
1664           yPos = SIGN;
1665           zPos = SIGN;}
1666           else
1667         {xPos = m_initPosition1[i].third();
                             /// STEP 30
1668             yPos = m_initPosition1[i].fourth();
1669         zPos = m_initPosition1[i].fifth();} }
1670
```

```
1671     }
1672        }
1673
1674
1675
1676        ///////// THROAT  /////////  THROAT ///////// THROAT ///////// THROAT
          ////////////
1677
1678              else if (elementID=='T'  && m_elemGeometry[refGeomIdx].first()
     =='T')                     /// STEP 3
1679
1680       {       if (m_elemGeometry[refGeomIdx].third()!= 0.0625)
1681
1682       {       assert (m_elemGeometry[refGeomIdx].second() == (refGeomIdx -
     m_origNumPores+1));
1683              BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
     refGeomIdx].third());
1684          BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
     refGeomIdx].fourth());
1685          base = BASE1+BASE2;
1686
1687                xBound =  BASE1+BASE2;
1688          yBound =  m_elemGeometry[refGeomIdx].fifth();
1689          zBound =  m_throatData[refPropIdx]->length;
1690
1691          beta1 = m_elemGeometry[refGeomIdx].third();
1692          beta2 = m_elemGeometry[refGeomIdx].fourth();
1693          height = m_elemGeometry[refGeomIdx].fifth();
1694
1695
1696              if (zPos<0.0)
                                  /// STEP 6
1697               {
1698           elementID = 'P';
                              /// STEP 8
1699             elemIdxNum = (m_throatData[refPropIdx]->poreOne);
1700           refPropIdx = elemIdxNum - 1;
1701                 refGeomIdx = elemIdxNum - 1;
1702             assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1703
1704               if (m_elemGeometry[refGeomIdx].third()!= 0.0625)  // diffusion
     Triangular-Triangular    /// STEP 9
1705
1706
1707         {     assert(m_elemGeometry[refGeomIdx].first()=='P');
                              /// STEP 11
1708             assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1709                     BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(
     m_elemGeometry[refGeomIdx].third());
1710               BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
     refGeomIdx].fourth());
1711             beta1 = m_elemGeometry[refGeomIdx].third();
1712               beta2 = m_elemGeometry[refGeomIdx].fourth();
1713                     height = m_elemGeometry[refGeomIdx].fifth();
1714             base = BASE1+BASE2;
1715
1716             int acct(0);
1717             while (acct<1)
```

```
1718          {xPos =  mtrand1()*base;
1719             yPos =  mtrand1()*height;
1720          zPos = m_poreData[refPropIdx].first()->length - abs(zPos);
1721                 if (xPos<BASE1)
1722                  {if (atan(yPos/xPos)<beta1)
1723           acct+=1;}
1724          else
1725          {xPosOpp = base - xPos;
1726                   if (atan(yPos/xPosOpp)<beta2)
1727             acct+=1;}}}
1728
1729      else              // diffusion Triangular-Square
                          /// STEP 10
1730
1731    {     assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1732              assert(m_elemGeometry[refGeomIdx].first()=='P');
1733        assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1734              xBound =  m_elemGeometry[refGeomIdx].fourth();
1735        yBound =  m_elemGeometry[refGeomIdx].fourth();
1736
1737        xPos =  mtrand1()*xBound;
1738          yPos =  mtrand1()*yBound;
1739        zPos = m_poreData[refPropIdx].first()->length - abs(zPos);
1740    }
1741
1742
1743
1744        countThroat2Pore = countThroat2Pore + 1;}
1745
1746             else  if (zPos>zBound)
                              /// STEP 5
1747             {elementID = 'P';
                              /// STEP 7
1748             setElemIdxNum = elemIdxNum;
1749         elemIdxNum = (m_throatData[refPropIdx]->poreTwo);
1750             refPropIdx = elemIdxNum - 1;
1751             refGeomIdx = elemIdxNum - 1;
1752             assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1753
1754           if (m_elemGeometry[refGeomIdx].third()!= 0.0625)  // diffusion
    Triangular-Triangular    /// STEP 9
1755
1756        {
1757          assert(m_elemGeometry[refGeomIdx].first()=='P');
                    /// STEP 11
1758          assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1759                  BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(
    m_elemGeometry[refGeomIdx].third());
1760           BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1761          beta1 = m_elemGeometry[refGeomIdx].third();
1762            beta2 = m_elemGeometry[refGeomIdx].fourth();
1763                  height = m_elemGeometry[refGeomIdx].fifth();
1764          base = BASE1+BASE2;
1765
1766          int acct(0);
1767          while (acct<1)
1768           {xPos =  mtrand1()*base;
```

```
1769              yPos =  mtrand1()*height;
1770                  zPos = zPos - zBound;
1771
1772           if (xPos<BASE1)
1773                  {if (atan(yPos/xPos)<beta1)
1774           acct+=1;}
1775          else
1776          {xPosOpp = base - xPos;
1777                   if (atan(yPos/xPosOpp)<beta2)
1778             acct+=1;}}}
1779
1780      else                  // diffusion Triangular-Square
                             /// STEP 10
1781
1782        {      assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1783               assert(m_elemGeometry[refGeomIdx].first()=='P');
1784        assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1785               xBound =  m_elemGeometry[refGeomIdx].fourth();
1786        yBound =  m_elemGeometry[refGeomIdx].fourth();
1787
1788        xPos =  mtrand1()*xBound;
1789          yPos =  mtrand1()*yBound;
1790        zPos =  zPos - zBound; }
1791
1792
1793
1794       countThroat2Pore = countThroat2Pore + 1;}
1795
1796
1797             else   if ( xPos>xBound || (xPos>0 && xPos<(height/tan(beta1))
     && // SMALLEST SIDE     /// STEP 13
1798           yPos>0 && yPos<yBound && atan(yPos/xPos)>beta1))
1799            {probab2 = mtrand1.randInt(deathProb-1);
1800          if (probab1==probab2)
                             /// STEP 19
1801               {hit2 = hit2 + 1;
1802                xPos = SIGN;
                                 /// STEP 25
1803            yPos = SIGN;
1804            zPos = SIGN;}
1805            else
1806         {xPos = m_initPosition1[i].third();
                             /// STEP 30
1807              yPos = m_initPosition1[i].fourth();
1808        zPos = m_initPosition1[i].fifth();} }
1809
1810
1811             else if (xPos<0 || (xPos>(height/tan(beta1)) && yPos>0 &&  //2
    nd LONGEST SIDE          /// STEP 13
1812       yPos<yBound && atan(yPos/(base- xPos))>beta2 ))
1813             {probab2 = mtrand1.randInt(deathProb-1);
1814           if (probab1==probab2)
                             /// STEP 19
1815               {hit1 = hit1 + 1;
1816                xPos = SIGN;
                                 /// STEP 25
1817            yPos = SIGN;
1818            zPos = SIGN;}
```

```
1819            else
1820         {xPos = m_initPosition1[i].third();
                    /// STEP 30
1821              yPos = m_initPosition1[i].fourth();
1822         zPos = m_initPosition1[i].fifth();} }
1823
1824
1825       else   if ( yPos<0 ||yPos>yBound)  // THE LONGEST SIDE
                      /// STEP 13
1826           {probab2 = mtrand1.randInt(deathProb-1);
1827          if (probab1==probab2)
                    /// STEP 19
1828              {hit3 = hit3 + 1;
1829               xPos = SIGN;
                           /// STEP 25
1830           yPos = SIGN;
1831           zPos = SIGN;}
1832           else
1833         {xPos = m_initPosition1[i].third();
                    /// STEP 30
1834              yPos = m_initPosition1[i].fourth();
1835         zPos = m_initPosition1[i].fifth();} }
1836 }
1837
1838
1839       else                // this element is square shaped
1840
1841     {       assert (m_elemGeometry[refGeomIdx].second() == (refGeomIdx -
    m_origNumPores+1));
1842            xBound =  m_elemGeometry[refGeomIdx].fourth();
1843       yBound =  m_elemGeometry[refGeomIdx].fourth();
1844       zBound =  m_elemGeometry[refGeomIdx].fifth();
1845
1846
1847           if (zPos<0.0)
                      /// STEP 6
1848           {
1849        elementID = 'P';
                    /// STEP 8
1850          elemIdxNum = (m_throatData[refPropIdx]->poreOne);
1851        refPropIdx = elemIdxNum - 1;
1852              refGeomIdx = elemIdxNum - 1;
1853
1854
1855           if (m_elemGeometry[refGeomIdx].third()!= 0.0625)  // diffusion
    Square-Triangular        /// STEP 9
1856
1857
1858     {     assert(m_elemGeometry[refGeomIdx].first()=='P');
                      /// STEP 11
1859           assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1860               BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry
    [refGeomIdx].third());
1861          BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1862        beta1 = m_elemGeometry[refGeomIdx].third();
1863          beta2 = m_elemGeometry[refGeomIdx].fourth();
1864              height = m_elemGeometry[refGeomIdx].fifth();
```

```
1865        base = BASE1+BASE2;
1866
1867          int acct(0);
1868          while (acct<1)
1869           {xPos =  mtrand1()*base;
1870              yPos =  mtrand1()*height;
1871          zPos = m_poreData[refPropIdx].first()->length - abs(zPos);
1872                 if (xPos<BASE1)
1873                  {if (atan(yPos/xPos)<beta1)
1874           acct+=1;}
1875          else
1876          {xPosOpp = base - xPos;
1877                   if (atan(yPos/xPosOpp)<beta2)
1878             acct+=1;}}}
1879
1880              else                          // diffusion square-Square
                                  /// STEP 10
1881
1882       {  assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
1883              assert(m_elemGeometry[refGeomIdx].first()=='P');
1884        assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1885              xBound =  m_elemGeometry[refGeomIdx].fourth();
1886        yBound =  m_elemGeometry[refGeomIdx].fourth();
1887
1888        xPos =  mtrand1()*xBound;
1889          yPos =  mtrand1()*yBound;
1890        zPos = m_poreData[refPropIdx].first()->length - abs(zPos); }}
1891
1892
1893              else  if (zPos>zBound)
                                  /// STEP 5
1894              {elementID = 'P';
                                  /// STEP 7
1895               setElemIdxNum = elemIdxNum;
1896          elemIdxNum = (m_throatData[refPropIdx]->poreTwo);
1897              refPropIdx = elemIdxNum - 1;
1898              refGeomIdx = elemIdxNum - 1;
1899
1900            if (m_elemGeometry[refGeomIdx].third()!= 0.0625)  // diffusion
     Square-Triangular       /// STEP 9
1901
1902     {      assert(m_elemGeometry[refGeomIdx].first()=='P');
                          /// STEP 11
1903          assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
1904              BASE1 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry
    [refGeomIdx].third());
1905         BASE2 = m_elemGeometry[refGeomIdx].fifth()/tan(m_elemGeometry[
    refGeomIdx].fourth());
1906        beta1 = m_elemGeometry[refGeomIdx].third();
1907          beta2 = m_elemGeometry[refGeomIdx].fourth();
1908              height = m_elemGeometry[refGeomIdx].fifth();
1909        base = BASE1+BASE2;
1910
1911          int acct(0);
1912          while (acct<1)
1913           {xPos =  mtrand1()*base;
1914              yPos =  mtrand1()*height;
1915          zPos =  zPos - zBound;
```

```
                  if (xPos<BASE1)
                   {if (atan(yPos/xPos)<beta1)
           acct+=1;}
          else
          {xPosOpp = base - xPos;
                   if (atan(yPos/xPosOpp)<beta2)
             acct+=1;}}}


       else                  // diffusion square-Square
                          /// STEP 10

      {  assert(m_poreData[refPropIdx].first()->index == elemIdxNum);
            assert(m_elemGeometry[refGeomIdx].first()=='P');
       assert(m_elemGeometry[refGeomIdx].second() == elemIdxNum);
            xBound =  m_elemGeometry[refGeomIdx].fourth();
       yBound =  m_elemGeometry[refGeomIdx].fourth();


       xPos =  mtrand1()*xBound;
         yPos =  mtrand1()*yBound;
       zPos = zPos - zBound;}}



        else  if (xPos>xBound || xPos<0.0|| yPos>yBound ||yPos<0 )
                          /// STEP 13
             {probab2 = mtrand1.randInt(deathProb-1);
         if (probab1==probab2)
                        /// STEP 19
              {hitSq = hitSq + 1;
               xPos = SIGN;
                               /// STEP 25
           yPos = SIGN;
           zPos = SIGN;}
           else
        {xPos = m_initPosition1[i].third();
                       /// STEP 30
             yPos = m_initPosition1[i].fourth();
       zPos = m_initPosition1[i].fifth();} }

  }
  }



           if ((xPos!=SIGN) && (yPos!=SIGN) && (zPos!=SIGN))
                               /// STEP 15
         {FiveSome< char, int, double, double, double > xyMinus(elementID
  ,elemIdxNum,xPos,yPos,zPos);
           m_finalPosition.push_back(xyMinus);}
                               /// STEP 14

}

           walkersAlive = static_cast< int > (m_finalPosition.size());
                               /// STEP 20

```

```
1965
1966            if ((timer%m_reportTime)==0)
1967       {
1968
1969         m_nmrResponse<<timer*m_timeStep<<space<<walkersAlive/init<<space<<
    space<<
1970            timer*m_timeStep<<space<<(walkersAlive/init)*exp(-(timer*
    m_timeStep)/m_bulkRelaxivity)<<
1971         space<<space<<timer*m_timeStep<<space<<(walkersAlive/init)*exp(-(
    timer*m_timeStep)/m_bulkRelaxivity)*
1972         exp(-(diffusionDecay*timer*m_timeStep))<<endl;
1973
1974         cout<<"      "<<timer*m_timeStep<<"                  "<<
1975            (walkersAlive/init)*exp(-(timer*m_timeStep)/m_bulkRelaxivity)*
    exp(-(diffusionDecay*timer*m_timeStep))<<endl;
1976
1977         double ampDecay = (walkersAlive/init)*exp(-(timer*m_timeStep)/
    m_bulkRelaxivity)*exp(-(diffusionDecay*timer*m_timeStep));
1978         double echoTime = timer*m_timeStep;
1979  //       if (ampDecay>m_decayCutOff)
1980         { TwoSome< double, double > decay(echoTime, ampDecay);
1981          amplitudeDecay.push_back(decay);}
1982       }
1983
1984
1985           m_initPosition1.clear();
1986          m_initPosition1 = m_finalPosition;
1987          m_finalPosition.clear();
1988           total = countX+countY+countZ;
1989
1990 }
1991
1992
1993           if (m_resultSum)
1994      {  m_resultSumm <<"Number of Throats        "<<m_origNumThroats<<endl;
1995            m_resultSumm <<"Number of Pores          "<<m_origNumPores<<
    endl;
1996       m_resultSumm <<"Number of Walkers        "<<walkersTot<<endl;
1997            m_resultSumm <<"Porosity                 "<<m_porosity<<endl;
1998            m_resultSumm <<"Average Conn Number      "<<m_avrgConnNum<<endl
    ;
1999            m_resultSumm <<"THROAT 2 PORE            "<<countThroat2Pore<<
    endl;
2000       m_resultSumm <<"PORE 2 THROAT            "<<countPore2Throat<<endl;
2001         m_resultSumm <<"HIT X-AXIS               "<<countX<<endl;
2002       m_resultSumm <<"HIT Y-AXIS               "<<countY<<endl;
2003            m_resultSumm <<"HIT FRACTION 1           "<<hit1/(hit1+hit2+
    hit3+hitSq)<<endl;
2004            m_resultSumm <<"HIT FRACTION 2           "<<hit2/(hit1+hit2+
    hit3+hitSq)<<endl;
2005            m_resultSumm <<"HIT FRACTION 3           "<<hit3/(hit1+hit2+
    hit3+hitSq)<<endl;
2006         m_resultSumm <<"HIT FRACTION SQ          "<<hitSq/(hit1+hit2+hit3+
    hitSq)<<endl;
2007       }
2008
2009          if (m_nmr)
2010         m_nmrResponse.close();
```

```
2011          if (m_initPos)
2012            m_initialPos.close();
2013          if (m_resultSum)
2014          m_resultSumm.close();
2015
2016            m_fractHit1T = hit1/(hit1+hit2+hit3+hitSq);
2017        m_fractHit2T = hit2/(hit1+hit2+hit3+hitSq);
2018            m_fractHit3T = hit3/(hit1+hit2+hit3+hitSq);
2019            m_fractHitSqT = hitSq/(hit1+hit2+hit3+hitSq);
2020
2021
2022
2023        cout <<"HIT FRACTION 1            "<<hit1/(hit1+hit2+hit3+hitSq)<<endl
    ;
2024            cout <<"HIT FRACTION 2            "<<hit2/(hit1+hit2+hit3+hitSq)<<
    endl;
2025            cout <<"HIT FRACTION 3            "<<hit3/(hit1+hit2+hit3+hitSq)<<
    endl;
2026            cout <<"HIT FRACTION SQ           "<<hitSq/(hit1+hit2+hit3+hitSq)
    <<endl;
2027
2028 }
```

Main.cpp

```
1
2  #include <iostream>
3  #include <fstream>
4  #include <sstream>
5  #include <iomanip>
6  #include <cmath>
7  #include <cstdlib>
8  #include <cassert>
9  #include <cstdio>
10 #include <ctime>
11 #include <set>
12 #include <vector>
13 #include <algorithm>
14 #include <functional>
15 #include <map>
16
17 using namespace std;
18
19 #include "threeSome.h"
20 #include "Input.h"
21 #include "NMRsim.h"
22 #include "MersenneTwister.h"
23
24 int main(int argc, char *argv[])
25 {
26     string inputFileName ("default.dat");
27
28   cout<<endl;
29   cout<<"      "<<
     "**********************************************************************"<<endl
     ;
30     cout<<endl;
31   cout<<"      "<<"           PETROLEUM ENGINEERING AND ROCK MECHANICS
     GROUP         "<<endl;
32   cout<<endl;
33   cout<<"      "<<"             DEPARTMENT OF EARTH SCIENCE AND ENGINEERING
               "<<endl;
34     cout<<endl;
35   cout<<"      "<<"                      IMPERIAL COLLEGE LONDON
               "<<endl;
36     cout<<endl;
37   cout<<endl;
38     cout<<endl;
39   cout<<"      "<<
     "======================================================================"<<endl
     ;
40   cout<<endl;
41     cout<<"      "<<"  NMR RESPONSE SIMULATION CODE FOR SINGLE-PHASE FLUID
     IN NETWORKS    "<<endl;
42     cout<<endl;
43     cout<<"      "<<
     "======================================================================"<<endl
     ;
44     cout<<endl;
45   cout<<endl;
46
```

```
47
48      Input input(inputFileName);
49
50      int seedNum;
51      input.randSeed(seedNum);
52
53      string baseFileName;
54    NMRsim NMRsim(seedNum);
55
56    NMRsim.readData(inputFileName);
57
58    int m_numPores, m_numThroats;
59    double  m_xSize, m_ySize, m_zSize;
60
61    NMRsim.init(input);
62    NMRsim.network (m_numPores, m_numThroats, m_xSize, m_ySize, m_zSize);
63    NMRsim.createFile();
64  //  NMRsim.dataFileOut();
65      NMRsim.weightedMatrix();
66    NMRsim.randomMotion();
67    NMRsim.writeDecay();
68    NMRsim.kernelMatrix();
69
70      system("PAUSE");
71      return EXIT_SUCCESS;
72      return 0;
73
74  }
```

Appendix C

Inversion Algorithm

This Matlab code was provided as part of the "Single-phase NMR network simulation code" developed by Talabi (2008) and available at the Imperial College of London's website (https://imperialcollegelondon.app.box.com/v/NMR-single-phase-network-model). It can be used to inverse NMR magnetisation decay data into relaxation times.

A description and derivation of the equations used in this code can be found in Section 2.6. An explanation on the input files required for this algorithm is detailed in Section 3.6.

```
% These files will be createed by the FILLING_LIST keyword

M = xlsread('M.xls')
K = xlsread('K.xls')
Wm = xlsread ('Wm.xls')
T2 = xlsread('T2.xls')

A1 = inv(K'*K+Wm)*K'*M
L = ((M-K*A1)'*(M-K*A1))/(A1'*A1) % Lambda on the text

KKex = K'*K
KMex = K'*M
Wmex = Wm;
KK = KKex

A = inv(KKex+(L*L*Wmex))*KMex
[nn, zz] = size(KKex)
[d,tt] = size(A)

mark = 0;

for mag = 1:zz
    if A(mag, 1)<0
        BB1(mag, 1)=0
    else
        BB1(mag, 1)=A(mag, 1)
    end
end

while (mark <1)
[mighty, tt] = size(A)
for me = 1:mighty
```

```
    if  A(me, 1)< 0
            KKex(:,me)=[999];
            KKex(me,:)=[999];
            KMex(me,:)=[999];
            Wmex(:,me)=[999];
            Wmex(me,:)=[999];
    end
end

count11 = 0
KKex1 = KKex
KMex1 = KMex
Wmex1 = Wmex

% This is for KKex
for m = 1:mighty
    if KKex(m,m)== 999
       count11 = count11+1
       if count11==1
       KKex1(:,m)=[]
       KKex1(m,:)=[]
       else
       j = m - count11+1
       KKex1(:,j)=[]
       KKex1(j,:)=[]

    end
    end
end

%   this is for Wmex
count11 = 0
for m = 1:mighty
    if Wmex(m,m)== 999
       count11 = count11+1
       if count11==1
       Wmex1(:,m)=[]
       Wmex1(m,:)=[]
       else
       j = m - count11+1
       Wmex1(:,j)=[]
       Wmex1(j,:)=[]

    end
    end
end

% This is for KMex
count11 = 0
for n = 1:mighty
   if KMex(n,1)==999
     count11 = count11+1;
     if count11==1
     j = n;
     KMex1(j,:)=[];
     else
     j = n - count11+1;
     KMex1(j,:)=[];
```

```
    end
 end
end

A = inv(KKex1+(L*L*Wmex1))*KMex1
[d, tt] = size(A)

signal = 0
for see = 1:d
    if A(see, 1)<0
        signal = signal + 1;
    end
end

sink = 0;
 for mini = 1:100
  if BB1(mini)~=0
        sink = sink+1;
        if A(sink, 1)<0
        BB1(mini, 1)=0;
        else
        BB1(mini, 1)=A(sink, 1);

end
end
end

KKex = KKex1
KMex = KMex1
Wmex = Wmex1
A
BB1
if signal==0
    mark = 2;
end
end

for res = 1:100 %size(BB1)
    finalResult(res, 1) = T2(res, 1)
    finalResult(res, 2) = BB1(res, 1)
end

%Magnet = K*BB1
%finalResult
xlswrite ('T2-Distribution.xls', finalResult)
```