GEOMETRY-BASED MODELING OF DISPERSION IN CORE-SHELL PARTICLE MEDIA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRE HATİPOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CHEMICAL ENGINEERING

SEPTEMBER 2017

Approval of the thesis:

GEOMETRY-BASED MODELING OF DISPERSION IN CORE-SHELL PARTICLE MEDIA

Submitted by **EMRE HATİPOĞLU** in the partial fulfillment of the requirements for the degree of **Master of Science** in **Chemical Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halil Kalıpçılar _____
Head of Department, **Chemical Engineering**

Asst. Prof. Dr. Harun Koku _____
Supervisor, **Chemical Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Deniz Üner _____
Chemical Engineering Dept., METU

Asst. Prof. Dr. Harun Koku _____
Chemical Engineering Dept., METU

Prof. Dr. Pınar Çalık _____
Chemical Engineering Dept., METU

Assoc. Prof. Dr. Çerağ Dilek-Hacıhabiboğlu _____
Chemical Engineering Dept., METU

Asst. Prof. Dr. Eda Çelik Akdur _____
Chemical Engineering Dept., Hacettepe University

Date: 07.09.2017

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Emre Hatipoğlu

Signature :

ABSTRACT

GEOMETRY-BASED MODELING OF DISPERSION IN CORE-SHELL PARTICLE MEDIA

Hatipoğlu, Emre
M.Sc., Department of Chemical Engineering
Supervisor : Asst. Prof. Dr. Harun Koku

September 2017, 203 pages

Dispersion is an important physical phenomenon that heavily influences performances of systems such as chromatographic separation processes. Mathematical modeling of this phenomenon is therefore widely investigated. This thesis study investigates dispersion of random-walking particles, or tracers, around core-shell particles, a type of recently commercialized spherical and porous stationary phase used in liquid chromatography that has a solid impermeable core that limits diffusion near the center and a porous shell covering the core. A random-walk approach was used for modelling the diffusion events, coupled with an external fluid velocity field to simulate convection and diffusion simultaneously. Impermeable boundaries of an unbound, without wall-effects, liquid chromatography column packed with core-shell particles were created using basic principles of analytical geometry, defining core-shell

particles as a collection of a large core spheres and much smaller shell side spheres coated around the core based on actual microscopy images of core-shell particles. Reconstruction method was very similar to the actual production methods of these type of materials where a silica core sphere is coated by silica nanospheres to create a core-shell particle with a very homogeneous geometry. Analytically reconstructed geometry was visually inspected using CAD images and found to be appropriate. The core-shell particle geometry was then copied into a periodic random jammed packing of monodisperse hardspheres generated independently by a software and scaled in size such that core-shell particles would flush-fit inside the hardsphere that make the random packing. Random packing of hardspheres were also used as the system boundaries of fluid flow calculations. Assuming no flow would occur inside the pores of core-shell particles, velocity field of the fluid flow obtained by these calculations were used in couple with random-walk diffusion to simulate dispersion in the periodic random jammed packing of core-shell particles. Predictions of the dispersion model were quantized in terms of reduced plate height at different operating Peclet numbers and the results were compared with experimental data found in the literature. Predictions of the model compares very well with the experimental data with deviations clearly explainable by the differences between the simulated system and the experimental system. Therefore the analytical geometry based reconstruction method of the core-shell particles was successful and it can potentially pose an alternative to complicated imaging and image processing for similar system geometries.

Keywords: Random-walk Diffusion, Dispersion, Analytical Geometry, Core-Shell, Chromatography

ÖZ

ÇEKİRDEK-KABUK PARÇACIK ORTAMLARINDA KÜTLE DAĞILMASININ GEOMETRİ TABANLI MODELLENMESİ

Hatipoğlu, Emre
Yüksek Lisans, Kimya Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Harun Koku

Eylül 2017, 203 sayfa

Kütle dağılması, kromatografik ayırma işlemleri gibi bazı sistemlerin performansını oldukça ileri seviyede etkileyen bir fiziksel olaydır. Dolayısıyla bu olayın matematiksel olarak modellenmesi yaygın bir şekilde çalışılmaktadır. Bu tez çalışması, sıvı kromatografisinde son yıllarda ticari olarak kullanılmaya başlanan bir durağan faz malzemesi olan, geçirimsiz bir çekirdek ve bu çekirdeğin etrafını kaplayan gözenekli bir tabakadan oluşan küresel çekirdek-kabuk parçacıklarının etrafında rastgele-yürüyüz yapan noktasal parçacıklara ait kütle dağılması olayını incelemeyi amaçlamaktadır. Ayrı olarak hesaplanan bir akışkan hız alanı, rastgele-yürüyüş metodu kullanan bir difüzyon modeliyle birlikte kullanılarak konveksiyon ve difüzyon olaylarını birlikte açıklayabilen bir kütle dağılması modeli oluşturulmuştur. Duvar etkilerinin olmadığı, çekirdek-kabuk parçacıklarla istiflenmiş sonsuz genişlikte teorik

bir sıvı kromatografi kolonu bu çalışmada basit analitik geometri prensipleri kullanılarak yeniden oluşturulmuştur. İlk aşamada, çekirdek- kabuk parçacıklar, mikroskop görüntüleri ve gerçek üretim metodları göz önünde bulundurularak büyük bir çekirdek küresinin etrafına katman-katman yerleştirilmiş kabuk kürelerinden oluşacak şekilde matematiksel olarak yeniden inşa edilmiştir. Sonrasında ise aynı büyüklükteki kürelerden oluşan periyodik olarak rastgele ve sıkıca istiflenmiş kürelerin geometrisi ayrı bir yazılımla oluşturulmuş, istiflenmiş küre geometrisi ise yeniden inşa edilen çekirdek-kabuk parçacıkların tam olarak periyodik istifteki kürelerin içine sığabileceği şekilde yeniden ölçeklendirilmiş ve nihayet periyodik olarak rastgele istiflenmiş bir çekirdek-kabuk parçacık geometrisi elde edilmiştir. Oluşturulan çekirdek parçacık geometrisi bilgisayar yardımlı çizim teknikleri kullanılarak görsel olarak incelenmiş ve uygun bulunmuştur. İstiflenmiş küre geometrisi ayrıca akışkan hız alanı hesaplanmasında kullanılmış, ve çekirdek-kabuk parçacıkların gözenekli kısımlarında akışın olmayacağı varsayılarak elde edilen akışkan hız alanı rastgele-yürüyüş modeliyle birleştirilerek çekirdek-kabuk malzemelerin etrafında kütle dağılımını tahmin edebilen bir matematiksel model oluşturulmuştur. Modelin tahminleri lteratürde bulunan uygun deneysel verilerle karşılaştırılmıştır. Model tahminleri ve deneysel verilerin aralarındaki farklar ise simülasyon sistemi ve deneysel sistem arasındaki temel farklar göz önünde bulundurularak açıklanabilmiştir. Sonuç olarak, çekirdek-kabuk malzemelerin analitik geometri prensipleri kullanılarak yeniden oluşturulmasının, benzer sistemlerde kullanılabilecek olan başarılı bir yöntem olduğu ortaya çıkmıştır. Bu yöntem görüntüleme ve görüntü işleme gibi zahmetli ve külfetli prosedürlere alternatif oluşturabilir.

Anahtar Kelimeler: Rastgele-yürüyüş Difüzyon, Kütle Dağılımı, Analitik Geometri, Çekirdek-Kabuk, Kromatografi

To family and friends,

ACKNOWLEDGMENTS

First and foremost, I would like to serve my gratitudes to my thesis supervisor, Asst. Prof. Dr. Harun Koku for his support and help. I am very grateful for the opportunities he presented me, that otherwise would most possibly not occur. His knowledge and diligence has always inspired me and always will.

I feel extremely lucky to have great friends all of whom I share darkest memories and brightest hopes of our lives, and even a profession with. I cannot thank enough to Veysi Halvacı, Zeynep Karakaş, Arzu Arslan Bozdağ, Berna Sezgin and Deniz Kaya for keeping me sane by their companionship on and off campus, their valuable advices, opinions, conversations on life and academia. I would like to thank to Görkem Bircan, Ekin Nural, Oğuzhan Paçal, Ali Dinç Bozat and Emre Can Ersoy, who are not different than a brother to me, for their life-long friendships and supports.

I want to thank to my beloved brother, Eren Hatipoğlu, for his love, support and friendship. Lastly, I would like to thank to my parents for their support, love and for encouraging me to take on this M.Sc. programme. Without them, this work literally would not exist.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

H	Height equivalent to a theoretical plate, or plate height
h	Reduced plate height
d_p	Particle diameter (core-shell, sphere)
D, D_{AB}	Bulk diffusivity
D_{eff}	Effective diffusivity
D_L	Longitudinal dispersion coefficient
L	Characteristic length
L_{pc}	Length of the periodic cell
N	Number of tracers used in a simulation
$n_{aux,k}$	Number of auxiliary circles around the core sphere or the *k*th sphere of influence
n_l	Number of shell layers in a core-shell particle
n_s	Number of random-steps a single tracer takes in a simulation
$n_{\mathrm{ssa,i,k}}$	Number of shell-side spheres that can be placed around the *i*th auxiliary circle of the *k*th sphere of influence
$n_{sse,k}$	Number of shell-side spheres that can be fit around the equator of core sphere, or the *k*th sphere of influence

$\mathrm{P}^{j,i,k}$	A vector that has the coordinates of the center points of *j*th shell-side sphere centered on the *i*th auxiliary circle in the *k*th shell layer of the core-shell geometry as its components
Pe	Peclet number
$\mathrm{r}_{\mathrm{aux,i,k}}$	Radius of the *i*th auxiliary circle of the the *k*th sphere of influence
r_c	Radius of the core of a core-shell particle
r_{hs}	Radius of monodisperse hardspheres in the random jammed packing
r_p	Radius of a core-shell particle
r_s	Radius of the shell-side spheres of a core-shell particle
r_{soi}	Radius of a sphere of influence
Re	Reynolds number
t_s	Duration of the diffusion/dispersion event in a simulation
u	Fluid velocity vector
u_z	Superficial fluid velocity
X	A vector that has the coordinates of a tracers position as its components
X_L	Counter-part of X localized in the main periodic cell

Greek Symbols

$\propto_k, \propto_{c,k}$	Spread angles and corrected spread angles between shell-side spheres placed around the equator of the core sphere or the *k*th sphere of influence
$\beta_{\mathrm{i,k}}, \beta_{\mathrm{c,i,k}}$	Spread angles and corrected spread angles between shell-side spheres placed around the *i*th auxiliary circle of the *k*th sphere of influence
ε	Void fraction

Δl	Random-step size
ΔP	Pressure drop
Δt	Time increment, or time-step of random-walk
φ	Core-to-particle diameter or radii ratio of a core-shell particle
Φ	Floor function
ξ	Unit vector with random direction
μ	Viscosity
ρ	Density
${\sigma_L}^2$	Longitudinal position variance
τ	Characteristic time
τ_C	Convective time
τ_D	Diffusive time
ϑ	Velocity field tensor

CHAPTER 1

INTRODUCTION

Dispersion, which can loosely be defined as the random spreading of concentrated mass due to bulk and molecular motion, is an important phenomena that affects performances of frequently encountered systems in chemical engineering. Chromatographic separations for example, are processes where dispersion, also referred to as band-broadening in chromatography, controls the majority of the separation performance.

Liquid chromatography is a separation process that takes advantage of different affinities of analytes or species in the mixture to the stationary phase to separate the mixture travelling along with the liquid mobile phase. Dispersion, or band-broadening, in liquid chromatography is typically quantized using the terms plate height or reduced plate height (Giddings,1965) which are very closely related to the longitudinal dispersion coefficient in the system (Maier et al., 2000). The quest for reducing dispersion in liquid chromatography column gave birth to different types of stationary phases such as fully porous spherical particles, monoliths and core-shell particles.

Core-shell particles constitute a specific type of stationary phase for high performance liquid chromatography columns. They were first introduced as a new concept by Horvath et al. (1967) 50 years ago, and are now widely used for purification of small molecules and peptides. Solid cores of these particles do not allow diffusion, effectively reducing the band broadening otherwise would be caused by analyte molecules unnecessarily spending very long time diffusing through tortuous pore

space that would be available in place of the solid core (Macnair et al., 1997). As a result of the reduced path length of diffusion inside the particle, internal mass transfer limitations in core-shell particles -especially for larger molecules- are very low for core-shell particles compared to fully porous ones. This difference in the mass transfer limitations gives core-shell particles a considerable advantage over fully porous particles at superficial velocities near minimum plate height and beyond (Gritti et al., 2007). Modern day commercial core-shell particles are typically produced with layer-by-layer preparation methods, a type of controlled growth process, (Gritti & Guiochon, 2011) that provides a narrow particle size, shell thickness and core size distributions (Bruns and Tallarek, 2011). These properties of core-shell particles makes them a good candidate for an example system to build a mathematical model that can predict dispersion around columns packed with these particles.

The equation of continuity and preservation of mass and momentum is typically invoked in modeling of diffusion and dispersion. However, continuum solutions of the governing equations can only be obtained for systems with relatively simple geometries (Choi et al., 2017) (Rani et al., 2017) (Wu and Chen, 2015). Even if the continuum solutions are obtained for systems with relatively more complex geometries, analytical solutions might yield results with poor accuracy (Sattin, 2008). Accordingly, numerical solution becomes the sole option, which are prone to numerical errors such as numerical dispersion. Fortunately there is an alternative approach for modeling of diffusion an dispersion, the random-walk and particle tracking methods. Since the random-motion of a point like tracer in stagnant and unhindered media results in the probability distribution of a tracer, the chance to find a tracer at a certain point after a certain amount of random steps are taken by it, which is mathematically identical to the analytical solution of the heat equation in the same system with same initial and boundary conditions (Chicone, 2017). The size of the random-step and corresponding time increment during the random walk is calculated by the Einstein's mean squared displacement formula, later given in Equation (14). If the random-walk experiment is repeated for a large amount of individual tracers, the mean squared displacement of all tracers becomes proportional to their diffusivity in the system. For systems with no-flux boundary conditions, impermeability is

simulated by collision control between tracers and the impermeable boundaries which can be done by simply using tracer positions and the equations describing the impermeable boundaries inside inequalities (Szymczak and Ladd, 2003). Random-walk models coupled with external velocity fields calculated separately resembles an Euler approximation of Fokker-Planck equation, which is mathematically analogous to the classic advection-diffusion equation (Maier et al., 2000). The coupled model is commonly referred to as particle tracking method of modeling dispersion and it is frequently used throughout the literature. There is already a particle tracking model by Daneyko et al. (2015) that can simulate dispersion of small molecules in unbound core-shell columns, but the model uses an effective medium approach (that uses an effective shell diffusivity, instead of defining the boundary geometry inside the shell side of the core-shell particles) for intraparticle diffusion.

This thesis work focuses on reconstructing a boundary system based on the production methods and microscopy images of modern core-shell particles using basic principles of analytical geometry and building a model that simulates dispersion in the reconstructed system using the principles of particle tracking methods and mainly open-source software. Analytical geometry methods allows minimal computer memory requirements for the storage of reconstructed boundary system that is essentially a collection of center coordinates of spheres. This method also allows taking on a microscopic approach to diffusion where diffusing mass is represented by point-like tracers with bulk diffusivity. Effective diffusivity inside the reconstructed core-shell particles is met innately due to collision control with definite pore structure in the system. In addition, the simple simulation model created in this study can potentially be scaled-up for wider applications involving large, adsorbing-desorbing and reacting molecules or even simulations of dispersion inside entire packings, given required facilities are available.

The rest of the thesis is organized as follows:

A literature survey in Chapter 2 is presented for introducing more detail related to the topic. In Chapter 3: Methods, every phase of the study leading to the final product, including the procedures for reconstructing a general core-shell particle geometry by

using only three user-given variables (core-to-particle ratio, particle diameter, amount of shell layers) and extending the geometry into a random packing, building a basic random-walk free diffusion model and coupling it with the boundary system, calculating fluid flow in the packing of core-shell particles and integrating the random-walk diffusion model with velocity field to finalize particle tracking simulation of dispersion. Application of these methods on utilized software, that is free form Fortran 95 programming language for this thesis study, is explained in detail in appendices.

In Chapter 4: Results & Discussion, validations of Fortran programs carrying out the calculations of the model, visualizations of the reconstructed boundary system and their inspection, predictions of the diffusion and dispersion simulations and their comparisons with experimental and theoretical data found in the literature are given and discussed. Conclusions and recommendations about the thesis study in Chapter 5 and Chapter 6, respectively, followed by the Appendices, mark the end for the thesis.

CHAPTER 2

LITERATURE SURVEY

2.1. Liquid Chromatography

2.1.1. Overview

Liquid chromatography is a separation process that takes advantage of different affinities of analytes or species in the mixture to the stationary phase to separate the mixture travelling along with the liquid mobile phase. A key factor that strongly influences separation performance in chromatography is dispersion, also commonly referred to as band-broadening, as roughly visualized in Figure 1, which is a measure of variance, σ_L^2, in the residence time distributions of concentrated bands of analyte that travel through the column.

Height equivalent to a theoretical plate, plate height in short, is the main term used in common that describes the performance of a chromatography column. Different plate heights for different species in the analyte mixture is responsible for distinct peaks visible in a typical chromatogram. Plate height of a species is defined as, by Giddings (1965):

$$\mathrm{H} = \frac{\delta \sigma_L^2}{\delta z} \tag{1}$$

Where σ_L^2 is the variance in the elution band, 'z' is the longitudinal position. Plate heights are typically reduced by the average particle diameter in columns packed with spherical particles to get the dimensionless parameter, *reduced plate height*, described as $h = H/d_p$. Plate height relation given in Equation (1) is very closely related to longitudinal dispersion coefficient in the column by the superficial mobile phase velocity, u_z. If longitudinal position, z, is substituted by (tu_z) the equation for transient dispersion coefficient is obtained and if the time derivative in the equation is constant the equation gives the Fickian dispersion coefficient in the conventional advection and convection equation (Maier et al., 2000), given in Equation (2), which is proportional to the time slope of the variance in residence times of tracers throughout a column that determines the shape of a chromatographic band. In other words, dispersion coefficient and plate height, or dispersion and band-broadening in a chromatography column are practically the same things with different names.

$$D_L = \frac{Hu_z}{2} = \frac{1}{2}\frac{\delta\sigma_L^2}{\delta t} \tag{2}$$

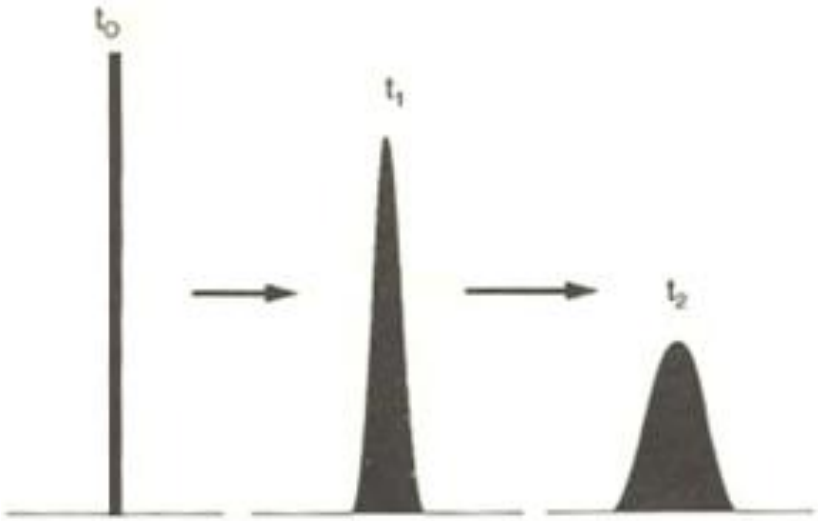


Figure 1: Band-broadening process. A rectangular concentration band morphs into a Gaussian shaped concentration band due to dispersion in the chromatographic system.

Hence, plate heights in a liquid chromatography column is in fact only related to the dispersion event in the system. Dispersion then, relates to certain characteristics of the system boundaries or physical properties of the mobile phase or the analyte. Still, reduced plate heights in different columns should emphasize differences in system boundaries due to the nature of reduced variables.

The most basic characterization of a liquid chromatography column can be made by fitting a van Deemter curve (van Deemter et al., 1956) to the experimentally determined data and determining the parameters A, B and C.

$$\mathrm{h} = \mathrm{A} + \frac{\mathrm{B}}{\mathrm{Pe}} + \mathrm{CPe} \tag{3}$$

The A term is regarded as a measure of the contributions resulting from the column geometry, structure of the packing and possible consequent flow irregularities throughout the column. The B term is the measure of plate height contributions due to longitudinal diffusion and the C term is a measure of mass transfer limitations in the system, such as adsorption of analyte molecules or highly tortuous diffusion of large molecules through the pores present in the system. Pe is the Peclet Number, $Pe = L.u_s/D_{AB}$, which is a dimensionless parameter that compares advective mass transfer in the system to the diffusive mass transfer, i.e. higher Pe corresponds to greater advection in the system. Another basic model is the Knox Equation (Knox, 2002) that uses similar fit parameters with similar physical significance, but the A term is modified with a power (typically 1/3) of Peclet number:

$$\mathrm{h} = \mathrm{APe}^{1/3} + \frac{\mathrm{B}}{\mathrm{Pe}} + \mathrm{CPe} \tag{4}$$

There are also more advanced models, some of which will be introduced later, but all of them are based on these main contribution parameters in van Deemter equation.

2.1.2. Porous Stationary Phases in Liquid Chromatography

The inverse relation between chromatographic column performance and dispersion motivates the developement of different porous stationary phases in liquid chromatography. Fully porous spherical stationary phase materials are well-understood, have well-established production techniques and are commonly used in liquid chromatography (Guiochon and Gritti, 2011). However they can reduce column efficiency at high speed separations very drastically, especially for large molecules due to very high internal mass transfer resistances (Macnair et al., 1997). They are still often preferred in liquid chromatography systems due to their high capacity.

Monolithic stationary phases are also commonly used and relatively new materials in liquid chromatography. Monoliths allow similar mobile phase flow rates at lower pressure gradients compared to spherical stationary phases due to their macroporous structure and low mass transfer limitations near the por surfaces due to macropore structure makes them efficient (Hlushkou et al., 2010). Second generation monoliths that became commercially available back in 2012 are even more efficient due to their radial homogeneity reducing velocity biases across the column, but the competition between spherical packing columns and monolithic columns still continues as progress in these areas still have momentum (Hormann and Tallarek, 2014).

Another relatively new stationary phase is core-shell particles, which is an older concept than monoliths but a still developing one due to its perhaps premature emergence near the era where fully porous particles were improving very rapidly in performance and popularity (Guiochon and Gritti, 2011). Core-shell particles are separately discussed in the next section.

2.2. Core-Shell Particles

2.2.1. Overview

Core-Shell particles, also frequently referred to as superficially-porous particles, or pellicular particles, are a type of porous stationary phase used in chromatography. They consist of a solid, impermeable core and a porous shell that allows obstructed diffusion coated around the core, as roughly visualized in Figure 2.

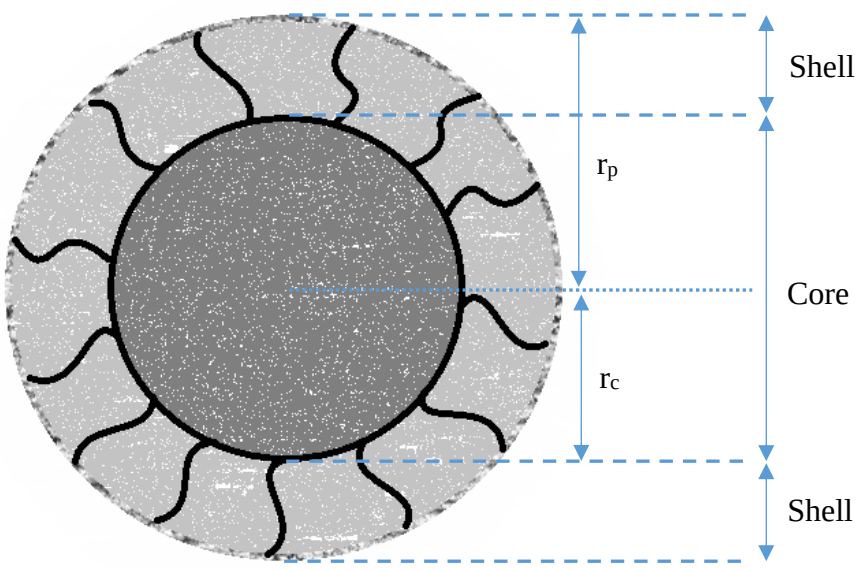


Figure 2: Basic visual representation of a core-shell particle, with impermeable solid core sphere and porous shell coated around it.

The impermeable solid core of the particles reduce the intra-particle mass transfer limitations encountered especially at high Peclet numbers and for large molecules as a very early study -that can be considered one of the first to explore the concept- reveals in which pellicular ion exchangers with a stationary phase resembling modern-day core-shell particles were investigated by Horvath et al. (1967). Dead volume created by the solid core intrinsically allows even more limited diffusion inside the core-shell particle compared to a fully porous particle, reducing the intra-particle residence times and relatively dispersion, leading to improved band-broadening in columns (Macnair et al., 1997). These improvements achieved by core-shell particles brings their performance close to that of fully-porous particles with smaller diameters, with the added benefit of requiring less pressure drop to operate at a similar efficiency (Gritti & Guiochon, 2012).

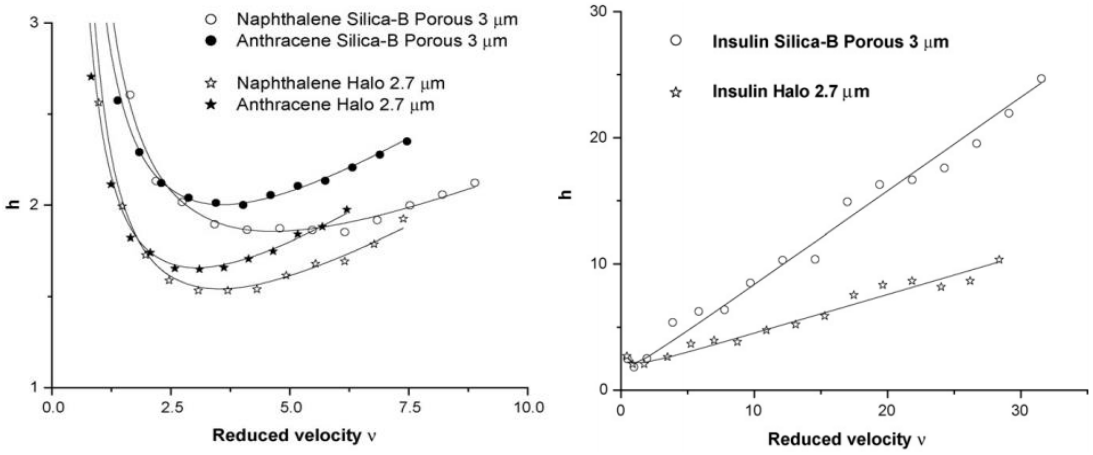


Figure 3: Improved C-branches in columns using core-shell particles (Halo) compared to the columns packed with fully porous particles (Silica-B). Left: Small molecules performance. Right: Large molecules performance (Adopted from the work of Gritti et al. (2007) with permission).

Oldest core-shell type particles were quite large in diameter compared to modern ones. Kirkland's (1969) study investigates the performances of early superficially porous particles ranging from 37 to 63 microns in diameter and a shell thickness of only 1/30 of the particle diameter. Another early study related to core-shell particles by Done and Knox (1972) investigates the performance of commercially available (at the time) core-shell particle series with average diameters ranging from 29 to 109 microns. In contrast, modern day core-shell particles are typically much smaller, possibly due to improvements in production methods over the years, such as the 5 micron "Poroshell" particles whose performance was investigated in another study of Kirkland et al. (2000) and 2.7 micron Halo particles investigated in Gritti et al.'s study (2007). Although such small sizes, newer core-shell particles such as Poroshell and Halo have much greater core-to-particle ratios compared to older ones. This gives new generation of core-shell particles a comparable capacity to fully porous particles at a smaller diameter, and also improved plate heights seen in Figure 3 (Guiochon & Gritti, 2011).

2.2.2. Production Methods and Imaging

Modern core-shell particles are prepared by coating solid silica spheres with shell layers. Core spheres are typically produced based on the method suggested very early on by Stöber et al. (1968), in which a process was proposed for controlled growth of monodisperse spherical silica particles that can help increase the size of seed spheres quite homogenously. Once the core-spheres are prepared, they are coated with a porous layer in a similar procedure proposed by Stöber and Fink, then the coated silica sphere is again used as a seed sphere and coated with another layer and so on (Gritti & Guiochon, 2011). Figure 5 displays a simplified visual representation of this process. This controlled growth approach to preparation of core-shell particles leads to very homogeneously spherical particles rather than egg-shaped ones. There are relatively earlier research on methods for controlling the pore sizes in such mesoporous materials and allowing production of particles with greater pore size while keeping the pore-size distribution as narrow as possible (Ma et al., 2003).

Gritti et al. (2010) inspects the performance of an HPLC column packed with commercially available core-shell particles, produced in a ‘layer-by-layer’ approach. According to the information provided by the manufacturers of these commercial particles in their work, these particles have a very narrow particle and pore size distributions and they confirm these by SEM images of the particles at different zoom levels. A very striking image that belongs to one of these particles is in Figure 4, where the solid silica core and incredibly smooth shell layers with very small silica nanospheres are very clearly visible.

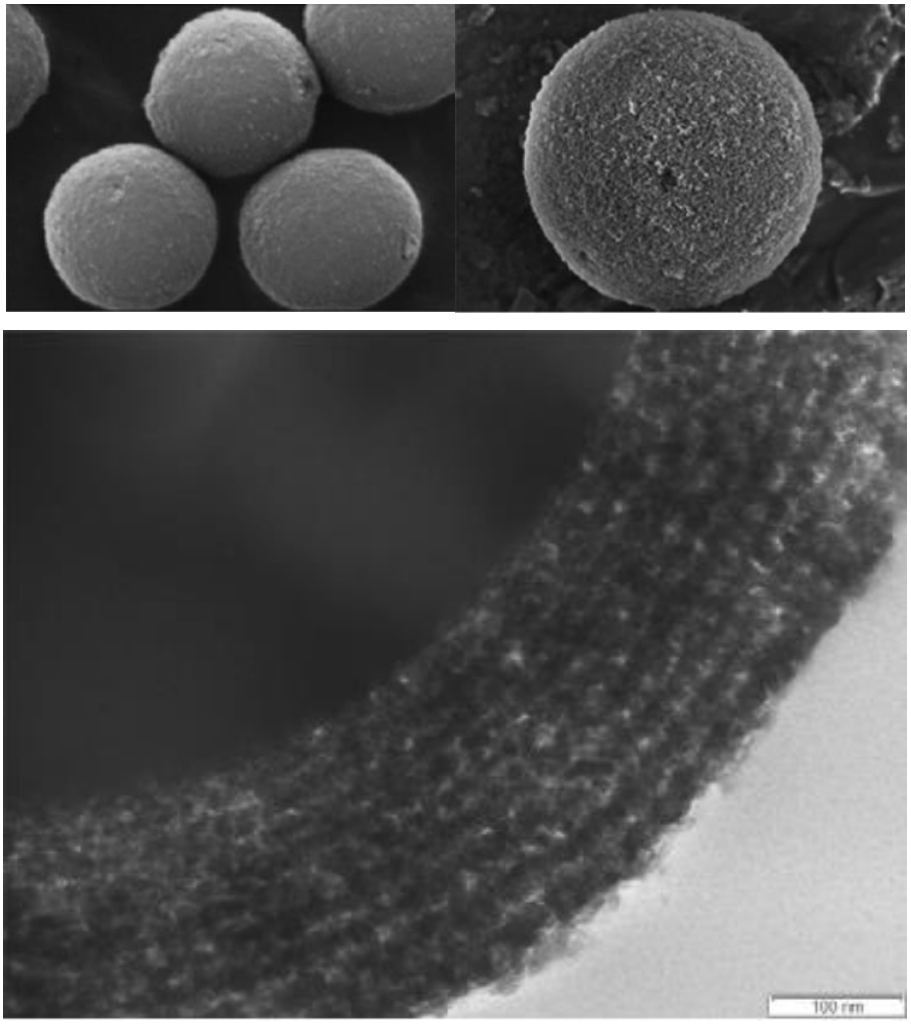


Figure 4: Images of commercially available core-shell particles. Attention for smoothness and similar sizes of the entire particles (top left and right) and the

morphology of the shell layers (bottom) (Adopted from the work of Gritti et al. (2010) with permission).

Bruns and Tallarek (2011) used confocal microscopy for imaging a portion of an entire capillary column packed with commercial Kinetex core-shell particles and reconstructed the packing geometry in 3-D (Figure 6), which is an important study for deeper understanding of the effects of packing preparation methods and particle properties on the morphology of the packing. Using image-processing techniques, they confirmed the narrow size distributions of the modern core-shell particle. Additionally, they also found that the size distributions of the core spheres and distributions of shell thickness in these particles are also very narrow (Figure 7).

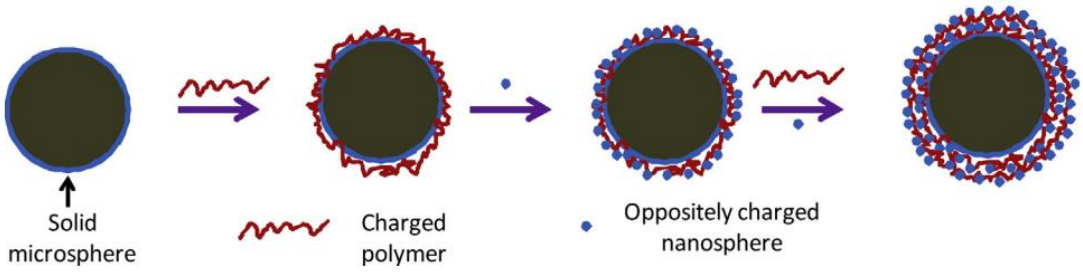


Figure 5: Representative flowchart of the layer-by-layer coating of solid core with nanospheres. Charged polymers are removed by heat treatment, their space becomes the pores between nanospheres coated in layers (Adopted from the work of Hayes et al. (2014) with permission)

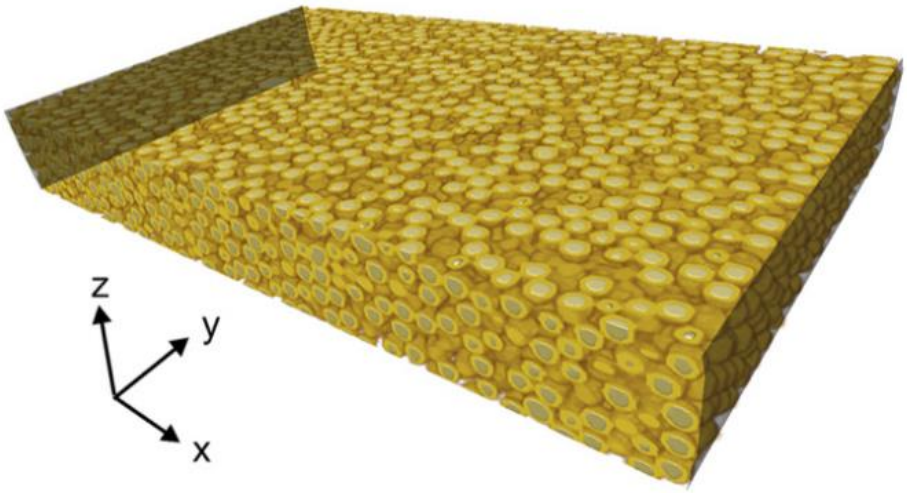


Figure 6: Digital reconstruction of capillary packed with core-shell particles. Core spheres and shell areas are clearly visible in grey and yellow respectively. Capillary walls are highlighted in dark shade (Adopted from the work of Bruns and Tallarek (2011) with permission).

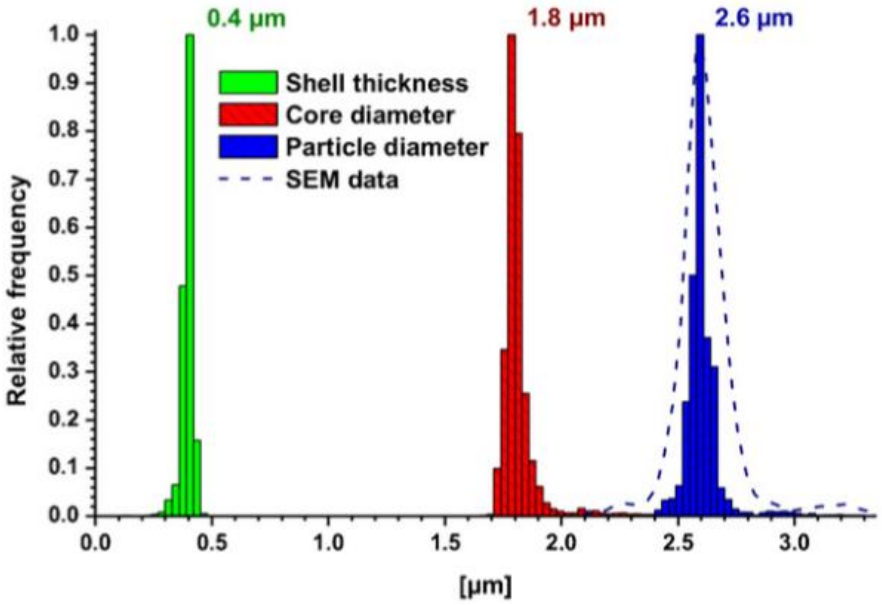


Figure 7: Distributions of shell thickness, core diameter and entire particle diameter of core-shell particles in the reconstructed portion of the capillary column, as determined by image-processing techniques (Adopted from the work of Bruns and Tallarek (2011) with permission).

2.3. Modelling of Diffusion and Dispersion

2.3.1. Continuum Solutions

Transient diffusion without externally-induced flow is described by the heat equation without generation terms, which is analogous to the Fick's second law, assuming a dilute medium where density and diffusion coefficient is constant. In one dimensions, the equation is the following (Choi et al., 2017)

$$\frac{\delta C}{\delta t} = D\frac{\delta^2 C}{\delta x^2} \tag{5}$$

The general solution for the heat equation given in Equation (5), for a general initial condition $C(x,0) = f(x)$ with open boundaries was given in the study of Zhukovsky and Srivastava (2017) as,

$$\mathrm{C(x,t)} = \frac{1}{\sqrt{4\pi D t}}\int_{-\infty}^{\infty} e^{-\frac{(x-\varepsilon)^2}{4Dt}} f(\varepsilon)d\varepsilon \tag{6}$$

If the initial concentration profile is described by delta function $\delta(x-\mu)$, Equation (6) transforms into a general gaussian distribution with variance $\sigma^2 = 2D$.

$$\mathrm{C(x,t)} = \frac{1}{\sqrt{4\pi D t}} e^{-\frac{(x-\mu)^2}{4Dt}} \tag{7}$$

In the study of Choi et al. (2017) other analytical solutions, some tailed and skewed probability distributions, for the heat equation with similar initial conditions but in a finite domain is proposed and proven to be solutions to the heat equation. These type of solutions would probably be applied to some analogous systems that might be encountered in chemical engineering. However proposing a similar solution and reversing to prove it is a solution for the heat equation in a system with complicated

geometry, such as a core-shell packing, would be extremely challenging if not impossible.

$$\frac{\delta C}{\delta t} + u\frac{\delta C}{\delta x} = D\frac{\delta^2 C}{\delta x^2} \tag{8}$$

Analytical solutions for advection-diffusion equation, given in Equation (8), is also available for different systems throughout the literature. Rani et al. (2017) derived analytical solutions for the transport of reduced and oxidized species to a rotating disc electrode for both transient and steady-state cases, with a depletion rate boundary condition at the surface of the cylinder, and a fairly complex velocity field around the electrode. They claim the solutions to be fairly accurate. Still, the solutions are one dimensional and for a relatively simple boundary geometry.

In the study of Wu and Chen (2015), they propose a one dimensional transient analytical solution for a Taylor dispersion in a packed bed that uses averaged phase properties. The solution takes on a macroscopic approach to the problem in which individual contributions to dispersion from different mechanisms cannot be separately quantized.

There are also numerical solutions of the mass balance and continuity equations, even derived for chromatographic systems involving core-shell particles (Kaczmarski and Guiochon, 2007). However these models use approximation methods such as lumping around the core-shell particles and are prone to numerical dispersion.

2.3.2. Random-Walk Diffusion

The random-walk and diffusion analogy first came up with the introduction of well-known mean squared displacement formula by Einstein (for n dimensions, $\Delta x = \sqrt{n2D_{AB}\Delta t}$), where it was revealed that the random displacements of diffusing particles in a certain time interval was proportional to the square root of the length of

that time interval. This finding was later used to show that the probability density function of the random-walking particle position, which has a variance equal to D_{AB}, becomes identical to the analytical solution of the heat equation for transient diffusion of a species initially point-injected to a free, stagnant diffusion domain as Δx and Δt approaches infinitely small amounts. In other words, the random-walk experiment repeated for a large number of times, or simultaneously for a large number of random-walking particles, results in a distribution of random-walking particles that resembles the normalized concentration profile of diffusing species. Since the integral of normal distribution throughout the entire real domain is equal to unity, so is the normalized concentration profile of random-walking particle ensemble after the experiment is repeated a large number of times. This effectively is equivalent to a mass balance in the system and it is innate to random-walks (Chicone, 2017).

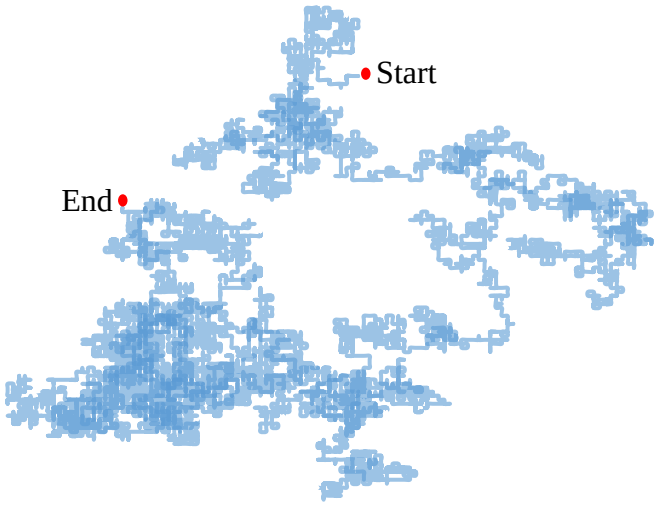


Figure 8: Path followed by a random-walking particle in 2-D. Start and end points shown in red. Darker blue paths are sampled multiple times by the particle.

Certain types of boundary conditions such as no-flux or constant-concentration can be adapted to random-walk models. For no-flux boundaries, Szymczak and Ladd (2003) proposes specular-reflection, can also be referred to as mirror reflection, or rejection

methods for the simulation of impermeability at the boundary. According to their study, specular-reflection method is the only one that can yield close to 100% accurate results but the rejection method has much less computing power demand and it can estimate nearly flat concentration profiles at the proximity of impermeable boundaries. These methods are discussed in further detail in Chapter 3.

Random-walk simulations of diffusion are used in different research areas. Gentile et al. (2015) uses a random-walk method to estimate local diffusion coefficients in a square-shaped capillary system, a potentially interesting study for the field of microfluidics where local transport properties are important. Khirevich et al. (2011) also uses a random-walk based diffusion model to probe the effect of microstructure of different spherical packings they generated, on the event of diffusion. Modified versions of random-walk diffusion models are even used for modelling so called "anomalous diffusion" phenomena where a non-linear relation between mean-squared displacements and time is observed (Angstmann et al., 2015). Examples can be increased. Random-walk approach for simulating diffusion can be concluded to be a robust and well-established method.

2.3.3. Particle Tracking Methods for Dispersion

Random-walk diffusion models are typically coupled with an external velocity field to simulate dispersion. The Fokker-Planck equation, given in Equation (9), describes the relation between the position of a random-walking particle affected by the drag forces in a velocity field and the total velocity that particle would have at that position due to random motion and the drag forces acting on it (Chen et al., 2017).

$$dx(t) = f\big(t, x(t)\big)dt + g\big(t, x(t)\big)dW(t) \tag{9}$$

The 'f' and 'g' functions in the equation are the velocity field and diffusion coefficient respectively. Under fully developed flow conditions ($f = f\big(x(t)\big)$) and constant bulk

diffusivity ($g(t, x(t)) = c$), an Euler approximation of Fokker-Planck equation very closely resembles the random-walk equation with an advective term added. In fact, Fokker-Planck equation is analogous to classic advection-diffusion equation under these conditions (Maier et al., 2000). This approximation of Fokker-Planck equation is used in particle tracking methods for the simulation of dispersion.

In the study of Widiatmojo et al. (2016) particle tracking methods were invoked to create a mathematical model of gas dispersion in underground tunnel systems. They created a simplified network of Taylor channels based on the tunnel network of some Kushiro Coal Mine and carried out the particle tracking simulations. Their simulation predictions are in almost perfect agreement with the actual gas measurements taken from Kushiro Coal Mine despite the usage of simplified system geometry. Brutz and Rajaram (2017), in another study, develops a particle tracking method based model to investigate dispersion of contaminants throughout fractured rock formations, essentially a porous media. Their results show very good agreements of particle tracking simulations with the analytical solutions for dispersion. Wang and Cardenas (2015) in a similar study involving fractured rock formations extends to three dimensions and reproduces similar results that supports the accuracy of particle tracking methods in approximating the analytical solutions for classic advection-diffusion equation.

Particle tracking models of dispersion are also developed for chromatographic systems. For example, Koku et al. (2012) reconstructs a monolithic stationary phase environment based on microscopy images of a monolithic media and uses the reconstructed geometry to simulate fluid flow and finally dispersion in a theoretical chromatography column using a particle tracking approach for modelling the phenomenon. It would be important to point out that the mentioned study was based on the Ph.D. thesis study of Koku (2011).

2.3.4. Dispersion Models Related to Core-Shell Particles

Kaczmarski and Guiochon (2007) proposes several different models based on the combination of mass balances of the components in the system in mobile phase and in the stationary phase. The most general model proposed in that study is the general rate model, which does not have any simplifications besides the main assumptions that are isothermal system, fully-developed steady incompressible flow, no radial concentration gradient, adsorption at equilibrium, non-transient dispersion and no flow in the shell sides of core-shell particles. Lumped pore diffusion model they propose simplifies the diffusion in shell side by using a volume-averaged concentration in the mass balance of species in stationary phase, introducing an internal mass transfer resistance term that can only be estimated. Equilibrium-dispersive model further simplifies the previous models by assuming the mass transfer resistances are negligible. They use the residence time distributions predicted by numerical solutions of these models to calculate plate heights for different molecules in the same column. Cavazzini et al. (2007), in another study, compares the general rate model predictions of Kaczmarski and Guiochon (2007) with experimental data. According to the comparisons, main problem with these models is the fact that it neglects radial contributions in the flowing mobile phase leading to an underestimation of plate height values at lower mobile phase velocities.

Another model that predicts dispersion in core-shell packings was developed by Daneyko et al. (2015) based on Giddings theory of coupled eddy diffusion (1963), which are given in Equations (10) and (11). The model is still empirical, but it can predict the plate height contributions from short (transchannel) and long (interchannel) range inter-particle channels (see Figure 9) separately by making use of parameters, λ'_i and ω'_i, related to the packing geometry. Contributions from longitudinal diffusion and mass transfer resistances related to the core-shell particles are represented by h_L and h_C terms respectively.

$$h = h_L + h_{short(1)} + h_{long(2)} + h_C \tag{10}$$

$$h = \frac{2D_{eff}}{D_{AB}R(Pe)} + \frac{2\lambda'_1}{1 + (2\lambda'_1/\omega'_1)(Pe^{-1})} + \frac{2\lambda'_2}{1 + (D_{eff}/D_{AB})(2\lambda'_2/\omega'_2)(Pe^{-1})} + C(Pe) \quad (11)$$

Daneyko et al. (2015) carried out particle tracking simulations of dispersion that takes place in random packings of core-shell particles in the same study they developed the modified Giddings theory. In their simulations, they adopt an effective medium approach where the core-shell particles are defined only by two parameters, core-to-particle ratio and effective diffusivity in the porous shell side. Effective medium approach allowed them to confirm the effectiveness of the simulations on the analysis of plate height contributions rising from different parts of the system. In addition, they state that the particle tracking approach in simulating dispersion can be coupled with digital reconstructions of chromatographic beds with morphological information retrieved from the actual tomography or microscopy images of the beds to computationally analyze dispersion in these type of columns even in more detail and accuracy. Studies of Bruns and Tallarek (2011) and Bruns et al. (2012) proves that microscopy images of capillaries packed with core-shell particles can be used for digital reconstruction of the system geometries with great convenience and even shell and core sides of particles are clearly distinguishable. Still, for the particle tracking simulations, either the effective medium approach must be used or individual core-shell particles must also be reconstructed using microscopy images.

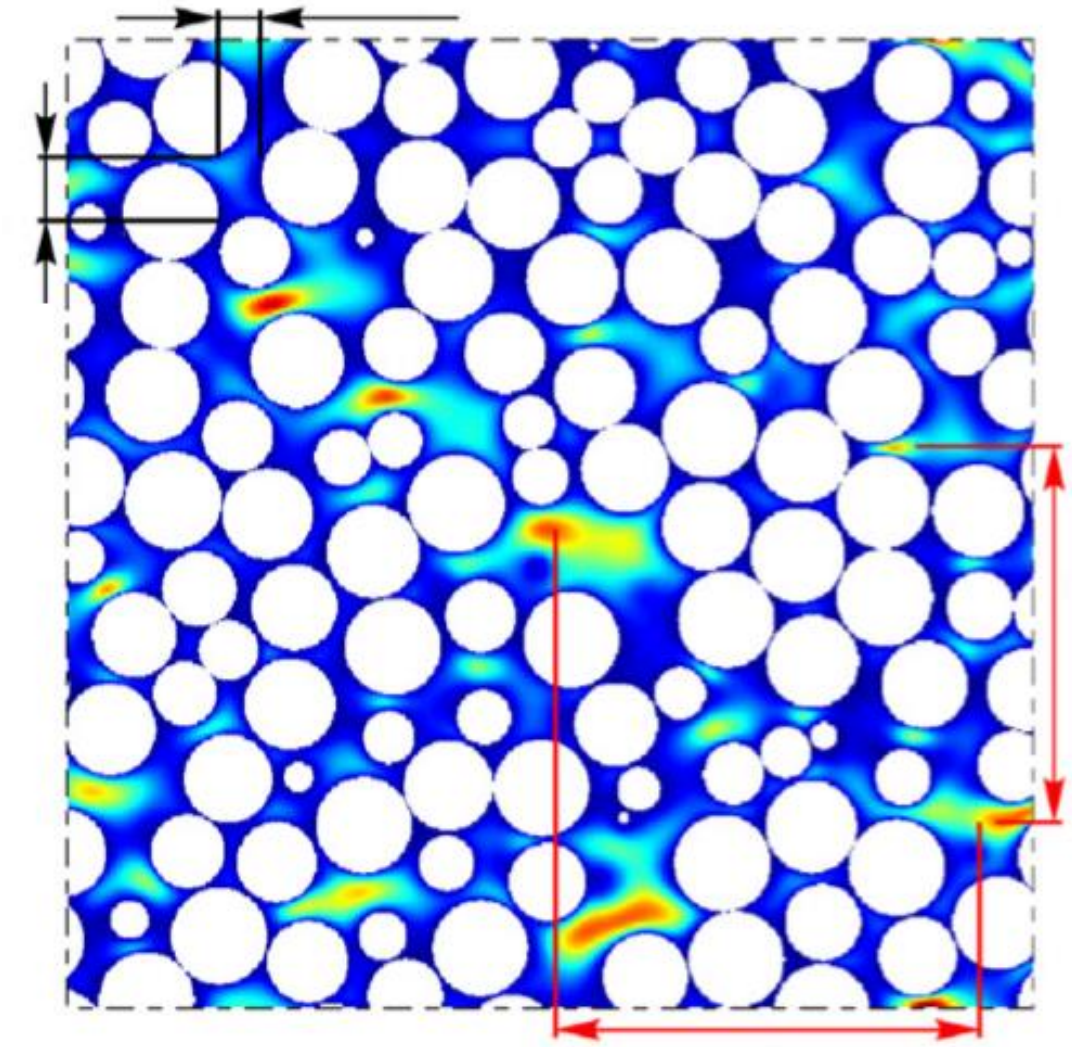


Figure 9: Ranges of transchannel (black brackets) -due to channeling of flow through narrow high porosity regions along the column- and interchannel (red brackets) -due to radial or transverse flow caused by the fluctuations in the porosity profile across the column- contributions to the dispersion, or plate height in a chromatographic system (Adopted from the work of Daneyko et al. (2015) with permission).

2.3.5. Time Scales and Dimensionless Measures of Time

Time scales in diffusion and dispersion events are important due to the transient nature of these events in systems with tortuous geometries. This behaviour of diffusion is experimentally observed in nuclear magnetic resonance studies involving pulsed-field-gradient techniques, where the time dependent diffusion coefficient of fluid molecules traveling through a porous media can be obtained and it is observed to be approaching an asymptotic effective diffusivity value in open pore structures after a certain amount of time (Latour et al., 1995) (Sen, 2004). Dispersion also behaves similarly and it is experimentally observed in the study of Gritti et al. (2014). In that study, they investigate a concept called "Parallel Segmented Flow Chromatography (PSFC)", where standard columns were effectively transformed into narrower columns by sending only the middle part of the outlet stream to outlet detector by splitting the flow in the center region after a short axial distance using an end-fitting. Comparing the chromatograms from standard columns and PSFC columns, they conclude that the plate height gain in PSFC columns is due to pre-asymptotic operation of the column, before the analyte travels for enough time to sample the entire radial distance in the column.

Commonly used time scales in the literature for diffusion and dispersion are diffusive-time (τ_D) and convective-time (τ_C), defined by the Equations (12) and (13).

$$\tau_D = \frac{L^2}{D_{AB}} \tag{12}$$

$$\tau_C = \frac{L}{u_z} \tag{13}$$

Note the units of τ_D and τ_C are in seconds. τ_D very closely resembles the mean squared-displacement formula of Einstein, it represents the amount of time required for a diffusing species with diffusion coefficient D_{AB} to sample a distance equal to L, which

is the characteristic length in the system. One can see that the ratio of diffusive time to convective time in a system becomes identical to the Peclet Number in that system, $Pe = \tau_D/\tau_C = Lu_z/D_{AB}$.

There are different variances of τ_D in different studies, for example in a study by Khirevich et al. (2011) τ_D is defined as a dimensionless time measure $\tau_D = t/({d_p}^2/2D_{eff})$, time required for a random-walker in the random packing they generated to travel a distance of a sphere diameter with the effective diffusivity inside the packing, which is essentially a direct application of Einstein's formula in 1-D. In some other similar study by Maier et al. (2000), that involves particle tracking simulations of dispersion in periodic random packings of monodisperse spheres, diffusive time is defined as $\tau_D = t/({d_p}^2/D_{AB})$. This definition uses bulk diffusivity instead of effective diffusivity, it still stands for more or less the same time-scale considering D_{eff} is around ~1.5 times D_{AB} in random packings of spheres. In the same study, τ_C is again defined as the dimensionless version of the one given in Equation (13), $\tau_C = tL/u_z$. Khirevich et al. (2009) in another study uses transverse dispersion coefficient D_T instead of diffusion coefficients in the definition of diffusive-time as $\tau_D = t\left({d_p}^2/2D_T\right)$. Although the definitions of τ_D is different in the latest two studies mentioned, they both essentially predict that the longitudinal dispersion in the system reaches asymptotic behavior after ~$0.1\tau_D$ seconds if the diffusive-time is defined as in Equation (12).

CHAPTER 3

METHODS

Construction of the dispersion model in this work consists following main procedures.

- Simulation of diffusion, in free and constrained media.
- Creation of core-shell particle geometry.
- Creation of periodic monodisperse random packings of hardspheres.
- Simulation of fluid flow in the boundary system.

Except for the dependency of fluid flow simulation on the hardsphere-packing geometry, these listed procedures can be independently developed and combined together to create a mathematical model that predicts the behaviour of dispersion in a chromatographic system involving core-shell particles as the stationary phase. The model has certain underlying assumptions, which limits its range, which will be stated in this dedicated chapter, along with detailed descriptions of the procedures. Their implementation on computers using mainly the Fortran programming language, as well as other software that are MS Excel, Octave and OpenSCAD and the integration of fluid flow and diffusion models are explained in Appedix C.

3.1 Diffusion Model

3.1.1 Free Molecular Diffusion

In Chapter 1 and Chapter 2, it has been discussed that random-walk particle tracking (RWPT) approach is a highly effective way for direct modelling of molecular diffusion in systems with high geometrical complexity, where continuum solutions are hard to obtain. RWPT methods use Einstein's relation to discretize the amount of displacement, Δl, a Brownian particle (tracer) undergoes in a certain time interval, Δt, depending on the molecular diffusivity of these particles.

$$\Delta l = \sqrt{2D_{AB}\Delta t} \tag{14}$$

Using the relation given in equation (14), time-dependent position vectors of tracers in one dimension are expressed by the following equation.

$$\mathbf{X}(\mathrm{t} + \Delta\mathrm{t}) = \mathbf{X}(\mathrm{t}) + \boldsymbol{\xi}\Delta l \tag{15}$$

Where $\boldsymbol{\xi}$ is a vector that results from addition of unit vectors with a random direction. Initial positions of tracers can be varied.

Equation (15) is equally valid for diffusion in higher dimensions since for isotropic diffusion random-walking particles will still have equal probability of proceeding in every possible direction, although this probability will decrease in value as more directions of movement are introduced (i.e. a random-walking particle in 1-D has a 50% chance to go either direction while a random-walking particle in 3-D has 16.7% chance to proceed in any of the 6 possible directions). In any case, magnitude of the net displacement vector, Δl, becomes $\Delta l\sqrt{d}$ for diffusion in *d* dimensions.

Equation (14) allows choosing either Δl or Δt as the basis and calculation of the other depending on the choice. Conventionally a suitable Δl value is chosen, depending on

the size and shape of impermeable boundaries in the system, which will be discussed in section 4.1.2. Order of magnitude in Δt selection for free diffusion is discussed in the section 4.1.1 where the verification of basic random-walk algorithm is shown.

The total number of random steps a single tracer takes throughout the simulation is then calculated simply by dividing duration of diffusion event, t_s, by Δt.

$$n_s = \frac{t_s}{\Delta t} \tag{16}$$

The diffusion coefficient of tracers is back-calculated using the initial and final positions of the tracers. The total displacement of a tracer is calculated by using its initial and final cartesian coordinates in the distance formula. For an ensemble of N tracers that are simulated to undergo diffusion for t_s seconds, molecular diffusion coefficient can be calculated by using following equation, which is basically the reversed version of Einstein's relation.

$$\mathrm{D}_{AB} = \frac{\sum_{i=1}^{n}(\Delta X_i)^2}{N}\frac{1}{2.d.t_s} \tag{17}$$

In a free diffusion event with open boundaries and no impermeable walls, Equation (17) is expected to yield the same D_{AB} value as the input diffusion coefficient value. As a matter of fact, it should yield this same value regardless of the simulation duration given. Therefore, it can be used for the verification of random-walk algorithm.

The model has underlying assumptions that affect the accuracy of it under certain circumstances. The assumptions are as follows.

1. No interaction between tracers.
2. Infinitesimal (point) tracers.
3. Constant & Isotropic free diffusion coefficient throughout the system.
4. Non-retained tracers (no adsorption in the system).
5. No chemical reactions.
6. Stagnant diffusion media.

Assumptions 1, 2 and 3 imply that the results obtained from the model would not accurately predict diffusion behavior of larger molecules, such as large proteins, that might have diffusion coefficients that change depending on their charge, molecular interactions and size. Still, the results would be comparable to experimental diffusion/dispersion coefficient data obtained by using small molecules in analyte level concentrations. Assumption 4 indicates that, if there are any impermeable boundaries in the system, any adsorption-desorption mechanism that may occur between boundaries and diffusing species in a real physical system is ignored. This becomes especially important in a system with pores. Adsorption event further affects the effective diffusivity of molecules in these pores, whereas the absence of adsorption allows diffusion only affected by the tortuosity of the pore structure. Assumption 5 supports constant free diffusion coefficient assumption and it guarantees the material balance of trace ensemble. Assumption 6 is the final assumption of the random-walk *diffusion* model. However it is a temporary assumption that is needed to verify the validity of random-walk diffusion algorithm, as the main purpose of this study is to construct a model that explains diffusion and convection together. All in all, the assumptions from 1 to 5 can be very well acceptable for columns used for analytical purposes that involve separation of small molecules.

With the equations and the assumptions introduced, the basic algorithm for random-walk simulation in open boundaries is given below. This algorithm will be added features throughout this chapter, as different aspects of diffusion and dispersion is explored and coupled with the random-walk diffusion model in empty media.

1. Set *N*.
2. Set D_{AB}.
3. Set simulation time, t_s.
4. Set Δt and calculate Δl using Equation (14)
5. Determine total number of random steps, using Equation (16).
6. Set initial positions for all tracers.
7. Generate n_s amount of unit vectors with random directions.
8.

9. Displace each tracer n_s times using Equation (15).
10. Use Equation (17) to recalculate D_{AB} and compare with initially set value.

3.1.2 Impermeability & Collision Control

Impermeable walls are defined as boundaries which prohibit any mass flux beyond the bound system. The equation of continuity emposes a Neumann boundary condition at impermeable walls of the system, such that the concentration profiles at those boundaries must satisfy the equation:

$$\nabla c = 0 \tag{18}$$

In random-walk methods, these no-flux conditions are typically simulated by simply monitoring positions of tracers, detecting boundary violations and preventing any tracer from traveling beyond the diffusion domain bound by impermeable walls. The mathematical implementation of this strategy uses the position vectors of tracers along with analytical equations representing shapes and locations of impermeable boundaries The impermeable regions in a system can be conveniently defined using analytical inequalities. Figure 10 illustrates a diffusion domain, bound by one circular and one linear impermeable walls, which essentially includes every point outside impermeable areas defined as the set of inequalities $x > 25$ and $x^2 + y^2 < 25$. If position of any Brownian particle in the system satisfies any of these inequalities after a random step is taken, then a collision between the particle and boundary is detected. Hence, the particle must be returned back to the diffusion domain.

Szymczak and Ladd (2003) suggest *specular reflection*, or *mirror reflection*, as the most reliable solution to effectively move Brownian particles back into diffusion domain, as it is the only method that yields zero gradient near impermeable boundaries in the system. According to their definition of specular reflection, a tracer that crosses an impermeable boundary is returned to such a point in the diffusion domain that corresponds to the mirror image of the tracer's position across the impermeable

boundary with respect to the boundary itself. This very closely resembles an elastic collision of a ball against a fixed straight-wall that has incomparably larger mass than the ball itself, in which case, if the velocity of the ball is known, its trajectory can easily be calculated even if it hits the wall at an angle. However, this is a considerably more complicated calculation if the wall has a curvature to it such as the circular boundary seen in Figure 11, as the problem in this case involves tangent lines near the point of contact between the boundary and the tracer. Then the boundary equation and the equation for the line that passes through two time-adjacent positions of the tracer before and after it crosses the boundary must be used to calculate the point of contact. Only then the tangent line to the curved wall at this point of contact can be found and used as the axis of mirror for the tracer. Considering the fact that a statistically significant population of tracers should be used in the simulation that results in a large count of collisions especially in a simulation involving a porous geometry, the specular reflection method should account for a significant portion of the computational requirements of such a simulation. In the same study of Szymczak and Ladd (2003), a method with much less computational demand is also suggested and it is called *the bounce-back method*. The bounce-back method simply returns any trans-boundary tracers back to their position inside the diffusion domain right before the collision. However, this method is shown to be not working as well near the boundaries as the specular reflection method. According to their simulations, the bounce-back method yields non-zero concentration gradients near the boundaries. However the concentration profile similar to the concentration profile yielded by specular reflection method, translated towards the boundary by a constant equal to the step-size of the tracer. This finding points to the fact that bounce-back method causes a numerical mass flux into the impermeable boundary, therefore increasing the effective diffusivity of tracer ensemble in the system. In other words, it effectively increases the space where diffusion is allowed. Still, the non-zero concentration gradient caused by bounce-back method is shown to be very small and its effect on the effective diffusivity will be neglected for the sake of reducing computational requirements of the model. Therefore, the bounce-back method is selected in this algorithm.

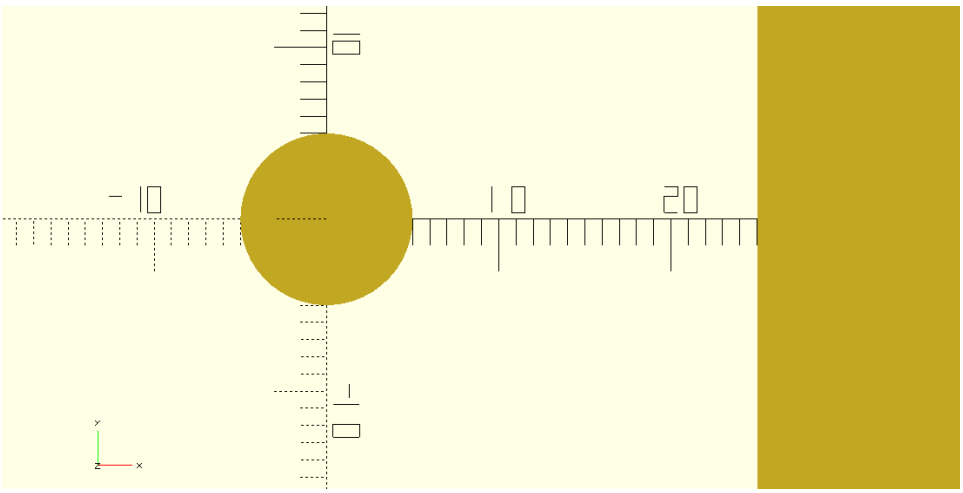


Figure 10: An arbitrary 2-D system. Dark shade areas are bound by two impermeable walls, the circle ($x^2 + y^2 = 25$) and the line ($x = 25$). Diffusion domain is the area illustrated in lighter shade.

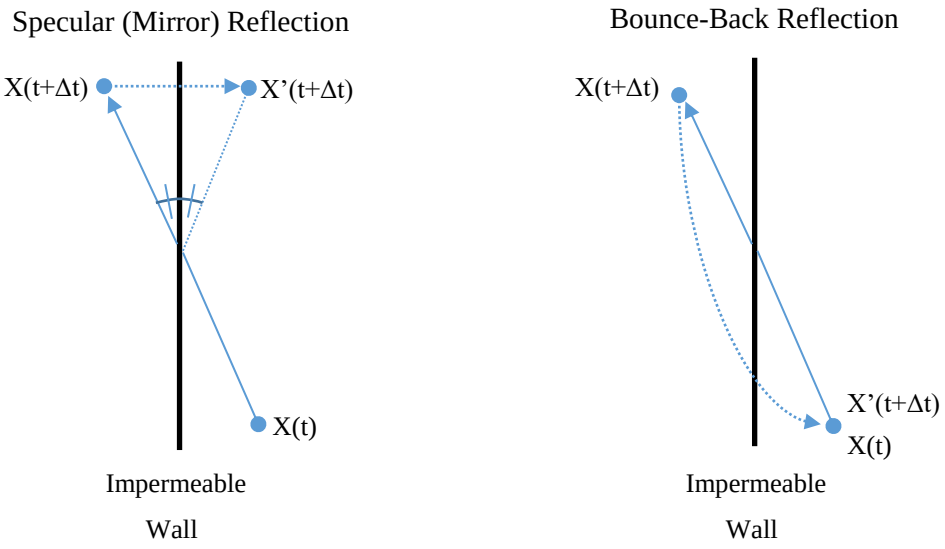


Figure 11: Specular Reflection and Bounce-Back methods (Szymczak and Ladd, 2003). Left side of impermeable wall is solid, right side allows diffusion. X(t) is the position of the tracer before collision. Tracer takes a random-step to the position X(t+Δt) which lies beyond the impermeable wall, and is reflected (either by mirror reflection or bounce-back) back to the position X'(t+Δt) in diffusion domain.

After the introduction of impermeable boundaries and collision control, random-walk algorithm evolves into the following shape. Changed and added steps are in *italic*.

1. Set *n*.
2. Set D_{AB}.
3. Set simulation time, t_s.
4. Set Δt and calculate Δl using Equation (14)
5. Determine total number of random steps, using Equation (16).
6. *Set boundary equations.*
7. Set initial positions for all tracers.
8. Generate n_s amount of unit vectors with random directions.
9. *Displace a single tracer, for one step using Equation (15).*
10. *Use $\boldsymbol{X}(t + \Delta t)$ in each boundary equation to check for collisions. If collision occurs, set $\boldsymbol{X}(t + \Delta t) = \boldsymbol{X}(t)$.*
11. *Repeat 9 and 10 until all tracers have taken n_s random steps each.*
12. Use Equation (17) to recalculate D_{AB} and compare with initially set value.

One should realize that recalculated time-dependent D_{AB} values converges to a value lower than the input value of D_{AB}, especially if the simulated system involves a tortuous pore structure. This occurrence is a very well-known property of diffusion in porous media, where the effective diffusion coefficient of diffusing species is a certain fraction of its free diffusion coefficient depending on the tortuosity of the pore structure.

3.1.3 Periodical Boundaries

In the previous section, it was explained how impermeability can be simulated by checking collisions between the tracers and impermeable boundaries. For smaller systems, such as the one depicted in Figure 10, storing information about the boundaries is rather simple since the example system has only 2 boundaries. However, a very large system like a packing of hardspheres may contain millions of spherical

boundaries depending on the size of hardspheres and packing container. In such cases, millions of sphere equations must be stored in memory for the collision control to be possible. To work around this impracticality, periodical boundaries in random-walk simulations are typically used. This type of a boundary uses a crystal unit cell system and virtually creates an infinitely large crystal structure of that unit cell. Figure 12 shows a basic illustration of how periodical boundaries can be used to create an infinitely large ensemble of circles by saving only one circle equation to memory. This becomes possible by defining a “local” tracer position, which is the main unit cell (where the boundaries are defined) equivalent of the global tracer position.

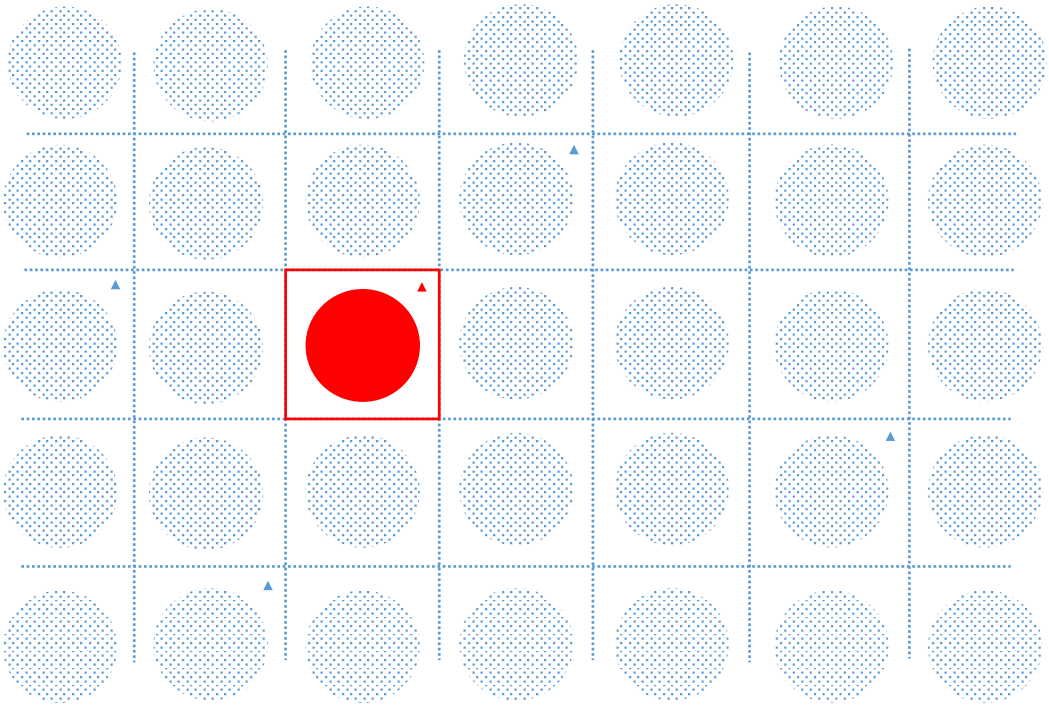


Figure 12: An illustration for the use of periodic boundaries to create a 2-D infinite array of circles in an ordered arrangement. Main periodic cell and a circular impermeable zone inside it are represented in solid red color, while the impermeable zones effectively created by the periodic boundaries are in dotted blue color. All points indicated by several small blue triangles are equivalent to the point indicated by the small red triangle in the main periodic cell. Crystal structure of the system extends to infinity without any bounds.

Let us assume a 2-D rectangular unit cell with dimensions W x L, cornered at the origin, occupying the corner of the first quadrant. The equivalent local position, $\mathbf{X_L}$, of a tracer wandering about a global position point (x_g, y_g) is then calculated by the following equation.

$$\mathbf{X_L} = \left(x_g - \Phi\left(x_g/W\right).W, y_g - \Phi\left(y_g/L\right).W\right) \quad (19)$$

Where Φ is a function that truncates its input to the nearest integer smaller than the input. Negative inputs are truncated down, for example $\Phi(-0{,}6) = -1$ while $\Phi(0{,}6) = 0$. It is, in fact, similar to "Floor" function used in several different programming languages.

Using the local tracer position to carry out collision control in the main unit cell is mathematically equivalent to translating impermeable boundary geometries defined in the main unit cell to the unit cell where the global tracer position is contained and carrying out collision control in that unit cell. Either way, if any collision occurs in the main unit cell, the global position is also set to the last known position before the collision, therefore creating infinitely many impermeable areas. Upon the addition of periodical boundaries, the random-walk algorithm is updated as following. Changed and added steps are, again, in *italic*.

1. Set N.
2. Set D_{AB}.
3. Set simulation time, t_s.
4. Set Δt and calculate Δl using Equation (14)
5. Determine total number of random steps, using Equation (16).
6. Set boundary equations.
7. Set initial positions for all tracers.
8. Generate n_s amount of unit vectors with random directions.
9. Displace a single tracer, for one step using Equation (15).
10. *Calculate $\boldsymbol{X_L}(t + \Delta t)$ using $\boldsymbol{X}(t + \Delta t)$ in Equation (19).*

11. *Use* $\boldsymbol{X_L}(t + \Delta t)$ *in each boundary equation to check for collisions. If collision occurs, set* $\boldsymbol{X}(t + \Delta t) = \boldsymbol{X}(t)$.
12. Repeat 9 and 10 until all tracers have taken n_s random steps each.
13. Use Equation (17) to recalculate D_{AB} and compare with initially set value.

3.1.4. Initial Conditions

Initial distributions of the tracers are important considering the random-walk method is a discrete representation of the continuum solutions. There can be other types of initial concentration profiles defined as initial conditions for a system, however this study uses only point injection or bulk (homogeneous) injection type initial conditions.

Point injection sets initial positions for all tracers to the same point, very much like a dirac delta function. Bulk or homogeneous injection, on the other hand assigns random initial positions to every tracer in a pre-determined volume, such as the diffusion-available volume in the main periodic cell.

The difference between these two initial condition types becomes important in a system with impermeable boundaries. All tracers in a tracer ensemble with a point injection type initial condition is bound to move across the same regular grid since the random-step size of all tracers are the same. Therefore one should expect dead zones around curved impermeable boundaries that cannot be sampled by any of the tracers due to discretization. In a homogeneously distributed tracer ensemble, on the other hand, every individual tracer still moves according to a regular grid but the grid is private for each tracer and relative positions of the grids are covering as much volume as possible throughout the system due to random initial positions of the tracers. Therefore bulk injection must be expected to have less dead volume around curved boundaries. This important difference between two initial condition types are discussed over the results given in section 4.1.2 of Chapter 4.

3.2. Construction of the Core-Shell Particle Geometry

3.2.1. Strategy

During the literature investigation, it was seen that some of the geometries such as a monolithic medium or a polydisperse spherical packing can be reconstructed digitally by using different imaging techniques and post-processing of these images, and these reconstructed geometries can be used in particle tracking simulation. However it was also revealed that these reconstruction methods are often experimentally difficult, computationally expensive and have high memory demand. Koku et al. (2012) reveal in their study that a digital reconstruction of a monolithic medium of size 18x14x18 micrometers cubed occupies over 150 gigabytes of memory. The main reason for such a requirement is the fact that the irregular shape of monolithic medium must be saved pixel by pixel to obtain an accurate representation of impermeable areas in the system. This problem is not exclusive to only monoliths and it will persist as long as the same method is used.

Core-Shell particles have a very suitable geometry to simplify the digital reconstruction approach in an attempt to eliminate these memory requirements. It was already discussed how these particles have a quite distinct geometry, thanks to the synthesis methods, that can be represented as almost perfectly arranged shell side spheres around a large core sphere. Therefore it is very convenient to represent a core-shell particle as an ensemble of analytical geometry entities, in other words, a group of radius and center coordinates data that represents each and every sphere that contributes to the entire core-shell particle geometry. This simpler analytical geometry approach is advantageos for certain reasons.

- It reduces memory demand, independent of any parameter that defines the exact geometry, such as shell thickness or number of shell layers. The data that represents a single element of the geometry is only the center point and the radius of that spherical element, which is always less than the requirements of image representation.

- It allows creating core-shell particle geometries with different shell thickness, radius and number of shell layers. This is a very important attribute of the approach since it will easily approximate the geometry of a real core-shell particle, given that its radius and shell thickness is known. Even the number of shell layers in a real core-shell can be identified using microscopy images and used as an input parameter.
- Created geometry is innately suitable for scaling due to the contributing elements being spheres, it will preserve shape after scaling, and it can easily be translated. Therefore it can easily be used in reconstructing monodisperse and -if wanted- polydisperse packings of core-shell particles.
- It will eliminate problems due to image-based reconstruction such as sampling errors.
- Most importantly, it is highly compatible with the diffusion/dispersion model being built in study.

The only drawback of this approach is the fact that it can only create a highly idealized core-shell particle geometry that is made of perfect spheres, closest elements to each other would be at least tangent to each other without any overlaps. Still, the approach is worth trying since it gets rid of the effort-intensive imaging process.

3.2.2. Single Layer Core-Shell Particle Geometry

Parameters that define the characteristics of a core-shell particle are entire particle radius (r_p), core radius (r_c) and the number of shell layers (n_l). Another parameter called core-to-particle ratio ($\varphi = r_c/r_p$) is introduced for convenience. Then radius of shell-side spheres (r_s) for a single layer core-shell particle can easily be calculated using $r_s = r_p(1-\varphi)/2$.

In the first step of reconstruction, imaginary geometrical entities called auxiliary circles are introduced. An auxiliary circle is an imaginary circle hovering around the core sphere, on which there lies center points of a certain number of shell side spheres

such that these shell side spheres would be tangent to core sphere and they would also be at closest tangent to other shell side spheres centered on the neighboring auxiliary circle. Total number of auxiliary circles and amount of shell side spheres on these circles varies depending on φ and r_s. A clear visualization of auxiliary circles can be seen in Figure 13.

Maximum number of auxiliary circles that can be placed around the core spheres is determined by figuring out how many shell side spheres can be placed around the equatorial circle of core sphere (n_{sse}), that are tangent to the circle and tangent to each other at closest distance. Imagine two shell side spheres tangent to each other and core sphere at its equatorial circle. Setting the center point of core sphere at the origin as basis, spread angle ($\propto$) between the line passing through the center of core sphere and the first shell sphere, and the line passing through the center of core sphere and the second shell sphere can be found by the following Equation (20), making use of isosceles triangle formed by center points of these three spheres. Then n_{sse} is calculated by the phi function introduced previously: $n_{sse} = \Phi(2\pi/\propto)$.

$$\propto = 2\arcsin\left(\frac{r_s}{r_s + r_c}\right) \tag{20}$$

Depending on n_{sse} being even or odd, maximum amount of auxiliary circles, n_{aux}, changes. One can visualize the auxiliary circles that can be placed around a core sphere with $n_{sse} = 12$ by imagining an analog clock where $n_{aux} = 5$. A core sphere with $n_{sse} = 11$ also has $n_{aux} = 5$ but only the appropriate part of the piecewise function given in Equation (21) can calculate the correct value of n_{aux} .

$$\mathrm{n_{aux}} = \begin{cases} \Phi((\mathrm{n_{sse}} - 2)/\,2), & \text{for even } \mathrm{n_{sse}} \\ \Phi(\mathrm{n_{sse}}/2), & \text{for odd } \mathrm{n_{sse}} \end{cases} \tag{21}$$

If the auxiliary circles are given number tags, from 1 to $\mathrm{n_{aux}}$ and starting from top to bottom, radii of these circles can be calculated by making use of right triangles. Since initially approximated spread angle changes due to truncations involved in calculating

n_{sse}, a corrected spread angle between auxiliary circles is calculated by $\propto_c = 2\pi/n_{sse}$. Then the radius of the *i*th auxiliary circle ($r_{aux,i}$) can be calculated by Equation (22).

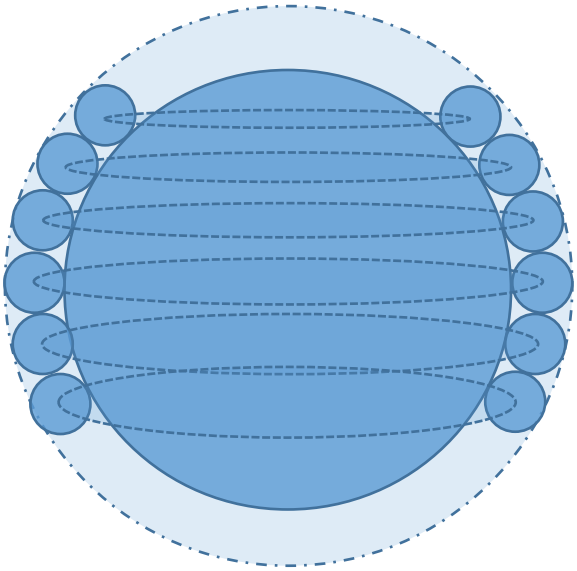


Figure 13: Rough visual representation of core-shell particle reconstruction. Different elements of the particle geometry (large core spheres and some of smaller shell spheres) and concepts created related to the calculations which are auxiliary circles (dashed circles, passing through the center of smaller shell spheres), sphere of influence (sphere with dot-dashed boundary) are visualized.

$$r_{aux,i} = \sin(i \propto_c) . (r_s + r_c) \tag{22}$$

The planes hosting auxiliary circles are all parallel to xy-plane and the z-coordinates of each auxiliary circle ($z_{aux,i}$) can be calculated using the same right triangles. Note that, any shell side sphere centered on an auxiliary circle will share the same z-coordinate with that auxiliary circle.

$$z_{aux,i} = \cos(i \propto_c) . (r_s + r_c) \tag{23}$$

The maximum number of shell side spheres that can be placed on the *i*th auxiliary circle ($n_{\mathrm{ssa,i}}$) can be calculated in a similar manner to the calculation of $\mathrm{n_{aux}}$. First a spread angle on that circle $\beta_{\mathrm{i}} = 2\arcsin\left(r_s/\mathrm{r_{aux,i}}\right)$ is defined. Then $n_{\mathrm{ssa,i}}$ is calculated by: $n_{\mathrm{ssa,i}} = \Phi(2\pi/\beta_{\mathrm{i}})$. Then the corrected spread angle becomes $\beta_{\mathrm{c,i}} = 2\pi/n_{\mathrm{ssa,i}}$

In the final step, x and y coordinates of the center points of shell side spheres placed an *i*th auxiliary circle can be calculated making use of more right triangles, similar to the ones used for the calculations of $\mathrm{r_{aux,i}}$ and $\mathrm{z_{aux,i}}$. Equations (24) and (25) calculate the x and y coordinates of the center point of *j*th shell side sphere centered on *i*th auxiliary circle respectively. These equations centers the 1st shell sphere on x-axis as basis and tags each shell side sphere on *i*th auxiliary circle from 1 to $n_{\mathrm{ssa,i}}$ counter-clockwise. Z-coordinates of the shell side spheres are already determined by $\mathrm{z_{i,j}} = \mathrm{z_{aux,i}}$ as mentioned previously.

$$x_{j,i} = \cos\left((j-1)\beta_{\mathrm{c,i}}\right).\mathrm{r_{aux,i}} \tag{24}$$

$$y_{j,i} = \sin\left((j-1)\beta_{\mathrm{c,i}}\right).\mathrm{r_{aux,i}} \tag{25}$$

Only remaining shell side spheres are one or two that needs to be placed tangent to the poles of core sphere depending on the value of n_{sse} being divisible by 4 or not. If divisible, all four quadrant points around the equator gets a sphere therefore two polar spheres are added. If not, region around the south pole becomes too narrow to fit another sphere for a realistic geometry. Their center points are predetermined without calculation. Sphere near the north pole is centered at the point $\eta(0,0,r_s + r_c)$ and the one near the south pole at $\sigma(0,0,-r_s - r_c)$.

3.2.3. Generalization to Multiple Shell-Layers

In the previous section, all required equations to calculate every element that creates an ideal single-layer core-shell particle geometry was explained and given. However core-shell particles often have more than one layer and variable shell thickness.

Thicker shells with a single layer would not realistically represent the void fraction in shell layer. Therefore the method should be extended to multiple shell-layers.

Introducing another helpful concept called a *sphere of influence* is a solution for the generalization of the method to cover multiple layers. A sphere of influence can be defined as an imaginary sphere that tightly wraps around core sphere and the first shell-layer around it. One can imagine this sphere of influence as a temporarily assigned, conceptual core sphere with a combined radius $r_{soi} = 2r_s + r_c$. As a matter of fact, the actual core sphere can be thought as the *zeroth* sphere of influence. Then the elements of another layer on top of the first one can be calculated by the same equations by setting $r_c = r_{soi}$. This procedure can be repeated for n_l number of times to create a core-shell geometry with n_l layers.

For a core-shell particle with n_l layers, radius of shell side spheres is calculated by the equation: $r_s = r_p(1 - \varphi)/2n_l$. Introducing the subscript *k*, which refers to the variables belonging to the elements located in *k*th layer, to the variables explained in previous section, equations evolve into the following. Note that *k* starts from 0 and ends at the value of n_l with $r_{soi,k} = 2kr_s + r_c$.

$$\propto_k = 2\arcsin\left(\frac{r_s}{r_s + r_{soi,k}}\right) \tag{26}$$

$$n_{sse,k} = \Phi(2\pi/\propto_k) \tag{27}$$

$$\mathrm{n_{aux,k}} = \begin{cases} \Phi((\mathrm{n_{sse,k}} - 2)/\,2), & \text{for even } \mathrm{n_{sse,k}} \\ \Phi(\mathrm{n_{sse,k}}/\,2), & \text{for odd } \mathrm{n_{sse,k}} \end{cases} \tag{28}$$

$$\propto_{c,k} = \frac{2\pi}{n_{sse,k}} \tag{29}$$

$$\mathrm{r_{aux,i,k}} = \sin(i_k \propto_{ck}) \,.\, (r_s + r_{soi,k}) \tag{30}$$

$$z_{\mathrm{aux,i,k}} = \cos(i_k \propto_{ck}) . (r_s + r_{soi,k}) \tag{31}$$

$$\beta_{\mathrm{i,k}} = 2\arcsin\left(\frac{r_s}{r_{\mathrm{aux,i,k}}}\right) \tag{32}$$

$$n_{\mathrm{ssa,i,k}} = \Phi(2\pi/\beta_{\mathrm{i,k}}) \tag{33}$$

$$\beta_{c,i,k} = \frac{2\pi}{n_{ssa,i,k}} \tag{34}$$

$$x_{j,i,k} = \cos\left((j-1)\beta_{\mathrm{c,i,k}}\right) . r_{\mathrm{aux,i,k}} \tag{35}$$

$$y_{j,i,k} = \sin\left((j-1)\beta_{\mathrm{c,i,k}}\right) . r_{\mathrm{aux,i,k}} \tag{36}$$

The resulting geometry is represented by vectors $\mathbf{P}^{j,i,k}$ pointing to the centers of each and every spherical element in the geometry, where the components of center point vectors are defined as in the following equations.

$$P_1^{i,j,k} = x_{j,i,k} \tag{37}$$

$$P_2^{i,j,k} = y_{j,i,k} \tag{38}$$

$$P_3^{i,j,k} = z_{j,i,k} = z_{\mathrm{aux,i,k}} \tag{39}$$

3.3 Periodical Random Packing of Core-Shell Particles

In the previous section, a method that utilizes basic principles of analytical geometry to allow reconstruction of a core-shell particle geometry was introduced. The method provides all defining parameters, being radii and center point vector components,

required for analytical representations of all elements in core-shell particle geometry. The resulting geometry can easily be translated across the coordinate system by using a translation vector on every element that collectively creates the entire geometry. The translation can be repeated an arbitrary amount of times to creates copies of the core-shell geometry without directly calculating the parameters for each core-shell particles. The translation vectors needed to copy a core-shell particle geometry into a random jammed packing are center point vectors of a computationally generated random jammed packing of hardpheres, the construction of which is described in the next section.

3.3.1. Random Packings of Monodisperse Hardspheres

Skoge et al. (2006), in their work related to maximally jammed packings of hyperspheres, investigate some properties of the periodic random jammed packings of spheres they generated using a modified Lubachevsky-Stillinger algorithm and generously share the computer code they created for their work with any researcher who needs it. If set to three dimensions, the algorithm calculates radius of spheres as well as center point vectors of all spheres in the main periodic unit cell with a single input that defines how many spheres are present in the packing. A periodic random jammed packing of 50 monodisperse spheres ($n_{hs} = 50$) with a radius $r_{hs} = 0.142079$ (depends on how many spheres there are in the unit cell, n_{hs}) and void fraction 0.355 was generated in a unit cube (periodic cell length: $L_{pc} = 1$) cornered at origin and used in diffusion/dispersion model by using the code Skoge et al. provided. For the sake of consistency, the same random jammed packing used in all simulations done for obtaining the results. The effect of this preference is discussed in the results and discussion chapter.

3.3.2. Visualization & Inspection of the Random Jammed Packing of Hardspheres

Computer generated visuals of the monodisperse spherical packing to be used in diffusion and dispersion simulations was done using an open source computer aided drawing program called OpenSCAD. Upon the visual inspection of generated packing a very crucial aspect of the packing for the model reveals itself. Packing generated by the code clearly has all 50 spheres centered in the unit cube however some of those spheres partially cross the boundaries defined by the unit cell. Consequently if the periodic unit cube is arranged into a crystal structure, these 'invading' spheres would partially appear in the main periodic unit cell even if their center point resides in the neighboring periodic cells. This prevents direct usage of the generated packing in the model because the specific collision control mechanism applied in this study needs access to parameters that define all main unit cell volume occupied by impermeable elements in the system. The problem can be solved by detecting invading spheres and including them in the packing in addition to 50 originally generated spheres.

The most convenient and systematic way that comes to mind for detection of invading spheres is to determine center points that lie within a radius distance inside the boundaries of the unit cube. If the hardspheres have a radius r_{hs} and *i*th sphere in the packing has its center point at $\boldsymbol{P}_{\boldsymbol{hs}}^{\boldsymbol{i}}$, *extreme points* on their surface in positive and negative x,y and z directions can be found by separate addition and substraction of r_{hs} on the compoents of the center point vector, $([P_{hs,1}^{i} \pm r_{hs}]\ P_{hs,2}^{i}\ P_{hs,3}^{i})$, $\left(P_{hs,1}^{i}\ [P_{hs,2}^{i} \pm r_{hs}]\ P_{hs,3}^{i}\right)$ and $(P_{hs,1}^{i}\ P_{hs,2}^{i}\ [P_{hs,3}^{i} \pm r_{hs}])$. If an extreme point lies beyond unit cube boundaries (any component is greater than 1 or less than 0), that sphere crosses the boundaries of its periodic cell. For the specific packing used in this work, individual hardspheres in the packing crossed over to 1, 2, 3 or 5 neighboring periodic cells. To be exact, 20 spheres crossed over to a single, 2 spheres to 2, 7 spheres to 3, and a single sphere to 5 other neighboring cells, making a total of 50 other spheres that need to be added into the packing in order for collision control to be properly carried out. Hence the sphere count inside the random packing increases to 100.

Figure 14 demonstrates the original packing and a partially modified version of it side by side. Red sphere seen in the figure is the one detected to be crossing over to 5 other cells, hence 5 copies of it is added to the packing (depicted in orange) that would otherwise be invading the main periodic cells from the neighboring cells. The figure also shows a black sphere crossing over to only one other neighboring cell. Therefore a single copy of it is added into the packing, which is visible at the back in grey. It would be important to point out that the packing on the right side of the figure, only demonstrates the inspection and manual modification method, therefore it is not the final packing used in any part of the model, since there are more over-hanging spheres in the packing that needs additional complementary copies to complete the procedure.

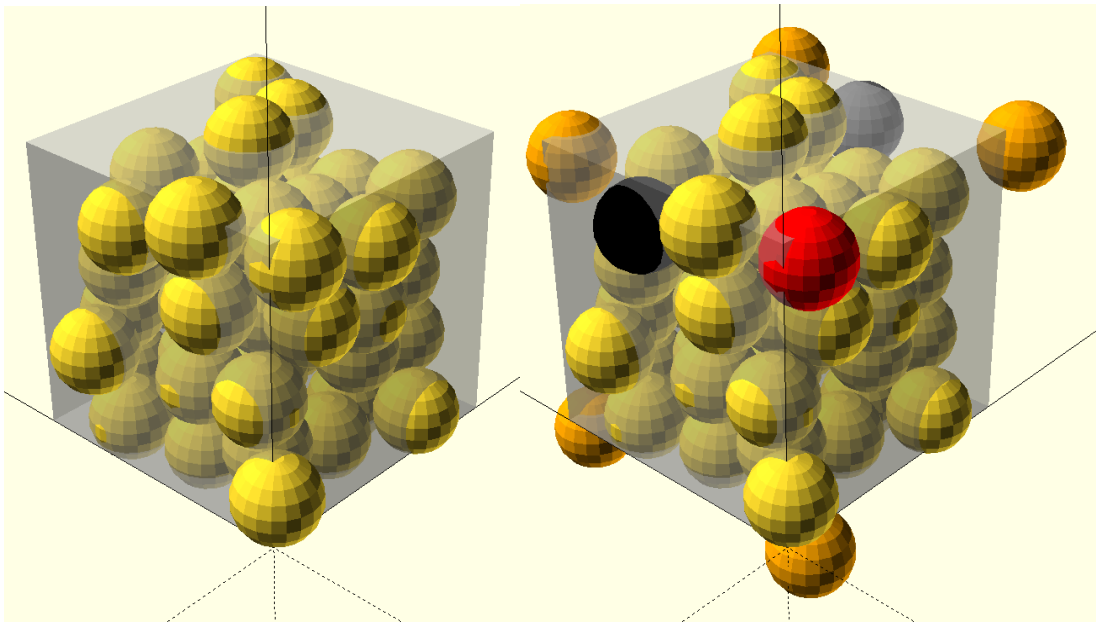


Figure 14: OpenSCAD images of the random jammed packing of monodisperse hardspheres. Cubic unit cell is visible in transparent. Left: Packing of 50 monodisperse spheres originally generated by the Skoge et al. code. Right: Packing after adding the required copies for two selected spheres, painted in red and black. Complementary copies of the red and black spheres are colored orange (near the corners) and grey (at the back-right), respectively.

3.3.3. Combination of Core-Shell Particle and Hardsphere Packing Geometries

Calculations of center point vector components for all the elements in reconstructed core-shell particle geometry was explained in the section 3.2.3. Center point vectors of the spheres in generated random packing is used as translation vectors to create all additional copies of calculated core-shell particle geometry. Labeling hardspheres in the packing from 1 to 100 and introducing a new dummy index *l*, representing the number tag of hardsphere inside of which core-shell geometry is copied, center point vectors of all spherical elements in the random jammed packing of core-shell particles can be calculated by Equation (40).

$$\mathbf{P}^{j,i,k,l} = \mathbf{P}^{j,i,k} + \mathbf{P}_{\boldsymbol{hs}}^{l} \tag{40}$$

However, before translating the core shell particles the packing geometry must be scaled such that a core-shell particle with a radius r_p would flush fit inside a hardsphere in the random packing, since the packing generation results in a cube of unit dimensions and spheres with diameter that varies with respect to the desired number of spheres in the packing as mentioned previously. Vectors pointing to the center points of the hardspheres can be scaled by a factor of r_p/r_{hs} and the spheres in the packing would still preserve their shape and relative positions while shrinking or expanding to match the size of the core-shell particle. It is possible because the scaling is basically done based on similar right triangles. The scaling is done by simply multiplying all components of $\mathbf{P}_{\boldsymbol{hs}}^{l}$ by the ratio r_p/r_{hs}.

$$\mathbf{P}_{\boldsymbol{hs}}^{l} = \mathbf{P}_{\boldsymbol{hs}}^{l}\frac{r_p}{r_{hs}} \tag{41}$$

Scaling also changes the length L_{pc} of the periodic cell by the same factor. Since the original cell was a unit cube, scaled length of the cube becomes $L_{pc} = r_p/r_{hs}$.

$$L_{pc} = r_p/r_{hs} \tag{42}$$

3.3.4. Integration of Core-Shell Packing Geometry and Collision Control

Information contained in $\mathbf{P}^{j,i,k,l}$ is complete. The center points of any element in any shell layer of core-shell particle and core spheres of particles in the packing is easily accessible once $\mathbf{P}^{j,i,k,l}$ is calculated for all j, i, k and l. $\mathbf{P}_{\boldsymbol{hs}}^{\boldsymbol{l}}$ can be used along with these data in sequence to carry out collision control around a specific shell-layer only, without having to check any other elements in the geometry.

$$\left(\mathrm{P}_{hs,1}^{l}{}^{2} - X_{L,1}(t+\Delta t)^2\right) + \left(\mathrm{P}_{hs,2}^{l}{}^{2} - X_{L,2}(t+\Delta t)^2\right) + \left(\mathrm{P}_{hs,3}^{l}{}^{2} - X_{L,3}(t+\Delta t)^2\right) \leq {r_{hs}}^2 \tag{43}$$

If the inequality given in Equation (43) holds for any l, the tracer must be traveling inside a core-shell particle. It may have been collided with an element of core-shell particle, or it may still be traveling in the pore space without colliding. If the inequality given in Equation (43) does not hold for every l, then the tracer is in inter-particle space hence a collision did not occur. Let us use the dummy index value l_t for the corresponding core-shell particle tracer travels in.

$$\left(\mathrm{P}_{hs,1}^{l_t}{}^{2} - X_{L,1}(t+\Delta t)^2\right) + \left(\mathrm{P}_{hs,2}^{l_t}{}^{2} - X_{L,2}(t+\Delta t)^2\right) + \left(\mathrm{P}_{hs,3}^{l_t}{}^{2} - X_{L,3}(t+\Delta t)^2\right) \leq {r_{soi,k}}^2 \tag{44}$$

The inequality given by Equation (44) helps determine the shell layer in which the tracer is traveling. If the inequality is tested for different values of $r_{soi,k}$ for $k = 0..n_l$, the first value of k the inequality holds for is the layer where tracer is currently travels in. If the inequality holds for $k = 0$, then the tracer is known to collide with the core sphere, $\mathbf{X}(\mathrm{t} + \Delta\mathrm{t}) = \mathbf{X}(\mathrm{t})$ is set (bounce-back occurs) and there is no need to check other elements for collision. If the inequality holds for $k > 0$, other elements in the corresponding layer, tagged by dummy index value k_t, must also be checked for collision.

$$\left(\mathrm{P}_1^{j,i,k_t,l_t}{}^2 - X_{L,1}(t+\Delta t)^2\right) + \left(\mathrm{P}_2^{j,i,k_t,l_t}{}^{2^2} - X_{L,2}(t+\Delta t)^2\right) + \left(\mathrm{P}_3^{j,i,k_t,l_t}{}^2 - X_{L,3}(t+\Delta t)^2\right) \leq r_s{}^2 \quad (45)$$

If the inequality given by Equation (45) holds for any value of $i = 0..\mathrm{n_{aux,k}}$ and $j = 0..\mathrm{n_{ssa,i,k}}$, then tracer ends up in the corresponding shell-side sphere hence a collision occurs. Then $\mathbf{X}(\mathrm{t}+\Delta\mathrm{t}) = \mathbf{X}(\mathrm{t})$ must be set and tracer must proceed taking its next random step.

After the construction of boundary system, the algorithm changes to:

1. Set n.
2. Set $\mathrm{D_{AB}}$.
3. Set simulation time, $\mathrm{t_s}$.
4. Set Δt and calculate Δl using Equation (14)
5. Determine total number of random steps, using Equation (16).
6. *Calculate* $\mathbf{P}^{j,i,k,l}$ *for all 100 core-shell particles in the packing, using Equation (40).*
7. Set initial positions for all tracers.
8. Generate $\mathrm{n_s}$ amount of unit vectors with random directions.
9. Displace a single tracer, for one step using Equation (15).
10. Calculate $\mathbf{X_L}(\mathrm{t}+\Delta\mathrm{t})$ using $\mathbf{X}(\mathrm{t}+\Delta\mathrm{t})$ in Equation (19).
11. *Use* $\boldsymbol{X_L}(t+\Delta t)$ *and* $\mathbf{P}_{\boldsymbol{hs}}^{\boldsymbol{l}}$ *in Equation (*43*) to determine if tracer travels in inter-particle void space (true) or not (false). If true, go to step 14. Else proceed to next step.*
12. *Use* $\boldsymbol{X_L}(t+\Delta t)$, $r_{soi,k}$ *and* $\mathbf{P}_{\boldsymbol{hs}}^{\boldsymbol{l}}$ *Equation (*44*) to determine in which shell layer the tracer is traveling in. If tracer is in zeroth layer (core sphere), detect collision and go to step 14. Else proceed to next step.*

13. *If step 11 is false, use* $\boldsymbol{X_L}(t + \Delta t)$ *and* $\mathbf{P}^{j,i,k,l}$ *of all elements in corresponding shell layer determined in previous step in Equation (*45*) to check for collision. If collision is detected, set* $\mathbf{X}(\mathrm{t} + \Delta\mathrm{t}) = \mathbf{X}(\mathrm{t})$.
14. Return to step 10 until all tracers have taken $\mathrm{n_s}$ random steps each.
15. Use Equation (17) to recalculate $\mathrm{D_{AB}}$ and compare with initially set value.

A flowchart of the diffusion/dispersion algorithm can be seen in Appendix D.1. In the flow chart, bypassing the steps related to fluid flow or simply assuming a velocity field with velocity vectors of magnitude zero gives the algorithm for diffusion.

3.4. Simulation of Fluid Flow in a Random Packing of Monodisperse Hardspheres in COMSOL

The simulation of fluid flow was carried out using COMSOL. Liquid water at room temperature was selected as the fluid. The system geometry was the visually inspected and modified random packing of hardspheres mentioned in section 3.3.2.

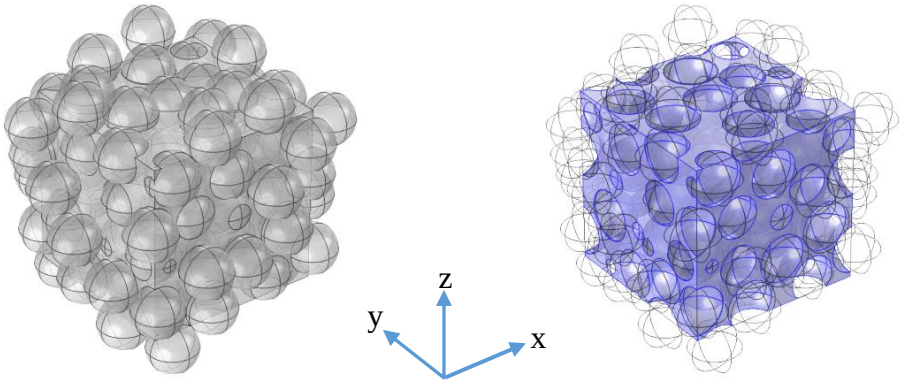


Figure 15: Left: Entire geometry of the system. Right: Fluid domain.

COMSOL allows importing certain CAD file formats to be used as boundaries for the modules. Geometry of the packing can be rendered in a compatible file format by OpenSCAD and imported to COMSOL. However the rendered files contained rough surfaces on hardspheres, preventing mesh generation. This problem related to geometry importing could not be solved, therefore all 100 spheres in the packing were scaled such that diameter of the hardspheres in the packing would be equal to $5\ \mu m$, which is the selected diameter of core-shell particles that were used in dispersion simulations, and all 100 hardspheres were manually added into the geometry. Finally, a cube geometry representing the main periodic cell with dimensions as defined in Equation (42) by $L_{pc} = r_p/r_{hs} = 2.5/0.14 \cong 17.6\ \mu m$, was added to be chosen as the domain that will be defined as the fluid. System geometry is shown in Figure 15, left.

The laminar Flow module was added to the component in model builder, since chromatographic velocities are almost always in the laminar flow region. Fluid domain was selected as the cube, at normal temperature and fluid properties was defined as "From material" belonging to physical property data of liquid water provided by COMSOL. Selected fluid domain can be seen in Figure 15, right.

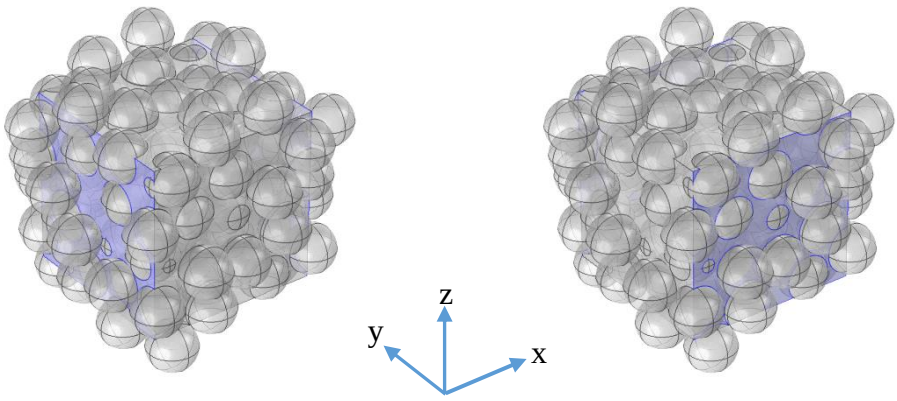


Figure 16: Two sets of periodic flow conditions with zero pressure difference.

Initial values of the velocity field and pressure in the system were set to zero. All boundaries in the fluid domain were set to no slip boundary conditions. Then two sets of periodic flow conditions were added for opposing boundaries perpendicular to xy-plane, that are the faces of the cube, and pressure difference between two pairs of opposing faces of the cube was set to zero Pascals in periodic flow conditions, shown in Figure 16. Finally another periodic flow condition was defined for the opposing pair of surfaces on the cube that are parallel to the xy-plane, as seen in the left of Figure 17, and the pressure difference between these boundaries was set to several different values in Pa, based on typical flow rates in core-shell particle operations as illustrated in more detail in the section 4.2.1 of the next chapter. Inlet and outlet boundary conditions were disabled. Selected boundaries in the three periodic flow conditions override previously defined selection of no slip boundaries and hardspheres become the only no slip boundaries in the system. The results for all pressure difference values in periodic flow conditions with non-zero pressure drop are given and discussed in sections under 4.2. in Chapter 4.

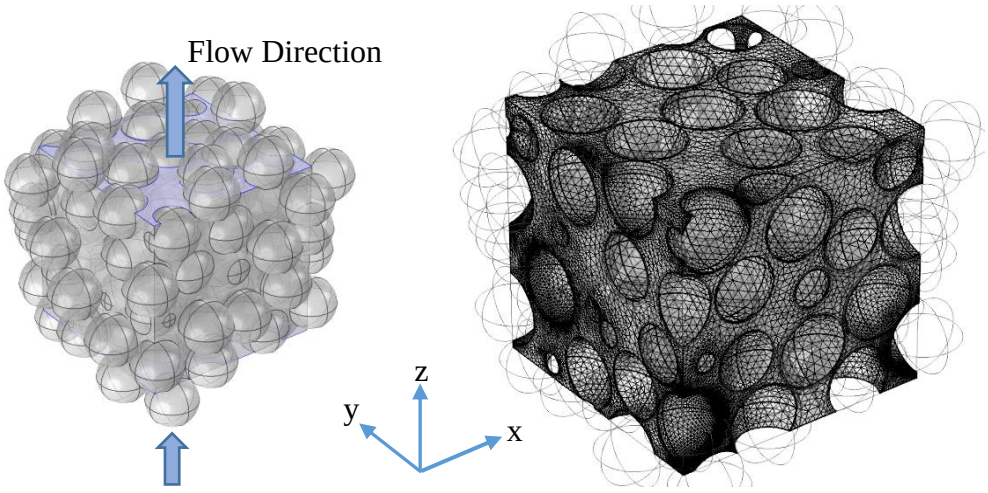


Figure 17: Left: Periodic flow condition with a set ΔP. Right: Fine mesh generated by COMSOL Multiphysics.

Meshes used in finite-element iterations are left in control of COMSOL, with a 'Fine' element size. Finally, a stationary study was added into the model builder and the steady-state velocity fields were obtained.

3.5. Integration of the Diffusion and Fluid Flow

Integrating separately developed random-walk diffusion model and velocity field obtained from the numerical continuum solution of flow equations in the system creates a particle tracking model that can explain dispersion. Recall that, Fokker-Planck equation given by Equation (9), under steady-state flow conditions and with a constant diffusion coefficient throughout the system becomes analogous to advection-diffusion equation and the random-walk diffusion equation given in Equation (15) with the addition of displacement due to external velocity field resembles an Euler approximation of Fokker-Planck Equation in the form:

$$\mathbf{X}(\mathrm{t}+\Delta\mathrm{t}) = \mathbf{X}(\mathrm{t}) + \boldsymbol{\vartheta}\big(\mathbf{X}(\mathrm{t})\big)\Delta\mathrm{t} + \boldsymbol{\xi}\Delta l \qquad (46)$$

This simple modification to Equation (15) converts entire diffusion model into a dispersion model that can use the same system geometry and collision control mechanism. Displacement due to the velocity field requires the velocity vector components at the current position of the tracer, $\boldsymbol{\vartheta}\big(\mathbf{X}(\mathrm{t})\big)$, that can be obtained by tri-linear interpolation of the nearest velocity vectors around and with respect to the local position of the tracer. Interpolation subroutine prepared for the study is explained in detail, in Appendix C.5. A flowchart of the dispersion algorithm can be seen in Appendix D.1.

Longitudinal displacement data is collected at a certain frequency for each tracer during the calculations for each tracer. The displacement data of the tracers is then easily converted to the variance of the longitudinal displacements of the entire tracer

ensemble to obtain a time-dependent longitudinal position variance. Then the time-dependent longitudinal position variance can be fitted an asymptotic function (explained in section 3.6.5) to find the asymptotic longitudinal dispersion coefficient of the tracer ensemble using the relation given in Equation (2). Dispersion coefficients can also be converted to the more appropriate plate height and reduced plate height values commonly used in chromatography terminology for referring to dispersion, using Equation (1).

3.6. Software Implementation of the Model

Implementation of the random-walk free diffusion model, computation and storage of core-shell packing boundaries, adapting random-walk model and preparing an appropriate collision control mechanism for the calculated system boundaries, storage of the external velocity field in computer memory, an adapted tri-linear interpolation subroutine for the stored velocity field and integration of the external velocity field with the random-walk model using free form Fortran are all explained in detail in Appendix C.

Following sections only includes some of the important remarks on the software implementation of the diffusion and dispersion models around core-shell particles. Reader is suggested to read the appendices for detailed information related to the programming done for this specific work.

3.6.1. Free Molecular Diffusion

The most important aspect in the implementation of random-walk diffusion on software is the random number generation. Computers cannot generate true random numbers. Instead, they use special subroutines that can generate uniformly distributed pseudo-random numbers based on an initial seed and every other pseudo-random

number is generated by using the previous one as the new seed. Therefore the generated pseudo-random numbers are only seemingly random if the numbers are used in a sequence. Considering this, random-walk must proceed until a tracer takes required number of random steps to complete the duration of the simulation before the next tracer begins its random-steps. Otherwise the unit vector with random direction, $\boldsymbol{\xi}$, in the Equation (15) might not contain random values at all.

In this work, the total displacements, or the transverse/longitudinal displacements, of individual tracers were saved to an independent matrix in a certain frequency during the execution of random-walk. After each and every tracer finishes all random-steps, this time-dependent displacement data was then converted into more convenient time-dependent diffusion coefficient -or time-dependent longitudinal position variance (which is then used for estimating longitudinal dispersion coefficients) for the case of dispersion- of the tracer ensemble. For free diffusion, time-dependent diffusion coefficient is expected to be not deviating from the input value except for the random fluctuations due to the probabilistic nature of the random-walk method.

3.6.2. Computation and Storage of Impermeable Boundaries

Previously, a method was proposed explained in detail how an ideal core-shell particle can be reconstructed by using basic principles of analytical geometry by using 3 user-defined parameters; core-to-particle ratio (φ), number of shell layers (n_l) and core-shell particle radius (r_p). Proposed method allows systematic calculation of the geometry elements. Total amount of elements in the system geometry as well as the exact positions of each and every element including in which core-shell particle and which shell layer they are located in are very well known after the calculations. This is very important for developing an efficient collision control mechanism for the model. Geometry data therefore, is systematically stored in higher-dimensional matrices allocated enough computer memory.

3.6.3. Adapting the Free Molecular Diffusion Code to Simulate Impermeability

In the section 3.1.2, basic principles of collision control which uses mathematical expressions that define the geometry of impermeable boundaries was explained. Later, a method that uses basic principles of analytical geometry was presented for digital reconstruction of a core-shell particle as spherical elements and this method was implemented on computer using Fortran programming language to finally obtain a four dimensional array that contains every parameter that defines the impermeable boundaries in a system of core-shell particles in a random jammed packing. The objective is to adapt the basic collision control mechanism in an efficient way so that impermeability at any point in the system is properly simulated.

A sphere centered at the point (x_0, y_0, z_0) with a radius r is defined by the sphere equation.

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2 \tag{47}$$

Then, the inequalities given in Equations (48) and (49), represents all points inside and outside the sphere defined by Equation (47), respectively, as well as the surface of the sphere itself.

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \leq r^2 \tag{48}$$

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \geq r^2 \tag{49}$$

These inequalities are used in a control structure to determine whether a tracer is inside any of the spherical elements that create the impermeable boundaries or not, by replacing (x_0, y_0, z_0) with the components of center point vectors and r by the radius of a specific sphere element, and dependent variables x,y,z by the local position $\mathbf{X_L}(\mathrm{t} + \Delta\mathrm{t})$ of a tracer. In order to make sure that a tracer is not in an impermeable volume hence a collision could not have occured, all sphere elements in the system must fail to satisfy the inequality given in Equation (48). However, one might appreciate the extraordinarily large amount of spherical elements that need to be tested before no

collision is detected. Therefore, the following algorithm is suggested in order to decrease total amount of control calculations needed to check all sphere elements.

1. Use $\mathbf{X_L}(\mathrm{t}+\Delta\mathrm{t})$, r_{hs} and `RJP` in Equation (48) for all hardspheres one by one, determine if tracer resides in any of the hardspheres in random jammed packing.
    a. If Equation (48) is not satisfied by any hardsphere, tracer is in inter-particle void space. Proceed to next random step.
    b. If Equation (48) is satisfied by any hardsphere, tracer resides in that hardsphere and might be in one of the core-shell particle elements. Proceed to next step of the collision control algorithm.
2. Use $\mathbf{X_L}(\mathrm{t}+\Delta\mathrm{t})$, $r_{soi,k}$, and `RJP` elements that belong to corresponding hardsphere detected in previous step, in Equation (48) for all spheres of influence one by one, determine the shell layer tracer currently resides in. The first sphere of influence that satisfies Equation (48) belongs to the shell layer that hosts tracer. Proceed to next step of the collision control algorithm.
3. Use $\mathbf{X_L}(\mathrm{t}+\Delta\mathrm{t})$, $r_{soi,k}$, and `CSPRJP` elements that belong to corresponding shell layer detected in previous step, in Equation (48) for all shell side elements in that shell layer one by one.
    a. If Equation (48) is not satisfied by any of the shell layer elements, tracer is in the pore space of core-shell particle hence no collision occured. Proceed to next random step.
    b. If Equation (48) is satisfied by any of the shell layer elements, tracer resides in the impermeable volume of that element hence a collision is detected. Stop checking remaining boundary elements and use bounce-back method and set $\mathbf{X}(\mathrm{t}+\Delta\mathrm{t}) = \mathbf{X}(\mathrm{t})$ to return tracer to its last known position outside impermeable volume. Proceed to next random step.

3.6.4. Storage of Velocity Field

COMSOL Multiphysics can create a data table of the velocity components of the vectors in the velocity field, on a user defined regular grid. For this work, a regular grid with 101 nodes in each dimension (one node per 1/100 of the periodic cell length) was used. The exported text file that contains the data table was around 200 mb in size. Number of nodes used in the grid and the size of the exported file has a cubic relation, for example using 10 times more nodes in each dimension would require 1000 times more hard disk space (200 GB, in this case) as well as memory considering the velocity field must be read and written to memory during the execution of the code. Therefore this must be considered while extracting the velocity field from the application. The data table also contains velocity components that cannot possibly be read by Fortran code, due to its incompatible format with the Fortran. This might also be the case for other different programming languages, therefore the table must be first arranged into a compatible format by using a different platform. Octave was used in this work to convert the velocity field data into a Fortran-compatible format.

3.6.5. Adaptation of Diffusion Program to Simulate Dispersion

The dispersion code essentially differs from the diffusion code only by the reading storage of the external velocity field and the added displacement term in the random-walk equation ($\left[\boldsymbol{\vartheta}\left(\mathbf{X}(t)\right)\Delta t\right]$ term in Equation (46)). Calculation of the core-shell packing geometry and collision control are the same as in the diffusion model. However the interpolation algorithm (explained in Appendix C.5) requires an additional calculation of local position prior to the random and flow displacement of the tracer. Integration of the random-walk diffusion model, the velocity field and interpolation subroutine using Fortran is explained in detail in Appendix C.6.

For this study, random-step size of tracers were selected as either 10% of the diameter of shell-side spheres or 1% of the entire core-shell particle diameter, whichever is the

smallest. While selecting a more refined random-step size, one should be aware of the fact that random-step size, Δl, is inversely proportional to the square root of the time increment, Δt, which directly influences the total amount of random-steps each tracer needs to take for the duration of simulation hence the wall-clock time of the execution. For example, halving the value of Δl increases the wall-clock time four times and such refinements on Δl might quickly lead to extremely long execution times for the computations given that the computer being used has enough memory to store required random-numbers.

Longitudinal dispersion coefficient is not directly calculated by the Fortran program. The reason behind it is related to the probabilistic nature of the model, as well as the actual nature of hydrodynamic dispersion. Random-walk creates fluctuations in variance, making it very hard to estimate over short time intervals. Longitudinal dispersion coefficient is by definition proportional to the time derivative of longitudinal displacement variance and it increases until reaching an asymptotic value. Therefore variance data was manually fitted an asymptotic function with decaying positive slope using a spreadsheet calculator, to estimate the time derivative of longitudinal displacement variance of tracer ensemble. Fitted function is given in Equation (50).

$$\mathrm{F(t)} = \mathrm{A}(\mathrm{t} + \frac{1}{\mathrm{k}}\mathrm{e}^{-\mathrm{kt}} - \frac{1}{\mathrm{k}}) \tag{50}$$

$F(t)$ is the integral of function $f(t)$, given in the following equation, with boundary conditions $\mathrm{f(0)} = 0$ and $\mathrm{f(t)}_{t\to\infty} = A$. The parameter A is equal to $\delta\sigma_L^2/\delta t|_{t\to\infty}$ and the parameter k is the decay rate of the increase in $\delta\sigma_L^2/\delta t$.

$$\mathrm{f(t)} = \mathrm{A}(1 - e^{-kt}) \tag{51}$$

Non-linear solver available in Excel can find the best fitting parameters to the variance data and the longitudinal dispersion coefficient is equal to half of the asymptotic time slope of the σ_L^2, which is finally estimated as $A/2$. It is important to point out that, the

parameters A and k does not necessarily have physical significance. They are just fitting parameters to estimate the asymptotic slope of the time-dependent variance.

3.6.6. Parallelization of Diffusion and Dispersion Programs

Due to the nature of random-walk and particle tracking methods, they are very intense iterative approaches for modelling these phenomenon. Regular serial execution of the programs written to carry out iterations might take very long time. Parallel computing can partially reduce the wall-clock times of the codes by distributing the computational load amongst available CPU threads. The model built in this study is very suitable for parallel computing. There is, however, a very important point that needs to be considered during the parallelization of the program that is the pseudo-random number generation. Fractions of the code that carries out the random number generation must be excluded from the parallel-worksharing region of the code (in other words, a thread must wait until the other thread completes generating its sequence of numbers) to avoid disruption in the sequence of pseudo-random numbers generated for random-displacements of the tracers. Otherwise, individual threads might 'steal' some of the pseudo-random numbers from the random number sequence of another thread, risking tracers to move according to a sequence of numbers that are not uniformly distributed. See Appendix C.7 for more details about the parallelization of the Fortran codes written for this study.

3.6.7. A Summary of Interactions Between Software Components

The dispersion model in its final state mainly depends on the Fortran code written in the study, however there are still other software, COMSOL, Octave, random packing generator, OpensCAD and a spreadsheet calculator (MS Excel) used in either in the process or obtaining the results. A flowchart representation of the overall work done

from the beginning to obtaining final results can be seen in Appendix D.3. Tasks taken on by each software can be summarized as follows.

Fortran code,

- Calculates core-shell particle geometry.
- Uses modified random jammed packing of hardspheres to generate core-shell packing.
- Carries out the actual simulation of dispersion.

Random packing generator written by Skoge et al. (2006),

- Generates random jammed packings of monodisperse hardspheres in periodic unit cells.

OpenSCAD is used for,

- Visual inspection of original random packing of hardspheres and its modification
- Visual inspection of created core-shell particle geometry, modified random packing, as well as the core-shell packing geometry.

COMSOL Multiphysics,

- Uses modified random jammed packing of monodisperse hardspheres to simulate fluid flow.
- Creates data tables for the velocity field obtained from the fluid flow simulation.

Octave,

- Re-organizes the data table into a certain format that can be used in Fortran.

MS Excel is used for,

- Modification of the original random packing of hardspheres to include invading spheres from neighboring periodic cells.

- Calculating dispersion coefficients from the variance data obtained from dispersion model by fitting an asymptotic function to the data points.

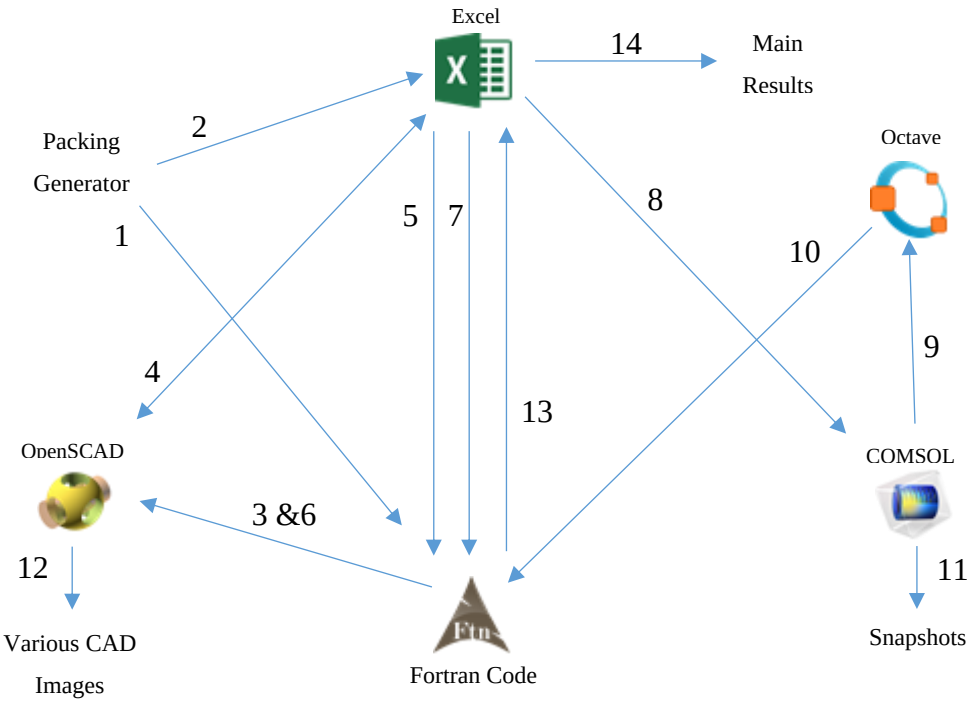


Figure 18: Interaction chart summarizing input/output relations between different software.

Different software interactions are visually demonstrated in Figure 18. Relations numbered in the figure are explained below.

1. Generated packing is sent to and read by a Fortran code
2. Generated packing is imported to Excel for calculating extreme points on hardspheres and detecting invading spheres.

3. Fortran generates a script that commands OpenSCAD to create an image of the generated packing.
4. OpenSCAD image along with extreme points of hardspheres calculated in Excel is used to visually determine how many copies of invading spheres need to be added to packing geometry.
5. Final modified packing geometry is read by Fortran code.
6. Fortran generates a script that commands OpenSCAD to create an image of the generated packing. Image is investigated, making sure hardspheres are properly replicated near periodic boundaries.
7. Modified packing geometry is read by dispersion code, and the packing of core-shell particles is created.
8. Modified packing geometry is used in COMSOL to create the system geometry in COMSOL simulation.
9. COMSOL creates a data table for the velocity field and the data table is sent to Octave. Octave reads the velocity field and re-organizes it into a certain format that can be used in Fortran.
10. Dispersion code reads velocity field from data file re-organized by Octave, proceeds simulating dispersion in the random packing of core-shell particles.
11. COMSOL is used for taking snapshot of the system geometry as well as contour plots and velocity vectors.
12. Various CAD images of core-shell particle and packing geometry is created.
13. Dispersion is simulated, variance data of tracer displacements is output into a file.
14. Variance data is imported to Excel to calculate dispersion coefficients and reduced plate heights as the final results.

CHAPTER 4

RESULTS AND DISCUSSION

4.1. Simulation of Diffusion in Stagnant Media

4.1.1. Validation of the Free Diffusion Program

Fortran program that simulates free molecular diffusion was tested by running it for various combinations of tracer population and set Δt values, for a one second diffusion event.

Diffusion coefficients were calculated every $0.001\ s$ and saved during simulation to obtain time-dependent diffusivity data containing 1000 data points. Simulation was repeated for a tracer population $N = 1000$ using a set of different time steps $\Delta t = (10^{-3}, 10^{-4}, 10^{-5})$.

$$D_0 = D_{AB}(t)/D_{AB} \tag{52}$$

In an unobstructed environment, diffusion coefficient is expected to be constant. Figure 19 shows normalized time-dependent diffusion coefficients, D_0 defined by Equation (52) for simulations carried out at different time increments. Sample means of D_0 is 1% different from unity for all Δt. Choosing a finer Δt only reduces the variance of the predictions.

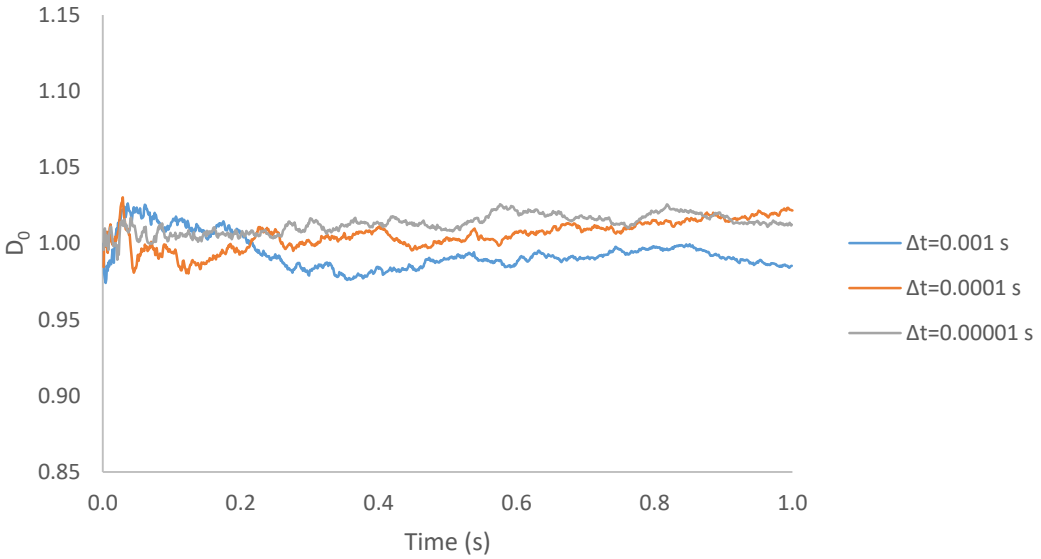


Figure 19: Normalized time dependent diffusion coefficients predicted by the model with respect to time. Results are for three different time steps, and a tracer population of 4000. Legend shows Δt values in seconds used for corresponding data set.

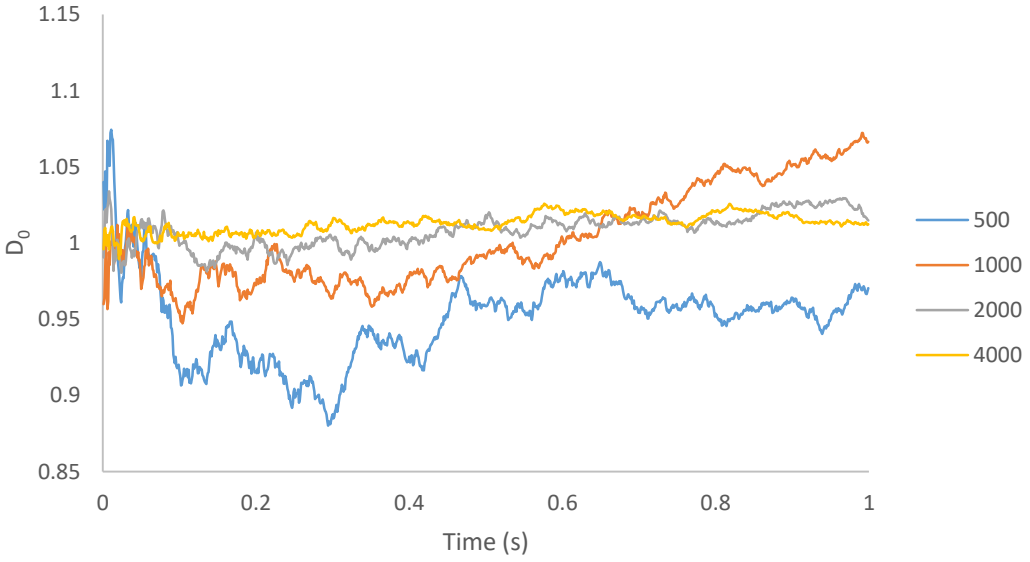


Figure 20: Normalized time dependent diffusion coefficients predicted by the model with respect to time. Results are for $N = (500,1000,2000,4000)$ and $\Delta t = 10^{-5}s$. Legend shows N values used for corresponding data set.

Free diffusion simulations were also conducted for a constant $\Delta t = 10^{-5} s$ and for 4 different tracer populations. Time dependent diffusion coefficient data was again normalized by the input value of D_{AB}.

Figure 20 shows that using larger tracer populations increases the precision of prediction as the results for the run where $N = 4000$ has, by far, the least standart deviation (0.006) compared to other simulations with 2000, 1000 and 500 tracers (0.011, 0.032 and 0.028 respectively). Mean values of diffusion coefficients are also shown in the figure. Maximum deviation of average D_0 from the unity is 5% in the simulation with $N = 500$, which can be expected from a coarse simulation using such a small population. The other runs with higher tracer populations have around 1% error from the expected value of D_0. Considering the decreasing standard deviations and error in higher tracer populations, the free diffusion algorithm and the corresponding Fortran code can be confirmed to accurately simulate diffusion.

4.1.2. Validation of Periodic Boundaries and Collision Control

Calculation of local tracer positions and the validity of periodic boundaries was tested by preparing a diffusion simulation with a single spherical boundary defined in the main periodic cell. Main periodic cell was defined as a cube with $5\ \mu m$ dimensions and a sphere $d = 5\ \mu m$ in diameter and centered in the geometric middle of the cube was used as the system. Diffusion of 5000 tracers, all placed at the origin with a point injection type initial condition, was simulated for 0.5 seconds of real time by setting $\Delta l = 0.5\ \mu m$ and calculating Δt accordingly. The last collision sites, positions of tracers before they are returned to diffusion domain by the bounce back method, between each tracer and the impermeable sphere was saved and visualized in a 3-D scatter plot.

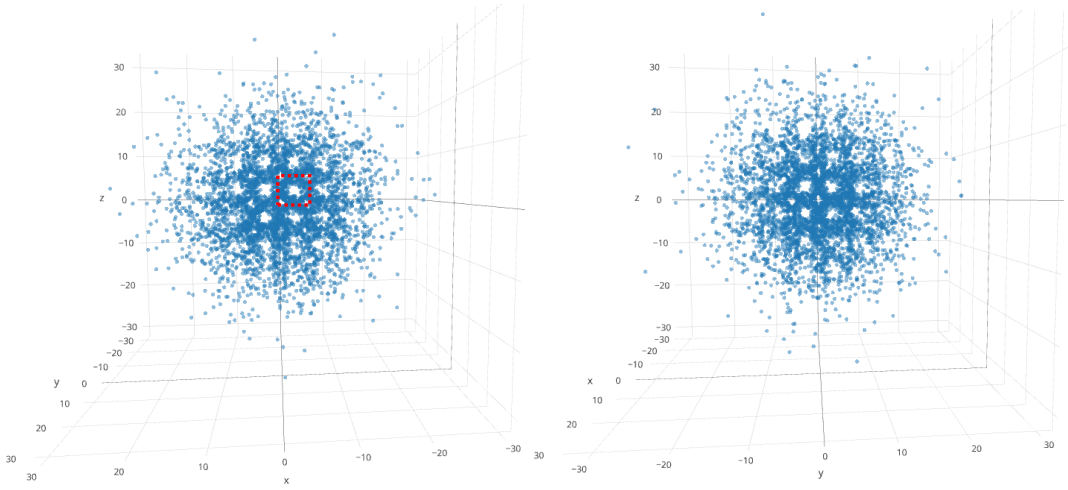


Figure 21: Two different side views of final collision sites between every tracer and the packing produced by the periodic boundaries from a single impermeable boundary defined in the main periodic cell. Units for all axis are in μm.The red frame indicates the scale and approximate position of a single periodic cell in the system.

Spherical impermeable volumes surrounded by collision sites are very distinct in the 3-D scatter plots given in Figure 21, as well as the smaller available volume between the spheres. These visuals can confirm that periodic boundaries are successful at creating an infinite arrangement of boundaries defined in the main periodic cell. Simulation was also repeated for a homogeneously distributed injection of tracers into the available space in the main periodic cell, to compare the local collision sites around the sphere located in the main periodic cell.

Point injection type initial conditions inevitably cause all tracers to randomly move in the same grid due to the uniform magnitude of random-displacement Δl, therefore collision sites for all tracers around the impermeable boundary lay on the same fixed points around the sphere. Figure 22, left, shows very sharp laddering occurring around the impermeable sphere for a simulation with point injection. However, when the initial positions of tracers are randomly distributed throughout the space in the periodic cell that allows diffusion, laddering is smoothened since each and every tracer gets a different random movement grid. Figure 22, right, clearly demonstrates the effect of

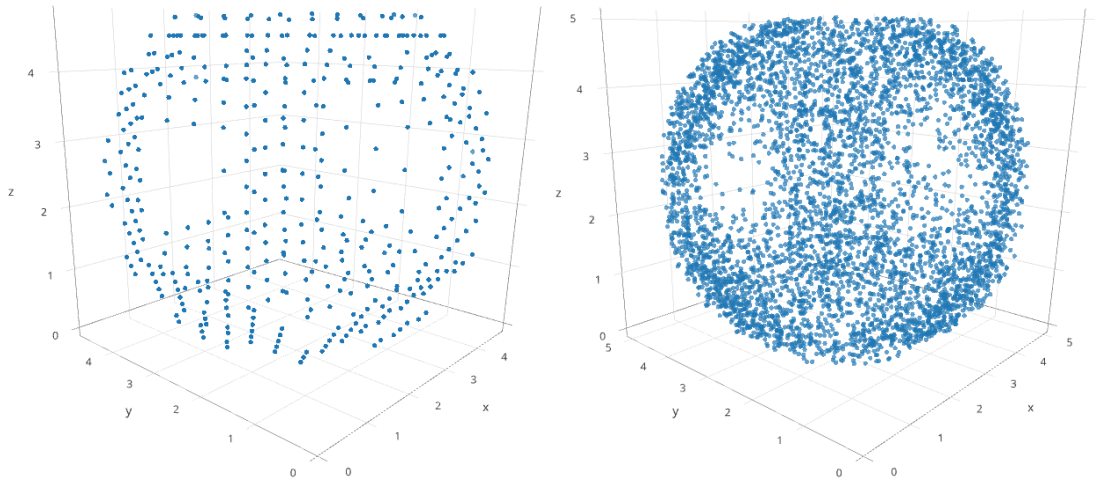


Figure 22: Local collision sites around the boundary defined in the main periodic cell. Left: Collision sites in point injection simulation. Right: Collision sites in distributed injection simulation. The box corresponds to the dimensions and the position of the main periodic cell.

choosing a distributed injection type initial condition on the geometry sampled by tracers. Sparse collision sites near the periodic cell faces, shown in Figure 22, are due to defined Δl value. These regions are narrower than the random-step size, consequently preventing a large portion of tracers to sample them in case of distributed injection, and all tracers in case of point injection. This effectively reduces the void fraction and tortuosity of the system. Since the narrow regions are not sampled by tracers, volume of these regions are practically treated as an impermeable zone.

Transient diffusion coefficients normalized by free diffusion coefficient D_{AB}, as described in Equation (52), in the same simple cubic equivalent system geometry was also obtained from simulations with Δl values of $d/10$, $d/20$, $d/30$ and $d/40$ in order to observe the effect of excluded free volume near the narrow regions between spheres due to usage of coarse step sizes. Asymptotic normalized diffusion coefficient D_0 was found to be 0.60, 0.66, 0.67 and 0.71 for simulations with Δl values of $d/10$, $d/20$, $d/30$ and $d/40$ respectively. D_0 vs. time graphs are shown in Figure 23.

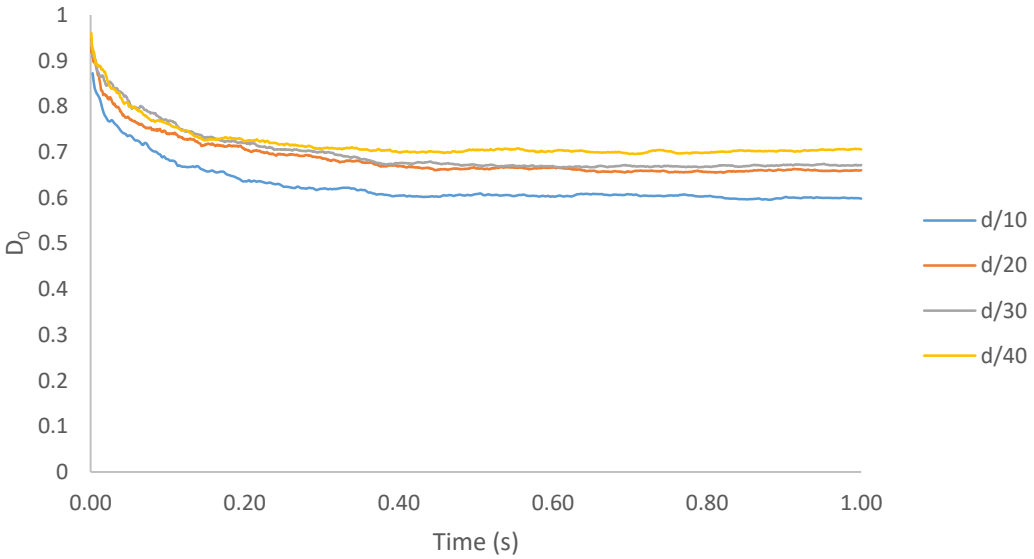


Figure 23: Normalized transient diffusion coefficients predicted by the model in simple cubic equivalent periodic cell, for random-step sizes between $\Delta l = d/10$ and $\Delta l = d/40$.

The results for the simulations clearly show the effect of dead volume inadvertently generated due to the selection of a large Δl. Smaller Δl reduces the dead volume, hence the higer effective diffusivity predicted by the model. Kim and Chen (2006) conducted a similar study on the prediction of effective diffusion coefficients in ordered and random packings of spheres by random-walk simulations. They have found, for simple cubic arrangements of spheres, normalized effective diffusivity must be approximately 0.72 when random-step size $\Delta l = d/100$ below which -they claim- dead volumes become very negligible. The model successfully predicts similar results to the independent study of Kim and Chen. However a coarse random-step size had to be selected for the diffusion and dispersion simulations in the packings of core-shell particles, results of which are discussed later, due to memory and processing power limitations in this study. Since Δt is proportional to the square of Δl, finer choices for random-step size can quickly lead to incredibly small Δt and consequently very large amount of random steps. Therefore memory requirement and wall-clock time of the

simulations are also increased proportionally, thus halving Δt will quadrouple the required memory and wall-clock time. Fortunately, the consequences are consistent and reproducible, and can be taken into account while comparing model predictions to any available experimental data.

4.1.3. Validation of Core-Shell Particle Geomtery

An analytical core-shell particle geometry was proposed and an analytical geometry approach was used to present a procedure for calculating each and every element that collectively makes up a core-shell particle. This procedure was implemented on computer by Fortran and the Fortran code that calculates the geometry of a single core-shell particle was tested by visual inspection of core-shell geometries with different properties created by the code. The code was compiled into a dedicated program that only calculates the geometry and uses a format resembling OpenSCAD syntax to write a script that can be read by OpenSCAD. Following code generates the script and saves it to a text file.

```
OPEN    (#,    FILE="CORE-SHELL    SCRIPT.TXT",    STATUS="UNKNOWN",
ACTION="WRITE")

DO I=1,NOL
  DO J=1,SC(J)

        WRITE(#,*)"translate([", CPGCC(I,J,1), "," , CPGCC(I,J,2),&
        & ",", CPGCC(I,J,3), "])"
        WRITE(#,*)"sphere(", CPGCC(I,J,4), ", $fn=10);"

  ENDDO
ENDDO
```

Figures 24, 25 and 26 shows section views of core-shell particles 5 μm in diameter with core-to-particle ratios 0.7 and 0.8, and with a single, double and triple layers respectively. Sections are taken as two symmetric octants of the particle geometry. Cross sections of cut elements are in green color.

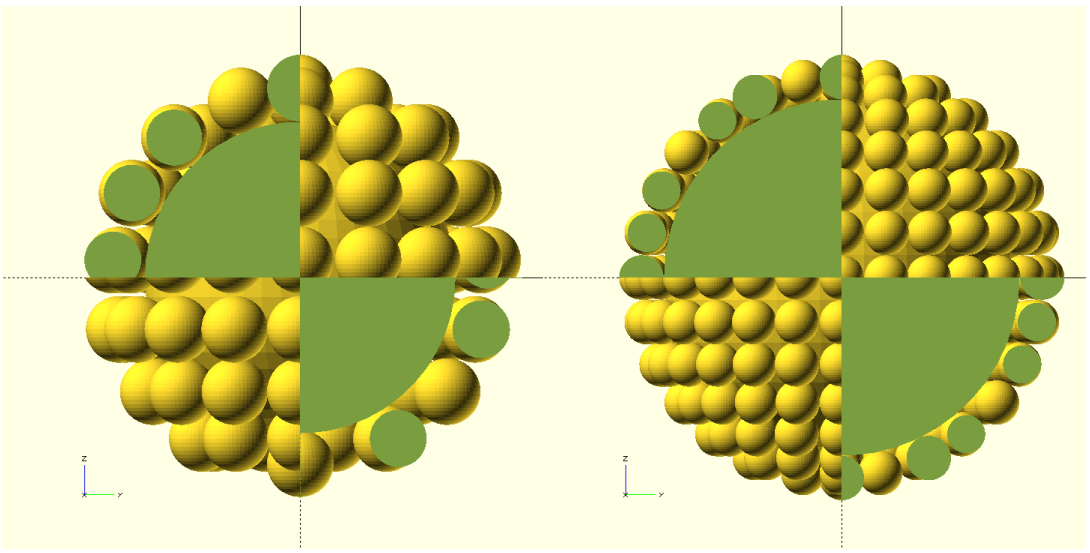


Figure 24: Section views of single layer core-shell particles. Left: Particle with $\varphi = 0.7$. Right: Particle with $\varphi = 0.8$.

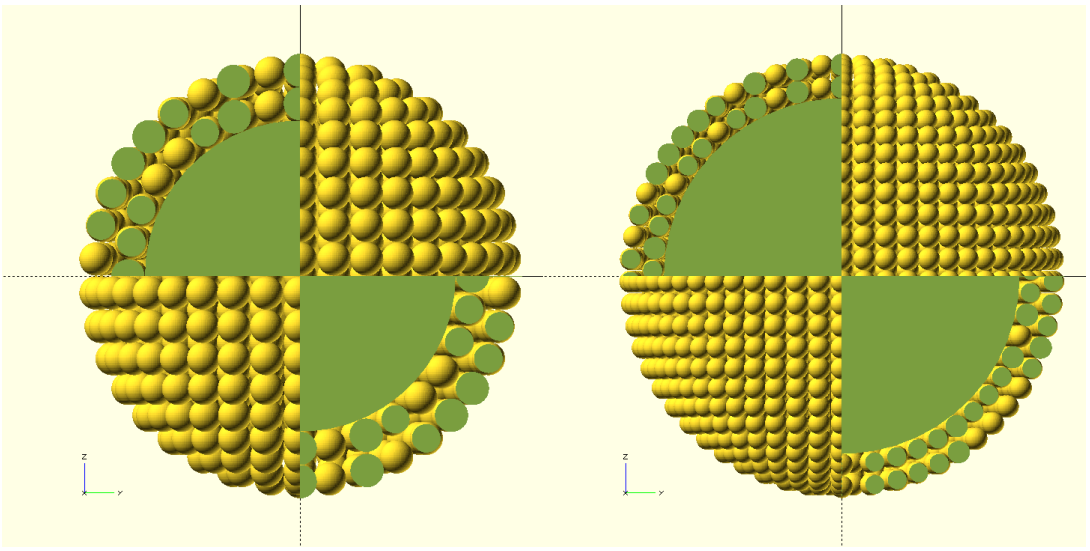


Figure 25: Section views of double layer core-shell particles. Left: Particle with $\varphi = 0.7$. Right: Particle with $\varphi = 0.8$.

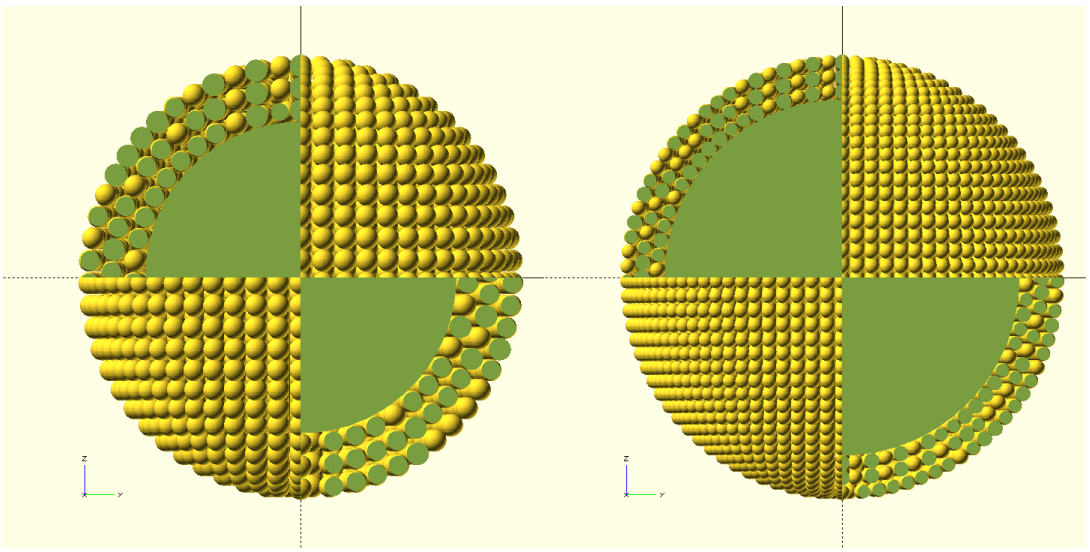


Figure 26: Section views of triple layer core-shell particles. Left: Particle with $\varphi = 0.7$. Right: Particle with $\varphi = 0.8$.

Visuals clearly show that the geometry calculations are done by the Fortran code as intended. Shell side spheres do not overlap or cross over to another shell layer or to core sphere. Layers are clearly distinguishable. Defining input parameters Ψ, r_p and n_l seem to be working properly to determine the final shape of the core-shell geometry, as the code is able to create particle with different Ψ and n_l, and radii of shell side spheres are appropriately determined. Therefore the algorithm and the code for re-construction of core-shell particles is safe to be used in diffusion and dispersion codes.

Shell porosity, and the entire particle porosity of the re-constructed geometries were also calculated. Since the amount of shell spheres are known, calculated as a necessary parameter for array allocation during runtime, the pore volume of the core-shell particle can simply be calculated by substracting the volumes of all elements in the geometry from the volume of outer-most sphere of influence. Similarly, pore volume of the shell layer only can be found by substracting from the shell volume only.

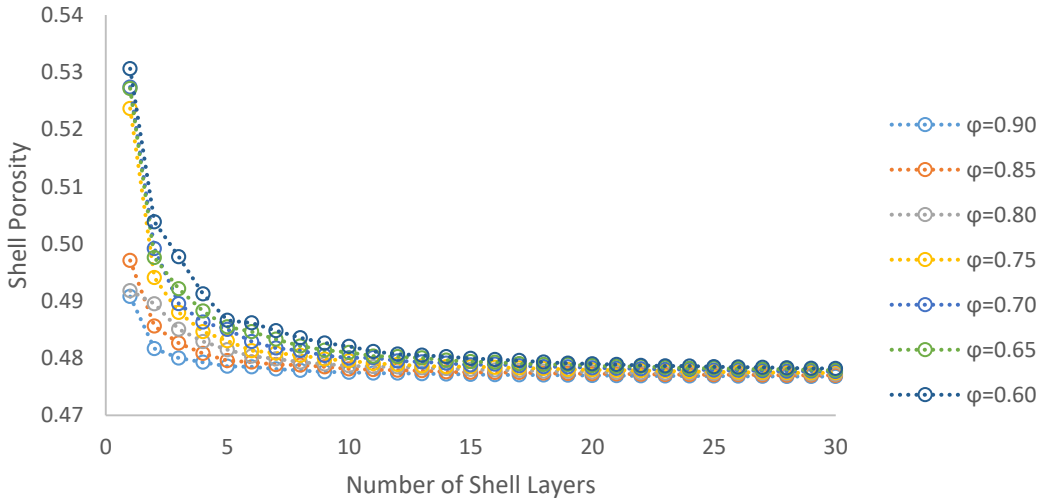


Figure 27: Shell porosity of a core-shell particle with certain φ values vs. the amount of shell layers it has. Note the convergence of shell porosity to 0.477 as N approaches to infinity.

Single layer core shell particles have a coarser structure compared to real core-shell particles. Consequently they would have an overestimated porosity. Real particles typically have multiple shell layers, and Figure 27 clearly shows how shell porosity can rapidly change especially at a small number of shell layers and smaller core-to particle ratios. In order to obtain a more accurate approximation of the real geometry, at least 2 or 3 shell layers must be calculated. A very large number of layers would also lead to more unrealistic results, since the shell layers of a real core-shell particle can only be made so thin with the current production methods, also it will lead to very large amount of shell side spheres in the layers and will drastically slow down the collision control since every spherical element in the layer is checked for collision according to the algorithm. Figure 27 also shows an interesting result, where the shell porosity converges to a certain value as the amount of shell layers is increased. Converged porosity is ~0.477, which is the porosity of a simple cubic packing of monodisperse hardspheres. As the number of layers in the calculated geometry approaches to infinity, the radius of shell side spheres becomes closer and closer to

zero and the curvature of core sphere becomes less and less ‘apparent’ to shell spheres. As a result, the shell side starts acting like a simple cubic packing of spheres.

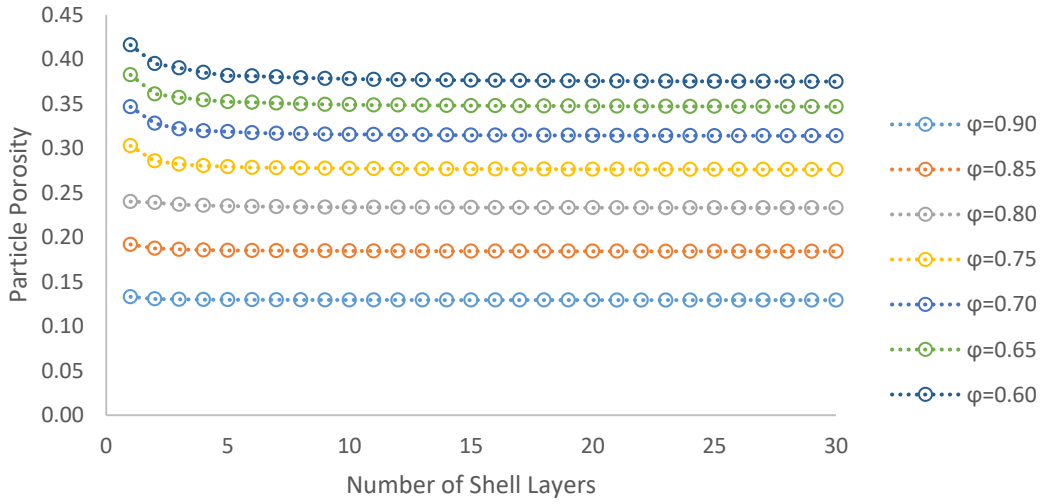


Figure 28: Entire porosities of core-shell particle with certain φ values vs. the amount of shell layers it has.

Porosity of the entire particle takes the dead volume of non-porous core sphere into account, therefore is different for different core-to-particle ratios, φ. Figure 28 shows how the porosity of the entire particle changes with respect to amount of shell layers for core-shell particles with different core-to-particle ratios. Particles with higher φ is better represented by a greater amount of shell layers for a better estimation of the porosity of a real core-shell particle. Dependency of the particle porosity on number of shell layers especially becomes more noticeable for $\varphi < 0.8$ since shell volume becomes greater than the volume of core sphere at $\varphi \cong 0.794$, hence the shell porosity starts having more impact on the porosity of entire core-shell particle. Therefore, after this point, it is more reasonable to segment the shell-side into several layers of spheres rather than a single layer of comparatively large spheres to avoid distortion in porosity values. For example, a core-shell particle with $\varphi = 0.75$ has a shell thickness of $(1-\varphi)r_p = 0.25r_p$ and it can be represented as a core-shell geometry with $n_l = 0.25r_p/0.1r_p = 2.5 \cong 3$ shell layers to avoid distortion in porosity values.

4.1.4. Validation of Core-Shell Packing Geometry

Random jammed packing of core-shell particles calculated by the model was visually inspected in a similar manner to the packing of hardspheres as described in section 3.3.2. and individual core-shell geometries in the previous section. Two packing geometries containing core-shell particles with $\varphi = 0.7$ and $\varphi = 0.8$ were calculated and a Fortran code was used to generate an OpenSCAD script, using the information contained in `CSPRJP` array (Refer to Appendix C for the details on Fortran strings used in the codes), which are the center points and radii of all elements in the packing geometry. Core-shell particles were set to have only a single layer, for multi-layer particles contain very large amounts of spherical elements that needs to be rendered by the computer. A packing containing triple layer core-shell particles with $\varphi = 0.7$ contains around 270,000 geometry elements which could not be rendered by the available computers, whereas the packing with single layer core-shell particle that has $\varphi = 0.8$ contains around 25,000 elements and the computer was able to render it although changing the view incredibly slowed the computer.

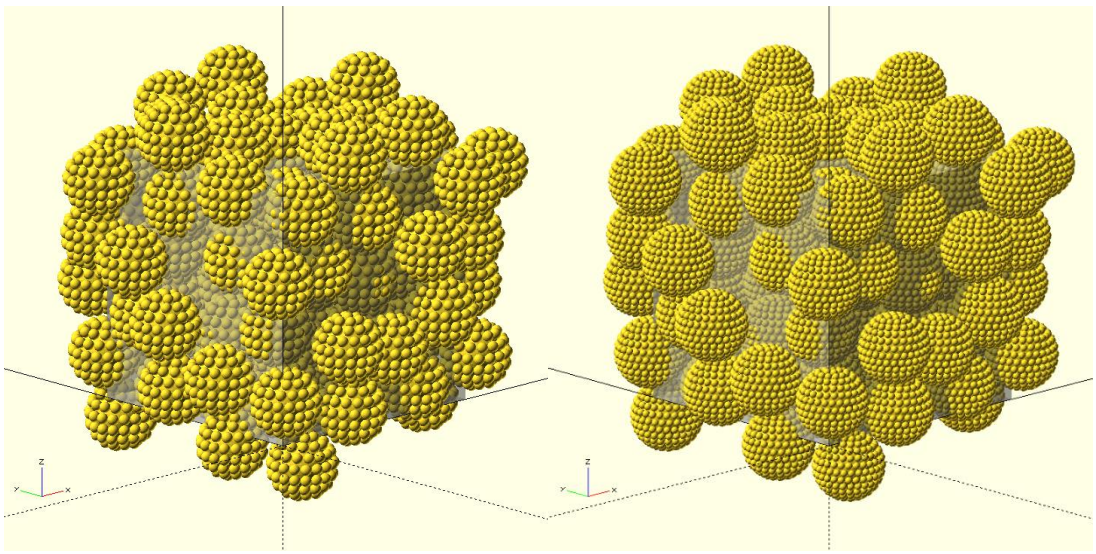


Figure 29: Random jammed packings of 100 single layer core-shell particles with $r_p = 2.5\ \mu m$. The main periodic cell is visible in translucent grey color. Left: Core-shell particles with $\varphi = 0.7$. Right: Core-shell particles with $\varphi = 0.8$.

Inspected packings of core-shell particles are seen in Figure 29. Renderings do not show any overlaps between core-shell particles or any abnormal large distance between them, an indication for successful scaling of hardsphere packing and successful translation of calculated core-shell particle geometry into the scaled packing of hardspheres. These visuals confirm that the system boundaries are calculated and stored properly. Hence they can be used in diffusion and dispersion simulations.

4.1.5. Diffusion in Random Jammed Packing of Core-Shell Particles

Diffusion of solute tracers were simulated in the packing of core-shell particles with $\varphi = 0.77$ with 1 and 2 shell layers, and $\varphi = 0.655$ with 3 shell layers was simulated. Bulk diffusivity of tracers were set as $110 \mu m^2/s$, a typical diffusivity for small proteins based on lysozyme (Bauer et al., 2016). Simulation was run for two different particle diameters, 5 μm and 3.4 μm based on the dimensions of some commercially

available core-shell particles introduced in the literature survey chapter, and the duration of simulation was set as five times diffusive time measure, $5\tau_D = 5(d_p)^2/D_{AB}$. Time-dependent normalized diffusivity was defined as $D_0 = D_{AB}(t)/D_{AB}$. Normalized effective diffusivity was calculated as the mean of D_0 data for $t/\tau_D \geq 1$. 500 data points were extracted from the simulation throughout the entire duration. 75000 tracers were used in each simulation. Random-step size were set to 10% of the diameters of shell spheres for all runs. Initial positions for all tracers are selected randomly, inside the inter-particle void space in the main periodic cell. Note that the initial condition is not a point injection but a homogeneous distribution of tracers throughout the inter-particle void space in the main periodic cell.

Figures 30 and 31 show the effective diffusivity values predicted by the model, for core-shell particles with different diameters but the same geometry defining parameters (i.e. the number of shell layers and core-to-particle diameter ratio). Predicted $D_{0,eff}$ is the same for repeated simulations for particles with both diameters, only differing after third decimal point. Since the defining parameters are the same for both diameters, tortuosity of the packing geometry must be and is practically the same. Or, in other words, calculated geometries for the packings of core-shell particles with diameters $5\ \mu m$ and $3.4\ \mu m$ are two different scaled versions of the exact same geometry. Therefore these results are consistent. Normalized time-dependent diffusivity for the same particles, but with 2 shell layers instead of 1 are given in Figure 32 and 33. Simulations predict the same $D_{0,eff}$ values as in single layer particles. This is also a self consistent result, considering the entire particle porosity changes a very small amount between 1 layer and 2 layer geometries (check Figure 28 in section 4.1.3) and the reflection of this change on the entire volume available for diffusion in the core-shell packing is almost zero. However, these results would definitely change if solute tracers were assigned a finite size, unlike point tracers used in this study. Finite-sized probes would not be able to sample spaces where point tracers can freely roam. Therefore, the consistency of the results between coarse and finer core-shell particle geometries is only valid for simulations of diffusion or dispersion of small molecules.

Simulation predictions for normalized time-dependent diffusivity in a packing of triple layer core-shell particles with $\varphi = 0.655$ is given in Figure 34. $D_{0,eff}$ for particles with different diameters is the same, due to almost identical tortuosity, but the effect of extra diffusion volume that becomes available for $\varphi = 0.655$ (see Figure 28 for the apparent change in the entire particle porosity between core-to-particle ratios $\varphi = 0.655$ and $\varphi = 0.77$) is clearly distinguishable in the ~2.5% increase in effective diffusivity. Diffusion predictions of the model are self consistent but more importantly of physical coherence, although direct comparisons with any experimental results would not be plausible or at least very hard to speculate on since, especially for these small size core-shell particles, differences in packing geometry in experiments and the packing used in the simulations would effect $D_{0,eff}$ quite significantly compared to actual contributions from core-shell particles. In dispersion, these differences would be amplified and the predictions of the model can be compared to an available data and discussed.

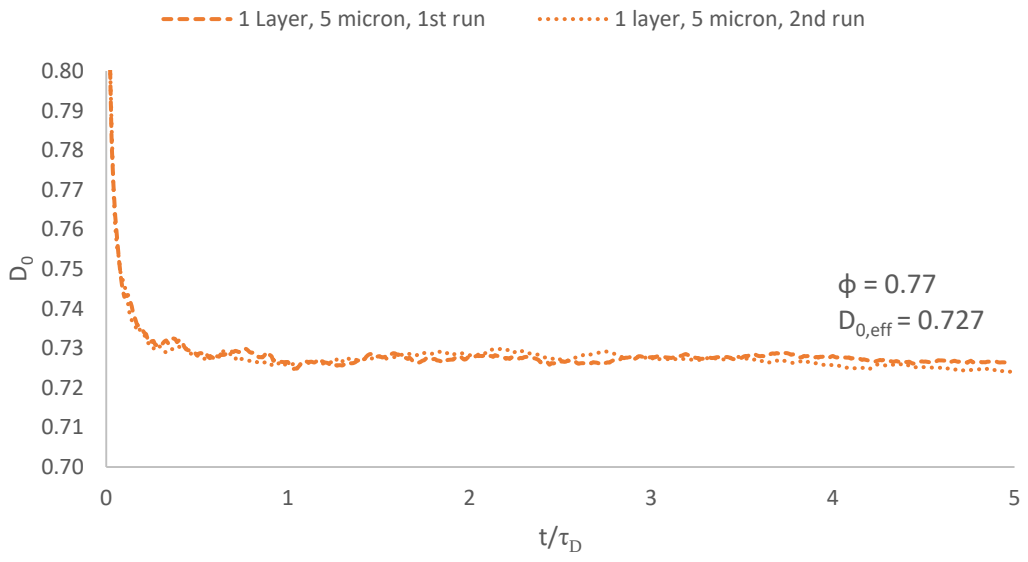


Figure 30: Normalized time-dependent diffusivity in the packing of 5 μm in diameter core-shell particles with single shell layer and core-to-particle ratio of 0.77. Normalized effective diffusivity, $D_{0,eff}$ is the same for two simulation runs in first 3 decimals.

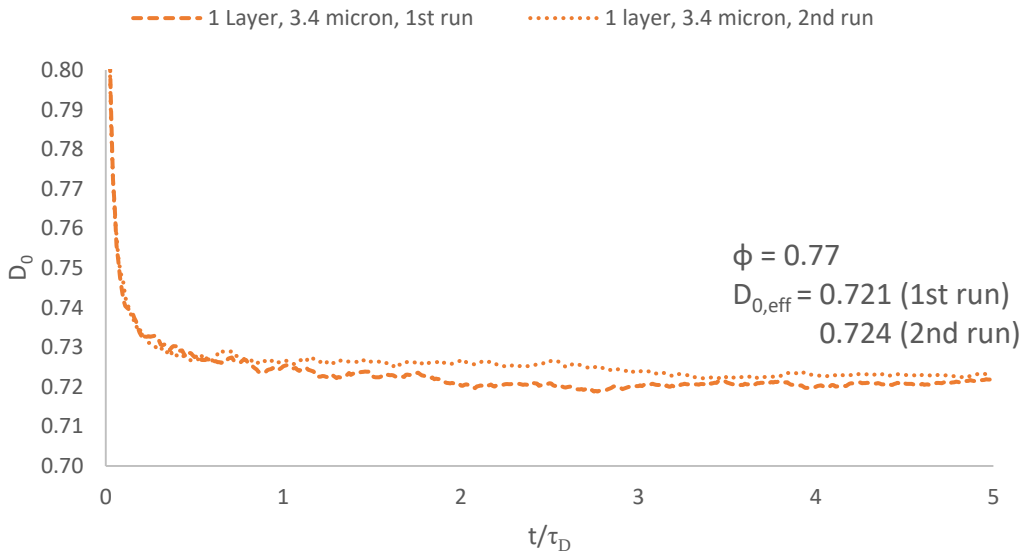


Figure 31: Normalized time-dependent diffusivity in the packing of 3.4 μm in diameter core-shell particles with single shell layer and core-to-particle ratio of 0.77. Normalized effective diffusivity, $D_{0,eff}$ is the same for two simulation runs in first 2 decimals.

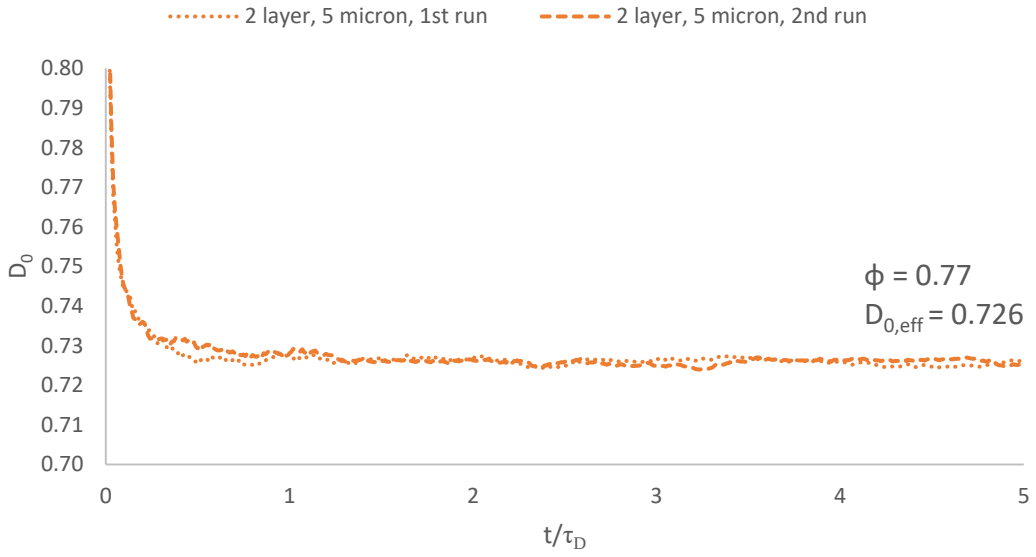


Figure 32: Normalized time-dependent diffusivity in the packing of 5 μm in diameter core-shell particles with 2 shell layers and core-to-particle ratio of 0.77. Normalized effective diffusivity, $D_{0,eff}$ is the same for two simulation runs in first 3 decimals.

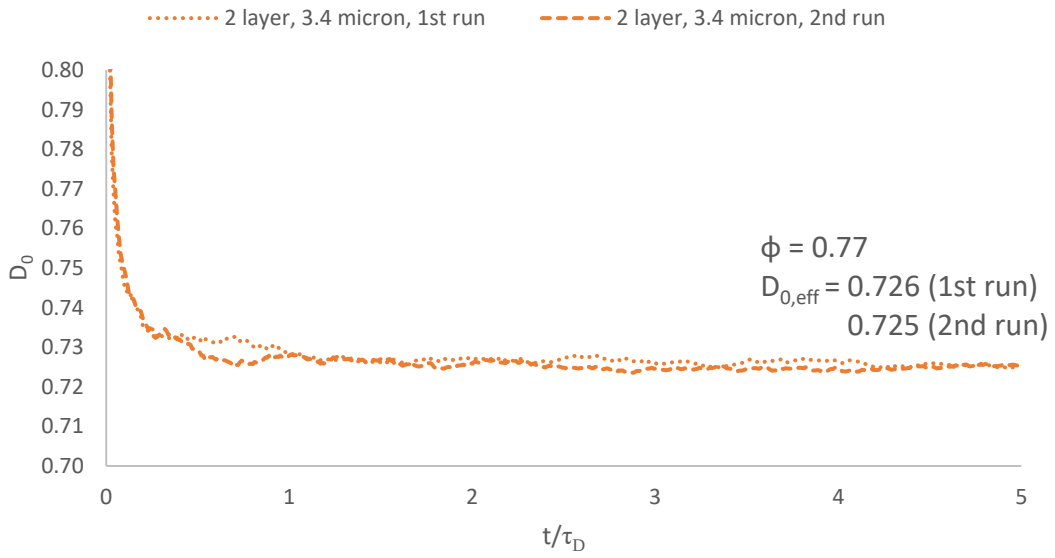


Figure 33: Normalized time-dependent diffusivity in the packing of 3.4 μm in diameter core-shell particles with 2 shell layers and core-to-particle ratio of 0.77. Normalized effective diffusivity, $D_{0,eff}$ is the same for two simulation runs in first 2 decimals.

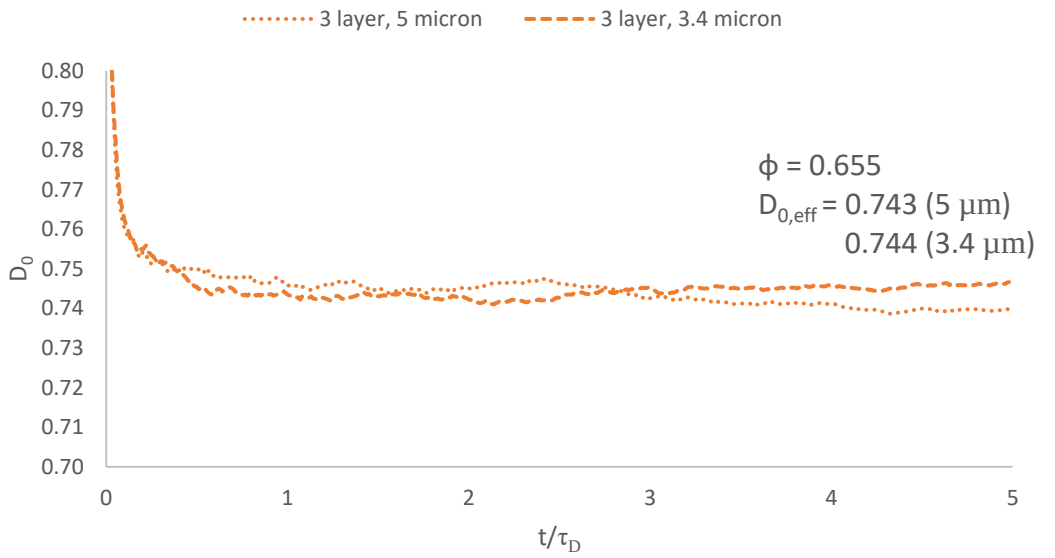


Figure 34: Normalized time-dependent diffusivity in the packing of 5 μm in diameter and 3.4 μm in diameter core-shell particles with 3 shell layers and core-to-particle ratio of 0.655. Normalized effective diffusivity, $D_{0,eff}$ is the same for two diameters in 2 significant figures.

Table 1: A summary of diffusion simulations. Results for double layer core-shell particles are averaged for Run 1 and Run 2 due to very close values in both runs.

	$\varphi = 0.77$						$\varphi = 0.665$	
	Single Layer Particle				Double Layer Particle		Triple Layer Particle	
	Run 1		Run 2					
d_p	3.4 μm	5 μm	3.4 μm	5 μm	3.4 μm	5 μm	3.4 μm	5 μm
$D_{0,eff}$	0.721	0.727	0.724	0.727	0.726	0.726	0.744	0.743

4.2. Fluid Flow Simulations

4.2.1. Validation of Periodic Flow Conditions

Simulation of water flow through the generated packing of hardspheres and obtaining the velocity field from the results were explained in section 3.4. Boundaries parallel to xy-plane were set as *periodic flow conditions*, such that the pressure difference between these boundaries would be a certain non-zero pressure drop value $\Delta P \neq 0$ so that flow occurs in positive z-direction. Opposing boundaries parallel to zy-plane and zx-plane were also set as periodic flow conditions, with no apparent pressure gradient, $\Delta P = 0$, in x and y directions. These boundary conditions assure the continuity of velocity field at the specified boundaries by definition, as in COMSOL they are defined as the velocity profile at the specified boundaries being equal to each other.

In order to validate the periodic boundary conditions set in the system, firstly the COMSOL model was computed as setting the pressure gradient in z-direction to $\Delta P = 400\ Pa$, an approximate pressure drop that would occur across the dimensions of the periodic cell ($\sim 18 \mu m$) estimated from Ergun Equation. Typical superficial velocities the HPLC columns are operated at, determined by the flow rate and inner-diameter of the column, was used in Ergun Equation to estimate the pressure drop in the flow direction across the main periodic cell with length $\sim 17{,}6\ \mu m$. In an HPLC

column with 4.6 mm inner diameter operating at 1 mL/min flow rate, superficial velocity of the mobile phase is very close to 1 mm/s. Corresponding pressure drop in the column over a length of 17,6 μm was calculated in Ergun Equation (Bird et al., 2007) by using void fraction of random close packings $\varepsilon = 0.355$ and viscosity and density of water, a typical mobile phase, at 25^oC. Pressure drop under specified conditions was found to be ~877 Pa.

$$\frac{\Delta P}{\rho {u_z}^2}\frac{d_p}{L}\frac{\varepsilon^3}{1-\varepsilon} = 150\frac{(1-\varepsilon)\mu}{d_p \rho u_z} + \frac{7}{4} \tag{53}$$

Contour plots of the z-components of velocity vectors, right at the periodic boundary couples were compared. Figures 35, 36 and 37 are showing the contour plots of periodic boundary couples parallel to xy, yz and zx-planes respectively. Plots show the same z-components for the velocity field at the boundaries, as it would be expected from the periodic flow conditions. Plots on the left and right sides of figures seem to be mirror images of each other with respect to y,z and x-axis due to the flip-rotation of view. If one of the boundaries is viewed from the opposite side, it would look exactly the same as the other.

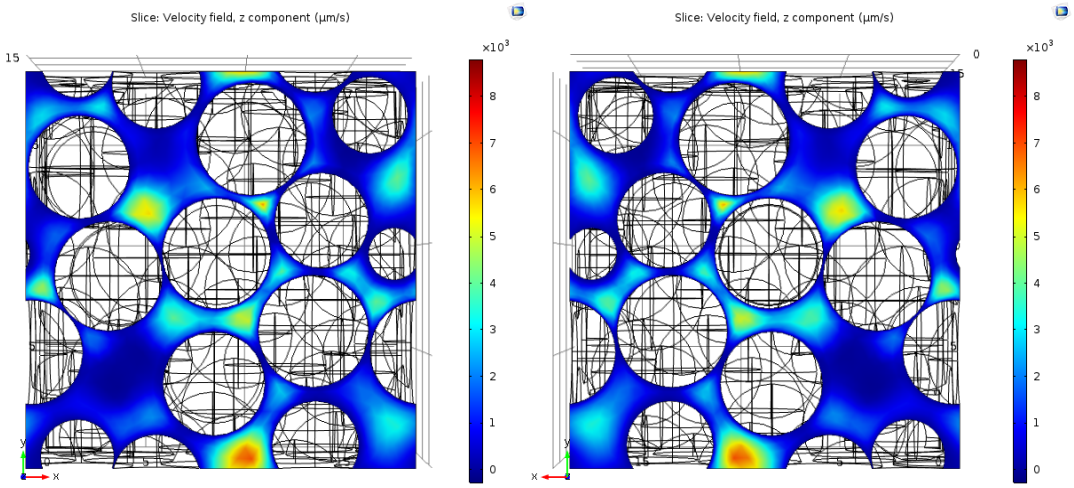


Figure 35: Contour plots of velocity field z-components at the periodic boundary couple parallel to xy-plane. Left: Top view of the main periodic cell. Right: Bottom view of the main periodic cell.

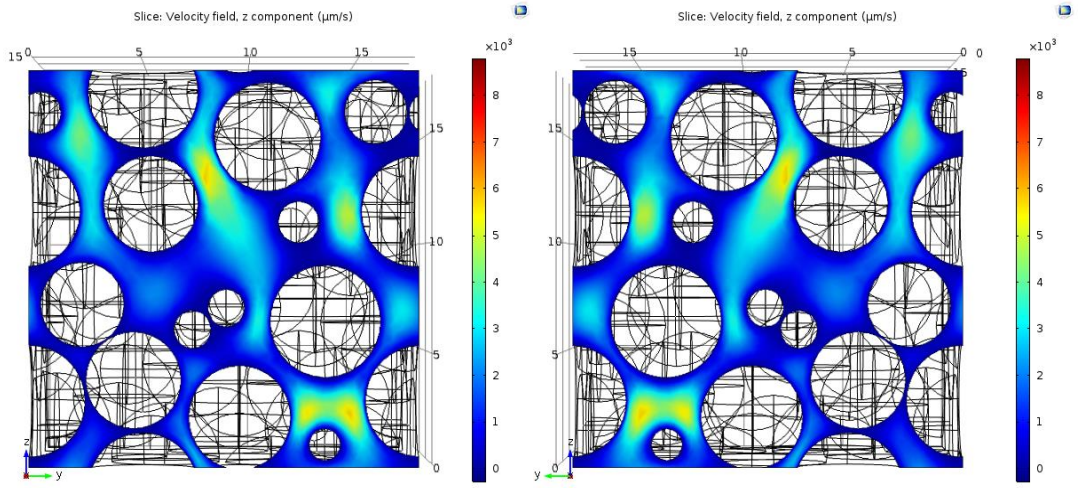


Figure 36: Contour plots of velocity field z-components at the periodic boundary couple parallel to yz-plane. Left: Right side view of the main periodic cell. Right: Left side view of the main periodic cell.

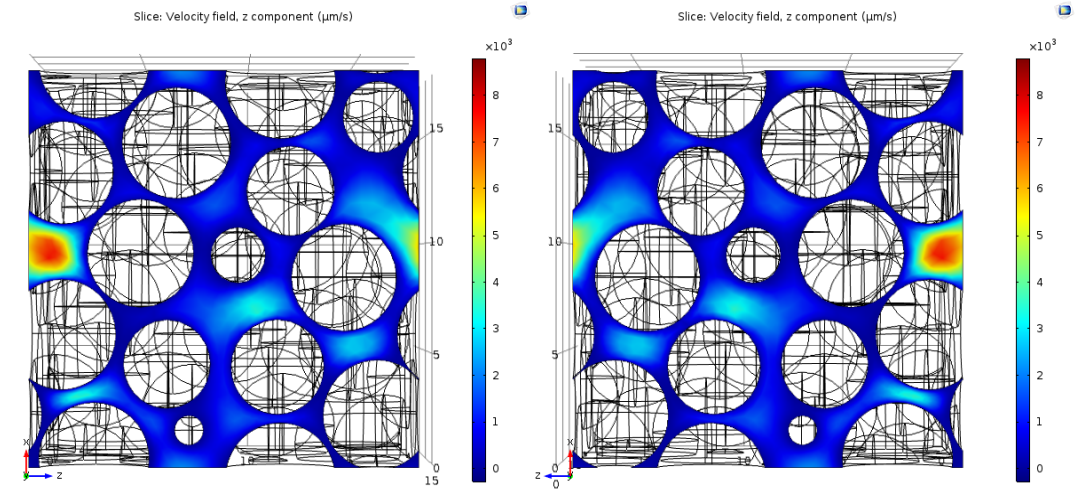


Figure 37: Contour plots of velocity field z-components at the periodic boundary couple parallel to zx-plane. Left: Back view of the main periodic cell. Right: Front view of the main periodic cell.

Velocity components at the periodic boundary couples were also compared quantitatively using Fortran. The velocity field was read by the Fortran code and stored into the `VF` array as described in Appendix C.4 and components of the velocity vectors on the boundary couples were compared. For fine resolution, 93.2% of the grid nodes on the coupled boundary surfaces have less than 1% difference in the z-components of velocity vectors and 96.5% have less than 5% difference. Therefore velocity profiles at the boundary couples are very close to each other and can practically be considered the same. This supports the contour plots and confirms periodic flow conditions are calculated properly. Matching velocity profiles at the boundaries also confirm that the random jammed packing of hardspheres was properly modified manually, as described in section 3.3.2. Otherwise, either the solution would not converge or it would converge to erroneously different velocity profiles at coupled periodic boundaries.

Grid dependence of the solution was also tested by running the COMSOL simulation at normal and coarse resolutions in addition to fine resolution. For normal and coarse resolutions, 93.6% and 93.1% of the nodes have less than 1%, 95.1% and 95.6% have less than 5% difference respectively, very close to that of fine resolution results. The x and y components were not compared since they are significantly smaller (around 1% of the z-component in average) than z-components. It can be concluded that the resolution does not affect the validity of periodic boundaries. Also, the average velocity magnitude throughout the entire velocity field changes by 2% and 2.7% relative to fine resolution, for normal and coarse meshes respectively. The x and y components are significantly different in lower resolutions. Yet the directions of the velocity vectors are not affected significantly, due to very small size of x and y components relative to z-components. Therefore the entire velocity field can be considered grid independent. Still, the velocity field obtained from fine resolution is used in the simulations of dispersion, later presented in this chapter.

4.2.2. Stokes Flow Range Inside the Packing

Determination of Stokes Flow range in the packing of hardspheres was required for practical purposes. Velocity field that belongs to a Stokes Flow, or Creeping Flow, is linearly scalable since the inertial terms in the equation of continuity are negligible compared to linear viscous terms. Linear scaling of the velocity field allows calculating once and using the same velocity field to simulate dispersion at different Peclet numbers determined by the average superficial velocity in the system linearly scaled from the original velocity field obtained.

Fluid flow simulations for five different ΔP values (400, 800, 8000, 16000 and 24000 Pa) around the Ergun Equation estimation, as explained in the previous section, were carried out and the velocity fields obtained at each pressure drop was analyzed. Table 2 shows the maximum Reynolds numbers occurring in the nodes, as well as the volume-average Reynolds numbers of all nodes in the 101x101x101 grid and average velocity components and velocity magnitudes for all five simulations.

Table 2: Maximum and average Reynolds numbers in the velocity fields, average velocity components and average velocity magnitudes obtained from the solutions at pressure drops between 400 and 24000 Pa.

ΔP (Pa)	Re_{max}	Re_{avg}	Volume average of velocity components (m/s)			$\lvert u \rvert_{avg}$ (m/s)	% Difference with linear scaling	
			u_x	u_y	u_z		Avg.	Max.
400	0.05	0.004	-4.35×10^{-6}	-4.35×10^{-6}	5.39×10^{-4}	6.48×10^{-4}	-	-
800	0.10	0.007	-8.70×10^{-6}	-8.70×10^{-6}	1.08×10^{-3}	1.30×10^{-3}	0.005	0.02
8000	1.03	0.073	-8.71×10^{-5}	-8.71×10^{-5}	1.08×10^{-2}	1.30×10^{-2}	0.088	0.38
16000	2.06	0.145	-1.74×10^{-4}	-1.74×10^{-4}	2.16×10^{-2}	2.59×10^{-2}	0.181	0.78
24000	3.09	0.218	-2.62×10^{-4}	-2.62×10^{-4}	3.24×10^{-2}	3.89×10^{-2}	0.273	1.17

Maximum Reynolds number in any of the grid nodes becomes closer to even turbulent flow limit of the water for $\Delta P = 24\ kPa$, however these are only calculated for the velocity vector with greatest magnitude in the entire velocity field. Local Reynolds numbers can reach up to 30 before Stokes Flow loses its validity around high-velocity regions in a random close packing of monodisperse spheres and eddy seeds start popping up (Hlushkou & Tallarek, 2006), whereas the maximum local Reynolds number is between 1 and 2 in this specific packing being dealt with. Average Reynolds number in the packing becomes greater than the Stokes Flow limit, conservatively assumed as $Re = 0.1$ in this thesis study, after pressure drop across the periodic cell reaches some value between $8\ kPa$ and $16\ kPa$. Therefore the error caused by linear scaling should start increasing after this value. In order to observe this error, velocity field obtained for $\Delta P = 400\ Pa$ was linearly scaled to match the velocity fields obtained by direct solutions of velocity fields that belong to other pressure drop values $800\ Pa$, $8\ kPa$, $16\ kPa$ and $24\ kPa$. Average magnitudes of the velocity vectors linearly scaled to $\Delta P = 800\ Pa$ and $\Delta P = 8\ kPa$ have 0.005% and 0.088% absolute error with respect to direct solutions. Average absolute errors for the velocity vectors linearly scaled to $\Delta P = 16\ kPa$ and $\Delta P = 24\ kPa$ was found to be 0.181% and 0.273% respectively. However, maximum absolute errors for local velocity vectors reaches 0.78% and 1.17% for scaling to $\Delta P = 16\ kPa$ and $\Delta P = 24\ kPa$ solutions, compared to 0.02% and 0.38% maximum absolute error when the velocity field solution for $\Delta P = 400\ Pa$ is linearly scaled to the solutions for $\Delta P = 800\ Pa$ and $\Delta P = 8\ kPa$ respectively. It is still a small error but considering the average Reynolds number at the same pressure drop, it would not be safe to assume the scaled velocity vectors would have the same directions above the Stokes Flow limit $Re = 0.1$, which occurs at $\Delta P_{SFL} \cong 11\ kPa$ and $u_{z,SFL} \cong 0.015\ m/s$.

The range of Peclet numbers that dispersion can safely be simulated depends on the bulk diffusion coefficient of solute tracers, given as an input parameter for the simulation. Limiting Peclet number, Pe_{SFL}, can simply be calculated by its defining equation. The characteristic length in the packing is equal to the diameter of spheres in the packing, d_p.

$$Pe_{SFL} = \frac{d_p u_{z,SFL}}{D_{AB}} \tag{54}$$

Equation (54) will be used in discussions related to simulations of dispersion in the random jammed packing of core-shell particles.

4.3. Dispersion Model

4.3.1. Validation of Dispersion Model by Simulating Taylor Dispersion in a Pipe

The Fortran code written for simulating dispersion was tested by simulating Taylor dispersion of a tracer ensemble at Peclet numbers ranging from near 0 to 1000, in a pipe with $d_p = 1\ \mu m$ diameter. The analytical solution for mass dispersion is readily available for this case (Taylor, 1953), which enables an opportunity for validating the code. Flow of water through the pipe with was simulated in COMSOL and the velocity field was obtained in a 101x101x101 grid similar to that used in simulations of dispersion in core-shell packings. The velocity field was read by the Fortran program the same way and the same trilinear interpolation subroutine was used. Velocity field solution was linearly scaled in Stokes Flow range to simulate Taylor dispersion of 2000 tracers at up to $Pe \cong 900$. Simulation duration was set to 0.2 seconds, a considerably high duration with respect to minimum required time for transient behavior of dispersion to stop and reach steady-state, which is the amount of time needed for all tracers to sample entire velocity field, $\tau_{min} = {d_{pipe}}^2/6D_{AB}$, which is equal to 0.038 seconds for this case. Longitudinal position variance data of the tracer ensemble with respect to time was obtained from the simulation and the data was used to determine time slopes of longitudinal position variance of tracer ensemble (See Appendix B.1 for variance vs. time plots). Then the longitudinal dispersion coefficients were calculated using the equation $D_L = 0.5(\delta\sigma_L^2/\delta t)$.

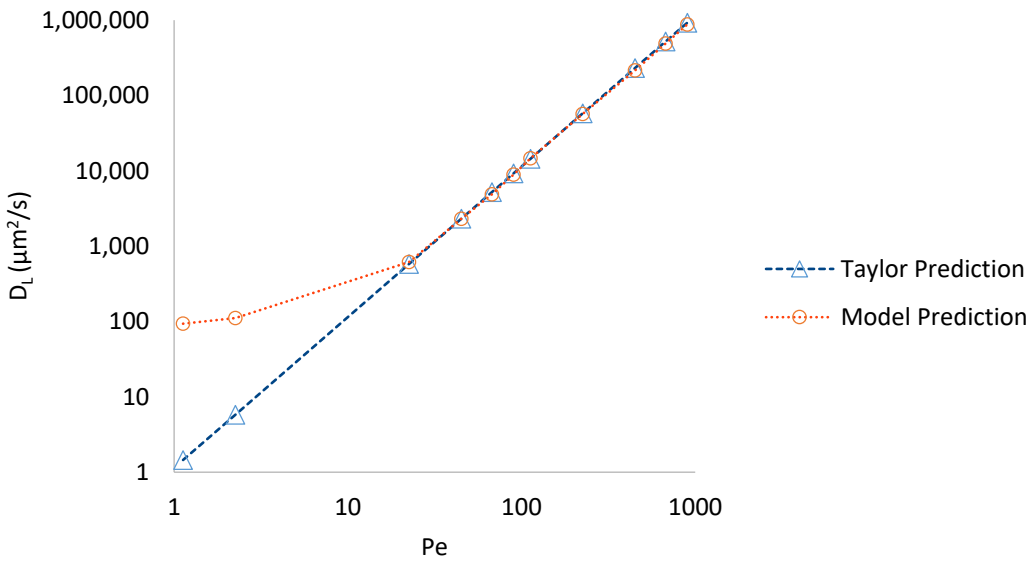


Figure 38: Longitudinal dispersion coefficients in a pipe, predicted by Taylor Dispersion Model (1953) and the dispersion model built in the thesis study, at different Peclet numbers. Both axis are logarithmic. Note the deviation between two models at low Pe.

Predictions of the prepared dispersion model matches with the Taylor Dispersion Model developed by Geoffrey Taylor (1953) which has reasonable accuracy compared to experiments carried out for the same study. Deviation of two models at low Peclet numbers occurs due to the fact that Taylor model is linear and it does not take the contributions from axial diffusion into account while predicting dispersion coefficients. Therefore it erroneously underestimates longitudinal dispersion coefficients at low Pe, as opposed to the random-walk based dispersion model, where the contributions from diffusion are very apparent as the distance traveled by tracers due to random-steps taken. D_L even approaches around the input solute tracer diffusion coefficient, $110 \mu m^2/s$, as Peclet number approaches zero. It can be concluded that the dispersion code works properly and predicts accurate results. Therefore it can be used for simulating dispersion in packings of core-shell particles.

4.3.2. Longitudinal Dispersion Coefficients of Tracers

4.3.2.1. In the Random Packing of Monodisperse Hardspheres

Dispersion of solute tracers were simulated in a packing of Monodisperse hardspheres with $d_p = 5\,\mu m$. The velocity field for water flow through the periodic cell with $\Delta P = 400\,Pa$, inspected in section 4.2.2, was used as the main velocity field and linearly scaled to obtain results at different Peclet numbers. Bulk diffusion coefficient of tracers was set as $D_{AB} = 110\,\mu m^2/s$. Tracers were randomly distributed into the inter-particle space between hardspheres as the initial condition. Duration of simulation was set to either ten times the convective time measure $\tau_C = 2r_p/u_{avg}$, defined as the time required for a tracer moving at the average fluid velocity to travel a distance equal to the diameter of spheres in the packing, or 0.2 times the diffusive time measure $\tau_D = (2r_p)^2/D_{AB}$, which is equal to 2 times the the minimum required time for dispersion to reach asymptotic behaviour, whichever is the greatest. This ensures that at low Pe, tracers will sample at least 3 periodic cells and also the simulation would proceed for enough time at higher Pe to allow reaching asymptotic behavior. Variance vs. time data was collected from the simulations carried out at various peclet numbers and fit an asymptotic curve, given by Equation (50) in section 3.6.5, which estimates the time slope of variance as the parameter A. Variance data and fitted curves at $Pe = 10$ and $Pe = 50$, and dispersion coefficients are given in Figure 39. Other variance and dispersion coefficient data obtained from the simulations carried out at the rest of Peclet numbers are in Appendix B.2.

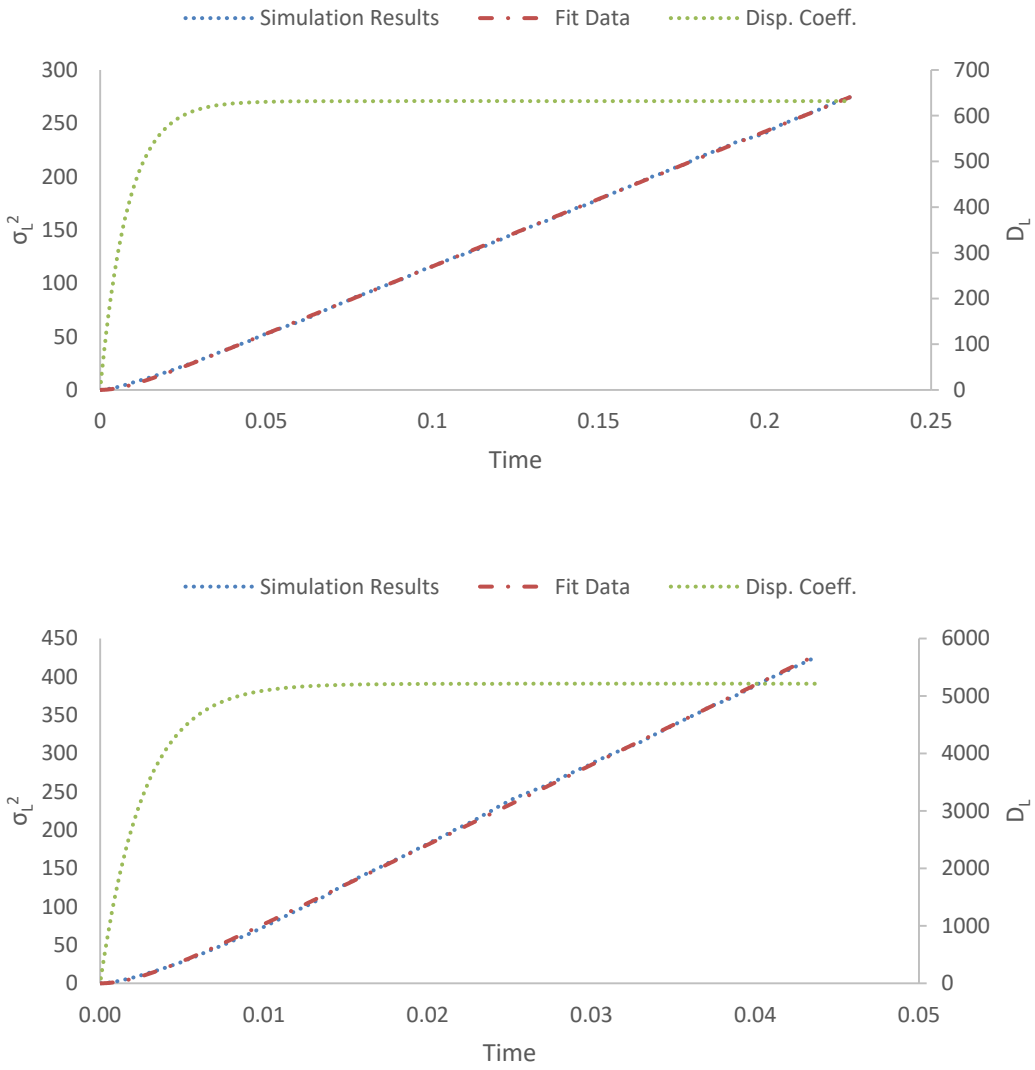


Figure 39: Longitudinal displacement variance vs. time at Peclet numbers 10 (top) and 50 (bottom). Longitudinal dispersion coefficient vs. time is on the secondary axis to the right.

Average mobile phase velocities, Peclet numbers and normalized longitudinal dispersion coefficients predicted at the corresponding Peclet numbers are given in Table 3. A longitudinal dispersion coefficient vs. Peclet number graph is also given in Figure 40.

Table 3: Volume-average velocity magnitudes in the linearly scaled velocity field at different Peclet numbers, corresponding estimated asymptotic time-slopes of σ_L^2 and normalized longitudinal dispersion coefficients.

$u_{avg}\ (\mu m/s)$	Pe	$\delta\sigma_L^2/\delta t\ (\mu m/s)$	D_L/D_{AB}
220	10	1.26×10^3	5.7
330	15	2.17×10^3	9.8
440	20	3.14×10^3	14.2
550	25	4.44×10^3	20.1
660	30	5.52×10^3	25.0
770	35	7.17×10^3	32.5
880	40	7.60×10^3	34.5
990	45	9.54×10^3	43.3
1100	50	1.04×10^4	47.3
1320	60	1.31×10^4	59.6
1540	70	1.60×10^4	72.7
1760	80	1.93×10^4	87.8
1980	90	2.16×10^4	98.1
2200	100	2.51×10^4	113.9

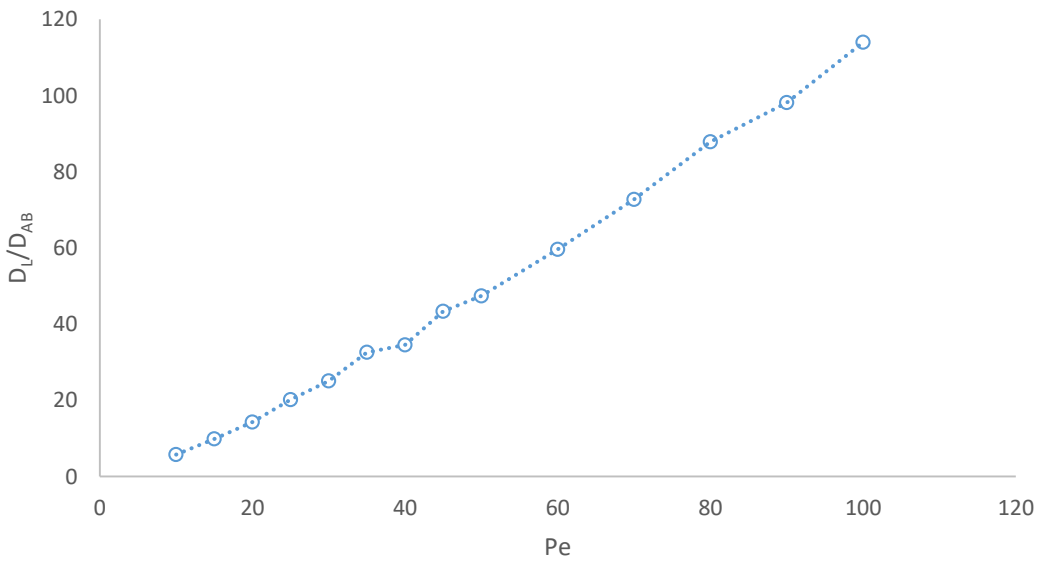


Figure 40: Normalized longitudinal dispersion coefficients, D_L/D_{AB}, predicted by the model vs. Pe.

Longitudinal dispersion coefficients predicted by the model in an unbound packing of hardspheres are in good agreement with experimental data of Han et al. (1985) and a compilation of other experimentally determined dispersion coefficients present in the same work, in which the dispersion coefficients follow a straight line between roughly similar values in the same range of Peclet numbers. Reader is referred to the article of Han et al. (1985) for these experimental data, since the figure containing the data could not be 'digitized' and permissions for using the figure here directly could not be received.

As an end note for the section, one should not forget that the fit parameters A and k are not necessarily physically significant. They are only fit parameters to estimate the asymptotic slope of the time dependent longitudinal position variance.

4.3.2.2. In the Random Packing of Core-Shell Particles

Dispersion in the random packing of core-shell particles were simulated in the same packing geometry as in the previous section, dispersion in packing of hardspheres, where this time the hardspheres were replaced by core-shell particles with the same diameter, $\varphi = 0.73$ and $n_l = 3$. Velocity field that belongs to the water flow through the periodic cell with $\Delta P = 400\ Pa$, again, was used as the base velocity field that can be linearly scaled within creeping flow range. Initial conditions were also the same. Tracers were randomly distributed into the inter-particle space between core-shell particles, initially. Same bulk diffusion coefficient was used and durations of simulations were also determined by the same reasoning. Determination of longitudinal dispersion coefficient was similarly done using variance vs. time data. Variance data and fitted curves at $Pe = 1$ and $Pe = 50$, and dispersion coefficients are given in Figure 41. Other variance and dispersion coefficient data obtained from the simulations carried out at the rest of Peclet numbers are in Appendix B.3.

Average mobile phase velocities, Peclet numbers and normalized longitudinal dispersion coefficients predicted at the corresponding Peclet numbers are given in

Table 4. A longitudinal dispersion coefficient vs. Peclet number graph is also given in Figure 42.

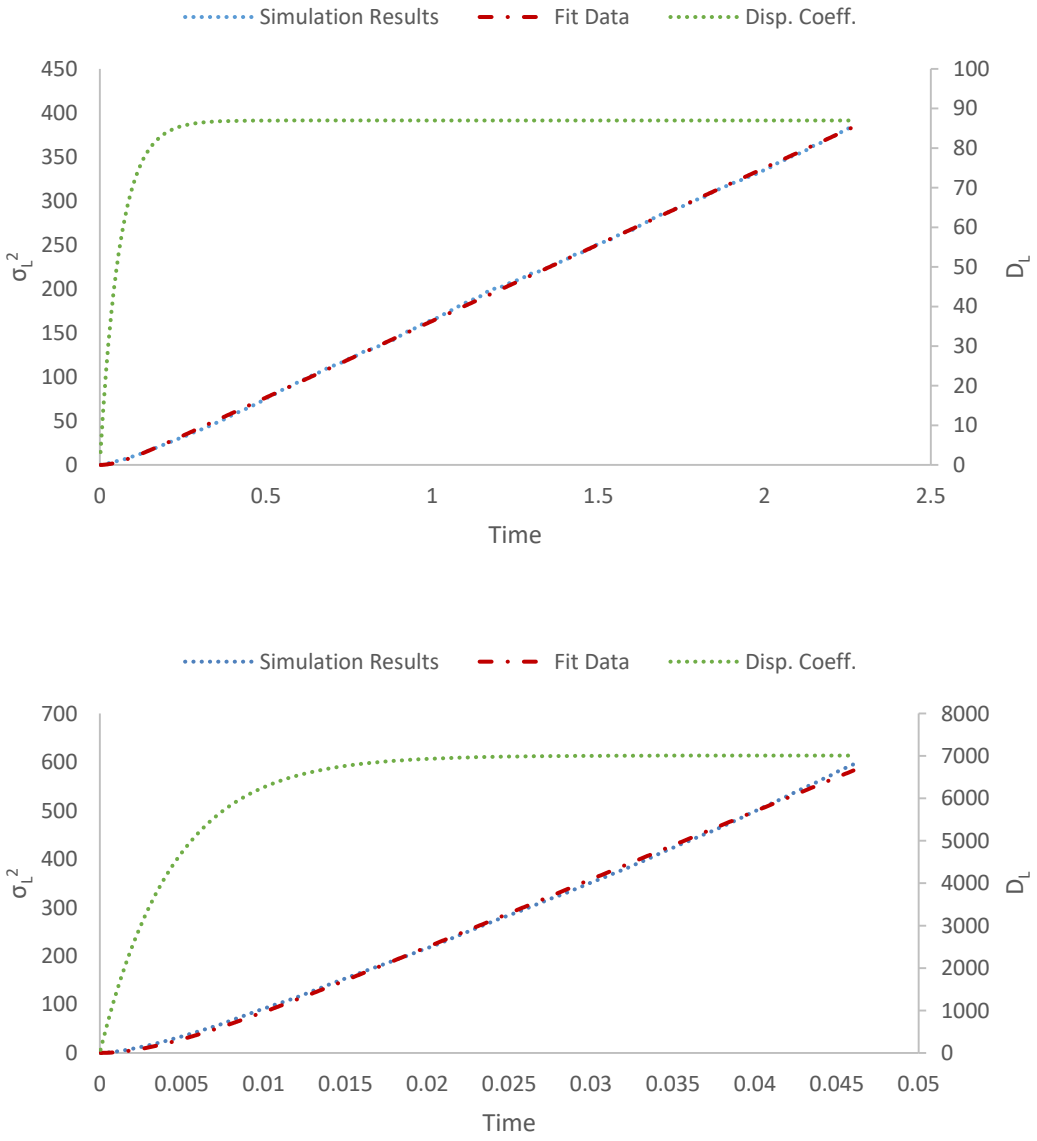


Figure 41: Longitudinal displacement variance vs. time at Peclet numbers 1 (top) and 50 (bottom). Longitudinal dispersion coefficient vs. time is on the secondary axis to the right.

Table 4: Volume-average velocity magnitudes in the linearly scaled velocity field at different Peclet numbers, corresponding estimated time-slopes of σ_L^2 and normalized longitudinal dispersion coefficients.

$u_{avg}\ (\mu m/s)$	Pe	$\delta\sigma_L^2/\delta t\ (\mu m/s)$	D_L/D_{AB}
22	1	1.74×10^2	0.8
44	2	2.42×10^2	1.1
66	3	3.12×10^2	1.4
88	4	4.25×10^2	1.9
110	5	5.38×10^2	2.4
132	6	6.46×10^2	2.9
154	7	8.50×10^2	3.9
176	8	1.01×10^3	4.6
198	9	1.11×10^3	5.0
220	10	1.37×10^3	6.2
330	15	2.34×10^3	10.6
440	20	3.52×10^3	16.0
550	25	4.86×10^3	22.1
660	30	6.45×10^3	29.3
770	35	8.00×10^3	36.4
880	40	1.00×10^4	45.5
990	45	1.21×10^4	55.1
1100	50	1.40×10^4	63.7
1650	75	2.67×10^4	121.4
2200	100	3.96×10^4	180.0

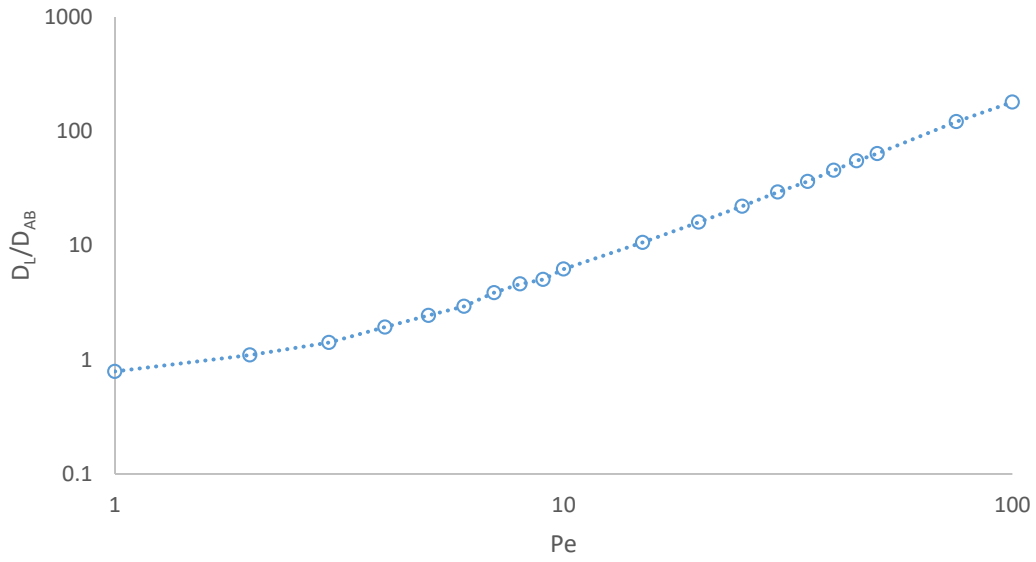


Figure 42: Normalized longitudinal dispersion coefficients, D_L/D_{AB}, predicted by the model vs. Pe.

Results show a linear increase in the normalized dispersion coefficients after the Pe in the system becomes large enough for the convection to take over the dispersion mechanism, as it should be expected from the simulation. However there is a barely noticeable increase in D_L after $Pe = 50$ mark (although the reduced plate height graphs given in the next section makes it more noticeable after that Pe), where longitudinal displacements of tracers must have already been started to be dominated by the intra-particle diffusion limitations and mechanical dispersion and following a straight line. This occurrence must be related to, with strong possibility, the contribution of velocity auto-correlation as it was explored in the study of Maier et al. (2000). They had found that periodic arrays of spherical random packings are prone to velocity auto-correlation, which consequently overestimates D_L compared to what it would have been in a non-periodic random packing of spheres, because tracers in periodic arrays can get caught by a high velocity region and repeatedly sample these same high velocities in the periodic array a large amount of times before they can diffuse away to low velocity regions. They found that the contributions from velocity

auto-correlation eventually decay. However the decay rate depends on the ratio of main periodic cell length to sphere diameter, $L_{pc}/2r_p$, where higher ratios allow quicker decay of velocity auto-correlation. In a spherical packing with similar $L_{pc}/2r_p$ to the packing used in this thesis study, they found that partially decayed auto-correlation contribution to D_L starts increasing again at around $10\tau_C$. At exactly $Pe = 50$, dispersion code starts using $0.2\tau_D$ as the simulation duration because the average mobile phase velocity in the system increases to such a turning point that $10\tau_C$ becomes less than $0.2\tau_D$. Consequently, velocity auto-correlation kicks in causing the unexpected increase in the predicted D_L. This also limits the model to work in the range $0 < Pe < 50$ without the effects of velocity auto-correlation. This occurrence cannot be related to linear scaling of the velocity field because the limiting Peclet number for Stokes Flow, Pe_{SFL}, is greater than 600 for tracer D_{AB} used in simulations.

4.3.2.3. Reduced Plate Heights in an Unbound Liquid Chromatography Column

Longitudinal dispersion coefficients of tracers obtained in the previous section were converted to reduced plate heights in order to be able to compare the predictions of the model to experimental data, since the dispersion coefficient data is typically available as reduced plate heights of HPLC columns determined by peak parking method which is quite similar to the way this dispersion model is used for obtaining dispersion coefficients.

Recall that, plate heights in chromatography columns are defined as proportional to the derivative of longitudinal position variance of the analyte, or the tracer ensemble in theoretical case, with respect to longitudinal distance. If normalized by the diameter of core-shell particles in the column, reduced plate height values for the column is obtained. Accordingly, D_L can be divided by superficial mobile phase velocity u_z and core-shell particle radius r_p for conversion to dimensionless reduced plate height h, as in the following equation.

$$h = \frac{H}{2r_p} = \frac{D_L}{u_z r_p} = \frac{1}{u_z d_p}\frac{\delta\sigma_L^2}{\delta t} = \frac{1}{d_p}\frac{\delta\sigma_L^2}{\delta z} \tag{55}$$

Reduced plate heights were calculated using longitudinal dispersion coefficients obtained in the previous section, at the same corresponding Peclet numbers. Average longitudinal velocity component u_z, as well as corresponding Peclet numbers, predicted plate heights and reduced plate heights at these Peclet numbers are given in Table 5. A graphical representation of h vs. Pe is also available in Figure 43.

Table 5: Volume-average z-components of velocity in the linearly scaled velocity field at different Peclet numbers, corresponding plate heights and reduced plate heights calculated by Equation (55) using variance slopes determined previously in Table 4.

$\boldsymbol{Pe}$	$\boldsymbol{u_z}$ $(\boldsymbol{\mu m/s})$	$\boldsymbol{H}$	$\boldsymbol{h}$
1	18.31	9.50	1.90
2	36.62	6.61	1.32
3	54.93	5.67	1.13
4	73.24	5.80	1.16
5	91.55	5.88	1.18
6	109.86	5.88	1.18
7	128.17	6.63	1.33
8	146.48	6.93	1.39
9	164.79	6.73	1.35
10	183.10	7.48	1.50
15	274.66	8.53	1.71
20	366.21	9.61	1.92
25	457.76	10.61	2.12
30	549.31	11.75	2.35
35	640.86	12.49	2.50
40	732.41	13.66	2.73
45	823.97	14.71	2.94
50	915.52	15.32	3.06
75	1373.28	19.45	3.89
100	1831.04	21.63	4.33

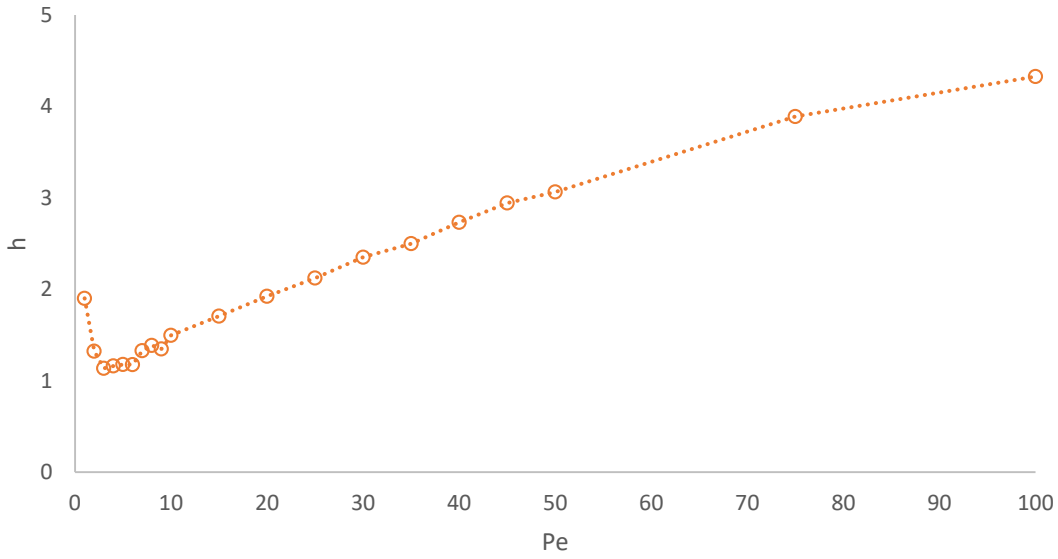


Figure 43: Reduced plate height of tracer ensemble vs. Peclet number.

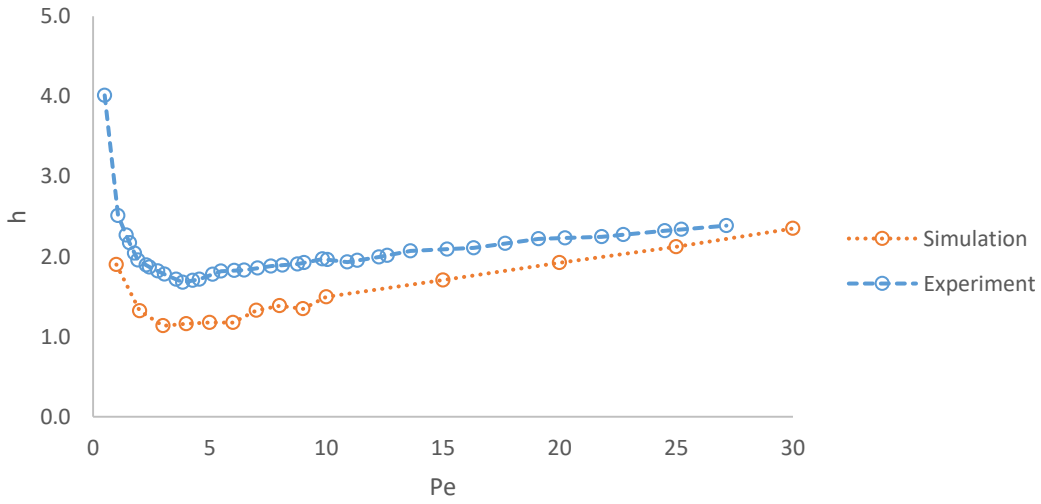


Figure 44: Reduced plate heights predicted by the model and experimental reduced plate height data for non-retained small molecule; uracil. Experimental data was taken from the study of Guiochon and Gritti (2011).

Reduced plate height predictions of the model compares well with experimental data collected by Guiochon and Gritti (2011), as seen in Figure 44. The experimental data belongs to uracil, a non-retained small molecule, analysed in a column with $4.6\,mm$ diameter and $150\,mm$ length, packed with $d_p = 2.6\,\mu m$ Kinetex-C_{18} core-shell particles that have $0.35\,\mu m$ shell thickness which corresponds to $\varphi = 0.73$. Uracil as a non-retained small molecule shares characteristics with dimensionless and non-adsorbed point tracers and the core-shell particles in used in this experimental study is practically the same as the core-shell geometry used in simulations of this work. The only difference between the experimental system and simulation system is the presence of wall effects, in the Pe interval the experiment was carried on. Mobile phase flow in confined systems like HPLC columns are typically divided into wall region, transition region and bulk regions in related studies, such as Bruns et al.'s (2012), Daneyko et al.'s (2011) and Khirevich et al.'s (2012). In these studies, it was pointed out that the porosity of the packing increases drastically near wall regions compared to the bulk region that lies around the center of the column. Consequently, mobile phase velocity profile follows a similar trend to the porosity profile across the packing, as detected by also in the study of Das et al. (2017), causing drastic flow irregularities which creates additional contribution to the plate height that would not exist in an unconfined packing. This effect is shown to be more visible in columns with smaller diameter and at lower Peclet numbers -or Reynolds numbers- in all of these studies. Considering the model only represents the bulk region in an HPLC column, majority of the difference between the model and experimental data of Guiochon and Gritti (2011) can be explained by the effects of confinement. The model underestimates dispersion in the column near the minimum reduced plate heights, and predictions of reduced plate heights converges to that of confined HPLC column at higher Peclet numbers as the wall effects start losing their dominance. In another series of studies released by Gritti and Guiochon (2013, 2013) and Gritti et al. (2014) effects of so-called "Parallel Segmented Flow Chromatography", PSFC in short, on the plate heights in columns was shown. According to their description, PSFC takes advantage of transient behavior of dispersion to improve the plate heights in analytical columns by splitting the outlet and taking measurements only from the middle section of the

column where less biased bulk flow occurs. Their results comparing standart columns and PSFC columns are consistent with the comparison of model predictions to experimental data given in Figure 44, for retained and non-retained solutes. However they are not directly comparable, since data is available for a very short range of Pe and the characteristics of the core-shell particle used in the experiment is unclear. Still, their findings support the predictions of the model. The difference, also, cannot be caused by the input bulk diffusivity assigned to tracers $D_{AB} = 110\ \mu m^2/s$, compared to bulk diffusivity of uracil in water, which is around $1160\ \mu m^2/s$ (Song et al., 2016). In fact, reduced plate height vs. Peclet number plots for dispersion simulations carried out in the range $0 < Pe < 20$ can be seen in Figure 45, and they are very similar to the main results. In reduced variables, simply no difference is observed. Higher tracer diffusivity only reduces Peclet number at Stokes Flow limit, to ~70 for $D_{AB} = 1160\ \mu m^2/s$.

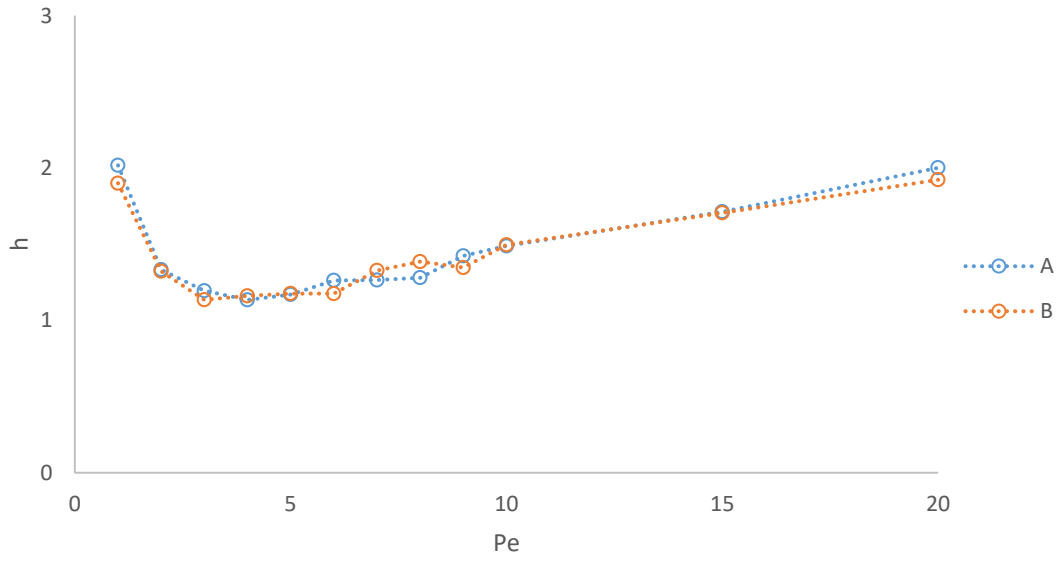


Figure 45: Reduced plate height vs. Pe predicted by simulations done with tracer bulk diffusion coefficients $1160\ \mu m^2/s$ (A) and $110\ \mu m^2/s$ (B).

Experimental data of Guiochon and Gritti (2011) was extended to $Pe = 100$ by fitting a Knox (2002) equation to it for full comparison with model predictions, since the experimental data was only available for $0 < Pe < 27$. Figure 46 shows extended data along with the predictions of the model.

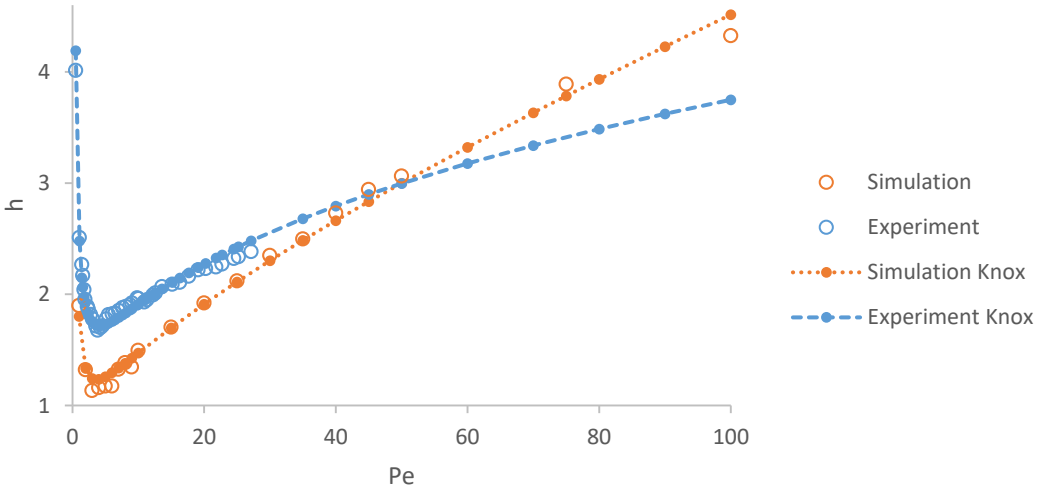


Figure 46: Experimental reduced plate height data of Guiochon and Gritti (2011) extended to $Pe = 100$ by Knox equation best fit with $A = 0.80$, $B = 1.77$ and $C = 0$. Reduced plate heights predicted by the model is also available, along with the best fitted Knox curve with $A = 0.53$, $B = 1.25$ and $C = 0.02$, for comparison.

A and B terms of the experimental data is greater than the terms of Knox curve best fitted to model predictions, which can be explained by flow irregularities and consequent trans-column velocity bias present in the confined column, as explained before. Best fit C terms are 0 and 0.2, very small values that are characteristic to core-shell particles thanks to solid cores. Still, the C term for the model predicts greater plate height contribution from intra-particle mass transfer compared to the

experimental data. There can be three reasons for it. First, contribution from velocity auto-correlation after is certainly present but this would be minimally reflected on the fit parameter since most of the data is at low Pe where the effects of velocity auto-correlation contributions are very small. Second, the effect of discretization mentioned in section 4.1.2 equally applies to the core-shell particle geometry, effectively and artificially decreasing the volume available for diffusion in the shell layer. Unlike tracers, uracil molecules would freely sample narrow regions in the shell layer with a higher effective diffusivity, hence the smaller C term. Third, the assumption that intra-particle flow does not occur starts losing its validity at higher Peclet numbers. Development of a mobile phase flow in the shell side would enhance intra-particle mass transfer and reduce the plate height contribution from the C term. Although the intra-particle Peclet number would be very small for a core-shell particle like Kinetex-C_{18} (Heeter & Liapis, 1996), still the model does not have that small intra-particle fluid velocity. It is most probably the second and third reasons has the most contribution to the differences in C terms best fitted to data. This justification can also be supported by the predictions of effective medium based model created by Daneyko et al. (2015) where individual contributions of diffusion, flow irregularities and intra-particle mass transfer limitations were determined computationally and intra-particle resistances was found to becoming more and more contributing to the plate heights as the Pe increases in the system.

Extended version of Giddings model (1963) for core-shell particles -which was introduced in Chapter 2- built by Daneyko et al. (2015) was used to determine the individual contributions to plate heights, as mentioned just previously, for the total reduced plate height predictions of thesis model and experiments of Gritti & Guiochon (2011). Void fraction of the column used in the experiment was assumed as 0.4, as suggested in the related study. Void fraction of the periodic packing was 0.355. Shell porosity of the core-shell particles were set as 0.49, determined from Figure 27 in section 4.1.3. Normalized effective diffusivity in the packing was taken as 0.73, an approximate value extracted from the results in section 4.1.5. Same value was assumed to be true for the experiment column, although the higher void fraction in the physical packing in the column is higher than that of periodic packing used in simulations. This

would result in a slight underestimation of effective diffusivity in the column, and overestimation of longitudinal diffusion and interchannel dispersion contributions. Modified Giddings model fits and individual contributions from longitudinal diffusion, interchannel dispersion, transchannel dispersion and intra-particle mass transfer limitations to both simulation and experiment data can be seen in Figure 47, in the range of Pe experimental data was available. Comparison of individual contributions to plate height between the experimental system and the system used in simulation can also be seen through Figures 48 to 51.

Transchannel eddy dispersion is significantly higher for the experiment. This can be explained by the greater void fraction in the column, compared to the periodic packing used in the simulations, as well as the higher void fraction areas near the column walls where the higher superficial velocity than the bulk region causes additional dispersion. The wall effects must also cause a slight radial flow from the bulk region towards near the column walls, consequently the plate height contribution from interchannel eddy dispersion must become slightly greater in the confined system. Modified Giddings model certainly detects this slight difference between simulation data and experimental data. Still the C term, related to intra-particle mass transfer limitation, is again greater for the simulated data, 0.020 compared to a smaller 0.013. As the modified Giddings model relies on more detail than Knox equation, it would be safe to assume that the overestimation of intra-particle mass transfer limitations by the simulation is not as severe as the fit Knox parameter. It can simply be explained by the stagnant mobile phase assumed in the shell layers and the dead volume in the shell layer resulting from random-walk discretization.

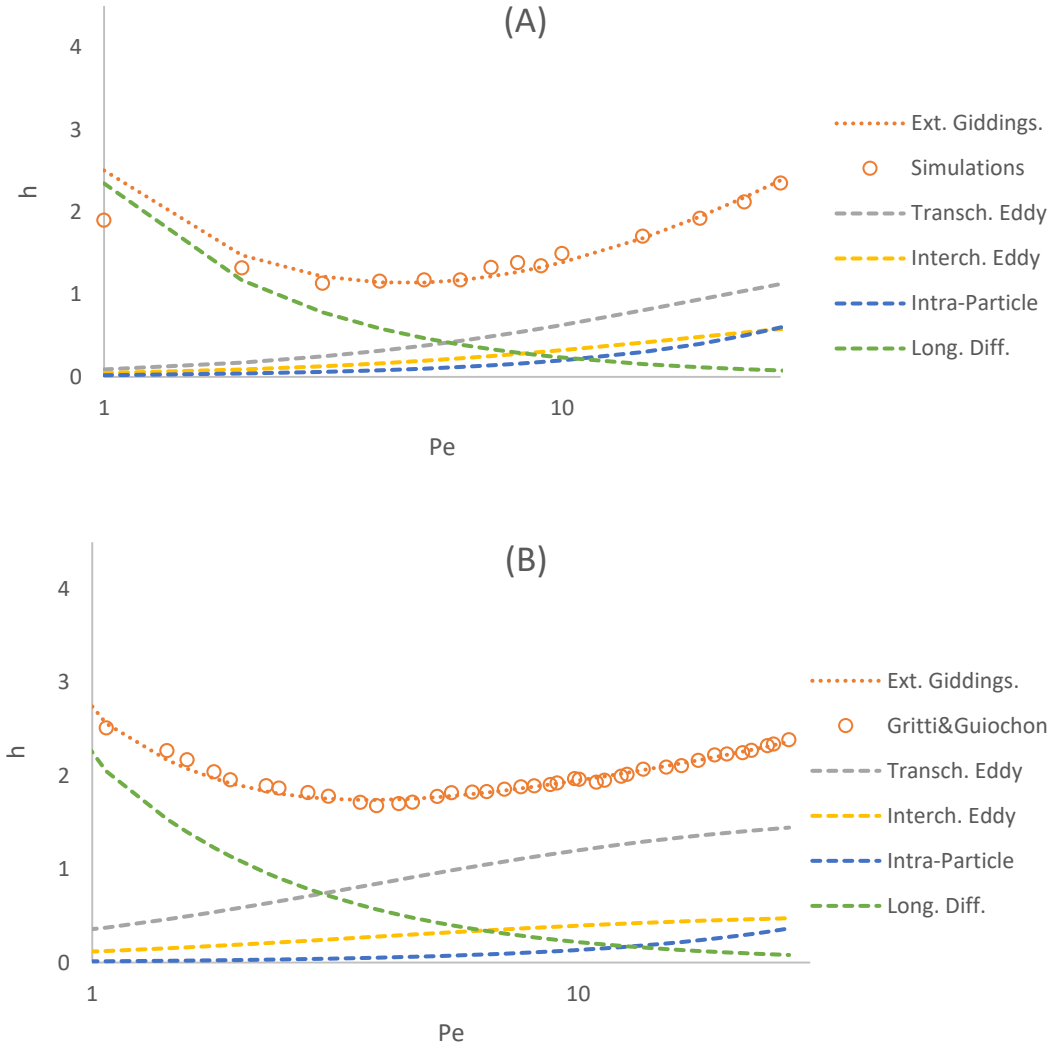


Figure 47: Extended Giddings model equations fit (dotted lines) to simulation data (A) and the experimental data from the study of Gritti & Guiochon (2011) (B). Contributions to reduced plate heights from transchannel eddy dispersion, interchannel eddy dispersion, intra-particle mass transfer limitations and longitudinal diffusion are dashed lines.

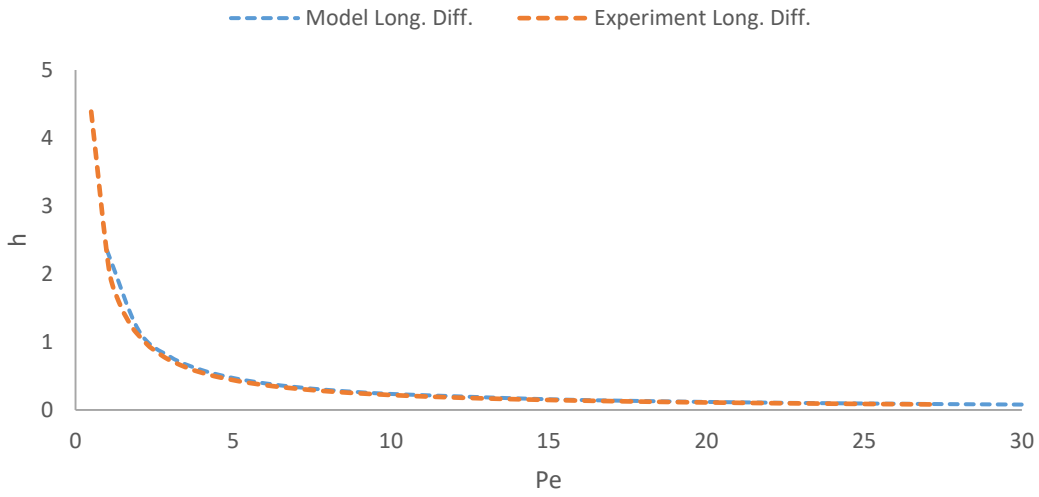


Figure 48: Comparison of longitudinal diffusion contributions to reduced plate heights predicted by the model to experimental data.

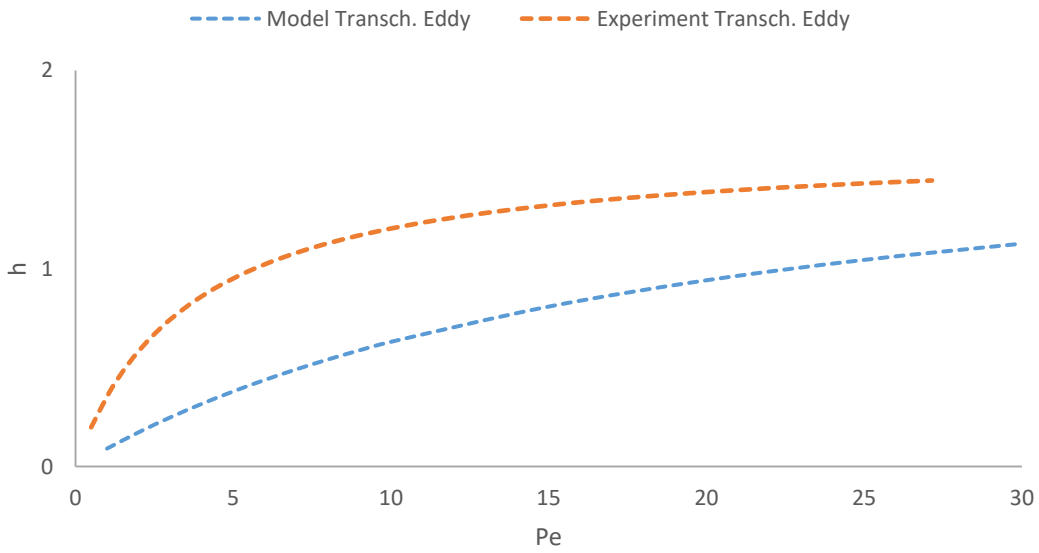


Figure 49: Comparison of transchannel eddy diffusion contributions to reduced plate heights predicted by the model to experimental data.

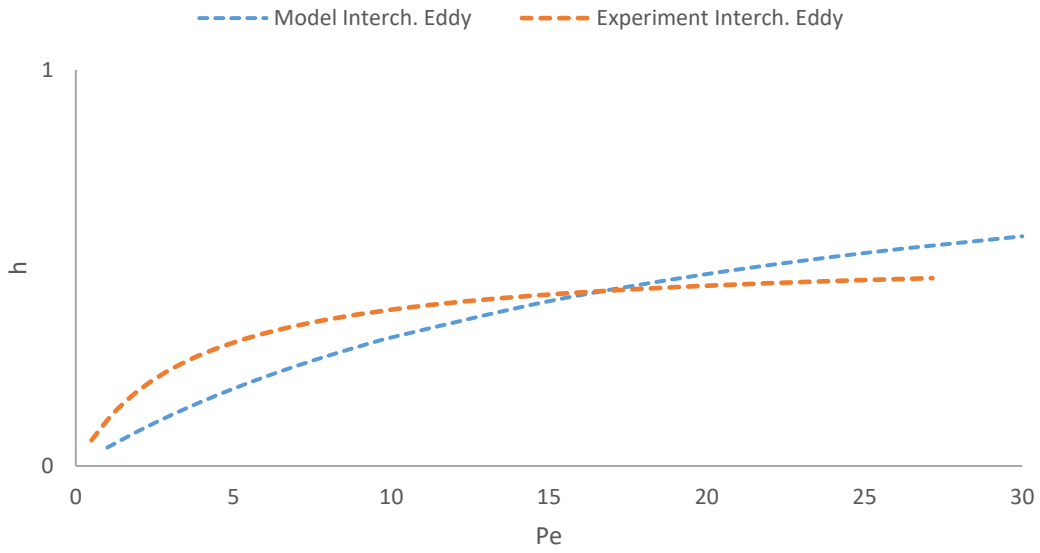


Figure 50: Comparison of interchannel channel eddy diffusion contributions to reduced plate heights predicted by the model to experimental data.

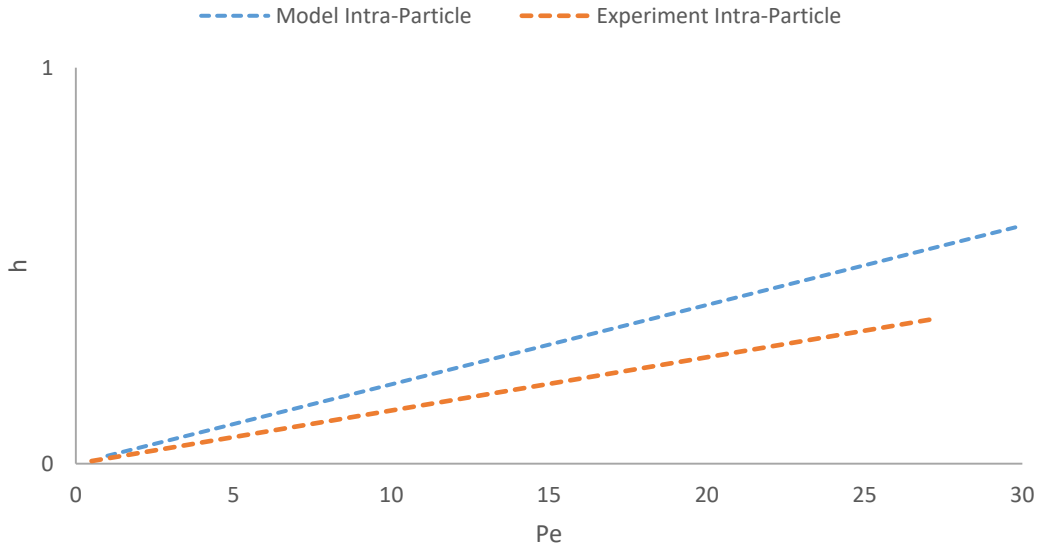


Figure 51: Comparison of intra-particle mass transfer limitation contributions to reduced plate heights predicted by the model to experimental data.

CHAPTER 5

CONCLUSIONS

A dispersion model that simulates dispersion of small molecules in an unbound liquid chromatography column was created throughout the thesis study. As the preliminary steps of building the model, random-walk diffusion simulations in unhindered stagnant media and periodic simulations with basic no-flux boundaries were prepared and their predictions were used to confirm validity of algorithms and Fortran codes.

An analytical geometry approach was adopted for reconstruction of core-shell particles. Core-to-particle ratio, particle diameter and amount of shell layers in the particle were used as defining parameters of the particle geometry to calculate center coordinates of spherical elements that collectively creates an ideal core-shell particle, coating a large core sphere with smaller shell side spheres very similar to actual production methods of core-shell particles. Resulting geometry was visually inspected by creating CAD images of the particle using the open-source drawing software, OpenSCAD. Then a periodic random jammed packing of 50 monodisperse hardspheres inside a unit cube was computationally generated using an algorithm developed Skoge et al. (2006). Packing of hardspheres was also inspected using OpenSCAD and it was found to be not usable directly in the simulations due to exclusion of any spheres located in neighboring periodic cells that would also appear in the main periodic cell. These spheres were dubbed invading spheres and they were manually inserted into the packing, rising total amount of spheres in the packing to 100. Corrected packing was easily scaled such that diameters of spheres in the packing

would be equal to that of reconstructed core-shell particles. Then the calculated geometry of core-shell particle was translated into each sphere in the packing to create the core-shell packing geometry. The packing geometry was also inspected using OpenSCAD and was found to be appropriate to use in collision control. Diffusion in the geometry was simulated for different core-shell particles and the predictions of effective diffusivity throughout the entire system was found to be of physically reasonable.

Pore-space of the core-shell particles was assumed to be stagnant, and corrected hardsphere packing geometry was used in COMSOL to obtain velocity field of periodic flow of water in the system geometry. Periodic boundaries were confirmed both visually by contour plots and direct comparison of the velocity profile at the coupled boundaries. Velocity field was obtained for different pressure gradients across the system. Stokes Flow limit for the system was determined by linear scaling and comparison of scaled velocity field and separately solved velocity field. A limiting Peclet number definition was derived and considered while carrying out the simulations.

A trilinear interpolation subroutine was written for interpolating velocity vectors near grid nodes and separately obtained velocity field was integrated to diffusion model to create a dispersion model. Dispersion model and interpolation subroutine was tested by simulating Taylor dispersion in a pipe. Predictions of the model in the pipe was in good agreement with the predictions of correlation derived by Taylor. Then dispersion in core-shell packing was simulated at different Peclet numbers. Predictions of the simulation was found to be in good agreement with experimentally obtained reduced plate height data found in the literature. Deviations of the model predictions from the experimental data were discussed. Lower reduced plate height values predicted by the model at lower Peclet numbers, the lower A and B terms in a Van Deemter type equation, were found to be due to effects of confinement being non-existent in the simulated geometry since periodic boundaries effectively creates an unbound column. The model was found to be over-predicting internal mass transfer resistances in the core-shell particles, the C term, most possibly due to dead-zones created by relatively

coarse definition of random-step size as 10% of the diameter of a shell side sphere where no path followed by a tracer can lead to. Although velocity recorrelation in simulations and stagnant pore-space assumption might also have minor contributions for the overestimation of the C term.

Overall, the dispersion model created for the study was found to be successfully predicting dispersion event in a very complex system like a random packing of porous materials, with explainable effects of simplifying assumptions. Basic analytical geometry approach for digital reconstruction of core-shell particles seems to be successful. Similar approaches for different systems might also be used for creating memory efficient simulations.

CHAPTER 6

RECOMMENDATIONS

Several recommendations for similar studies that can possibly be conducted in the future, or continuation of this specific study can be made as follows.

- Finer choice for the step-size of the random-walk should increase the accuracy of the model at higher Peclet numbers. However the non-linear relation between step-size and time increment must be very seriously taken into account, since reducing the step-size by a factor will increase the amount of random-steps that needs to be taken by a tracer by the square of the same factor which can dramatically increase the wall-clock time of Fortran programs.
- Point-like tracers can be assigned a finite size and shape, spherical preferably, and modifying the collision control algorithm accordingly would allow simulating dispersion of large molecules like some globular non-adsorbing proteins. This might also eliminate the need for finer step-size for better accuracy at high Pe if the tracer size is comparable to the chosen step-size.
- Bounce-back method applied when a collision is detected, requires less processing power but, for better accuracy, specular-reflection can be traded-off with wall-clock time.
- Adsorption-desorption could potentially be integrated to the model by using a probabilistic subroutine that would be invoked upon detecting collisions between tracers and the impermeable boundaries.

- COMSOL is an easy but non-free solution for fluid flow problem. For the liberation of the work from all software copyrights, velocity field can be obtained by Lattice-Boltzmann methods alike on open-source software such as Octave or Fortran.
- Velocity auto-correlation is known to be less apparent in periodic cells of random packings with greater cell length to sphere diameter ratios. The results are self-consistent in this study due to similar simulation durations in terms of convective-time measure. However using a more appropriate periodic packing would also increase the accuracy of the model as well as computation times.

REFERENCES

Angstmann, C. N., Donnelly, I. C., Henry, B. I., & Nichols, J. A. (2015). A discrete time random walk model for anomalous diffusion. Journal of Computational Physics, 293, 53–69. https://doi.org/10.1016/j.jcp.2014.08.003

Bauer, K. C., Göbel, M., Schwab, M.-L., Schermeyer, M.-T., & Hubbuch, J. (2016). Concentration-dependent changes in apparent diffusion coefficients as indicator for colloidal stability of protein solutions. International Journal of Pharmaceutics, 511(1), 276–287. https://doi.org/10.1016/j.ijpharm.2016.07.007

Bruns, S., Stoeckel, D., Smarsly, B. M., & Tallarek, U. (2012). Influence of particle properties on the wall region in packed capillaries. Journal of Chromatography A, 1268, 53–63. https://doi.org/10.1016/j.chroma.2012.10.027

Bruns, S., & Tallarek, U. (2011). Physical reconstruction of packed beds and their morphological analysis: Core-shell packings as an example. Journal of Chromatography A, 1218(14), 1849–1860. https://doi.org/10.1016/j.chroma.2011.02.013

Brutz, M., & Rajaram, H. (2017). Coarse-scale particle tracking approaches for contaminant transport in fractured rock. Applied Mathematical Modelling, 41, 549–561. https://doi.org/10.1016/j.apm.2016.09.023

Busani, O. (2017). Finite dimensional Fokker–Planck equations for continuous time random walk limits. Stochastic Processes and Their Applications, 127(5), 1496–1516. https://doi.org/10.1016/j.spa.2016.08.008

Cavazzini, A., Gritti, F., Kaczmarski, K., Marchetti, N., & Guiochon, G. (2007). Mass-transfer kinetics in a shell packing material for chromatography. Analytical Chemistry, 79(15), 5972–5979. https://doi.org/10.1021/ac070571a

Chen, F., Han, Y., Li, Y., & Yang, X. (2017). Periodic solutions of Fokker–Planck equations. Journal of Differential Equations, 263(1), 285–298. https://doi.org/10.1016/j.jde.2017.02.032

Chicone, C. (2017). Random Walks and Diffusion. An Invitation to Applied Mathematics. https://doi.org/10.1016/B978-0-12-804153-6.50009-9

Choi, B. S., Kang, H., & Choi, M. Y. (2017). Emergence of heavy-tailed skew distributions from the heat equation. Physica A: Statistical Mechanics and Its Applications, 470, 88–93. https://doi.org/10.1016/j.physa.2016.11.095

Daneyko, A., Höltzel, A., Khirevich, S., & Tallarek, U. (2011). Influence of the particle size distribution on hydraulic permeability and eddy dispersion in bulk packings. Analytical Chemistry, 83(10), 3903–3910. https://doi.org/10.1021/ac200424p

Daneyko, A., Hlushkou, D., Baranau, V., Khirevich, S., Seidel-Morgenstern, A., & Tallarek, U. (2015). Computational investigation of longitudinal diffusion, eddy dispersion, and trans-particle mass transfer in bulk, random packings of core-shell particles with varied shell thickness and shell diffusion coefficient. Journal of Chromatography A, 1407, 139–156. https://doi.org/10.1016/j.chroma.2015.06.047

Das, S., Deen, N. G., & Kuipers, J. A. M. (2017). A DNS study of flow and heat transfer through slender fixed-bed reactors randomly packed with spherical particles. Chemical Engineering Science, 160(October 2016), 1–19. https://doi.org/10.1016/j.ces.2016.11.008

Dentz, M., Kang, P. K., & Le Borgne, T. (2015). Continuous time random walks for non-local radial solute transport. Advances in Water Resources, 82, 16–26. https://doi.org/10.1016/j.advwatres.2015.04.005

Done, J. N., & Knox, J. H. (1972). The Performance of Packings in High Speed Liquid Chromatography II. ZIPAX® The Effect of Particle Size. Journal of Chromatographic Science , 10(10), 606–612. https://doi.org/10.1093/chromsci/10.10.606

Gao, Q., & Zou, M. Y. (2016). An analytical solution for two and three dimensional nonlinear Burgers' equation. Applied Mathematical Modelling, 45, 255–270. https://doi.org/10.1016/j.apm.2016.12.018

Gentile, F. S., Santo, I. De, D'Avino, G., Rossi, L., Romeo, G., Greco, F., … Maffettone, P. L. (2015). Hindered Brownian diffusion in a square-shaped geometry. Journal of Colloid and Interface Science, 447, 25–32. https://doi.org/10.1016/j.jcis.2015.01.055

Gharib, M., Khezri, M., & Foster, S. J. (2017). Meshless and analytical solutions to the time-dependent advection-diffusion-reaction equation with variable coefficients and boundary conditions. Applied Mathematical Modelling, 49, 220–242. https://doi.org/10.1016/j.apm.2017.04.033

Giddings, J. C. (1963). Reduced plate height equation: a common link between chromatographic methods. Journal of Chromatography, 13, 301–304. https://doi.org/10.1016/S0021-9673(01)95123-4

Giddings, J. C. (1965). Dynamics of Chromatography: Principles and Theory. New York: Marcel Dekker, Inc.

Giesche, H. (1994). Synthesis of monodispersed silica powders I. Particle properties and reaction kinetics. Journal of the European Ceramic Society, 14(3), 189–204. https://doi.org/10.1016/0955-2219(94)90087-6

Gritti, F., Cavazzini, A., Marchetti, N., & Guiochon, G. (2007). Comparison between the efficiencies of columns packed with fully and partially porous C18-bonded silica materials. Journal of Chromatography A, 1157(1–2), 289–303. https://doi.org/10.1016/j.chroma.2007.05.030

Gritti, F., & Guiochon, G. (2012). Measurement of the eddy dispersion term in chromatographic columns. II. Application to new prototypes of 2.3 and 3.2mm I.D. monolithic silica columns. Journal of Chromatography A, 1227, 82–95. https://doi.org/10.1016/j.chroma.2011.12.065

Gritti, F., & Guiochon, G. (2013). Effect of parallel segmented flow chromatography on the height equivalent to a theoretical plate II - Performances of 4.6mm×30mm columns packed with 2.6μm Accucore-C18 superficially porous particles. Journal of Chromatography A, 1314(70), 44–53. https://doi.org/10.1016/j.chroma.2013.08.060

Gritti, F., & Guiochon, G. (2013). Effect of parallel segmented flow chromatography on the height equivalent to a theoretical plate II - Performances of 4.6mm×30mm columns packed with 2.6μm Accucore-C18 superficially porous particles. Journal of Chromatography A, 1314, 44–53. https://doi.org/10.1016/j.chroma.2013.08.060

Gritti, F., Leonardis, I., Shock, D., Stevenson, P., Shalliker, A., & Guiochon, G. (2010). Performance of columns packed with the new shell particles, Kinetex-C18. Journal of Chromatography A, 1217(10), 1589–1603. https://doi.org/10.1016/j.chroma.2009.12.079

Gritti, F., Pynt, J., Soliven, A., Dennis, G. R., Shalliker, R. A., & Guiochon, G. (2014). Effect of parallel segmented flow chromatography on the height equivalent to a theoretical plate III - Influence of the column length, particle diameter, and the molecular weight of the analyte on the efficiency gain. Journal of Chromatography A, 1333, 32–44. https://doi.org/10.1016/j.chroma.2014.01.055

Guiochon, G., & Gritti, F. (2011). Shell particles, trials, tribulations and triumphs. Journal of Chromatography A, 1218(15), 1915–1938. https://doi.org/10.1016/j.chroma.2011.01.080

Hayes, R., Ahmed, A., Edge, T., & Zhang, H. (2014). Core-shell particles: Preparation, fundamentals and applications in high performance liquid chromatography. Journal of Chromatography A, 1357, 36–52. https://doi.org/10.1016/j.chroma.2014.05.010

Hlushkou, D., Bruns, S., & Tallarek, U. (2010). High-performance computing of flow and transport in physically reconstructed silica monoliths. Journal of Chromatography A, 1217(23), 3674–3682. https://doi.org/10.1016/j.chroma.2010.04.004

Hormann, K., & Tallarek, U. (2013). Analytical silica monoliths with submicron macropores: Current limitations to a direct morphology-column efficiency scaling. Journal of Chromatography A, 1312, 26–36. https://doi.org/10.1016/j.chroma.2013.08.087

Hormann, K., & Tallarek, U. (2014). Mass transport properties of second-generation silica monoliths with mean mesopore size from 5 to 25nm. Journal of Chromatography A, 1365, 94–105. https://doi.org/10.1016/j.chroma.2014.09.004

Horvath, C. G., Preiss, B. A., & Lipsky, S. R. (1967). Fast liquid chromatography: an investigation of operating parameters and the separation of nucleotides on pellicular ion exchangers. Analytical Chemistry, 39(12), 1422–8. https://doi.org/10.1021/ac60256a003

Jansi Rani, P. G., Kirthiga, M., Molina, A., Laborda, E., & Rajendran, L. (2017). Analytical solution of the convection-diffusion equation for uniformly accessible rotating disk electrodes via the homotopy perturbation method. Journal of Electroanalytical Chemistry, 799(April), 175–180. https://doi.org/10.1016/j.jelechem.2017.05.053

Kaczmarski, K., & Guiochon, G. (2007). Modeling of the mass-transfer kinetics in chromatographic columns packed with shell and pellicular particles. Analytical Chemistry, 79(12), 4648–4656. https://doi.org/10.1021/ac070209w

Khirevich, S., Höltzel, A., Daneyko, A., Seidel-Morgenstern, A., & Tallarek, U. (2011). Structure-transport correlation for the diffusive tortuosity of bulk, monodisperse, random sphere packings. Journal of Chromatography A, 1218(37), 6489–6497. https://doi.org/10.1016/j.chroma.2011.07.066

Khirevich, S., Höltzel, A., Seidel-Morgenstern, A., & Tallarek, U. (2009). Time and length scales of eddy dispersion in chromatographic beds. Analytical Chemistry, 81(16), 7057–7066. https://doi.org/10.1021/ac901187d

Khirevich, S., Höltzel, A., Seidel-Morgenstern, A., & Tallarek, U. (2012). Geometrical and topological measures for hydrodynamic dispersion in confined sphere packings at low column-to-particle diameter ratios. Journal of Chromatography A, 1262, 77–91. https://doi.org/10.1016/j.chroma.2012.08.086

Kim, A. S., & Chen, H. (2006). Diffusive tortuosity factor of solid and soft cake layers: A random walk simulation approach. Journal of Membrane Science, 279(1–2), 129–139. https://doi.org/10.1016/j.memsci.2005.11.042

Kirkland, J. J. (1969). Controlled surface porosity supports for high-speed gas and liquid chromatography. Analytical Chemistry, 41(1), 218–220. https://doi.org/10.1021/ac60270a054

Kirkland, J. J., Truszkowski, F. A., Dilks, C. H., & Engel, G. S. (2000). Superficially porous silica microspheres for fast high-performance liquid chromatography of macromolecules. Journal of Chromatography A, 890(1), 3–13. https://doi.org/10.1016/S0021-9673(00)00392-7

Knox, J. H. (2002). Band dispersion in chromatography - A universal expression for the contribution from the mobile zone. Journal of Chromatography A, 960(1–2), 7–18. https://doi.org/10.1016/S0021-9673(02)00240-6

Koku, H. (2011). Microstructure-based analysis and simulation of flow and mass transfer in chromatographic stationary phases (Ph.D Thesis, 2011). University of Delaware.

Koku, H., Maier, R. S., Schure, M. R., & Lenhoff, A. M. (2012). Modeling of dispersion in a polymeric chromatographic monolith. Journal of Chromatography A, 1237, 55–63. https://doi.org/10.1016/j.chroma.2012.03.005

Latour, L. L., Kleinberg, R. L., Mitra, P. P., & Sotak, C. H. (1995). Pore-Size Distributions and Tortuosity in Heterogeneous Porous Media. Journal of Magnetic Resonance, Series A. https://doi.org/10.1006/jmra.1995.1012

Ma, Y., Qi, L., Ma, J., Wu, Y., Liu, O., & Cheng, H. (2003). Large-pore mesoporous silica spheres: Synthesis and application in HPLC. Colloids and Surfaces A: Physicochemical and Engineering Aspects, 229(1–3), 1–8. https://doi.org/10.1016/j.colsurfa.2003.08.010

Maier, R. S., Kroll, D. M., Bernard, R. S., Howington, S. E., Peters, J. F., & Davis, H. T. (2000). Pore-scale simulation of dispersion. Physics of Fluids, 12(8), 2065–2079. https://doi.org/10.1063/1.870452

Müllner, T., Unger, K. K., & Tallarek, U. (2016). Characterization of microscopic disorder in reconstructed porous materials and assessment of mass transport-relevant structural descriptors. New J. Chem., 3993–4015. https://doi.org/10.1039/C5NJ03346B

Reising, A. E., Godinho, J. M., Jorgenson, J. W., & Tallarek, U. (2017). Bed morphological features associated with an optimal slurry concentration for reproducible preparation of efficient capillary ultrahigh pressure liquid chromatography columns. Journal of Chromatography A, 1504, 71–82. https://doi.org/10.1016/j.chroma.2017.05.007

Sabelfeld, K. K. (2017). A mesh free floating random walk method for solving diffusion imaging problems. Statistics & Probability Letters, 121, 6–11. https://doi.org/10.1016/j.spl.2016.10.006

Sattin, F. (2008). Fick's law and Fokker-Planck equation in inhomogeneous environments. Physics Letters, Section A: General, Atomic and Solid State Physics, 372(22), 3941–3945. https://doi.org/10.1016/j.physleta.2008.03.014

Sen, P. N. (2004). Time-dependent diffusion coefficient as a probe of geometry. Concepts in Magnetic Resonance Part A: Bridging Education and Research, 23(1), 1–21. https://doi.org/10.1002/cmr.a.20017

Skoge, M., Donev, A., Stillinger, F. H., & Torquato, S. (2007). Publisher's note: Packing hyperspheres in high-dimensional Euclidean spaces. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, 75(2), 1–11. https://doi.org/10.1103/PhysRevE.75.029901

Song, H., Vanderheyden, Y., Adams, E., Desmet, G., & Cabooter, D. (2016). Extensive database of liquid phase diffusion coefficients of some frequently used test

molecules in reversed-phase liquid chromatography and hydrophilic interaction liquid chromatography. Journal of Chromatography A, 1455, 102–112. https://doi.org/10.1016/j.chroma.2016.05.054

Stöber, W., Fink, A., & Bohn, E. (1968). Controlled growth of monodisperse silica spheres in the micron size range. Journal of Colloid and Interface Science, 26(1), 62–69. https://doi.org/10.1016/0021-9797(68)90272-5

Szymczak, P., & Ladd, A. J. C. (2003). Boundary conditions for stochastic solutions of the convection-diffusion equation. Physical Review E, 68(3), 36704. https://doi.org/10.1103/PhysRevE.68.036704

Taylor, G. (1953). Dispersion of Soluble Matter in Solvent Flowing Slowly through a Tube. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 219(1137), 186–203. https://doi.org/10.1098/rspa.1953.0139

Van Deemter, J. J., Klinkenberg, A., & Zuiderweg, F. J. (1956). Longitudinal diffusion and resistance to mass transfer as causes of nonideality in chromatography. Chemical Engineering Science, 5(6), 271–289. https://doi.org/10.1016/0009-2509(56)80003-1

Wang, L., & Cardenas, M. B. (2015). An efficient quasi-3D particle tracking-based approach for transport through fractures with application to dynamic dispersion calculation. Journal of Contaminant Hydrology, 179, 47–54. https://doi.org/10.1016/j.jconhyd.2015.05.007

Widiatmojo, A., Sasaki, K., Widodo, N. P., Sugai, Y., Sahzabi, A. Y., & Nguele, R. (2016). Predicting gas dispersion in large scale underground ventilation: A particle tracking approach. Building and Environment, 95, 171–181. https://doi.org/10.1016/j.buildenv.2015.07.025

Wu, Z., & Chen, G. Q. (2015). Axial diffusion effect on concentration dispersion. International Journal of Heat and Mass Transfer, 84, 571–577. https://doi.org/10.1016/j.ijheatmasstransfer.2015.01.045

Yang, B., & Liu, S. (2017). Closed-form analytical solutions of transient heat conduction in hollow composite cylinders with any number of layers. International Journal of Heat and Mass Transfer, 108, 907–917. https://doi.org/10.1016/j.ijheatmasstransfer.2016.12.020

Zhukovsky, K. V., & Srivastava, H. M. (2017). Analytical solutions for heat diffusion beyond Fourier law. Applied Mathematics and Computation, 293, 423–437. https://doi.org/10.1016/j.amc.2016.08.038

APPENDIX A

FORTRAN CODES

A.1. Validation of Free Diffusion Program

```
PROGRAM FREEDIFFUSION
IMPLICIT NONE
DOUBLE PRECISION :: DAB,DURATION,DT,DX,PI,FREQ,SUMM
INTEGER :: MAXRNG,NP,NS,B,AODP,I,J,K
DOUBLE PRECISION, ALLOCATABLE :: POSITIONOLD(:,:),POSITIONNEW(:,:)
DOUBLE PRECISION, ALLOCATABLE :: DISPLACEMENT(:,:),DABVSTIME(:)
DOUBLE PRECISION, ALLOCATABLE :: NUMBERSX(:),NUMBERSY(:),NUMBERSZ(:)

DAB=100
DURATION=1
NP=4000
DT=DURATION/100000
DX=SQRT(2*DAB*DT)
NS=DURATION/DT

MAXRNG=NS
ALLOCATE(NUMBERSX(MAXRNG))
ALLOCATE(NUMBERSY(MAXRNG))
ALLOCATE(NUMBERSZ(MAXRNG))

FREQ=DURATION/1000
B=INT(FREQ/DT)
AODP=INT(NS/B)
ALLOCATE(POSITIONNEW(NP,NS))
ALLOCATE(POSITIONOLD(NP,NS))
ALLOCATE(DISPLACEMENT(NP,AODP))

PI=ACOS(-1.0)

!!SETTING INITIAL POSITIONS!!
DO I=1,NP
    POSITIONOLD(I,1)=0
    POSITIONOLD(I,2)=0
    POSITIONOLD(I,3)=0
ENDDO

!!RANDOM-WALK!!
CALL INIT RANDOM SEED()
DO I=1,NP
```

```
  CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)

  DO J=1,NS
    POSITIONNEW(I,1)=POSITIONOLD(I,1)+(NUMBERSX(J)*DX)
    POSITIONNEW(I,2)=POSITIONOLD(I,2)+(NUMBERSY(J)*DX)
    POSITIONNEW(I,3)=POSITIONOLD(I,3)+(NUMBERSZ(J)*DX)
    POSITIONOLD (I,1)=POSITIONNEW (I,1)
    POSITIONOLD (I,2)=POSITIONNEW (I,2)
    POSITIONOLD (I,3)=POSITIONNEW (I,3)
     IF (MOD(J,B).EQ.0) THEN
      DISPLACEMENT(I,J/B)=SQRT(POSITIONNEW(I,1)&
      **2+ POSITIONNEW(I,2)**2+ POSITIONNEW(I,3)**2)
     ENDIF
  ENDDO
  WRITE(*,*) "FINISHED TRACER:",I
ENDDO

!!DATA TREATMENT AND SAVING TO FILE!!
ALLOCATE(DABVSTIME(AODP))

OPEN(10, FILE='DAB VS TIME DATA.TXT',STATUS='NEW',ACTION='WRITE')

DO I=1,AODP
  SUMM=0
  DO J=1,NP
    SUMM=SUMM+DISPLACEMENT(J,I)**2
  ENDDO
  DABVSTIME(I)=(SUMM/NP)/(6*I*B*DT)
  WRITE(10,*) DABVSTIME(I)
ENDDO


CONTAINS
!!SUBROUTINES!!
SUBROUTINE NUMBERS(NX,NY,NZ)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(OUT):: NX(MAXRNG),NY(MAXRNG),NZ(MAXRNG)
INTEGER:: A
DO A=1,MAXRNG
  CALL RANDOM NUMBER (NX(A))
  NX(A)=COS((FLOOR(2*NX(A)))*PI)
  CALL RANDOM_NUMBER (NY(A))
  NY(A)=COS((FLOOR(2*NY(A)))*PI)
  CALL RANDOM NUMBER (NZ(A))
  NZ(A)=COS((FLOOR(2*NZ(A)))*PI)
ENDDO
END SUBROUTINE NUMBERS

SUBROUTINE INIT RANDOM SEED()
INTEGER :: I, N, CLOCK
INTEGER, DIMENSION(:), ALLOCATABLE :: SEED
CALL RANDOM SEED(SIZE = N)
ALLOCATE(SEED(N))
CALL SYSTEM CLOCK(COUNT=CLOCK)
SEED = CLOCK + 37 * (/ (I - 1, I = 1, N) /)
CALL RANDOM SEED(PUT = SEED)
DEALLOCATE(SEED)
END SUBROUTINE

END PROGRAM FREEDIFFUSION
```

A.2. Validation of Periodic Boundaries

```
PROGRAM DIFFUSIONPERIODIC
IMPLICIT NONE
DOUBLE PRECISION :: DAB,DURATION,DT,DX,PI,FREQ,SUMM
INTEGER :: MAXRNG,NP,NS,B,AODP,I,J,K
DOUBLE PRECISION, ALLOCATABLE :: POSITIONOLD(:,:),POSITIONNEW(:,:)
DOUBLE PRECISION, ALLOCATABLE :: POSITIONI(:,:)
DOUBLE PRECISION, ALLOCATABLE :: DISPLACEMENT(:,:),DABVSTIME(:)
DOUBLE PRECISION, ALLOCATABLE :: NUMBERSX(:),NUMBERSY(:),NUMBERSZ(:)
DOUBLE PRECISION, ALLOCATABLE :: COLLISIONSITES(:,:)
DOUBLE PRECISION :: PLOCAL(3),C(3),R,LPC

!!SETTING SPHERE CENTER AND RADIUS
LPC=5
C(1)=LPC/2
C(2)=LPC/2
C(3)=LPC/2
R=LPC/2

DAB=100
NP=5000
DX=R/10
DT=(DX**2)/(2*DAB)
DURATION=3*LPC*LPC/(6*DAB)

NS=INT(DURATION/DT)
DURATION=NS*DT
write(*,*)duration, ns, dt
MAXRNG=NS
ALLOCATE(NUMBERSX(MAXRNG))
ALLOCATE(NUMBERSY(MAXRNG))
ALLOCATE(NUMBERSZ(MAXRNG))

FREQ=DURATION/399
B=INT(FREQ/DT)
AODP=INT(NS/B)
ALLOCATE(POSITIONNEW(NP,NS))
ALLOCATE(POSITIONOLD(NP,NS))
ALLOCATE(POSITIONI(NP,NS))
ALLOCATE(DISPLACEMENT(NP,AODP))
ALLOCATE(COLLISIONSITES(NP,3))

PI=ACOS(-1.0)

!!SETTING INITIAL POSITIONS!!
CALL INIT RANDOM SEED()

!RANDOM DISTRIBUTION
DO I=1,NP
100     CALL RANDOM NUMBER(POSITIONOLD(I,1))
    POSITIONOLD(I,1)=POSITIONOLD(I,1)*LPC
        CALL RANDOM NUMBER(POSITIONOLD(I,2))
    POSITIONOLD(I,2)=POSITIONOLD(I,2)*LPC
        CALL RANDOM_NUMBER(POSITIONOLD(I,3))
    POSITIONOLD(I,3)=POSITIONOLD(I,3)*LPC

    PLOCAL(1)=MODULO(POSITIONOLD(I,1),LPC)
    PLOCAL(2)=MODULO(POSITIONOLD(I,2),LPC)
    PLOCAL(3)=MODULO(POSITIONOLD(I,3),LPC)

        IF ((((PLOCAL(1)-C(1))**2)+((PLOCAL(2)-C(2))**2)+&
    &((PLOCAL(3)-C(3))**2)).LE.(R**2)) GOTO 100

        POSITIONI(I,1)=POSITIONOLD(I,1)
        POSITIONI(I,2)=POSITIONOLD(I,2)
        POSITIONI(I,3)=POSITIONOLD(I,3)
ENDDO

!POINT INJECTION
DO I=1,NP
```

```
    POSITIONOLD(I,1)=0

    POSITIONOLD(I,2)=0

    POSITIONOLD(I,3)=0

ENDDO
!!SELECT ONE INITIAL CONDITION, COMMENT THE OTHER

!!RANDOM-WALK!!
CALL INIT RANDOM SEED()

DO I=1,NP

  CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)

  DO J=1,NS
    POSITIONNEW(I,1)=POSITIONOLD(I,1)+(NUMBERSX(J)*DX)
    POSITIONNEW(I,2)=POSITIONOLD(I,2)+(NUMBERSY(J)*DX)
    POSITIONNEW(I,3)=POSITIONOLD(I,3)+(NUMBERSZ(J)*DX)
    PLOCAL(1)=MODULO(POSITIONNEW(I,1),LPC)
    PLOCAL(2)=MODULO(POSITIONNEW(I,2),LPC)
    PLOCAL(3)=MODULO(POSITIONNEW(I,3),LPC)

    IF ((((PLOCAL(1)-C(1))**2)+((PLOCAL(2)-C(2))**2)+&
    &((PLOCAL(3)-C(3))**2)).LE.(R**2)) THEN

        POSITIONNEW (I,1)=POSITIONOLD (I,1)
    POSITIONNEW (I,2)=POSITIONOLD (I,2)
    POSITIONNEW (I,3)=POSITIONOLD (I,3)

    PLOCAL(1)=MODULO(POSITIONNEW(I,1),LPC)
    PLOCAL(2)=MODULO(POSITIONNEW(I,2),LPC)
    PLOCAL(3)=MODULO(POSITIONNEW(I,3),LPC)

        COLLISIONSITES(I,1)=PLOCAL(1)
        COLLISIONSITES(I,2)=PLOCAL(2)
        COLLISIONSITES(I,3)=PLOCAL(3)

    ENDIF

        POSITIONOLD (I,1)=POSITIONNEW (I,1)
    POSITIONOLD (I,2)=POSITIONNEW (I,2)
    POSITIONOLD (I,3)=POSITIONNEW (I,3)

     IF (MOD(J,B).EQ.0) THEN
      DISPLACEMENT(I,J/B)=SQRT((POSITIONNEW(I,1)-POSITIONI(I,1))**2+ &
         &(POSITIONNEW(I,2)-POSITIONI(I,2))**2+                 (POSITIONNEW(I,3)-
POSITIONI(I,3))**2)
     ENDIF

  ENDDO
  WRITE(*,*) "FINISHED TRACER:",I
ENDDO


!!DATA TREATMENT AND SAVING TO FILE!!
ALLOCATE(DABVSTIME(AODP))

!DIFFUSIVITY
OPEN(10, FILE='DAB VS TIME DATA.TXT',STATUS='NEW',ACTION='WRITE')
WRITE(10,*)AODP
WRITE(10,*)DURATION
DO I=1,AODP
  SUMM=0
  DO J=1,NP
    SUMM=SUMM+DISPLACEMENT(J,I)**2
  ENDDO
  DABVSTIME(I)=(SUMM/NP)/(6*I*B*DT)
  WRITE(10,*) DABVSTIME(I)
```

```
ENDDO

!COLLISION SITES
OPEN(20, FILE='COLLISION.TXT',STATUS='NEW',ACTION='WRITE')
DO I=1,NP
WRITE(20,*)COLLISIONSITES(I,1),COLLISIONSITES(I,2),COLLISIONSITES(I,3)
ENDDO


CONTAINS
!!SUBROUTINES!!
SUBROUTINE NUMBERS(NX,NY,NZ)
IMPLICIT NONE
DOUBLE PRECISION, INTENT(OUT):: NX(MAXRNG),NY(MAXRNG),NZ(MAXRNG)
INTEGER:: A
DO A=1,MAXRNG
  CALL RANDOM NUMBER (NX(A))
  NX(A)=COS((FLOOR(2*NX(A)))*PI)
  CALL RANDOM_NUMBER (NY(A))
  NY(A)=COS((FLOOR(2*NY(A)))*PI)
  CALL RANDOM NUMBER (NZ(A))
  NZ(A)=COS((FLOOR(2*NZ(A)))*PI)
ENDDO
END SUBROUTINE NUMBERS

SUBROUTINE INIT_RANDOM_SEED()
INTEGER :: I, N, CLOCK
INTEGER, DIMENSION(:), ALLOCATABLE :: SEED
CALL RANDOM SEED(SIZE = N)
ALLOCATE(SEED(N))
CALL SYSTEM_CLOCK(COUNT=CLOCK)
SEED = CLOCK + 37 * (/ (I - 1, I = 1, N) /)
CALL RANDOM SEED(PUT = SEED)
DEALLOCATE(SEED)
END SUBROUTINE

END PROGRAM DIFFUSIONPERIODIC
```

A.3. Validation of Core-Shell Particle Geometry and Packing

```
PROGRAM CORESHELLMULTILAYER
IMPLICIT NONE
INTEGER :: I,J,K,L,M,N
INTEGER :: NOL,NOSRJP
INTEGER, ALLOCATABLE :: NOAC(:),NOSOACINT(:,:),NOSISL(:),NOSS(:),SC(:)
DOUBLE PRECISION :: PI,PR,CPRATIO,RCORE,RSHELL,ALPHA,RRATIO,RRJP
REAL, ALLOCATABLE :: RAUX(:,:),ZAUX(:,:),RSOI(:),NOSOAC(:,:)
REAL, ALLOCATABLE :: PGCC(:,:,:,:),CPGCC(:,:,:)
REAL, ALLOCATABLE :: RJP(:,:),CSPRJP(:,:,:,:)
INTEGER(8) :: TOTALSHELL,B,AODP,MAXRNG,RNGDUMMY
DOUBLE PRECISION :: POROSITY,POROSITYSHELL,FREQ,SUMDSQ,SUMDISP,MEANDISP,VARIANCE
DOUBLE PRECISION, ALLOCATABLE :: DISPDAT(:,:),DISPDATL(:,:)
DOUBLE PRECISION, ALLOCATABLE :: DABVSTIME(:),DSPRCO(:)

PI=ACOS(-1.0)
PR=5.0/2
CPRATIO=0.8
NOL=1
RCORE=PR*CPRATIO
RSHELL=(PR-RCORE)/(NOL*2)

!CALCULATE THE NUMBER OF AUX. CIRCLES IN EACH SHELL LAYER
ALLOCATE(NOSS(NOL))
ALLOCATE(NOAC(NOL))

        DO I=1,NOL
```

```
               ALPHA=ASIN(RSHELL/(RSHELL+RCORE))*2
               NOSS(I)=FLOOR(2*PI/ALPHA)

                       IF (MOD(NOSS(I),2).EQ.0) THEN
                              NOAC(I)=INT((NOSS(I)-2)/2.0)
                              ELSE
                              NOAC(I)=FLOOR(REAL(NOSS(I)/2.0))
                       ENDIF
               RCORE=RCORE+RSHELL*2

       ENDDO
RCORE=PR*CPRATIO

!CALCULATE RADII,Z-COORD. OF, AND NUMBER OF SHELL SPHERES ON, EACH AUX. CIRCLE
ALLOCATE(RSOI(NOL))
ALLOCATE(RAUX(NOL,MAXVAL(NOAC)))
ALLOCATE(NOSOAC(NOL,MAXVAL(NOAC)))
ALLOCATE(NOSOACINT(NOL,MAXVAL(NOAC)))
ALLOCATE(ZAUX(NOL,MAXVAL(NOAC)))

       DO I=1,NOL

               RSOI(I)=RCORE+2*RSHELL
               ALPHA=ASIN(RSHELL/(RSHELL+RCORE))*2
               NOSS(I)=FLOOR(2*PI/ALPHA)
               ALPHA=2*PI/NOSS(I)

               DO J=1,NOAC(I)
                       RAUX(I,J)=SIN(J*ALPHA)*(RCORE+RSHELL)
               ENDDO

               DO J=1,NOAC(I)
                       NOSOAC(I,J)=PI/ASIN(RSHELL/RAUX(I,J))
                       NOSOACINT(I,J)=INT(NOSOAC(I,J))
               ENDDO

               DO J=1,NOAC(I)
                       ZAUX(I,J)=COS(J*ALPHA)*(RCORE+RSHELL)
               ENDDO

               RCORE=RSOI(I)

       ENDDO

RCORE=PR*CPRATIO


!CALCULATE TOTAL NUMBER OF SHELL SPHERES IN EACH LAYER
ALLOCATE(NOSISL(NOL))

       DO I=1,NOL
               NOSISL(I)=0
       ENDDO

       DO I=1,NOL
               DO J=1,NOAC(I)
                       NOSISL(I)=NOSISL(I)+NOSOACINT(I,J)
               ENDDO
       ENDDO

       DO I=1,NOL
               IF (MOD(NOSS(I),4).EQ.0) THEN
                       NOSISL(I)=NOSISL(I)+2
                       ELSE
                       NOSISL(I)=NOSISL(I)+1
               ENDIF
       ENDDO

       NOSISL(1)=NOSISL(1)+1
```

```
!CALCULATE CENTER COORDINATES OF SHELL SPHERES

ALLOCATE(PGCC(NOL,MAXVAL(NOAC),MAXVAL(NOSOACINT),4))

        DO I=1,NOL
                DO J=1,NOAC(I)
                        DO K=1,NOSOACINT(I,J)

                                ALPHA=2*PI/NOSOACINT(I,J)

                                PGCC(I,J,K,1)=RAUX(I,J)*COS((K-1)*ALPHA)
                                PGCC(I,J,K,2)=RAUX(I,J)*SIN((K-1)*ALPHA)
                                PGCC(I,J,K,3)=ZAUX(I,J)
                                PGCC(I,J,K,4)=RSHELL

                        ENDDO
                ENDDO
        ENDDO

!COMBINE PGCC DIMENSIONS, ADD POLAR SHELL SPHERES AND CORE SPHERE
ALLOCATE(CPGCC(NOL,MAXVAL(NOSISL)+1,4))
ALLOCATE(SC(NOL))

        !COMBINING 2ND AND 3RD DIMENSIONS OF PGCC
        DO I=1,NOL
                SC(I)=0
                DO J=1,NOAC(I)
                        DO K=1,NOSOACINT(I,J)

                                SC(I)=SC(I)+1
                                CPGCC(I,SC(I),1)=PGCC(I,J,K,1)
                                CPGCC(I,SC(I),2)=PGCC(I,J,K,2)
                                CPGCC(I,SC(I),3)=PGCC(I,J,K,3)
                                CPGCC(I,SC(I),4)=PGCC(I,J,K,4)

                        ENDDO
                ENDDO
        ENDDO

        !ADDING POLAR SHELL SPHERES CENTER COORDINATES
        DO I=1,NOL
                IF (MOD(NOSS(I),2).EQ.0) THEN

                        CPGCC(I,SC(I)+1,1)=0
                        CPGCC(I,SC(I)+1,2)=0
                        CPGCC(I,SC(I)+1,3)=RSOI(I)-RSHELL
                        CPGCC(I,SC(I)+1,4)=RSHELL

                        CPGCC(I,SC(I)+2,1)=0
                        CPGCC(I,SC(I)+2,2)=0
                        CPGCC(I,SC(I)+2,3)=-(RSOI(I)-RSHELL)
                        CPGCC(I,SC(I)+2,4)=RSHELL

                        SC(I)=SC(I)+2

                        ELSE

                        CPGCC(I,SC(I)+1,1)=0
                        CPGCC(I,SC(I)+1,2)=0
                        CPGCC(I,SC(I)+1,3)=RSOI(I)-RSHELL
                        CPGCC(I,SC(I)+1,4)=RSHELL

                        SC(I)=SC(I)+1

                ENDIF
        ENDDO

        !ADDING CORE SPHERE   CENTER COORDINATES
        CPGCC(1,SC(1)+1,1)=0
        CPGCC(1,SC(1)+1,2)=0
        CPGCC(1,SC(1)+1,3)=0
```

```
        CPGCC(1,SC(1)+1,4)=RCORE
        SC(1)=SC(1)+1

!READING HARSPHERE PACKING GEOMETRY
OPEN (30, FILE= "packing.dat", STATUS="OLD", ACTION="READ")
300     FORMAT(F16.8,2F17.8)

ALLOCATE (RJP(NOSRJP,3))

        DO I=1,NOSRJP
                READ(30,300)RJP(I,1),RJP(I,2),RJP(I,3)
        ENDDO

        DO I=1,NOSRJP
                DO J=1,3
                        RJP(I,J)=(RJP(I,J))*RRATIO
                ENDDO
        ENDDO

ALLOCATE(CSPRJP(NOSRJP,NOL,MAXVAL(SC),4))

!COPYING CORE-SHELL PARTICLES INTO THE PACKING
        !XYZ COORDINATES ONLY
        DO I=1,NOSRJP
                DO J=1,NOL
                        DO K=1,SC(J)
                                DO L=1,3

                                        CSPRJP(I,J,K,L)=RJP(I,L)+CPGCC(J,K,L)

                                ENDDO
                        ENDDO
                ENDDO
        ENDDO

        !RADIUS ONLY
        DO I=1,NOSRJP
                DO J=1,NOL
                        DO K=1,SC(J)

                                        CSPRJP(I,J,K,4)=CPGCC(J,K,4)

                        ENDDO
                ENDDO
        ENDDO


!GENERATING OPENSCAD SCRIPT OF CORE-SHELL PACKING GEOMETRY FOR VISUAL INSPECTION
OPEN (40, FILE= "CORE-SHELL SCRIPT.TXT", STATUS="UNKNOWN", ACTION="WRITE")


WRITE(40,*)"union(csp) {"
DO I=1,NOSRJP
DO J=1,NOL
  DO K=1,SC(J)

        WRITE(40,*)"translate([", CSPRJP(I,J,K,1), "," , CSPRJP(I,J,K,2),&
        & ",", CSPRJP(I,J,K,3), "])"
                WRITE(40,*)"sphere(", CSPRJP(I,J,K,4), ", $fn=50);"

  ENDDO
ENDDO
ENDDO
WRITE(40,*)"}"


END PROGRAM CORESHELLMULTILAYER

!TO GENERATE THE SCRIPT FOR A SINGLE CORE-SHELL PARTICLE, SKIP
```

```
!"READING HARSPHERE PACKING GEOMETRY" AND "COPYING CORE-SHELL !PARTICLES INTO THE
PACKING", REMOVE THE OUTER-MOST DO LOOP IN THE !LAST NESTED DO LOOPS AND CHANGE THE
"CSPRJP(I,J,K,'1-4')" WITH !"CPGCC(J,K,1-4)"
```

A.4. Diffusion/Dispersion in Random Jammed Packing of Core-Shell Particles

```
!LINE NUMBERS,
!FOR FILES: MULTIPLES OF 10, EXCEPT MULTIPLES OF 100
!FOR FORMAT: ODD MULTIPLES OF 100
!FOR GOTO: EVEN MULTIPLES OF 100

PROGRAM CORESHELLDISPERSION
IMPLICIT NONE
INTEGER :: I,J,K,L,M,N

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DECLARATIONS - GEOMETRY RELATED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER :: NOL,NOSRJP
INTEGER(8), ALLOCATABLE :: NOAC(:),NOSOACINT(:,:),NOSISL(:),NOSS(:),SC(:)
DOUBLE PRECISION :: PI,PR,CPRATIO,RCORE,RSHELL,ALPHA,RRATIO,RRJP
REAL, ALLOCATABLE :: RAUX(:,:),ZAUX(:,:),RSOI(:),NOSOAC(:,:)
REAL, ALLOCATABLE :: PGCC(:,:,:,:),CPGCC(:,:,:)
REAL, ALLOCATABLE :: RJP(:,:),CSPRJP(:,:,:,:)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DECLARATIONS - DIFFUSION RELATED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER(8) :: NP,NS,COLLISIONCOUNT,MAXDIST
DOUBLE PRECISION :: DAB,DT,DX,DURATION,DIFTIME,DIVIDER,POLD(3),PNEW(3)
DOUBLE PRECISION :: SOIC(3),PLOC(3),DURATIOND
DOUBLE PRECISION, ALLOCATABLE :: POSITIONI(:,:),POSITIONOLD(:,:),POSITIONNEW(:,:)
REAL, ALLOCATABLE :: NUMBERSX(:),NUMBERSY(:),NUMBERSZ(:)
DOUBLE PRECISION, ALLOCATABLE :: PLOCAL(:)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DECLARATIONS - FLUID MECHANICS RELATED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER :: GRID
DOUBLE PRECISION :: CFSF,MAGSUM,MAG,PEAVG,PE,DURATIONC,UAVG
DOUBLE PRECISION, ALLOCATABLE :: VF(:,:,:,:),VELOCITY(:)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DECLARATIONS - DATA EXTRACTION RELATED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
INTEGER(8) :: TOTALSHELL,B,AODP,MAXRNG,RNGDUMMY
DOUBLE PRECISION :: POROSITY,POROSITYSHELL,FREQ,SUMDSQ,SUMDISP,MEANDISP,VARIANCE
```

```
DOUBLE PRECISION, ALLOCATABLE :: DISPLACEMENT(:,:),DISPDATL(:,:)
DOUBLE PRECISION, ALLOCATABLE :: DABVSTIME(:),DSPRCO(:)
CHARACTER(LEN=30) :: DATETAG
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!






!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DEFINITIONS - GEOMETRY RELATED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!ALPHA: HOLDS THE ANGLE OF SEPARATION BETWEEN AUX.CIRCLES AND SHELL SPHERES ON
!              THESE CIRCLES.

!PI: AS IN 3.14
PI=ACOS(-1.0)

!PR: RADIUS OF ENTIRE CORE-SHELL PARTICLE (IN MICROMETERS)
PR=5.0/2

!CPRATIO: CORE TO PARTICLE RADII RATIO
CPRATIO=0.73

!NOL: NUMBER OF SHELL LAYERS
NOL=3

!RCORE:RADIUS OF CORE SPHERE
RCORE=PR*CPRATIO


!RSHELL: RADIUS OF SHELL SPHERES
RSHELL=(PR-RCORE)/(NOL*2)

!::SC: SPHERE COUNTER IN A CERTAIN LAYER            (LAYER)
!          NECESSARY FOR KEEPING TRACK OF HOW MANY SHELL SPHERES THERE ARE
!          IN EACH SHELL LAYER SINCE SOME ELEMENTS IN CPGCC WILL BE EMPTY.

!::RAUX: RADIUS OF A CERTAIN AUX. CIRCLE            (LAYER,CIRCLE NUMBER)

!::ZAUX: Z-COORD. OF A CERTAIN AUX. CIRCLE          (LAYER,CIRCLE NUMBER)
!               DETERMINES Z-COORD. OF SHELL SPHERES ON THAT AUX. CIRCLE

!::RSOI: RADIUS OF A CERTAIN SPHERE OF INFLUENCE FROM INSIDE TO OUTSIDE

!::NOAC: NUMBER OF AUX.CIRCLES IN A CERTAIN SHELL LAYER            (LAYER)

!::NOSOAC: NUMBER OF SHELL SPHERES ON A CERTAIN AUX. CIRCLE
       (LAYER,CIRCLE NUMBER)
!                 REAL TYPE, FOR THE SAKE OF CALCULATIONS

!::NOSOACINT: INTEGER COUNTERPART OF NOSOAC
!                        HOLDS INTEGER PARTS OF THE ELEMENTS IN NOSOAC

!::NOSISL: TOTAL NUMBER OF SHELL SPHERES IN A CERTAIN SHELL LAYER  (LAYER)

!::NOSS: NUMBER OF SHELL SPHERES THAT CAN BE PLACED AROUND THE EQUATOR
!               DETERMINES NOAC AND NOSISL          (LAYER)

!::PGCC: CENTER COORDINATES OF THE SPHERES IN PROTOTYPE GEOMETRY
!               (LAYER, NOAC, NOSOAC, 4)
!               ELEMENTS  (:,:,:,4)  ARE  X,Y,Z  COORDINATES  AND  SPHERE  RADIUS
ACCORDINGLY
!               IE. (2,1,1,3) IS THE Z-COORD OF THE FIRST SHELL SPHERE LOCATED ON THE
!               FIRST AUX.CIRCLE OF THE SECOND SHELL LAYER

!::CPGCC: ANOTHER VERSION OF PGCC WITH 2ND AND 3RD DIMENSIONS COMBINED INTO ONE
!                (LAYER,SHELL SPHERE NUMBER,4)
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!NOSRJP: NUMBER OF SPHERES IN RANDOM JAMMED PACKING OF HARDSPHERES
NOSRJP=100                                 !DO NOT CHANGE

!RRJP: RADIUS OF HARDSPHERES IN RANDOM JAMMED PACKING OF HARDSPHERES
RRJP=0.2841570815121517/2     !DO NOT CHANGE

!RRATIO: RADII RATIO OF CORE-SHELL PARTICLE TO HARDSPHERES
RRATIO=PR/RRJP

!::RJP: RANDOM JAMMED PACKING OF HARDSPHERES IN UNIT CUBE  (NOSRJP,3)
!              HOLD CENTER COORDINATES OF HARDSPHERES IN THE PACKING

!::CSPRJP: CORE-SHELL PARTICLES IN RANDOM JAMMED PACKING
!                 HOLDS CENTER COORDINATES AND RADII OF CORE-SHELL PARTICLE ELEMENTS
!                 IN RANDOM JAMMED PACKING
!                 (NOSRJP,NOL,SHELL SPHERE NUMBER,4)


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!





!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DEFINITIONS - DIFFUSION RELATED
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DAB: DIFFUSION COEFFICIENT OF TRACERS (IN MICROMETER^2/SEC)
DAB=110

!NP: NUMBER OF TRACERS
NP=5000

!DX: RANDOM-STEP SIZE IN ALL AXIS
DX=RSHELL*0.2
WRITE(*,*)"DX:",DX
!DX=SQRT(2*DT*DAB)

!DT: TIME INCREMENT OF RANDOM-WALK (IN SEC)
DT=(DX**2)/(2.0*REAL(DAB))
!DT=0.00000284091
WRITE(*,*)"DT:",DT

!DURATION: DURATION OF THE EVENT (IN SEC)
!                 CHOSEN AS DIFFUSE TIME OR 15 X CONVECTIVE TIME (GREATER ONE)
!                 CONVECTIVE TIME (DURATIONC) IS CALCULATED IN FLUID MECHANICS
!                 DEFINITIONS
DURATIOND=0.2*((2*PR)**2)/(DAB)

!NS: TOTAL NUMBER OF RANDOM STEPS THAT WILL BE TAKEN BY A SINGLE TRACER
!        DEFINED AFTER DURATION IS DETERMINED AT FLUID MECHANICS DEFINITIONS

!::POSITIONI: HOLDS INITIAL POSITIONS OF TRACERS
        (NP,3)
ALLOCATE(POSITIONI(NP,3))

!::POSITIONOLD: HOLDS POSITIONS OF TRACER FROM THE PREVIOUS STEP          (NP,3)
ALLOCATE(POSITIONOLD(NP,3))

!::POSITIONNEW: HOLDS THE CALCULATED NEW POSITIONS OF TRACERS
        (NP,3)
ALLOCATE(POSITIONNEW(NP,3))

!MAXRNG: MAXIMUM AMOUNT OF RANDOM NUMBERS TO BE HOLD

!RNGDUMMY: KEEPS TRACK OF RANDOM NUMBER USAGE IN NUMBERSX/Y/Z ARRAYS
```

```
!::NUMBERSX: HOLDS RANDOM NUMBERS (-1 OR +1) FOR STEPS TAKEN IN X-AXIS

!::NUMBERSY: HOLDS RANDOM NUMBERS (-1 OR +1) FOR STEPS TAKEN IN Y-AXIS

!::NUMBERSZ: HOLDS RANDOM NUMBERS (-1 OR +1) FOR STEPS TAKEN IN Z-AXIS

!COLLISIONCOUNT: COUNTS THE TOTAL NUMBER OF ENCOUNTERS BETWEEN BOUNDARIES AND
!                             TRACERS. ASSIGNED VALUES LATER IN THE CODE.

!::PLOCAL: HOLDS LOCALIZED COORDINATES OF TRACERS. RELATED TO PERIODIC BOUNDARIES
ALLOCATE(PLOCAL(3))


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!







!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DEFINITIONS - DATA EXTRACTION
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DATETAG: HOLDS THE DATE FOR FILE NAMES
CALL FDATE(DATETAG)

!POROSITY: VOID VOLUME TO ENTIRE VOLUME RATIO FOR AN ENTIRE CORE-SHELL PARTICLE

!POROSITYSHELL: VOID VOLUME TO ENTIRE VOLUME RATIO FOR SHELL SIDE ONLY

!TOTALSHELL: TOTAL NUMBER OF SPHERES IN SHELL SIDE

!FREQ: FREQUENCY OF DISPLACEMENT DATA EXTRACTION.
!         I.E. ONCE EVERY 'FREQ' SECONDS

!B: EQUIVALENT AMOUNT OF RANDOM-STEPS REQUIRED FOR 'FREQ' SECONDS TO PASS

!AODP: TOTAL AMOUNT OF DATA POINTS COLLECTED

!::DISPDAT: DISPLACEMENT DATA        (NP,AMOUNT OF DATA POINTS)
!              HOLDS TRANSVERSE DISPLACEMENTS FOR DISPERSION SIMULATIONS

!::DABVSTIME: HOLDS CALCULATED DIFFUSION COEFFICIENT DATA (AMOUNT OF DATA POINTS)

!SUMDSQ: SUM OF SQUARED DISPLACEMENT FOR ALL TRACERS

!DISPDATL:: LONGITUDINAL DISPLACEMENT DATA

!DSPRCO:: HOLDS DISPERSION COEFFICIENTS

!MEANDISP: MEAN OF TRACER DISPLACEMENTS (L OR T)

!SUMDISP: SUM OF TRACER DISPLACEMENTS (L OR T)

!VARIANCE: VARIANCE OF TRACER DISPLACEMENTS (L OR T)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!







!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!OPEN FILES & FORMAT LINES
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
OPEN(10, FILE='VF.TXT',STATUS='OLD',ACTION='READ')  !VELOCITY FIELD - READ
```

```
100 FORMAT(98X,3ES34.15E2)                                                    !VELOCITY
FIELD FORMAT


OPEN (20, FILE="SIM SUMMARY "//DATETAG//".TXT", &   !SUMMARY FILE
&STATUS="UNKNOWN", ACTION="WRITE")

OPEN (30, FILE="PACKING.DAT", STATUS="OLD", &              !RANDOM PACKING - READ
&ACTION="READ")
300     FORMAT(F16.8,2F17.8)                                                  !RANDOM
PACKING FORMAT

!OPEN (40, FILE="DAB VS TIME "//DATETAG//".TXT", &  !DIFFUSIVITY DATA FILE
!&STATUS="UNKNOWN", ACTION="WRITE")                                  !WORKS        FOR
DIFFUSIVITY ONLY

500 FORMAT(F14.5)
        !DATA OUTPUT FORMAT

OPEN (50, FILE="LDISP VAR "//DATETAG//".TXT", &            !LONGITUDINAL
DISPLACEMENT
&STATUS="UNKNOWN", ACTION="WRITE")                                   !VARIANCE    DATA
OUTPUT FILE

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DEFINITIONS - FLIUD MECHANICS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!PE: DESIRED PECLET NUMBER (STOKES FLOW BECOMES NOT APPLICABLE FOR PE>500)
PE=6.0

!GRID: DIMENSIONS OF THE VELOCITY FIELD ARRAY
GRID=101
!VF: VELOCITY FIELD DATA IN REGULAR GRID
!       (X,Y,Z,VELOCITY VECTOR COMPONENT)
ALLOCATE(VF(GRID,GRID,GRID,3))

        !READ DATA
        DO I=1,GRID
                DO J=1,GRID
                        DO K=1,GRID
                                READ(10,100) VF(I,J,K,1),VF(I,J,K,2),VF(I,J,K,3)
                        ENDDO
                ENDDO
        ENDDO

        !CALCULATE AVERAGE PE
        MAGSUM=0.0
        DO I=1,GRID
                DO J=1,GRID
                        DO K=1,GRID

        MAG=SQRT(VF(I,J,K,1)**2+VF(I,J,K,2)**2+VF(I,J,K,3)**2)
                                MAGSUM=MAGSUM+MAG
                        ENDDO
                ENDDO
        ENDDO

        UAVG=1000000*MAGSUM/(GRID**3)
        WRITE(*,*)"AVERAGE U:",UAVG
        PEAVG=2*PR*UAVG/DAB
        WRITE(*,*)"AVERAGE PE:",PEAVG
        !CFSF: CREEPING FLOW SCALING FACTOR
        CFSF=PE/PEAVG
```

```
        WRITE(*,*)"CFSF:",CFSF
        !DURATIONC: CONVECTIVE TIME
        DURATIONC=10*(2*PR)/(UAVG*CFSF)
        WRITE(*,*)"10xCONVECTIVE TIME:",DURATIONC
        WRITE(*,*)"0.2xDIFFUSIVE TIME:",DURATIOND
        DURATION=MAX(DURATIOND,DURATIONC)
        NS=INT(DURATION/DT)
        WRITE(*,*)"NS:",NS
        DURATION=NS*DT

        !CALCULATE AVERAGE Z-COMPONENT
        MAGSUM=0.0
        DO I=1,GRID
                DO J=1,GRID
                        DO K=1,GRID
                                MAG=VF(I,J,K,3)
                                MAGSUM=MAGSUM+MAG
                        ENDDO
                ENDDO
        ENDDO
        UAVG=1000000*MAGSUM/(GRID**3)
        WRITE(*,*)"AVERAGE U-z:",UAVG*CFSF

        !SCALE FOR DT AND CONVERT TO MICROMETERS/SECOND
        DO I=1,GRID
                DO J=1,GRID
                        DO K=1,GRID
                                DO L=1,3
                                VF(I,J,K,L)=VF(I,J,K,L)*DT*CFSF*1000000.0
                                ENDDO
                        ENDDO
                ENDDO
        ENDDO

!VELOCITY: INTERPOLATED VELOCITY VECTOR COMPONENTS
!                 ALREADY SCALED FOR DT
ALLOCATE(VELOCITY(3))
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DEPENDENT DECLARATIONS FROM DATA EXTRACTION DEFINITIONS
!FREQ: FREQUENCY OF DISPLACEMENT DATA EXTRACTION.
!          I.E. ONCE EVERY 'FREQ' SECONDS
FREQ=DURATION/1000.0

!B: EQUIVALENT AMOUNT OF RANDOM-STEPS REQUIRED FOR 'FREQ' SECONDS TO PASS
B=FREQ/DT

!AODP: TOTAL AMOUNT OF DATA POINTS COLLECTED
AODP=NS/B
NS=AODP*B
DURATION=NS*DT
WRITE(*,*)"DURATION:",DURATION

!::DISPDAT: DISPLACEMENT DATA        (NP,AMOUNT OF DATA POINTS)
!              HOLDS TRANSVERSE DISPLACEMENTS FOR DISPERSION SIMULATIONS
ALLOCATE(DISPDAT(NP,AODP))

!::DABVSTIME: HOLDS CALCULATED DIFFUSION COEFFICIENT DATA (AMOUNT OF DATA POINTS)
ALLOCATE(DABVSTIME(AODP))

!SUMDSQ: SUM OF SQUARED DISPLACEMENT FOR ALL TRACERS

!DISPDATL:: LONGITUDINAL DISPLACEMENT DATA
ALLOCATE(DISPDATL(NP,AODP))

!DSPRCO:: HOLDS DISPERSION COEFFICIENTS
ALLOCATE(DSPRCO(AODP))
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DEPENDENT DECLARATIONS FROM DIFFUSION DEFINITIONS
!MAXRNG: MAXIMUM AMOUNT OF RANDOM NUMBERS TO BE HOLD
MAXRNG=NS
```

```
!::NUMBERSX: HOLDS RANDOM NUMBERS (-1 OR +1) FOR STEPS TAKEN IN X-AXIS
ALLOCATE(NUMBERSX(MAXRNG))

!::NUMBERSY: HOLDS RANDOM NUMBERS (-1 OR +1) FOR STEPS TAKEN IN Y-AXIS
ALLOCATE(NUMBERSY(MAXRNG))

!::NUMBERSZ: HOLDS RANDOM NUMBERS (-1 OR +1) FOR STEPS TAKEN IN Z-AXIS
ALLOCATE(NUMBERSZ(MAXRNG))
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!






!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!CALCULATE THE NUMBER OF AUX. CIRCLES IN EACH SHELL LAYER
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE(NOSS(NOL))
ALLOCATE(NOAC(NOL))

	DO I=1,NOL

		ALPHA=ASIN(RSHELL/(RSHELL+RCORE))*2
		NOSS(I)=FLOOR(2*PI/ALPHA)

			IF (MOD(NOSS(I),2).EQ.0) THEN
				NOAC(I)=INT((NOSS(I)-2)/2)
				ELSE
				NOAC(I)=FLOOR(REAL(NOSS(I)/2))
			ENDIF
		RCORE=RCORE+RSHELL*2

	ENDDO
RCORE=PR*CPRATIO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!





!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!CALCULATE RADII,Z-COORD. OF, AND NUMBER OF SHELL SPHERES ON, EACH AUX. CIRCLE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE(RSOI(NOL))
ALLOCATE(RAUX(NOL,MAXVAL(NOAC)))
ALLOCATE(NOSOAC(NOL,MAXVAL(NOAC)))
ALLOCATE(NOSOACINT(NOL,MAXVAL(NOAC)))
ALLOCATE(ZAUX(NOL,MAXVAL(NOAC)))

	DO I=1,NOL

		RSOI(I)=RCORE+2*RSHELL
		ALPHA=ASIN(RSHELL/(RSHELL+RCORE))*2
		NOSS(I)=FLOOR(2*PI/ALPHA)
		ALPHA=2*PI/NOSS(I)

		DO J=1,NOAC(I)
			RAUX(I,J)=SIN(J*ALPHA)*(RCORE+RSHELL)
		ENDDO

		DO J=1,NOAC(I)
			NOSOAC(I,J)=PI/ASIN(RSHELL/RAUX(I,J))
			NOSOACINT(I,J)=INT(NOSOAC(I,J))
		ENDDO

		DO J=1,NOAC(I)
```

```
                        ZAUX(I,J)=COS(J*ALPHA)*(RCORE+RSHELL)
                ENDDO

                RCORE=RSOI(I)

        ENDDO

RCORE=PR*CPRATIO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!





!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!CALCULATE TOTAL NUMBER OF SHELL SPHERES IN EACH LAYER
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE(NOSISL(NOL))

        DO I=1,NOL
                NOSISL(I)=0
        ENDDO

        DO I=1,NOL
                DO J=1,NOAC(I)
                        NOSISL(I)=NOSISL(I)+NOSOACINT(I,J)
                ENDDO
        ENDDO

        DO I=1,NOL
                IF (MOD(NOSS(I),2).EQ.0) THEN
                        NOSISL(I)=NOSISL(I)+2
                        ELSE
                        NOSISL(I)=NOSISL(I)+1
                ENDIF
        ENDDO

        NOSISL(1)=NOSISL(1)+1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!





!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!CALCULATE CENTER COORDINATES OF SHELL SPHERES
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE(PGCC(NOL,MAXVAL(NOAC),MAXVAL(NOSOACINT),4))

        DO I=1,NOL
                DO J=1,NOAC(I)
                        DO K=1,NOSOACINT(I,J)

                                ALPHA=2*PI/NOSOACINT(I,J)

                                PGCC(I,J,K,1)=RAUX(I,J)*COS((K-1)*ALPHA)
                                PGCC(I,J,K,2)=RAUX(I,J)*SIN((K-1)*ALPHA)
                                PGCC(I,J,K,3)=ZAUX(I,J)
                                PGCC(I,J,K,4)=RSHELL

                        ENDDO
                ENDDO
        ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!COMBINE PGCC DIMENSIONS, ADD POLAR SHELL SPHERES AND CORE SPHERE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE(CPGCC(NOL,MAXVAL(NOSISL),4))
ALLOCATE(SC(NOL))

        !COMBINING 2ND AND 3RD DIMENSIONS OF PGCC
        DO I=1,NOL
                SC(I)=0
                DO J=1,NOAC(I)
                        DO K=1,NOSOACINT(I,J)

                                SC(I)=SC(I)+1
                                CPGCC(I,SC(I),1)=PGCC(I,J,K,1)
                                CPGCC(I,SC(I),2)=PGCC(I,J,K,2)
                                CPGCC(I,SC(I),3)=PGCC(I,J,K,3)
                                CPGCC(I,SC(I),4)=PGCC(I,J,K,4)

                        ENDDO
                ENDDO
        ENDDO

        !ADDING POLAR SHELL SPHERES CENTER COORDINATES
        DO I=1,NOL
                IF (MOD(NOSS(I),2).EQ.0) THEN

                        CPGCC(I,SC(I)+1,1)=0
                        CPGCC(I,SC(I)+1,2)=0
                        CPGCC(I,SC(I)+1,3)=RSOI(I)-RSHELL
                        CPGCC(I,SC(I)+1,4)=RSHELL

                        CPGCC(I,SC(I)+2,1)=0
                        CPGCC(I,SC(I)+2,2)=0
                        CPGCC(I,SC(I)+2,3)=-(RSOI(I)-RSHELL)
                        CPGCC(I,SC(I)+2,4)=RSHELL

                        SC(I)=SC(I)+2

                        ELSE

                        CPGCC(I,SC(I)+1,1)=0
                        CPGCC(I,SC(I)+1,2)=0
                        CPGCC(I,SC(I)+1,3)=RSOI(I)-RSHELL
                        CPGCC(I,SC(I)+1,4)=RSHELL

                        SC(I)=SC(I)+1

                ENDIF
        ENDDO

        !ADDING CORE SPHERE   CENTER COORDINATES
        CPGCC(1,SC(1)+1,1)=0
        CPGCC(1,SC(1)+1,2)=0
        CPGCC(1,SC(1)+1,3)=0
        CPGCC(1,SC(1)+1,4)=RCORE
        SC(1)=SC(1)+1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DATA EXTRACTION - SUMMARY OF CORE-SHELL GEOMETRY
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


        WRITE(20,*)"--------------------------------------------------------"

        WRITE(20,*)"GEOMETRY"

        WRITE(20,*)"--------------------------------------------------------"

        WRITE(20,*)"RADIUS (MICROMETER):",PR

        WRITE(20,*)"CORE RADIUS (MICROMETER):",RCORE

        WRITE(20,*)"SHELL RADIUS (MICROMETER):",RSHELL

        WRITE(20,*)"NUMBER OF AUX CIRCLES IN"
        DO I=1,NOL
                WRITE(20,*)"   LAYER",I,":",NOAC(I)
        ENDDO

        WRITE(20,*)"NUMBER OF SPHERES ON"
        DO I=1,NOL
                WRITE(20,*)"LAYER",I
                DO J=1,NOAC(I)
                        WRITE(20,*)"          AUX. CIRCLE",J,":",NOSOACINT(I,J)
                ENDDO
        ENDDO

        WRITE(20,*)"NUMBER OF SHELL SPHERES IN"
        DO I=1,NOL
                IF (I.EQ.1) THEN
                WRITE(20,*)"   LAYER",I,":",SC(I)-1
                ELSE
                WRITE(20,*)"   LAYER",I,":",SC(I)
                ENDIF
        ENDDO
        TOTALSHELL=SUM(SC)-1
        WRITE(20,*)"   TOTAL:",TOTALSHELL

        !((4/3)*PI*(RSHELL^3)*(SUM(SC)-1)): TOTAL SHELL SPHERE VOLUME
        !(4/3)*PI*(RCORE^3): CORE SPHERE VOLUME
        !(4/3)*PI*(PR^3): ENTIRE VOLUME
        !((4/3)*PI*(PR^3)-(4/3)*PI*(RCORE^3)): SHELL VOLUME
        POROSITY=(((4/3)*PI*(PR**3))-(((4/3)*PI*(RSHELL**3)*(SUM(SC)-
1))+((4/3)*PI*&
                        &(RCORE**3))))/((4/3)*PI*(PR**3))
        POROSITYSHELL=(((4/3)*PI*(PR**3)-(4/3)*PI*(RCORE**3))-
((4/3)*PI*(RSHELL**3)&
                        &*(SUM(SC)-1)))/((4/3)*PI*(PR**3)-(4/3)*PI*(RCORE**3))

        WRITE(20,*)"POROSITY:",POROSITY

        WRITE(20,*)"SHELL POROSITY:",POROSITYSHELL

        WRITE(20,*)"--------------------------------------------------------"

        WRITE(20,*)"DIFFUSION"

        WRITE(20,*)"--------------------------------------------------------"

        WRITE(20,*)"DAB (MICROMETER^2/S):",DAB

        WRITE(20,*)"TIME INCREMENT(S):",DT

        WRITE(20,*)"STEP SIZE (MICROMETER):",DX

        WRITE(20,*)"DIFFUSIVE TIME (FOR DP):",DIFTIME*DIVIDER

        WRITE(20,*)"NUMBER OF TRACERS:",NP

        WRITE(20,*)"NUMBER OF RANDOM STEPS PER TRACER:",NS
```

```
       WRITE(20,*)"DURATION:", DURATION

       WRITE(20,*)"-------------------------------------------------------"

       WRITE(20,*)"FLOW"

       WRITE(20,*)"-------------------------------------------------------"

       WRITE(20,*)"SCALING FACTOR(BASE 400 PA):",CFSF

       WRITE(20,*)"PECLET NUMBER:", PE

       WRITE(*,*)"    TOTAL SHELL SPHERES:",TOTALSHELL
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!






!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!READING & SCALING RANDOM JAMMED PACKING OF HARDSPHERES
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE (RJP(NOSRJP,3))

       DO I=1,NOSRJP
              READ(30,300)RJP(I,1),RJP(I,2),RJP(I,3)
       ENDDO

       DO I=1,NOSRJP
              DO J=1,3
                     RJP(I,J)=(RJP(I,J))*RRATIO
              ENDDO
       ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!






!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!PLACING CORE-SHELL GEOMETRY INTO HARDSPHERES INSIDE THE PACKING
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
ALLOCATE(CSPRJP(NOSRJP,NOL,MAXVAL(SC),4))

       !XYZ COORDINATES ONLY
       DO I=1,NOSRJP
              DO J=1,NOL
                     DO K=1,SC(J)
                            DO L=1,3

                                   CSPRJP(I,J,K,L)=RJP(I,L)+CPGCC(J,K,L)

                            ENDDO
                     ENDDO
              ENDDO
       ENDDO

       !RADIUS ONLY
       DO I=1,NOSRJP
              DO J=1,NOL
                     DO K=1,SC(J)

                                   CSPRJP(I,J,K,4)=CPGCC(J,K,4)

                     ENDDO
```

```
              ENDDO
       ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!






!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DETERMINING INITIAL CONDITIONS OF TRACERS
!RANDOMLY PLACED THROUGHOUT THE INTER-PARTICLE SPACE
!NO TRACER STARTS INSIDE THE PORES OF ANY CORE-SHELL PARTICLE
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
CALL INIT_RANDOM_SEED()

       DO I=1,NP

              COLLISIONCOUNT=0
200           CALL RANDOM_NUMBER(NUMBERSX(1))
              CALL RANDOM_NUMBER(NUMBERSY(1))
              CALL RANDOM_NUMBER(NUMBERSZ(1))

              POSITIONI(I,1)=(NUMBERSX(1))*RRATIO
              POSITIONI(I,2)=(NUMBERSY(1))*RRATIO
              POSITIONI(I,3)=(NUMBERSZ(1))*RRATIO

              PLOCAL(1)=POSITIONI(I,1)
              PLOCAL(2)=POSITIONI(I,2)
              PLOCAL(3)=POSITIONI(I,3)

                     DO J=1,NOSRJP

                            IF            ((((PLOCAL(1)-RJP(J,1))**2)+((PLOCAL(2)-
RJP(J,2))**2)+((PL&
                                   &OCAL(3)-RJP(J,3))**2)).LE. (PR**2)) THEN

                                   COLLISIONCOUNT=COLLISIONCOUNT+1
                                   GOTO 200

                            ENDIF

                     ENDDO

!             WRITE(*,*) "TRACER (",I,") PLACED AFTER (",COLLISIONCOUNT,") TRIES."

                     DO J=1,3
                            POSITIONOLD(I,J)=POSITIONI(I,J)
                     ENDDO
       ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!






!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!RANDOM-WALK & DISPERSION EVENT
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

CALL INIT_RANDOM_SEED()

       !$omp                          parallel                                   do
private(j,k,l,m,plocal,numbersx,numbersy,numbersz,rngdummy,maxdist,velocity,pold,pn
ew,soic,ploc)
       DO I=1,NP
```

```
               !$omp critical
               CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)
               !$omp end critical

                      DO J=1,NS
                      !!!COMMENT THIS BLOCK, AND CONVECTIVE TERMS BELOW FOR DIFFUSION
ONLY
                              !CALCULATE LOCAL POSITION FOR INTERPOLATION
                              PLOCAL(1)=MODULO(POSITIONOLD(I,1),RRATIO)
                              PLOCAL(2)=MODULO(POSITIONOLD(I,2),RRATIO)
                              PLOCAL(3)=MODULO(POSITIONOLD(I,3),RRATIO)


                              !CALL INTERPOLATION
                              !!$omp critical
                              CALL GIMME VELOCITY(PLOCAL,VF,RRATIO,GRID,VELOCITY)
                              !!$omp end critical
                      !!!/COMMENT  THIS  BLOCK,  AND  CONVECTIVE  TERMS  BELOW  FOR
DIFFUSION ONLY

                              !MOVE                                          TRACERS
(OLDPOSITION+DIFFUSIVETERM+CONVECTIVETERM)
                              RNGDUMMY=INT(MOD((J-1),MAXRNG))+1

       POSITIONNEW(I,1)=POSITIONOLD(I,1)+(NUMBERSX(RNGDUMMY)*DX)+VELOCITY(1)

       POSITIONNEW(I,2)=POSITIONOLD(I,2)+(NUMBERSY(RNGDUMMY)*DX)+VELOCITY(2)


       POSITIONNEW(I,3)=POSITIONOLD(I,3)+(NUMBERSZ(RNGDUMMY)*DX)+VELOCITY(3)

                              PLOCAL(1)=MODULO(POSITIONNEW(I,1),RRATIO)
                              PLOCAL(2)=MODULO(POSITIONNEW(I,2),RRATIO)
                              PLOCAL(3)=MODULO(POSITIONNEW(I,3),RRATIO)

                                      !CHECKING  IF  THE  TRACER  IS  IN  INTER-PARTICLE
SPACE OR NOT
                                      DO K=1,NOSRJP

                                             IF                        ((((PLOCAL(1)-
RJP(K,1))**2)+((PLOCAL(2)-RJP(K,2))**2&
                                                    &)+((PLOCAL(3)-RJP(K,3))**2))
.LE.(PR**2)) THEN

                                             !!!COMMENT THIS BLOCK FOR SIMLATIONS IN
HARDSPHERE PACKING ONLY
                                                    !CHECKING  IN  WHICH  SHELL  LAYER
THE TRACER IS
                                                    DO L=1,NOL

                                                           IF         ((((PLOCAL(1)-
RJP(K,1))**2)+((PLOCAL(2)-RJP(&

       &K,2))**2)+((PLOCAL(3)-RJP(K,3))**2)) .LE.(&
                                                                  &RSOI(L)**2))
THEN

                                                                  !CHECKING     ALL
SPHERES IN THE SHELL LAYER
                                                                  DO M=1,SC(L)

                                                                         IF
((((PLOCAL(1)-CSPRJP(K,L,M,1))**2)+(&

       &(PLOCAL(2)-CSPRJP(K,L,M,2))**2)+((&

       &PLOCAL(3)-CSPRJP(K,L,M,3))**2)) &

       &.LE.(CSPRJP(K,L,M,4)**2)) THEN
                                             !!!/COMMENT THIS BLOCK FOR SIMLATIONS IN
HARDSPHERE PACKING ONLY
```

```
       POSITIONNEW(I,1)=POSITIONOLD(I,1)

       POSITIONNEW(I,2)=POSITIONOLD(I,2)

       POSITIONNEW(I,3)=POSITIONOLD(I,3)

                                                                    GOTO 400

                                          !!!COMMENT THIS BLOCK FOR SIMLATIONS IN
HARDSPHERE PACKING ONLY
                                                                    ENDIF

                                                                ENDDO

                                                            ENDIF

                                                   ENDDO
                                          !!!/COMMENT THIS BLOCK FOR SIMLATIONS IN
HARDSPHERE PACKING ONLY

                                          ENDIF

                                    ENDDO

400                          POSITIONOLD(I,1)=POSITIONNEW(I,1)
                             POSITIONOLD(I,2)=POSITIONNEW(I,2)
                             POSITIONOLD(I,3)=POSITIONNEW(I,3)

                             IF (MOD(J,B).EQ.0) THEN
                                    !LONGITUDINAL DISPLACEMENTS (USED IN DISPERSION
SIMULATIONS)
                                    DISPDATL(I,J/B)=ABS(POSITIONNEW(I,3)-
POSITIONI(I,3))
                                    !TOTAL   DISPLACEMENTS   (USED   IN   DIFFUSION
SIMULATIONS)
                                    !   DISPLACEMENT(I,J/B)=SQRT((POSITIONNEW(I,1)-
POSITIONI(I,1))**2+&
                                    !                           &(POSITIONNEW(I,2)-
POSITIONI(I,2))**2+(POSITIONNEW(I,3)-PO&
                                    ! &SITIONI(I,3))**2)
                             ENDIF

                      ENDDO

600            WRITE(*,*)"FINISHED TRACER:",I

       ENDDO
       !$omp end parallel do
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!




!COMMENT 'THE OTHER' DATA EXTRACTION.
!I.E. COMMENT DIFFUSIVITY DATA BLOCK FOR DISPERSION SIMULATIONS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DATA EXTRACTION - LONGITUDINAL DISPERSION COEFFICIENT
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       DO I=1,AODP

              SUMDISP=0.0

              DO J=1,NP
                     SUMDISP=SUMDISP+DISPDATL(J,I)
              ENDDO
```

```
               MEANDISP=SUMDISP/NP

               VARIANCE=0.0

               DO J=1,NP
                      VARIANCE=VARIANCE+(((DISPDATL(J,I)-MEANDISP)**2)/NP)
               ENDDO

               DSPRCO(I)=0.5*(VARIANCE/(I*B*DT))

               WRITE(50,500) VARIANCE

       ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!







!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!DATA EXTRACTION - TIME DEPENDENT DIFFUSIVITY
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DO I=1,AODP
  ! SUMM=0
  ! DO J=1,NP
    ! SUMM=SUMM+DISPLACEMENT(J,I)**2
  ! ENDDO
  ! DABVSTIME(I)=(SUMM/NP)/(6*I*B*DT)
  ! WRITE(40,500) DABVSTIME(I)
! ENDDO
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!







!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!WRITING START AND END DATES OF COMPUTATION TO SIMLULATION SUMMARY
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WRITE(20,*)"------------------------------------------------------------"
WRITE(20,*)"COMPUTATION HAD"
WRITE(20,*)"STARTED ON:",DATETAG
WRITE(*,*)"STARTED ON:",DATETAG
CALL FDATE(DATETAG)
WRITE(20,*)"ENDED ON:",DATETAG
WRITE(*,*)"ENDED ON:",DATETAG
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!



CONTAINS

SUBROUTINE INIT_RANDOM_SEED()
!SETS RANDOM SEED USING SYSTEM CLOCK
INTEGER :: I, N, CLOCK
INTEGER, DIMENSION(:), ALLOCATABLE :: SEED
CALL RANDOM_SEED(SIZE = N)
ALLOCATE(SEED(N))
CALL SYSTEM_CLOCK(COUNT=CLOCK)
SEED = CLOCK + 37 * (/ (I - 1, I = 1, N) /)
CALL RANDOM_SEED(PUT = SEED)
DEALLOCATE(SEED)
END SUBROUTINE
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SUBROUTINE NUMBERS(NX,NY,NZ)
!GENERATES RANDOM NUMBERS, EITHER -1 OR +1
IMPLICIT NONE
REAL, INTENT(OUT):: NX(MAXRNG),NY(MAXRNG),NZ(MAXRNG)
INTEGER:: A
DO A=1,MAXRNG
       CALL RANDOM_NUMBER (NX(A))
       NX(A)=COS((FLOOR(2*NX(A)))*PI)
       CALL RANDOM NUMBER (NY(A))
       NY(A)=COS((FLOOR(2*NY(A)))*PI)
       CALL RANDOM NUMBER (NZ(A))
       NZ(A)=COS((FLOOR(2*NZ(A)))*PI)
ENDDO
END SUBROUTINE NUMBERS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SUBROUTINE GIMME VELOCITY(LOCALP,VFI,LENGTH,GRIDD,VEL)
!TRILINEAR INTERPOLATION SUBROUTINE
IMPLICIT NONE
INTEGER, INTENT(IN) :: GRIDD
DOUBLE PRECISION, INTENT(IN) :: VFI(GRIDD,GRIDD,GRIDD,3)
DOUBLE PRECISION, INTENT(IN) :: LOCALP(3),LENGTH
DOUBLE PRECISION, INTENT(OUT) :: VEL(3)
DOUBLE PRECISION :: XUNIT,YUNIT,ZUNIT,MAIN(8),FIRST(4),SECOND(2)
INTEGER :: EDGE(3),DUMMY
!FIELD ARRAY INCLUDES GRID^3 UNIT CUBES THAT HOLD
!VELOCITY VECTORS INSIDE PERIODIC CUBE

!FIND WHICH UNIT CUBE THE TRACER IS IN
EDGE(1)=INT(LOCALP(1)/(LENGTH/100))+1
EDGE(2)=INT(LOCALP(2)/(LENGTH/100))+1
EDGE(3)=INT(LOCALP(3)/(LENGTH/100))+1

!FIND NORMALIZED POSITION OF TRACER INSIDE UNIT CUBE
XUNIT=MOD(LOCALP(1),(LENGTH/100))/(LENGTH/100)
YUNIT=MOD(LOCALP(2),(LENGTH/100))/(LENGTH/100)
ZUNIT=MOD(LOCALP(3),(LENGTH/100))/(LENGTH/100)

!INTERPOLATE (COMPONENT BY COMPONENT)
DO DUMMY=1,3

!SET VECTOR COMPONENT AT 8 CORNERS OF UNIT CUBE
MAIN(1)=VFI(EDGE(1),EDGE(2),EDGE(3),DUMMY)
MAIN(2)=VFI(EDGE(1)+1,EDGE(2),EDGE(3),DUMMY)
MAIN(3)=VFI(EDGE(1),EDGE(2),EDGE(3)+1,DUMMY)
MAIN(4)=VFI(EDGE(1)+1,EDGE(2),EDGE(3)+1,DUMMY)
MAIN(5)=VFI(EDGE(1),EDGE(2)+1,EDGE(3),DUMMY)
MAIN(6)=VFI(EDGE(1)+1,EDGE(2)+1,EDGE(3),DUMMY)
MAIN(7)=VFI(EDGE(1),EDGE(2)+1,EDGE(3)+1,DUMMY)
MAIN(8)=VFI(EDGE(1)+1,EDGE(2)+1,EDGE(3)+1,DUMMY)

!INTERPOLATE ALONG X-AXIS (REDUCES TO BILINEAR)
FIRST(1)=MAIN(1)+((MAIN(2)-MAIN(1))*XUNIT)
FIRST(2)=MAIN(3)+((MAIN(4)-MAIN(3))*XUNIT)
FIRST(3)=MAIN(5)+((MAIN(6)-MAIN(5))*XUNIT)
FIRST(4)=MAIN(7)+((MAIN(8)-MAIN(7))*XUNIT)

!INTERPOLATE ALONG Z-AXIS (REDUCES TO LINEAR)
SECOND(1)=FIRST(1)+((FIRST(2)-FIRST(1))*ZUNIT)
SECOND(2)=FIRST(3)+((FIRST(4)-FIRST(3))*ZUNIT)

!INTERPOLATE ALONG Y-AXIS
VEL(DUMMY)=SECOND(1)+((SECOND(2)-SECOND(1))*YUNIT)

ENDDO

END SUBROUTINE GIMME VELOCITY
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

END PROGRAM CORESHELLDISPERSION
```

APPENDIX B

DISPERSION COEFFICIENTS

B.1. In Taylor Dispersion Simulation

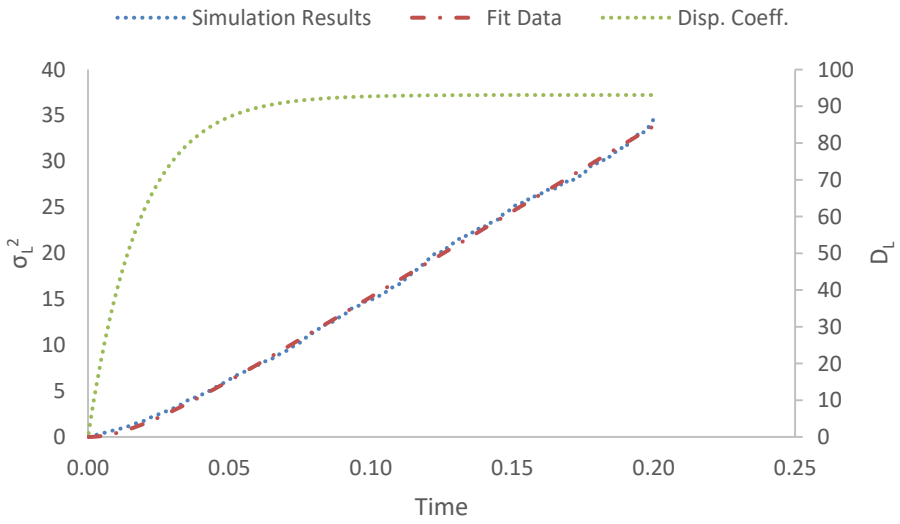


Figure B.1: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 1.13$. Fit parameters are, $A = 186.2$ and $k = 54.9$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

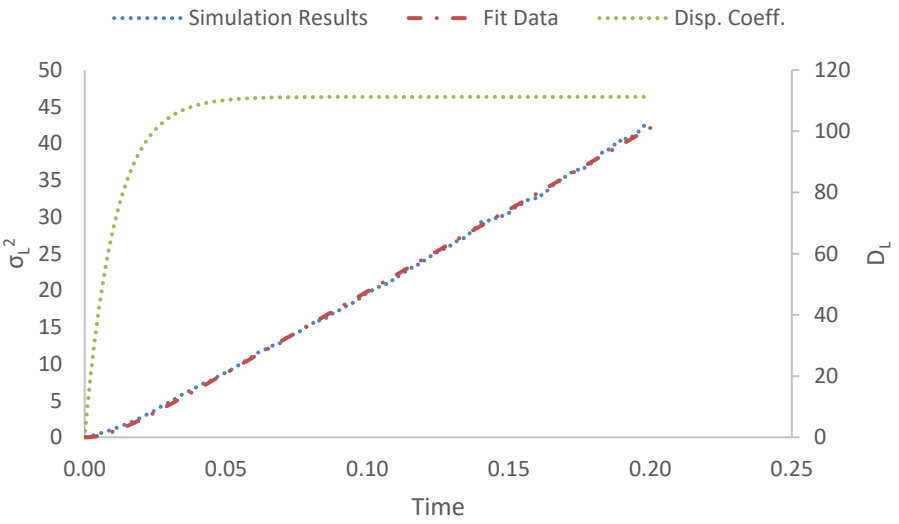


Figure B.2: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 2.25$. Fit parameters are, $A = 222.4$ and $k = 93.6$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

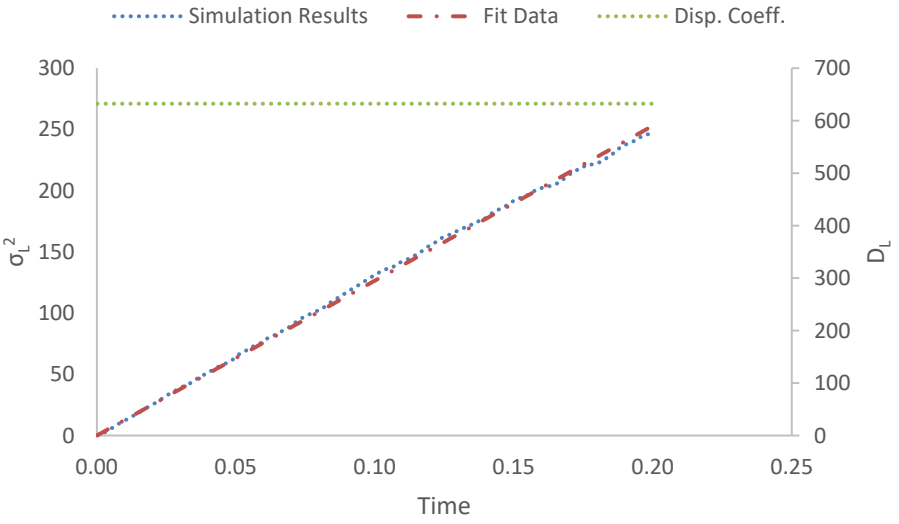


Figure B.3: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 22.5$. Fit parameters are, $A = 1264.6$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

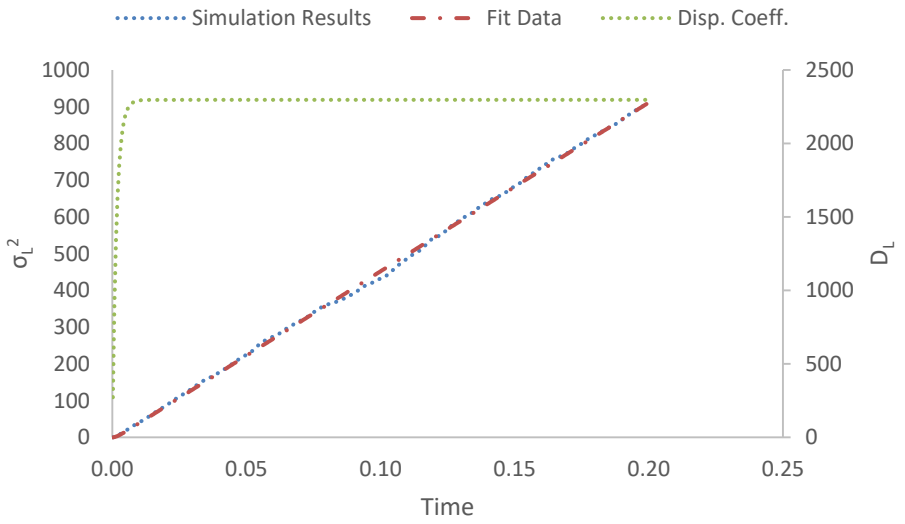


Figure B.4: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 45.0$. Fit parameters are, $A = 4593.4$ and $k = 632.1$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

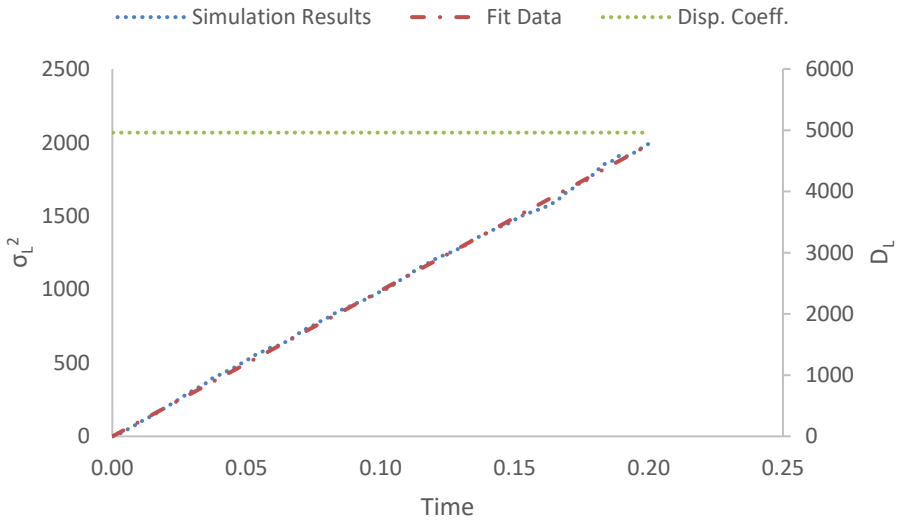


Figure B.5: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 67.5$. Fit parameters are, $A = 9922.0$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

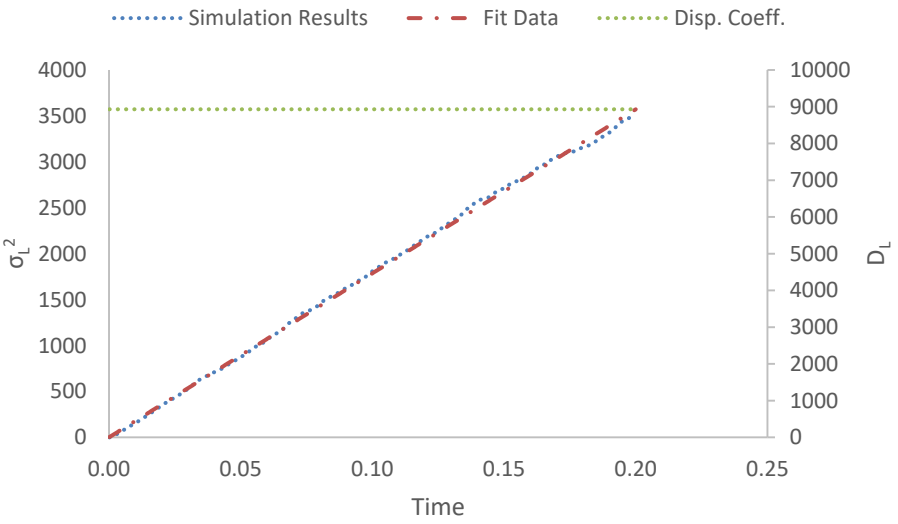


Figure B.6: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 90.1$. Fit parameters are, $A = 17859.5$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

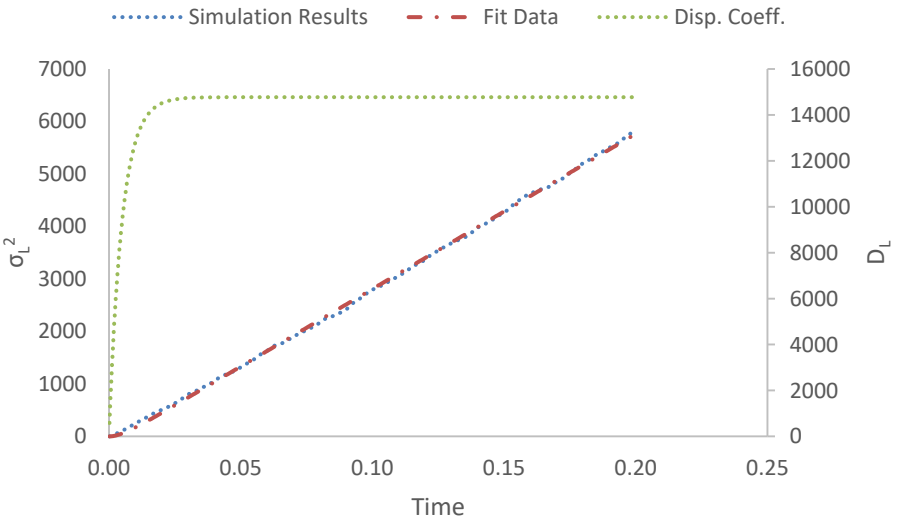


Figure B.7: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 112.6$. Fit parameters are, $A = 29542.8$ and $k = 201.2$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

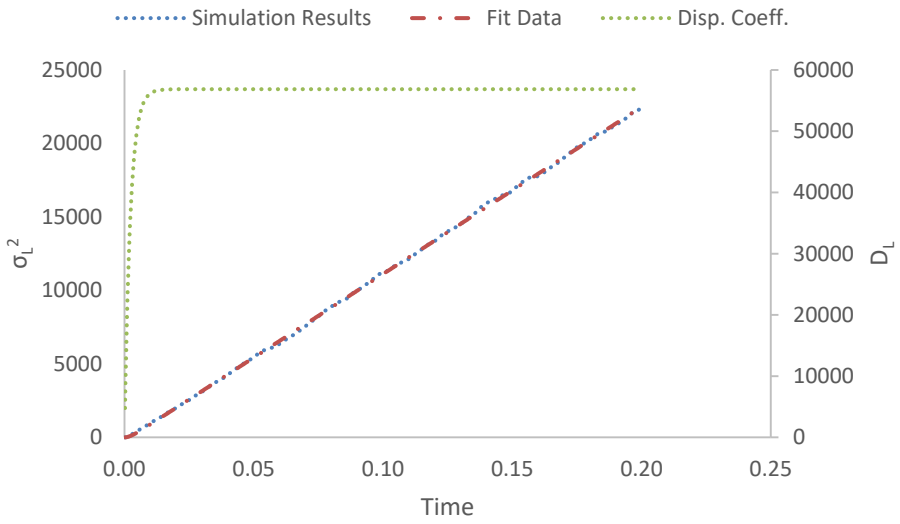


Figure B.8: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 225.2$. Fit parameters are, $A = 113724.9$ and $k = 439.2$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

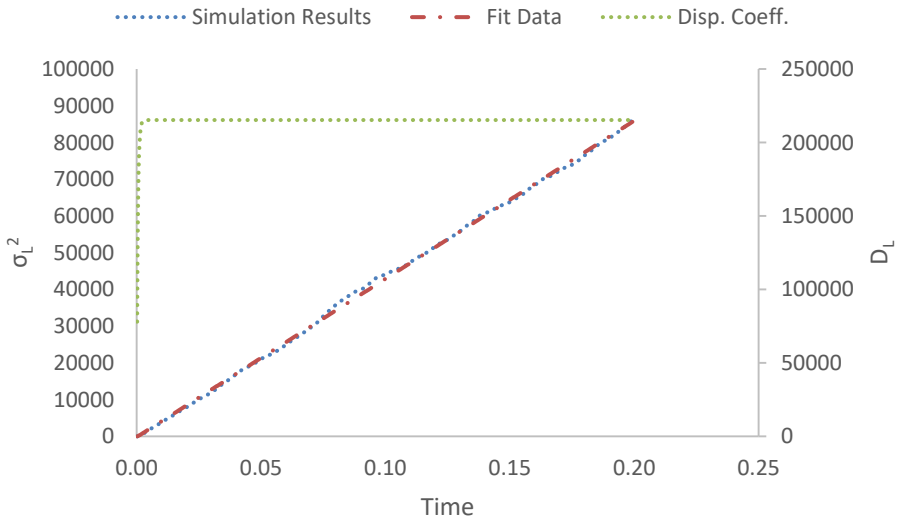


Figure B.9: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 450.4$. Fit parameters are, $A = 430558.9$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

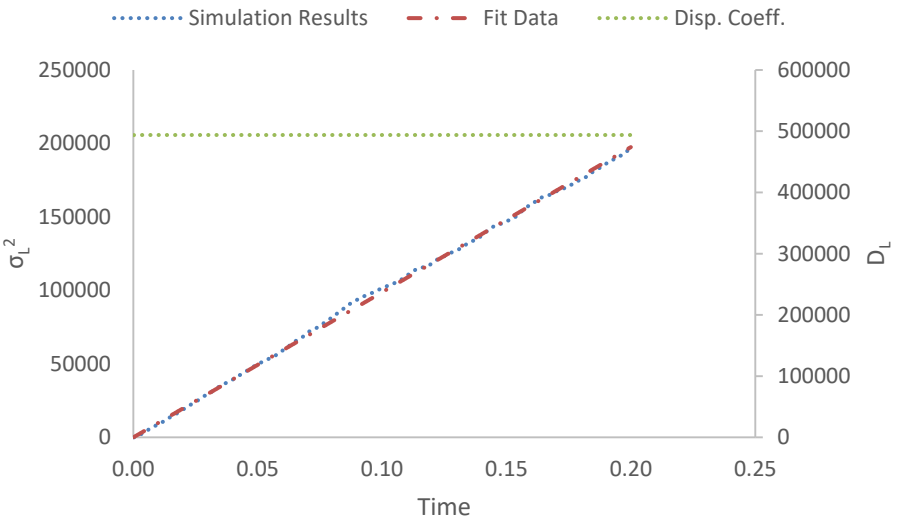


Figure B.10: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 675.5$. Fit parameters are, $A = 987587.8$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

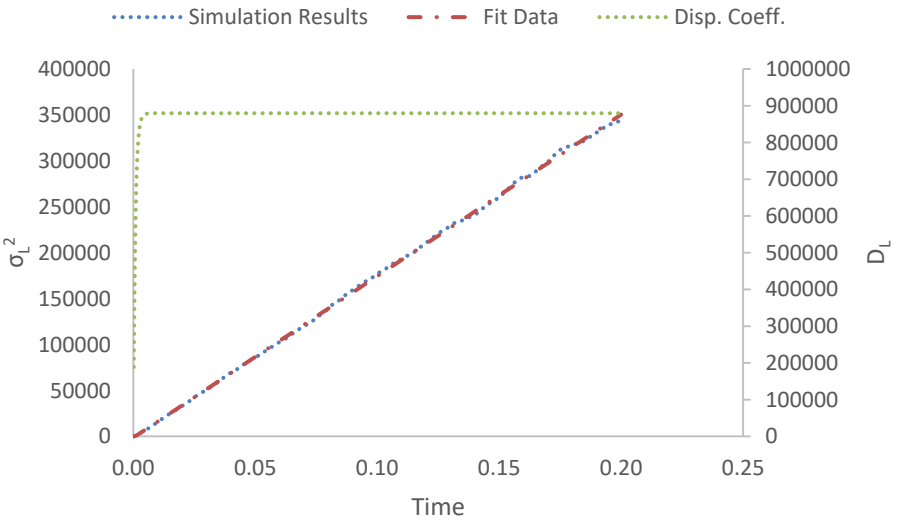


Figure B.11: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 900.8$. Fit parameters are, $A = 1759189$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

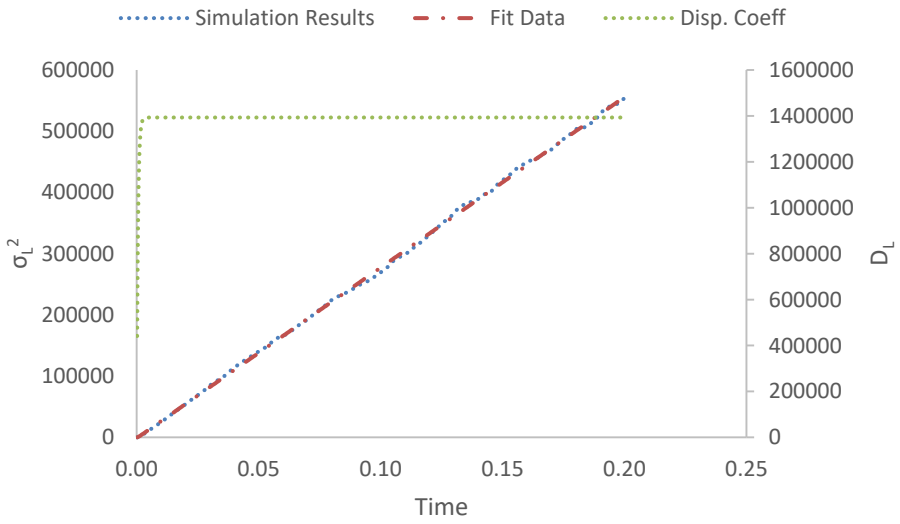


Figure B.12: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 900.8$. Fit parameters are, $A = 2786138$ and $k \gg 1000$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

B.2. In the Random Packing of Monodisperse Hardspheres

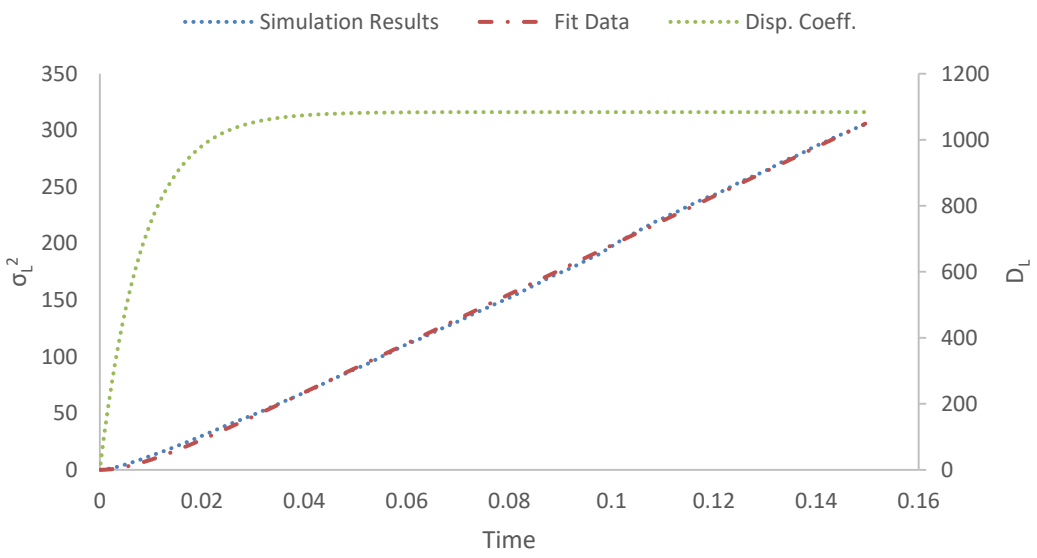


Figure B.13: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 15$. Fit parameters are, $A = 2167.8$ and $k = 118.1$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

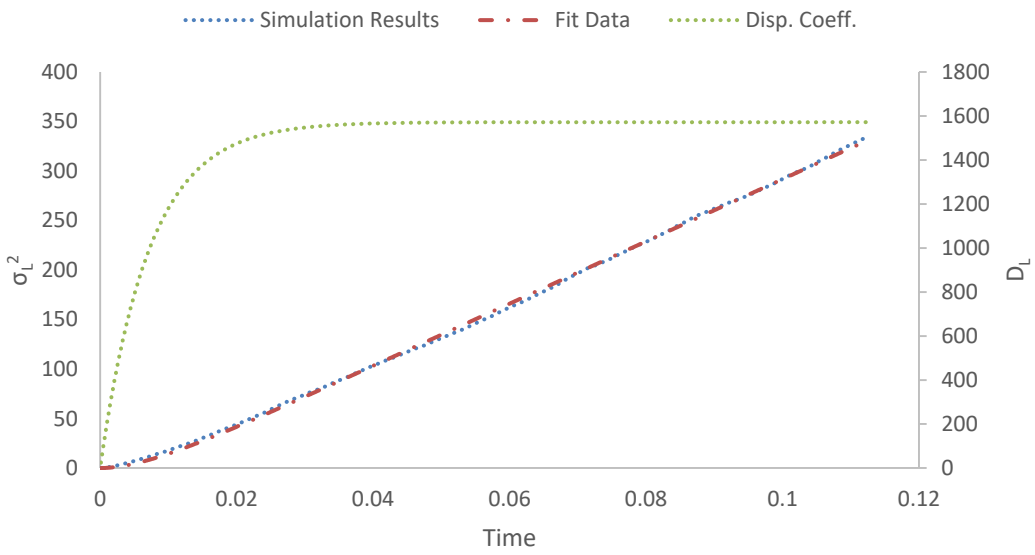


Figure B.14: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 20$. Fit parameters are, $A = 3143.8$ and $k = 139.0$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

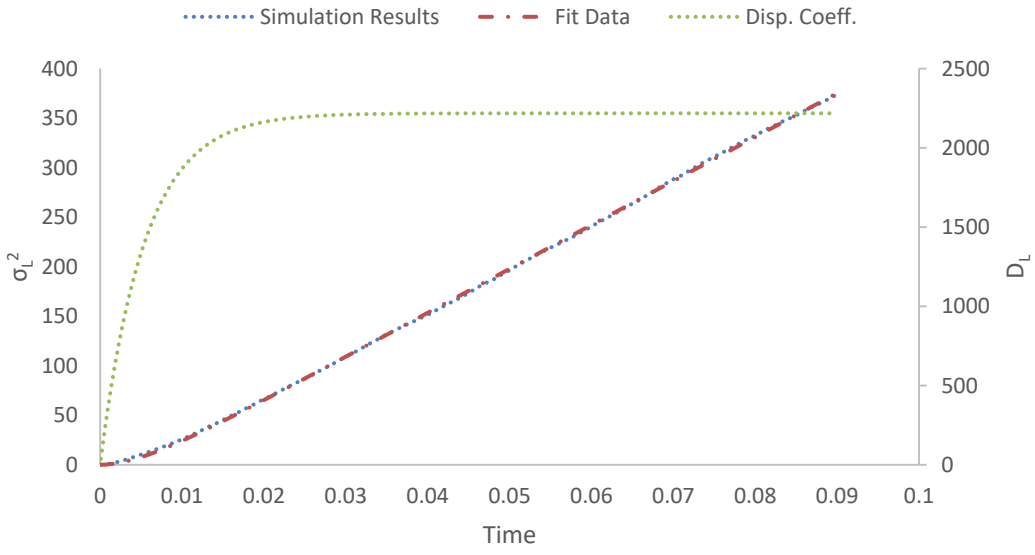


Figure B.15: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 25$. Fit parameters are, $A = 4436.9$ and $k = 184.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

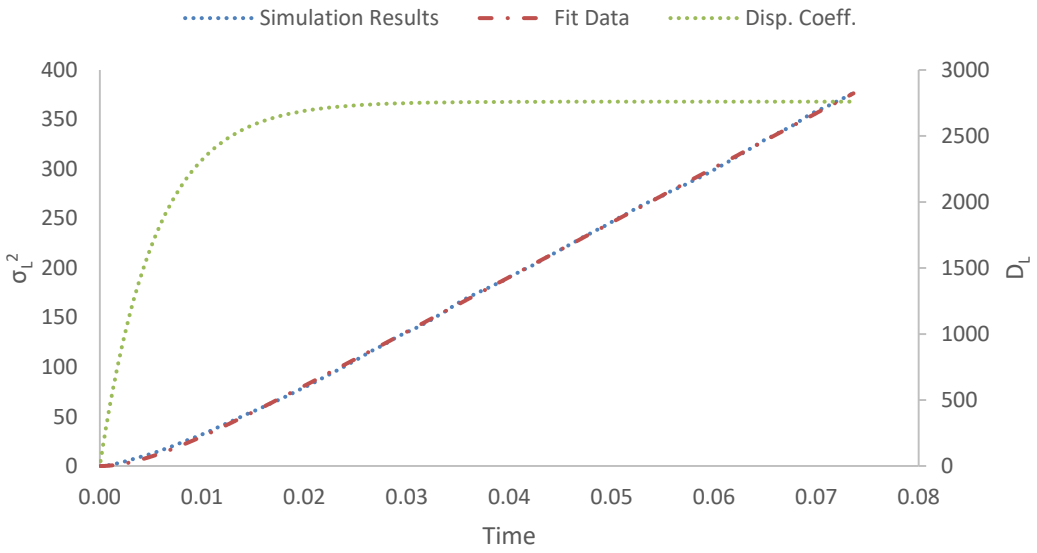


Figure B.16: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 30$. Fit parameters are, $A = 5520.0$ and $k = 183.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

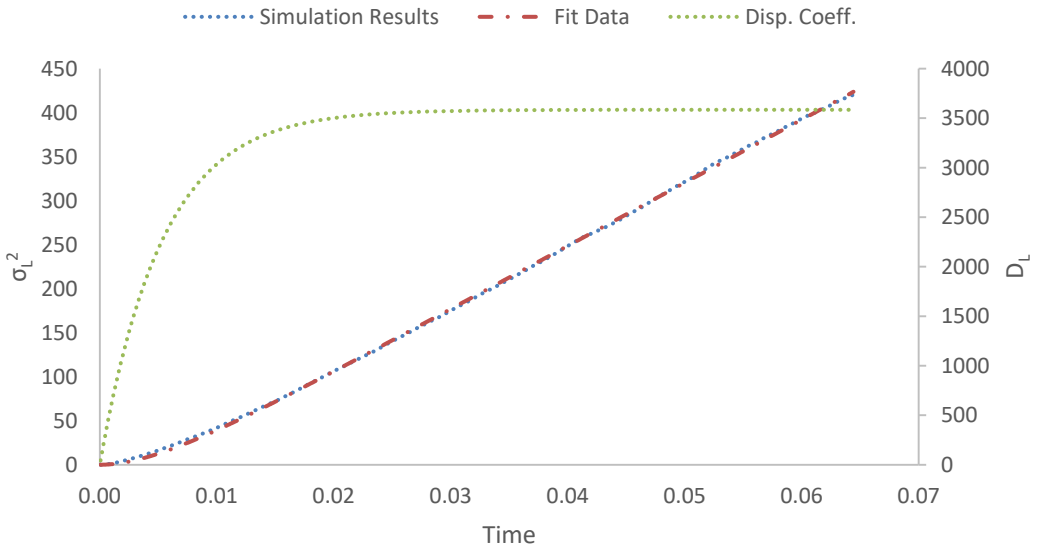


Figure B.17: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 35$. Fit parameters are, $A = 7170.6$ and $k = 187.5$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

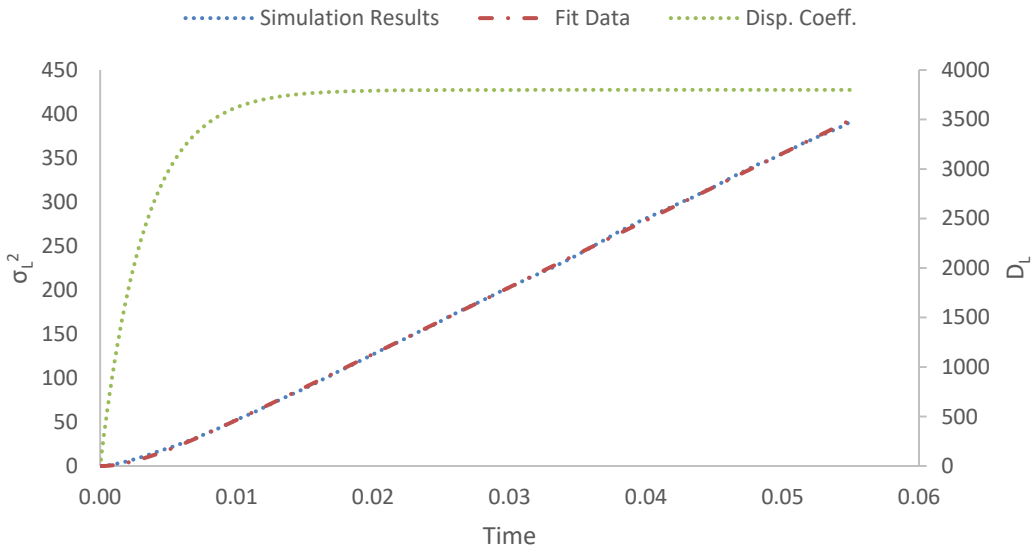


Figure B.18: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 40$. Fit parameters are, $A = 7597.7$ and $k = 306.9$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

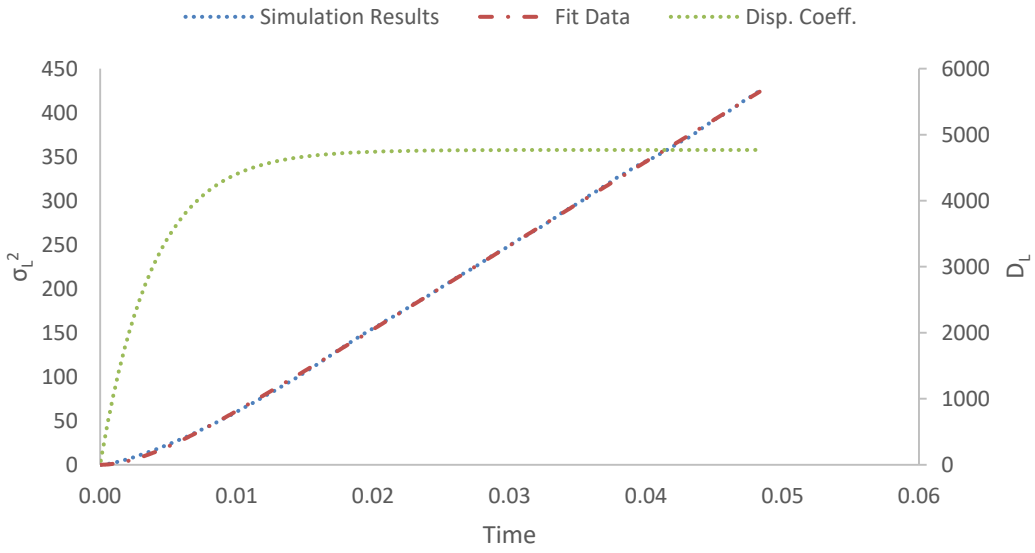


Figure B.19: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 45$. Fit parameters are, $A = 9539.2$ and $k = 256.82$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

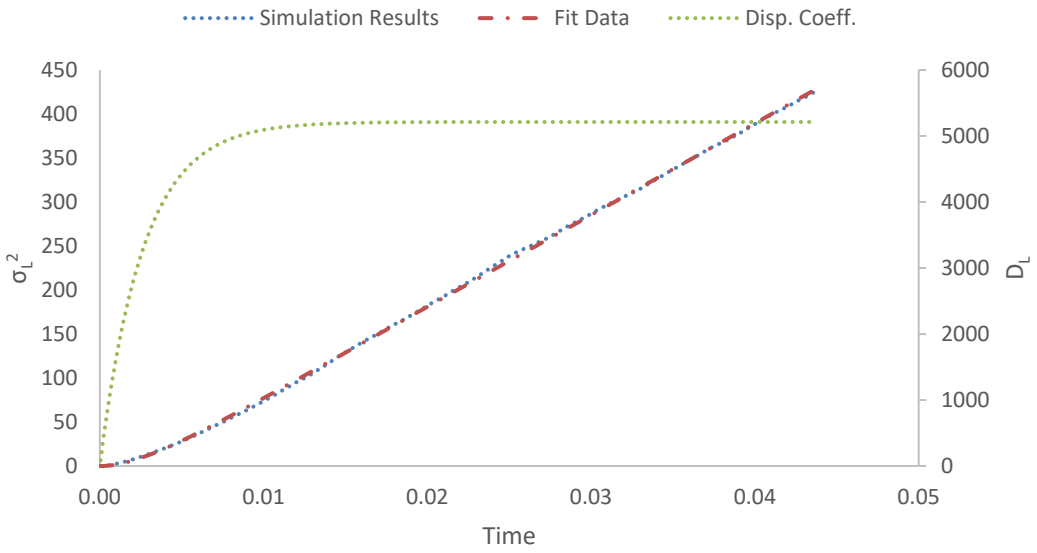


Figure B.20: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 50$. Fit parameters are, $A = 10425.2$ and $k = 377.1$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

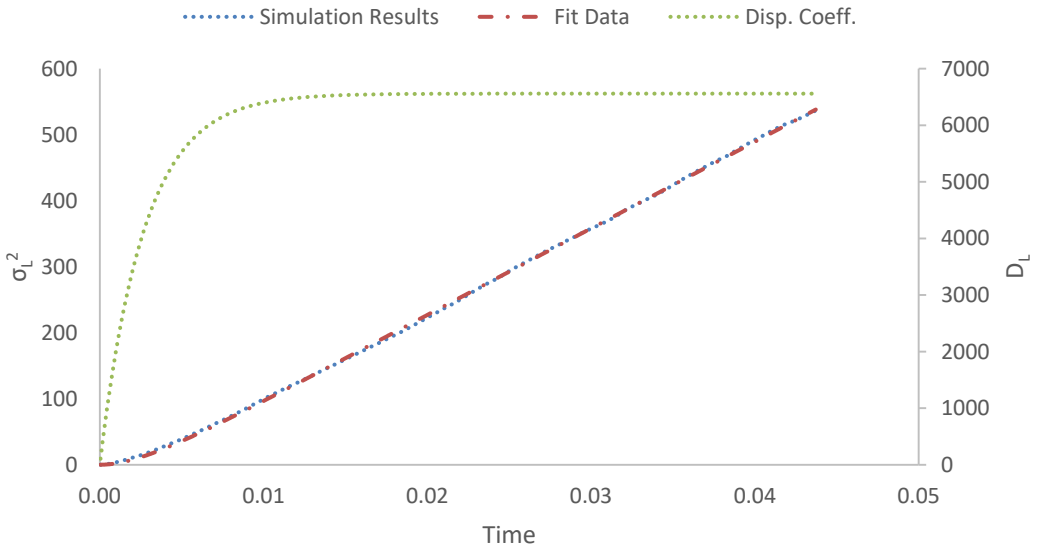


Figure B.21: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 60$. Fit parameters are, $A = 13117.7$ and $k = 370.0$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

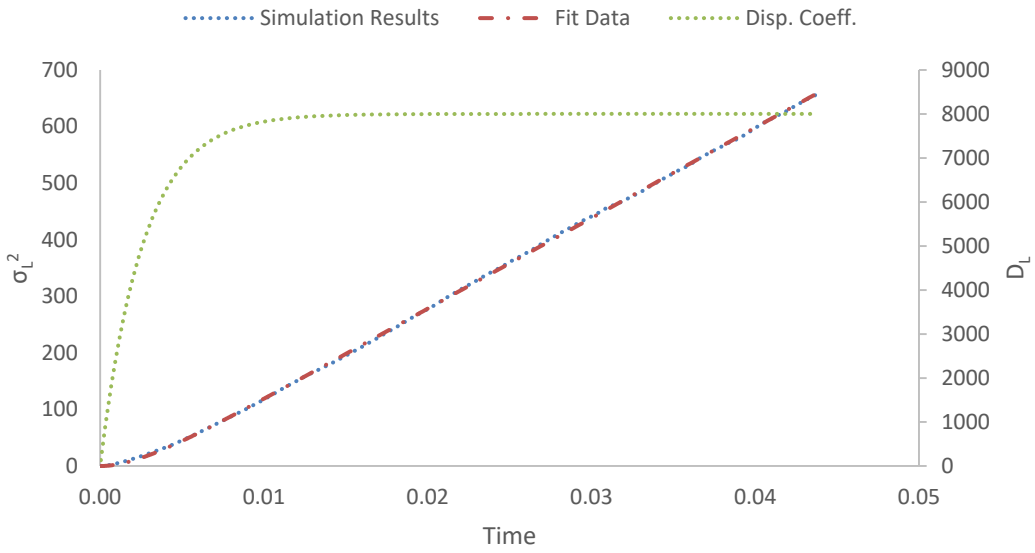


Figure B.22: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 70$. Fit parameters are, $A = 16003.4$ and $k = 381.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

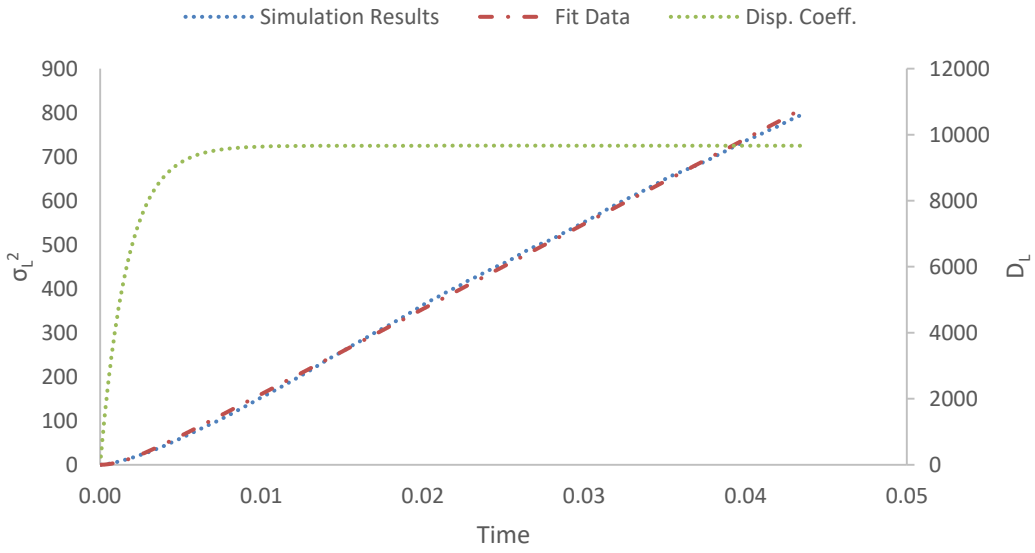


Figure B.23: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 80$. Fit parameters are, $A = 19331.5$ and $k = 589.6$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

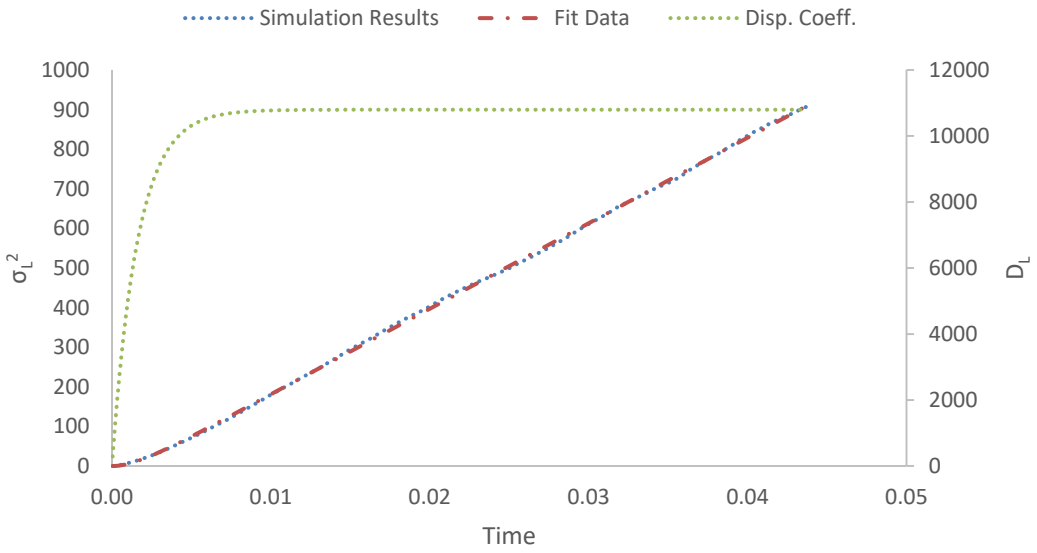


Figure B.24: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 90$. Fit parameters are, $A = 21591.7$ and $k = 621.0$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

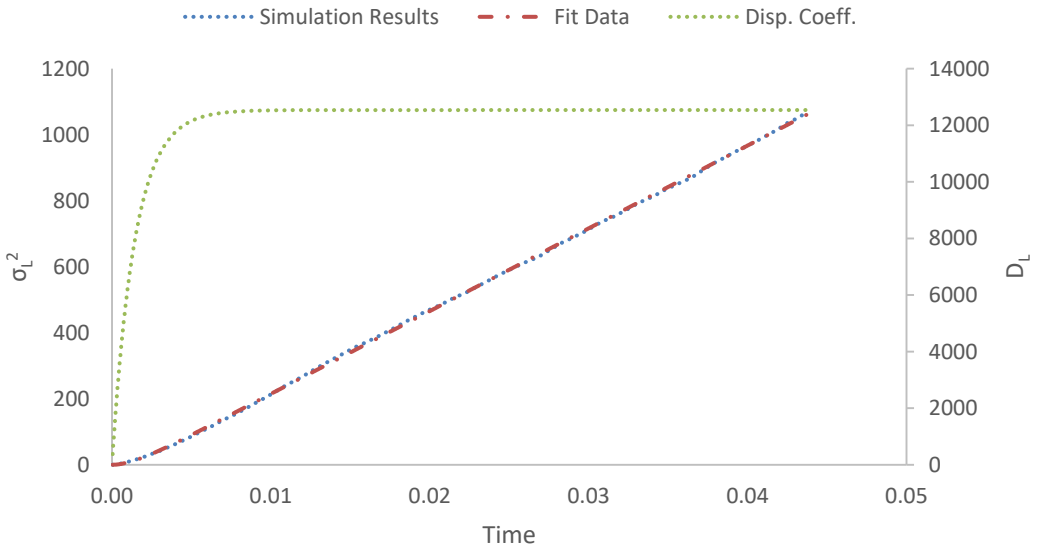


Figure B.25: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 100$. Fit parameters are, $A = 25076.5$ and $k = 695.1$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

B.3. In the Random Packing of Core-Shell Particles

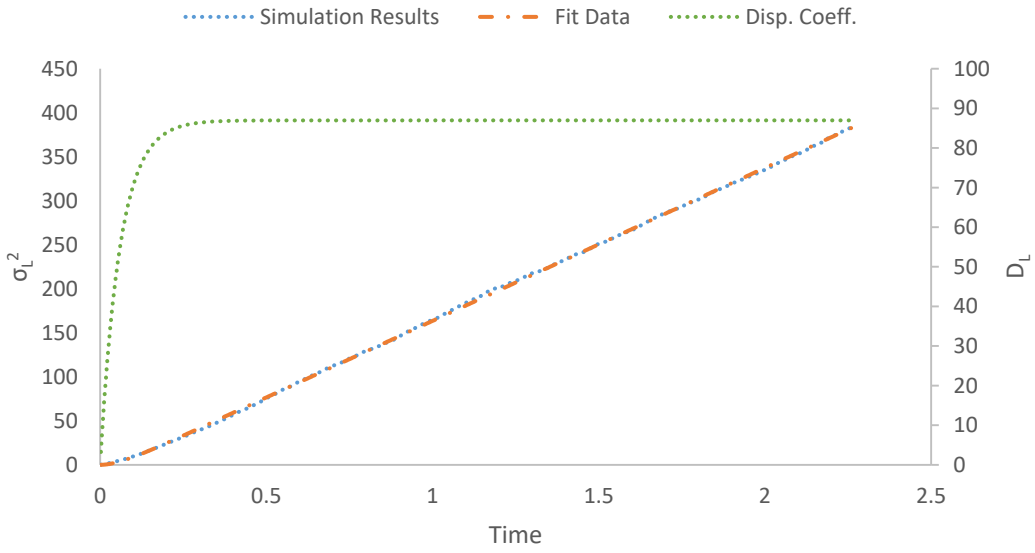


Figure B.26: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 1$. Fit parameters are, $A = 173.9$ and $k = 16.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

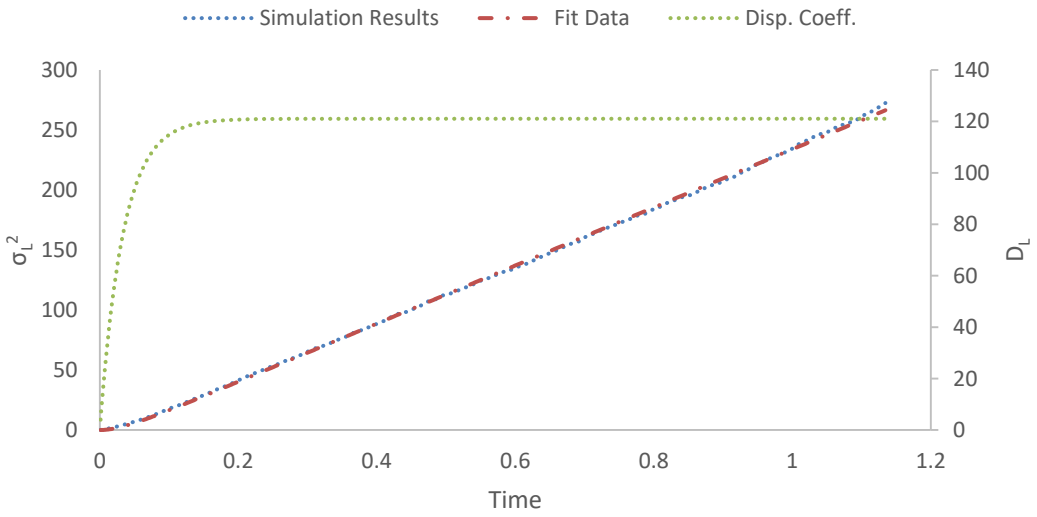


Figure B.27: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 2$. Fit parameters are, $A = 242.1$ and $k = 29.6$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

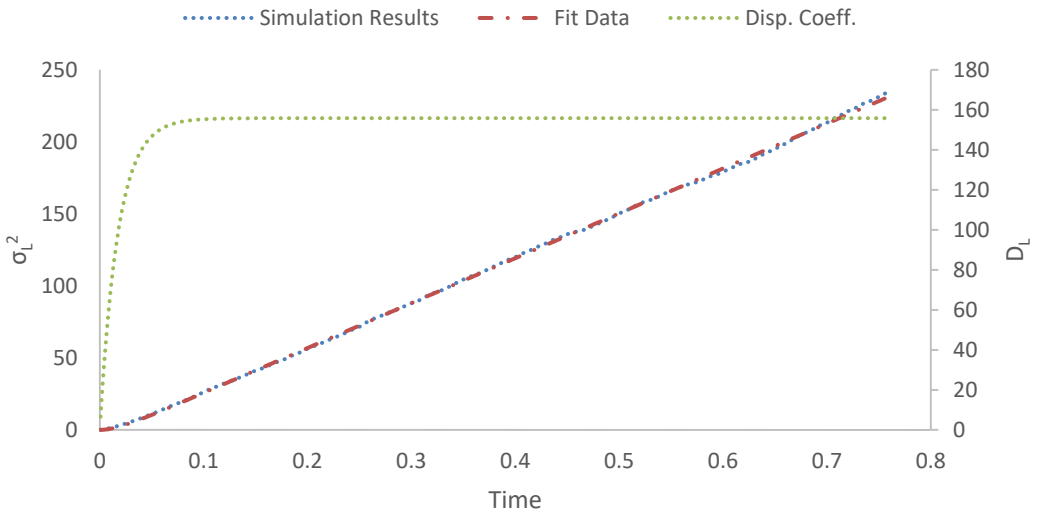


Figure B.28: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 3$. Fit parameters are, $A = 311.7$ and $k = 56.9$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

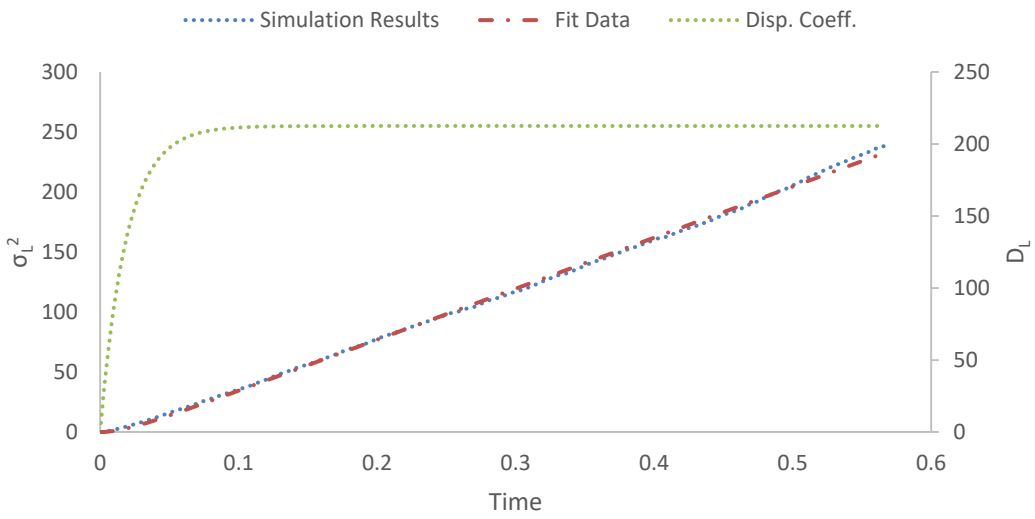


Figure B.29: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 4$. Fit parameters are, $A = 425.0$ and $k = 52.8$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

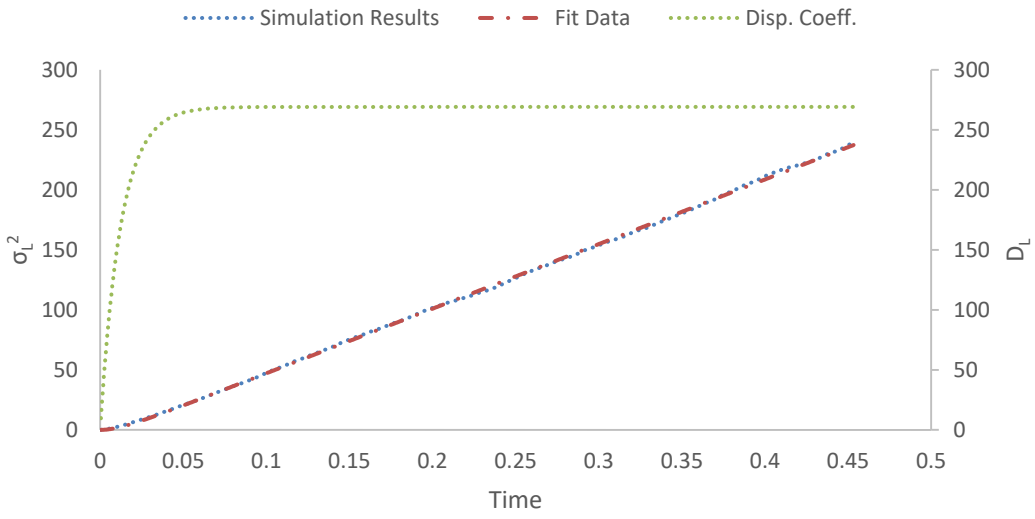


Figure B.30: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 5$. Fit parameters are, $A = 538.2$ and $k = 80.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

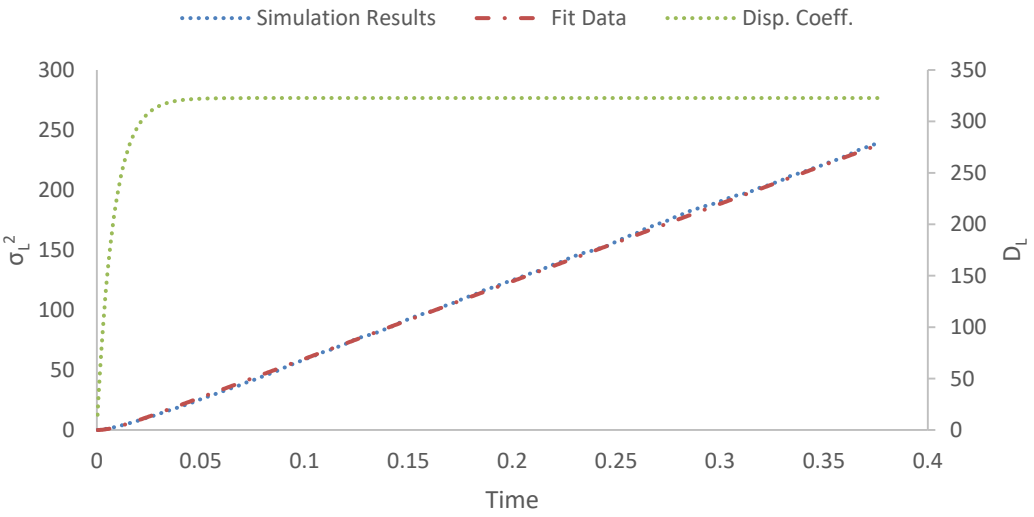


Figure B.31: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 6$. Fit parameters are, $A = 645.5$ and $k = 124.4$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

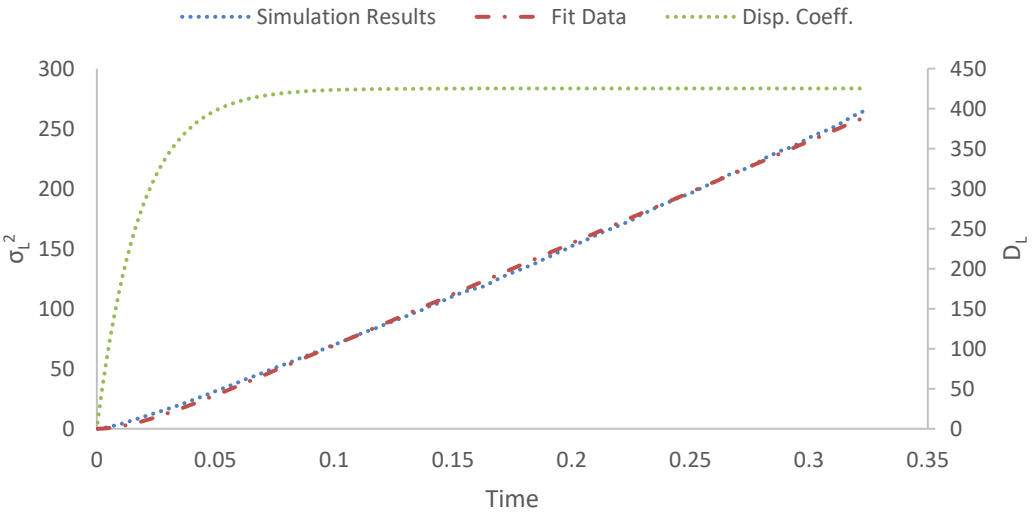


Figure B.32: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 7$. Fit parameters are, $A = 850.4$ and $k = 54.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

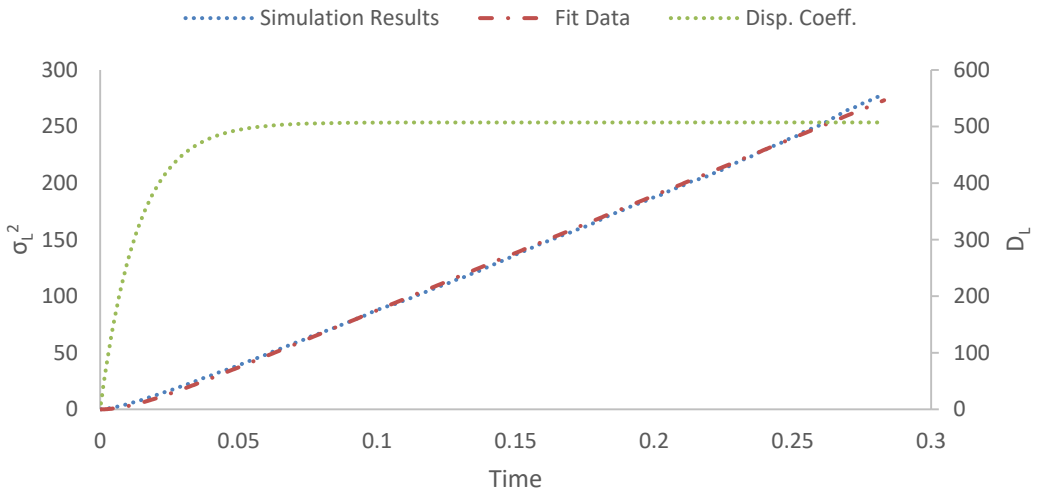


Figure B.33: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 8$. Fit parameters are, $A = 1014.5$ and $k = 73.1$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

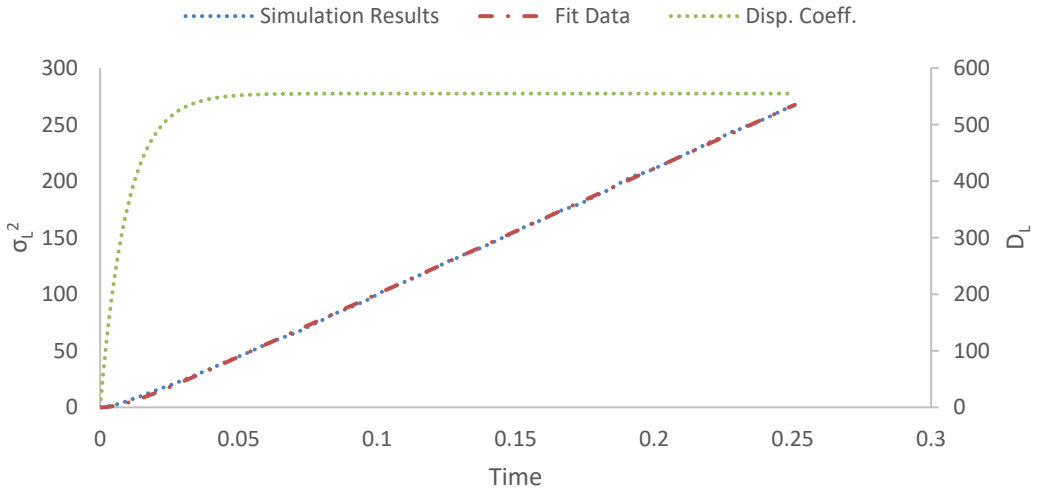


Figure B.34: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 9$. Fit parameters are, $A = 1109.5$ and $k = 102.8$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

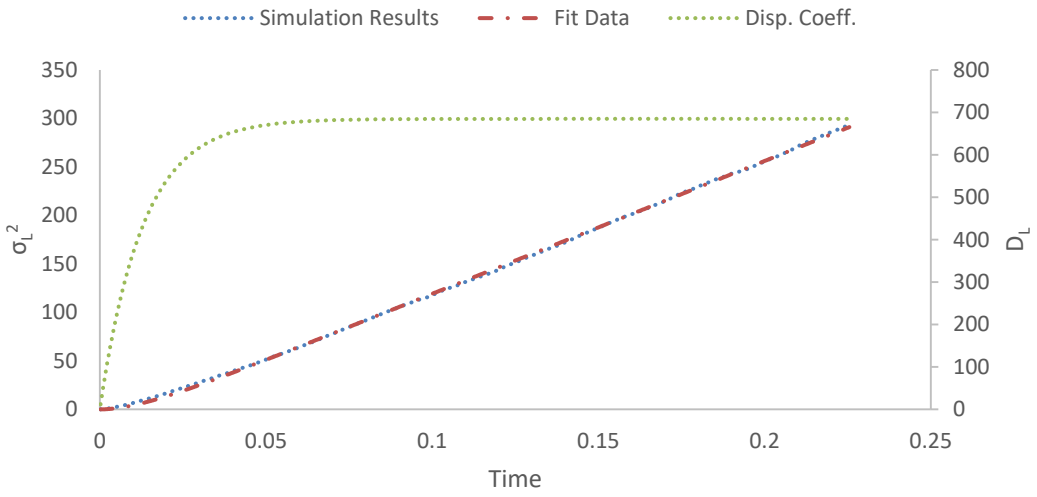


Figure B.35: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 10$. Fit parameters are, $A = 1369.5$ and $k = 77.2$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

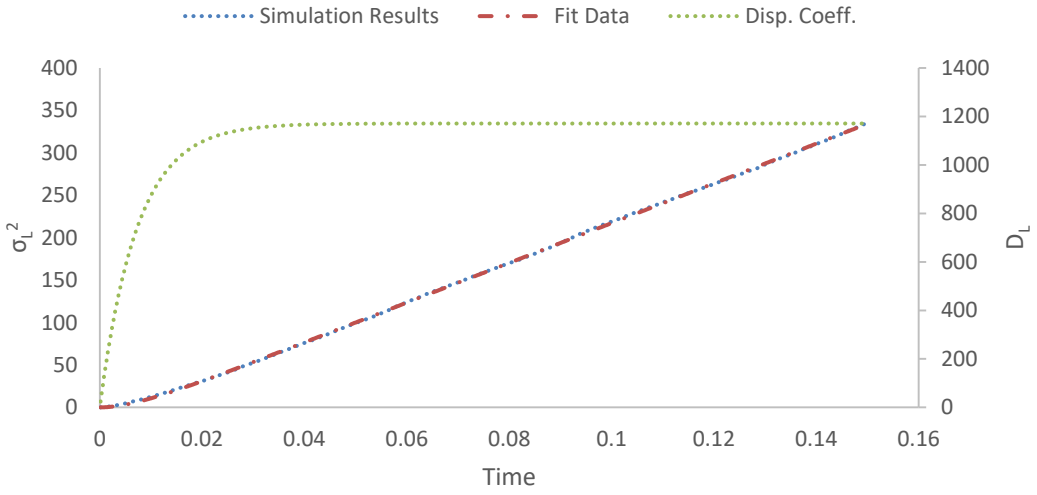


Figure B.36: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 15$. Fit parameters are, $A = 2342.0$ and $k = 137.2$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

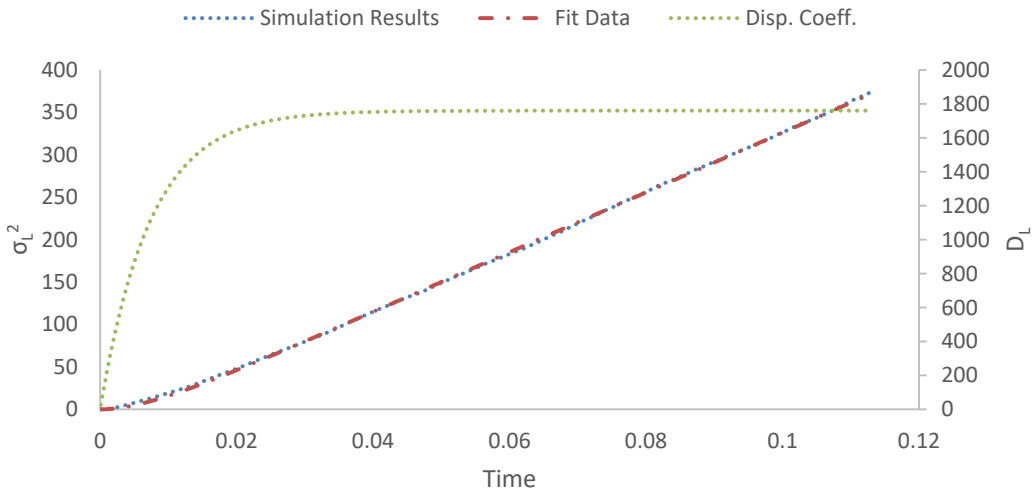


Figure B.37: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 20$. Fit parameters are, $A = 3520.5$ and $k = 135.8$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

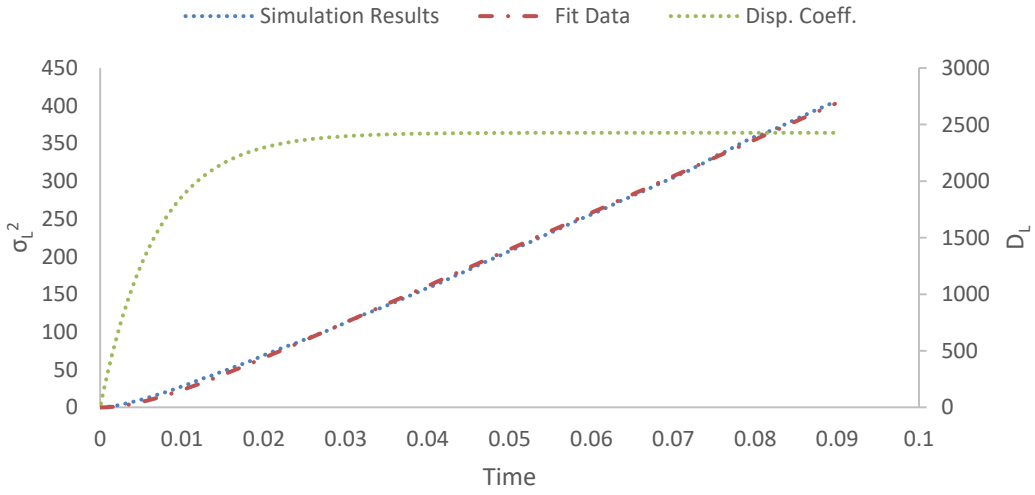


Figure B.38: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 25$. Fit parameters are, $A = 4855.0$ and $k = 146.0$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

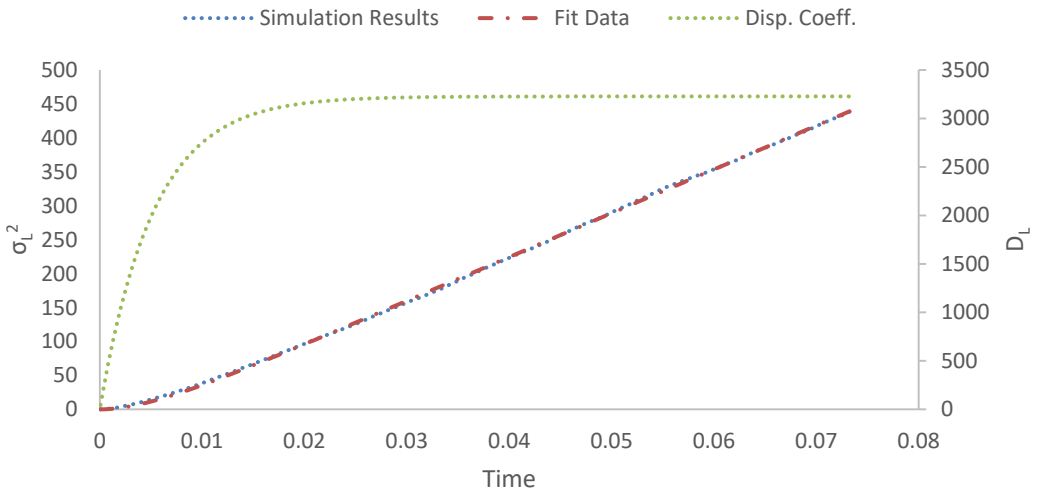


Figure B.39: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 30$. Fit parameters are, $A = 6454.7$ and $k = 191.0$ Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

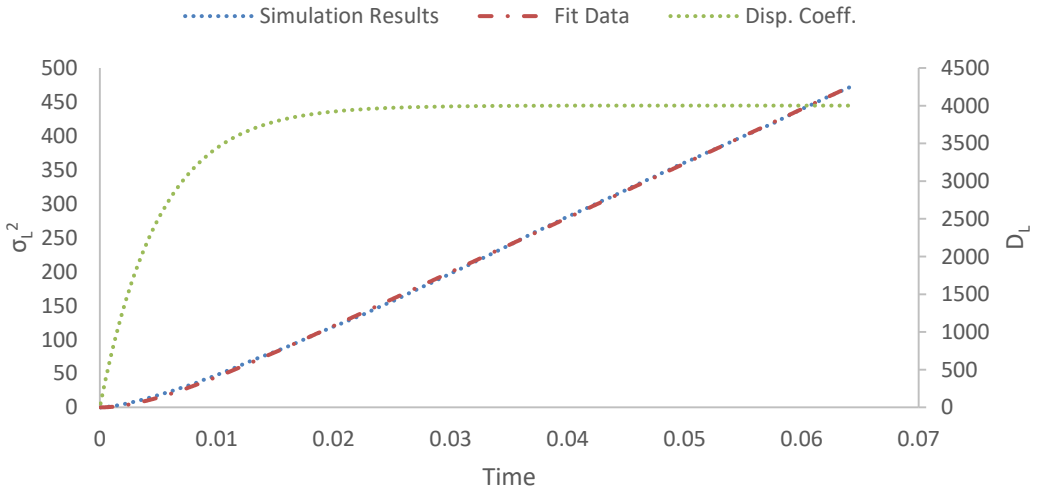


Figure B.40: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 35$. Fit parameters are, $A = 8003.9$ and $k = 195.9$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

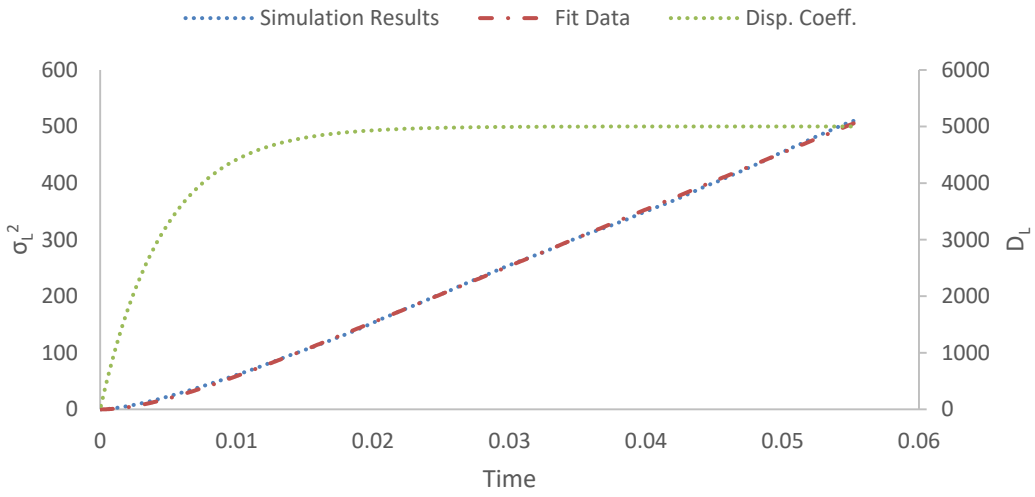


Figure B.41: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 40$. Fit parameters are, $A = 10003.1$ and $k = 214.6$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

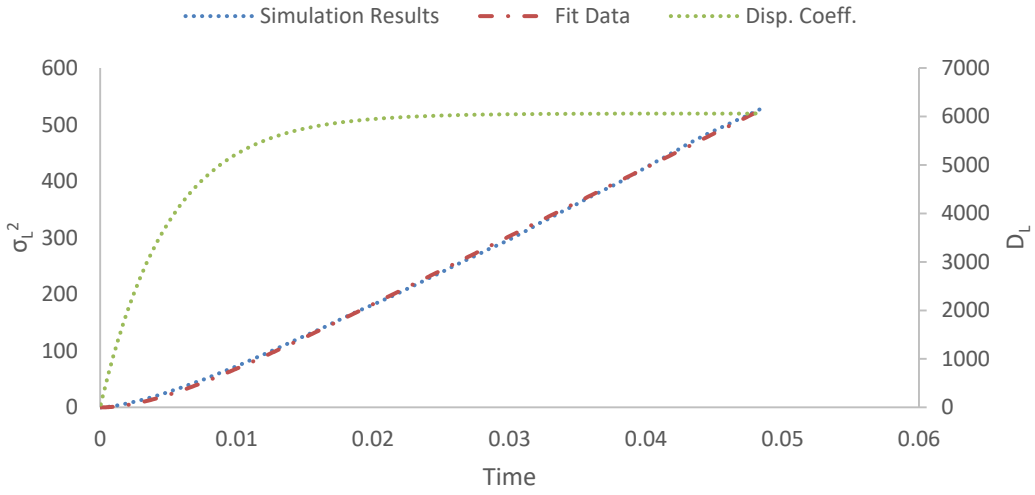


Figure B.42: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 45$. Fit parameters are, $A = 12122.3$ and $k = 198.3$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

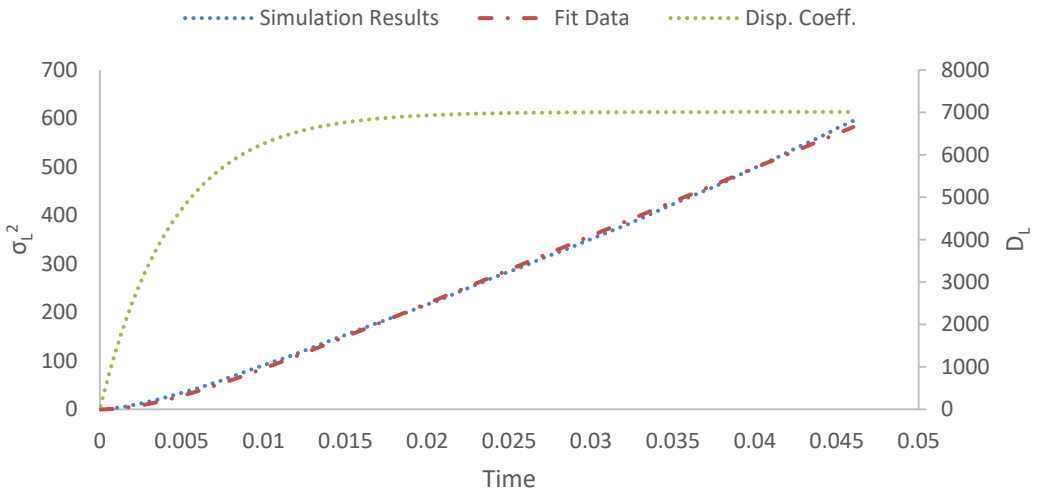


Figure B.43: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 50$. Fit parameters are, $A = 14021.3$ and $k = 223.6$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

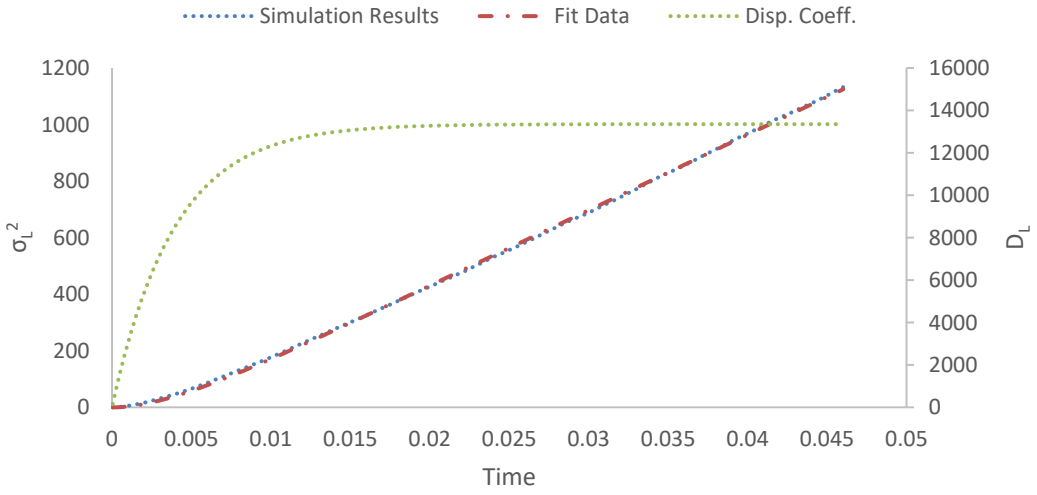


Figure B.44: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 75$. Fit parameters are, $A = 26705.8$ and $k = 256.1$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

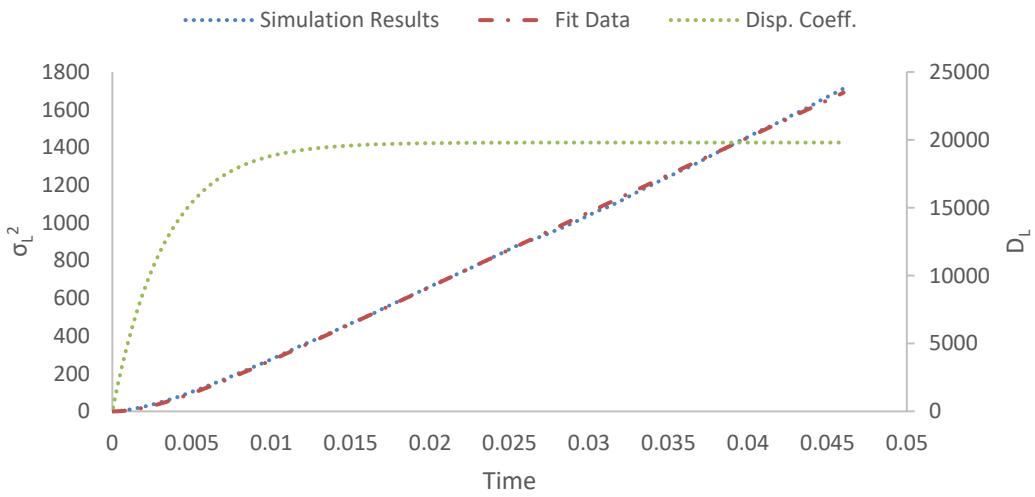


Figure B.45: Longitudinal position variance (μm^2) vs. time (s) at $Pe = 100$. Fit parameters are, $A = 39604.1$ and $k = 299.7$. Longitudinal dispersion coefficient ($\mu m^2/s$) vs. time (s) plot is also on the secondary axis to the right.

APPENDIX C

FORTRAN IMPLEMENTATIONS

Fortran programming language is often preferred for scientific computations. Its roots extend across a very long time, it has open source well written compilers as well as having open source parallel computing modules and online resources available. Therefore it is very suitable for the application of the model being created on computer. In the next three sections, details of creating a free form Fortran program to carry out calculations required for the model will be explained.

C.1. Free Molecular Diffusion

Random-walk simulations of free molecular diffusion was explained in section 3.1.1. and it was shown that Equation (15) is the main equation that calculates random motion of tracers and an algorithm was presented. Variables and parameters that need to be declared at the beginning of the code as well as the types and strings used for corresponding variables or parameters are given in Table C.1. There are other Fortran strings, which do not appear in the equation given in section 3.1, that needed to be declared throughout the code as auxiliary variables to carry out calculations and a list of them can be found at the end of this section.

Table C.1: Declared main parameters and variables and corresponding strings and their declaration types used for free diffusion model.

Parameter/Variable	String	Type
D_{AB}	DAB	Real
t_s	DURATION	Real
n	NP	Integer
Δt	DT	Real
Δl	DX	Real
n_s	NS	Integer
$\boldsymbol{\xi}$ (components)	NUMBERSX, NUMBERSY, NUMBERSZ	Real, Rank=1, Allocatable
$\mathbf{X}(t)$	POSITIONOLD	Real, Rank=2, Allocatable
$\mathbf{X}(t + \Delta t)$	POSITIONNEW	Real, Rank=2, Allocatable
ΔX_i	DISPLACEMENT	Real, Rank=2, Allocatable
D_{AB} (re-calculated)	DABVSTIME	Real, Rank=1, Allocatable

All the strings in program appear in declarations and fixed parameters are calculated by the following code. Note, decimal is lost during the calculation of n_s due to type conversion. Therefore duration of the simulation may effectively decrease if the result is not an integer. Still the amount of decrement is very negligible since Δt is typically chosen as a very small fraction of a second and t_s is very large compared to it. Duration can be re-calculated to get rid of this negligible error.

```
DAB=D_AB
DURATION=t_s
NP=n
DT=Δt
DX=SQRT(2*DAB*DT)
NS=DURATION/DT
DURATION= DT*NS
ALLOCATE(POSITIONNEW(NP,NS))
ALLOCATE(POSITIONOLD(NP,NS))
```

The vector $\boldsymbol{\xi}$ in Equation (15) was declared as individual components as rank 1 arrays with allocatable size. The reason behind it is the fact that amount of random numbers required, which is equal to n_s, is variable depending on t_s and Δt. Declaring a new string, `MAXRNG` (Integer), which determines how many random numbers will be generated and written in $\boldsymbol{\xi}$ component arrays, as well as the size alllocated to them. `MAXRNG` was set equal to n_s in the simulations done in this work so that all random numbers required for a single tracer is pre-generated. It can also be set to a lesser number and new random numbers can be generated upon running out of random numbers, if memory is limited.

```
MAXRNG=NS
ALLOCATE(NUMBERSX(MAXRNG))
ALLOCATE(NUMBERSY(MAXRNG))
ALLOCATE(NUMBERSZ(MAXRNG))
```

Random number generation (RNG) subroutines are available in GFortran compiler libraries. `RANDOM_NUMBER` is one of them but it only generates pseudo-random numbers in the interval $0 \leq RNG < 1$. Therefore a subroutine which converts random numbers into either +1 or -1 was written. This subroutine makes use of Fortran's `FLOOR` function and cosinus function to carry out the conversion. Declaring another parameter string `PI` of type real, as in the number π, and the conversion subroutine is given as:

```
PI=ACOS(-1.0)
```

```
SUBROUTINE NUMBERS(NX,NY,NZ)
IMPLICIT NONE
REAL, INTENT(OUT):: NX(MAXRNG),NY(MAXRNG),NZ(MAXRNG)
INTEGER:: A
DO A=1,MAXRNG
  CALL RANDOM_NUMBER (NX(A))
  NX(A)=COS((FLOOR(2*NX(A)))*PI)
  CALL RANDOM_NUMBER (NY(A))
  NY(A)=COS((FLOOR(2*NY(A)))*PI)
  CALL RANDOM_NUMBER (NZ(A))
  NZ(A)=COS((FLOOR(2*NZ(A)))*PI)
ENDDO
END SUBROUTINE NUMBERS
```

To explain more clearly, subroutine first assigns a value in the interval $0 \leq RNG < 1$ to a component array element, then uses 2 times the assigned value in `FLOOR` function to convert the result into 0 or 1, then multiplies the converted value with π and gives input to `COS` function. `COS` gets 0 or π as equally likely inputs to yield +1 or -1 respectively.

A useful subroutine was found available as open source code in several different online Fortran resources as an example. GFortran compiler's `RANDOM_SEED` routine, if not given input, repeatedly selects the same initial seed every time the program is executed hence causing `RANDOM_NUMBER` to generate same sequence of pseudo-random numbers. The open source subroutine that uses system clock and modifies it through a function to set a random initial seed was used in the code. As a result, program generates different sequences of random numbers for every execution. The subroutine can be seen below.

```
SUBROUTINE INIT_RANDOM_SEED()
INTEGER :: I, N, CLOCK
INTEGER, DIMENSION(:), ALLOCATABLE :: SEED
CALL RANDOM_SEED(SIZE = N)
ALLOCATE(SEED(N))
CALL SYSTEM_CLOCK(COUNT=CLOCK)
SEED = CLOCK + 37 * (/ (I - 1, I = 1, N) /)
CALL RANDOM_SEED(PUT = SEED)
DEALLOCATE(SEED)
END SUBROUTINE
```

Actual random-walk part of the code is based on Equation (15). Initial positions of all tracers are set to origin, based on a point injection initial condition. Then tracers are randomly moved one by one. Therefore calculations of next tracer waits until the previous tracer finishes taking n_s amount of random steps. RNG is done in bulk for each tracer. Declaring a new real string `FREQ` (frequency of data collection in seconds, i.e. once every `FREQ` seconds) two new integer strings `AODP` (for amount of data points) and `B` (equivalent amount of random-steps required for `FREQ` seconds to pass) for the allocation of `DISPLACEMENT` to collect data at equally distant time intervals during random-walk. This study sets `FREQ=DURATION/1000` unless stated otherwise.

```
FREQ=DURATION/1000
B=INT(FREQ/DT)
AODP=INT(NS/B)

ALLOCATE(DISPLACEMENT(NP,AODP))

!!SETTING INITIAL POSITIONS!!
DO I=1,NP
    POSITIONOLD(I,1)=0
    POSITIONOLD(I,2)=0
    POSITIONOLD(I,3)=0
ENDDO

!!RANDOM-WALK!!
CALL INIT_RANDOM_SEED()

DO I=1,NP
  CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)
  DO J=1,NS
    POSITIONNEW(I,1)=POSITIONOLD(I,1)+(NUMBERSX(J)*DX)
    POSITIONNEW(I,2)=POSITIONOLD(I,2)+(NUMBERSY(J)*DX)
    POSITIONNEW(I,3)=POSITIONOLD(I,3)+(NUMBERSZ(J)*DX)
    POSITIONOLD (I,1)=POSITIONNEW (I,1)
    POSITIONOLD (I,2)=POSITIONNEW (I,2)
    POSITIONOLD (I,3)=POSITIONNEW (I,3)
     IF (MOD(J,B).EQ.0) THEN
      DISPLACEMENT(I,J/B)=SQRT(POSITIONNEW(I,1)&
      **2+ POSITIONNEW(I,2)**2+ POSITIONNEW(I,3)**2)
     ENDIF
  ENDDO
ENDDO
```

Displacement data collected during random-walk phase is compiled into D_{AB} data by using Equation (17) (t_s swapped with elapsed time when the data was taken), intrinsically creating a set of data that represents the time evolution of diffusion coefficient. A new real string `SUMM` that can be used as a dummy to hold any summation, was declared. Results are written in a file right after calculation. Optionally, formatting and character string allocation to output file name to prevent any over-writes after re-execution of the program, can be done. In the case of diffusion in free environments, time dependent D_{AB} should not deviate from input value except for random fluctuations that can occur due to probabilistic nature of random-walk method.

```
ALLOCATE(DABVSTIME(AODB))

OPEN(#, FILE='DAB VS TIME DATA.TXT',STATUS='NEW',ACTION='WRITE')
```

```
DO I=1,AODP
  SUMM=0
  DO J=1,NP
    SUMM=SUMM+DISPLACEMENT(J,I)**2
  ENDDO
  DABVSTIME(I)=(SUMM/NP)/(6*I*B*DT)
  WRITE(#,*) DABVSTIME(I)
ENDDO
```

A list of Fortran strings that appear in the program, but not in the equations in section 3.1 is given in Table C.2.

Table C.2: Additional strings declared for the free diffusion code as necessary parameters for calculations.

String	Definition	Type
MAXRNG	Size allocated to NUMBERS arrays. Set equal to NS for this study.	Integer
PI	Ratio of circumference to diameter.	Real
FREQ	Size of the time interval for data collection. Collect data once every FREQ seconds.	Real
AODP	Amount of data points collected for the duration	Integer
B	Amount of random-steps equivalent for FREQ seconds to pass	Integer
DISPLACEMENT	Holds spatial displacements of all tracers at different times	Real, Rank=2, Allocatable

C.2. Computation and Storage of Impermeable Boundaries

In the sections 3.2.2 and 3.2.3, it was explained how an ideal mathematical representation of a multiple layer core-shell particle was created by using basic principles of analytical geometry. Reconstructed core-shell geometry was entirely made of spherical elements. Components of center point vectors for each element, total amount of them present in the geometry, as well as information regarding the exact locations of these elements in the core-shell particle such as the layer and auxiliary circle number was shown to be calculated using Equations (26) to (36). Systematic calculation and storage of the calculated parameters is very crucial for implementing an efficient application of collision control, as will be explained in the next section. Main objective is to store all parameters that define the geometry of every element in the periodic random jammed packing of core-shell particles.

Variables and parameters declared in the Fortran code that calculates the geometry is listed in Table C.3. There are other Fortran strings, which do not appear in the equation given in section 3.2, that needed to be declared throughout the code as auxiliary variables to carry out calculations and a list of them can be found at the end of this section.

Calculation of the core-shell geometry follows the same steps in section 3.2.2 generalized for multiple layers. Core-Shell particle radius (r_p), number of shell layers (n_l) and core-to-particle ratio (φ) is given as input to calculate core radius (r_c) and shell side spheres radii (r_s). Arrays that hold the values for $n_{sse,k}$ and $\mathrm{n_{aux,k}}$ are allocated to have a size equal to defined number of shell layers. Then spread angle and $n_{sse,k}$ is calculated. Finally a control structure determines the value of $\mathrm{n_{aux,k}}$ before setting core radius equal to the radius of next sphere of influence. Do loop calculates the vales for all shell layers. After the loop, core radius is set to original value. Note that this code does not calculate anything other than amount of auxiliary circles that can be placed in each individual layer. Reason is the fact that the values of $\mathrm{n_{aux,k}}$ for each layer is needed to be known in order to allocate memory to arrays that hold the other information.

Table C.3: Declared parameters and variables and corresponding strings and their declaration types for the code fragment that calculates a core-shell particle geometry.

Parameter/Variable	String	Type
r_p	PR	Real
n_l	NOL	Integer
Ψ	CPRATIO	Real
r_c	RCORE	Real
r_s	RSHELL	Real
$r_{soi,k}$	RSOI	Real, Rank=1, Allocatable
$\propto_k$	ALPHA	Real
$n_{sse,k}$	NOSS	Real, Rank=1, Allocatable
$n_{ssa,k}$	Uses the same string as $n_{sse,k}$	-
$n_{aux,k}$	NOAC	Real, Rank=1, Allocatable
$\propto_{c,k}$	Uses the same string as $\propto_k$	-
$r_{aux,i,k}$	RAUX	Real, Rank=2, Allocatable
$z_{aux,i,k}$	ZAUX	Real, Rank=2 Allocatable
$\beta_{i,k}$	Uses the same string as $\propto_k$	-
$n_{ssa,i,k}$	NOSOAC, NOSOACINT	Real, Rank=2 Allocatable
$\beta_{c,i,k}$	Uses the same string as $\propto_k$	-
$x_{j,i,k}$, $y_{j,i,k}$	Do not have dedicated strings	-
$\mathbf{P}^{j,i,k}$	PGCC	Real, Rank=4 Allocatable

PR=r_p
NOL=n_l
CPRATIO= Ψ

```
RCORE=PR*CPRATIO

ALLOCATE(NOSS(NOL))
ALLOCATE(NOAC(NOL))

DO I=1,NOL
  ALPHA=ASIN(RSHELL/(RSHELL+RCORE))*2
  NOSS(I)=FLOOR(2*PI/ALPHA)
    IF (MOD(NOSS(I),2).EQ.0) THEN
    NOAC(I)=INT((NOSS(I)-2)/2)
    ELSE
    NOAC(I)=FLOOR(REAL(NOSS(I)/2))
    ENDIF
  RCORE=RCORE+RSHELL*2
ENDDO

RCORE=PR*CPRATIO
```

The amount of auxiliary circles in outer shell layers is different and greater than previous shell layers. However the arrays `ZAUX`, `RAUX`, `NOSOAC` and `NOSOACINT` (defined as integer counterpart of `NOSOAC` for convenience because direct usage of integer type array in a calculation involving real types causes problems) must have enough size to hold information regarding every shell layer. Therefore they were allocated the maximum value of `NOAC` array to their second dimension and they will have non-assigned array elements for inner shell layers. This does not cause any problem since the empty array elements are not going to be accessed. Following code allocates mentioned arrays and calculates the values for them. Inner loops are separately written for easier reading of the code.

```
ALLOCATE(RSOI(NOL))
ALLOCATE(RAUX(NOL,MAXVAL(NOAC)))
ALLOCATE(NOSOAC(NOL,MAXVAL(NOAC)))
ALLOCATE(NOSOACINT(NOL,MAXVAL(NOAC)))
ALLOCATE(ZAUX(NOL,MAXVAL(NOAC)))

DO I=1,NOL
  RSOI(I)=RCORE+2*RSHELL
  ALPHA=ASIN(RSHELL/(RSHELL+RCORE))*2
  NOSS(I)=FLOOR(2*PI/ALPHA)
  ALPHA=2*PI/NOSS(I)

    DO J=1,NOAC(I)
      RAUX(I,J)=SIN(J*ALPHA)*(RCORE+RSHELL)
    ENDDO
```

```
    DO J=1,NOAC(I)
      NOSOAC(I,J)=PI/ASIN(RSHELL/RAUX(I,J))
      NOSOACINT(I,J)=INT(NOSOAC(I,J))
    ENDDO

    DO J=1,NOAC(I)
      ZAUX(I,J)=COS(J*ALPHA)*(RCORE+RSHELL)
    ENDDO

  RCORE=RSOI(I)
ENDDO

RCORE=PR*CPRATIO
```

Now that the amounts of auxiliary circles in each layer and the amounts of shell side spheres on these auxiliary circles are known, corrected spread angle can be calculated for each and every auxiliary circle and the `PGCC` (initials: particle geometry center coordinates) array that holds center point vector components of shell side spheres can be allocated and assigned values for components $x_{j,i,k}$ and $y_{j,i,k}$ by using Equations (35) and (36) respectively to its first two elements in dimension 4. as well as the z-components and a radius to 3rd and 4th elements. For example, the elements (2,7,16,3) and (2,7,16,4) of `PGCC` array hold the z-component of center point vector and the radius for the 16th shell sphere centered on the 7th auxiliary circle in 2nd shell layer respectively.

```
ALLOCATE(PGCC(NOL,MAXVAL(NOAC),MAXVAL(NOSOACINT),4))

DO I=1,NOL
  DO J=1,NOAC(I)
    DO K=1,NOSOACINT(I,J)
      ALPHA=2*PI/NOSOACINT(I,J)
      PGCC(I,J,K,1)=RAUX(I,J)*COS((K-1)*ALPHA)
      PGCC(I,J,K,2)=RAUX(I,J)*SIN((K-1)*ALPHA)
      PGCC(I,J,K,3)=ZAUX(I,J)
      PGCC(I,J,K,4)=RSHELL
    ENDDO
  ENDDO
ENDDO
```

PGCC array at this point only holds information about shell side elements of the geometry except two spheres that need to be placed tangent to the poles of the core sphere or additional spheres of influence. The core sphere must be added into the array and the amount of polar spheres that needs to be added may differ depending on input parameters therefore a control structure is needed to determine the amount. Additionally, 2nd and 3rd dimensions of PGCC array can be combined into a single one that holds information about all elements located in a shell layer. In order to combine these dimensions, number of shell side elements in each shell layer must be calculated by the following code. An integer array NOSISL (initials: number of spheres in shell layer) with rank 1 that holds the amount of spherical elements in each shell layer is declared, allocated size equal to the number of shell layers and all array elements are assigned the initial value 0. All contributions from all auxiliary spheres located in all shell layers are calculated by the nested do loop and finally the amount of polar spheres that needs to be added for each layer is determined by the control structure inside the last do loop. For convenience, core sphere is adopted as a first shell layer element contributes one more count to the amount of elements in the first shell layer.

```
ALLOCATE(NOSISL(NOL))

DO I=1,NOL
  NOSISL(I)=0
ENDDO

DO I=1,NOL
  DO J=1,NOAC(I)
    NOSISL(I)=NOSISL(I)+NOSOACINT(I,J)
  ENDDO
ENDDO

DO I=1,NOL

  IF (MOD(NOSS(I),4).EQ.0) THEN
    NOSISL(I)=NOSISL(I)+2
  ELSE
    NOSISL(I)=NOSISL(I)+1
  ENDIF
ENDDO

NOSISL(1)=NOSISL(1)+1
```

In order not to deallocate PGCC array, another rank 3 array CPGCC (initials: combined PGCC) was declared and dimensions are allocated n_l, maximum value in NOSISL and 4 respectively, similar to PGCC. CPGCC array has the same size for each shell layer but it has non-assigned array elements for any possible inner shell layer since inner layers have less spherical elements than outer layers. Again, this does not give rise to any problems since the non-assigned elements will not be accessed by the program because another rank 1 array called SC (initials: sphere count) is declared and allocated size equal to number of shell layers to hold exact amounts of assigned array elements for each layer. First, the new arrays are allocated and the existing elements in PGCC is transferred into CPGCC by the following code.

```
ALLOCATE(CPGCC(NOL,MAXVAL(NOSISL),4))
ALLOCATE(SC(NOL))

DO I=1,NOL
  SC(I)=0
  DO J=1,NOAC(I)
    DO K=1,NOSOACINT(I,J)

      SC(I)=SC(I)+1
      CPGCC(I,SC(I),1)=PGCC(I,J,K,1)
      CPGCC(I,SC(I),2)=PGCC(I,J,K,2)
      CPGCC(I,SC(I),3)=PGCC(I,J,K,3)
      CPGCC(I,SC(I),4)=PGCC(I,J,K,4)

    ENDDO
  ENDDO
ENDDO
```

Then polar elements are added to the CPGCC by the following code.

```
DO I=1,NOL

  IF (MOD(NOSS(I),4).EQ.0) THEN
   CPGCC(I,SC(I)+1,1)=0
   CPGCC(I,SC(I)+1,2)=0
   CPGCC(I,SC(I)+1,3)=RSOI(I)-RSHELL
   CPGCC(I,SC(I)+1,4)=RSHELL

   CPGCC(I,SC(I)+2,1)=0
   CPGCC(I,SC(I)+2,2)=0
   CPGCC(I,SC(I)+2,3)=-(RSOI(I)-RSHELL)
   CPGCC(I,SC(I)+2,4)=RSHELL
   SC(I)=SC(I)+2
  ELSE
   CPGCC(I,SC(I)+1,1)=0
   CPGCC(I,SC(I)+1,2)=0
   CPGCC(I,SC(I)+1,3)=RSOI(I)-RSHELL
   CPGCC(I,SC(I)+1,4)=RSHELL
   SC(I)=SC(I)+1
  ENDIF

ENDDO
```

Finally the core sphere is added as the last array element of the 2nd dimension of CPGCC by the following code.

```
      CPGCC(1,SC(1)+1,1)=0
      CPGCC(1,SC(1)+1,2)=0
      CPGCC(1,SC(1)+1,3)=0
      CPGCC(1,SC(1)+1,4)=RCORE
      SC(1)=SC(1)+1
```

After the calculations done until this point, CPGCC array holds very detailed information about the core-shell particle geometry, including center coordinates and radius of all elements creating the geometry itself, as well as at which shell layer the element is located in. The geometry stored in CPGCC is ready to undergo a geometric translation in order to create the packing geometry. As the first step, program reads a file that includes the components of center point vectors that belong to 100 hardspheres in the random jammed packing initially created as a packing of 50 spheres by the program written by Skoge et al. (2006) and modified in this study by adding 50 more invading spheres from neighboring periodic cells. However center point vectors of

hardspheres must be resized first, such that the radius of hardspheres would be equal to the radius of the core-shell particle.

A new integer string NOSRJP (initials: number of spheres in random jammed packing) is declared and set equal to 100, which is the amount of hardspheres that has their partial or total volume lying inside the main periodic cell as it was found in section 3.3.2. Then a real string RRJP is declared to hold radius of hardspheres in random jammed packing along with the real string RRATIO which is the ratio of core-shell particle radius to hardsphere radius.

```
NOSRJP=100
RRJP=0.2841570815121517/2
RRATIO=PR/RRJP
```

Then another rank 2 array RJP (initials: random jammed packing) was declared and allocated NOSRJP and 3 to dimensions 1 and 2 respectively, which holds the components of center point vectors of 100 hardspheres read from the file made ready previously. Then all the components in RJP was rescaled by a factor of RRATIO. Note that this rescaling also effectively changes the dimensions of the original unit cube whence the packing was generated by the program of Skoge et al. (2006) from unity to RRATIO. Therefore the value of RRATIO must be used in the calculation of local positions of tracers.

```
OPEN (#, FILE="RJP.DAT", STATUS="OLD", ACTION="READ")

ALLOCATE (RJP(NOSRJP,3))

DO I=1,NOSRJP

  READ(#,'format')RJP(I,1),RJP(I,2),RJP(I,3)

ENDDO

DO I=1,NOSRJP
  DO J=1,3

    RJP(I,J)=(RJP(I,J))*RRATIO

  ENDDO
ENDDO
```

In the final step, a rank 4 array CSPRJP, which is to be hold center coordinates and radius of all core-shell particle elements in the packing, was declared and allocated sizes NOSRJP, n_l, maximum value of SC and 4 to its dimensions respectively. Then the center point vectors held by CPGCC was translated by the following code.

```
ALLOCATE(CSPRJP(NOSRJP,NOL,MAXVAL(SC),4))

DO I=1,NOSRJP
  DO J=1,NOL
    DO K=1,SC(J)
      DO L=1,3

        CSPRJP(I,J,K,L)=RJP(I,L)+CPGCC(J,K,L)

      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Radius values of the elements had to be separately assigned by the following code due to rank and size of RJP.

```
DO I=1,NOSRJP
  DO J=1,NOL
    DO K=1,SC(J)

      CSPRJP(I,J,K,4)=CPGCC(J,K,4)

    ENDDO
  ENDDO
ENDDO
```

Final product, CSPRJP holds every information related to any element of the impermeable boundaries in the system of core-shell particles arranged in a random jammed packing. In the following section, adaptation of free diffusion program to include a collision control mechanism that uses this array is explained.

A list of Fortran strings that appear in the codes given in this section, but not in the equations in section 3.2.3 is given in Table C.4.

Table C.4: Additional strings declared for the code taking care of calculating and storing a random packing of core-shell particles as necessary parameters for calculations.

String	Definition	Type
CPGCC	Another version of PGCC where its 2nd and 3rd dimensions are combined into a single one representing an entire shell layer	Real, Rank=3, Allocatable
SC	Holds amounts of spheres in shell layers	Real, Rank=1, Allocatable
NOSRJP	Amount of hardspheres in modified random packing	Integer
RJP	Center points of hardspheres in modified random packing	Real, Rank=2, Allocatable
RRATIO	Ratio of core-shell diameter to hardsphere diameter	Real
CSPRJP	Holds center points and radii of every spherical geometry element inside the random packing of core-shell particles	Real, Rank=4, Allocatable

C.3. Adapting Free Molecular Diffusion Code to Simulate Impermeability

The collision control algorithm given in section 3.6.3 in Chapter 3 was implemented to the Fortran code given in section 3.4.1 which governs the actual random-walking part, by using additional individual nested do loops inside the master loop to carry out necessary calculations required by the control structures in collision control algorithm and adding a function that calculates local positions of tracers.

```
CALL INIT_RANDOM_SEED()

DO I=1,NP

  CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)

  DO J=1,NS

    POSITIONNEW(I,1)=POSITIONOLD(I,1)+NUMBERSX(J)*DX
    POSITIONNEW(I,2)=POSITIONOLD(I,2)+NUMBERSY(J)*DX
    POSITIONNEW(I,3)=POSITIONOLD(I,3)+NUMBERSZ(J)*DX

    !CALCULATING LOCAL POSITION VECTOR COMPONENTS
    PLOCAL(1)=MODULO(POSITIONNEW(I,1),RRATIO)
    PLOCAL(2)=MODULO(POSITIONNEW(I,2),RRATIO)
    PLOCAL(3)=MODULO(POSITIONNEW(I,3),RRATIO)

    !COLLISION CONTROL ALGORITHM STEP 1
    DO K=1,NOSRJP

      IF ((((PLOCAL(1)-RJP(K,1))**2)+((PLOCAL(2)-RJP(K,2))**2)&
      &+((PLOCAL(3)-RJP(K,3))**2)).LE.(PR**2)) THEN

      !COLLISION CONTROL ALGORITHM STEP 2
      DO L=1,NOL

        IF ((((PLOCAL(1)-RJP(K,1))**2)+((PLOCAL(2)-RJP(K,2))**&
        &2) +((PLOCAL(3)-RJP(K,3))**2)) .LE. (RSOI(L)**2)) THEN

        !COLLISION CONTROL ALGORITHM STEP 3
        DO M=1,SC(L)

          IF     ((((PLOCAL(1)-CSPRJP(K,L,M,1))**2)+((PLOCAL(2)-CS&
          &PRJP(K,L,M,2))**2) +((PLOCAL(3)-CSPRJP(K,L,M,3))**2&
          &)).LE.(CSPRJP(K,L,M,4)**2)) THEN

          POSITIONNEW(I,1)=POSITIONOLD(I,1)
          POSITIONNEW(I,2)=POSITIONOLD(I,2)
          POSITIONNEW(I,3)=POSITIONOLD(I,3)

          GOTO 200

          ENDIF
```

```
        ENDDO
        ENDIF
      ENDDO
      ENDIF

200 IF (MOD(J,B).EQ.0) THEN
      DISPLACEMENT(I,J/B)=SQRT(POSITIONNEW(I,1)**2+ POSITIONNE&
      &W(I,2)**2+ POSITIONNEW(I,3)**2)
    ENDIF

    POSITIONOLD(I,1)=POSITIONNEW(I,1)
    POSITIONOLD(I,2)=POSITIONNEW(I,2)
    POSITIONOLD(I,3)=POSITIONNEW(I,3)

  ENDDO

ENDDO
```

If the condition in algorithm step 1.a is met, program by-passes any nested do loops inside the 3rd loop. If step 1.b is satisfied, then step 2 is inevitably carried out, 4th do loop is iterated and the shell layer that needs to be checked, which is represented by do loop dummy index `L`, is detected by the control structure. Then step 3 is applied by the inner-most nested do loop in the code. If the control structure fails to satisfy the condition after checking all elements in the shell layer, condition in step 3.a is met, program exits to 2nd do loop, data is collected and global position variable are swapped for the next random step. If the the control structure satisfies the inequality for any shell layer element, collision is detected, global position of tracer is set to previous global position before the collision, program is forced to exit to 2nd do loop by `GOTO` command to prevent any possible unnecessary iterations remaining for the rest of shell layer elements. Then data is collected and global position variable are swapped for the next random step. The algorithm in this state requires the least amount of control structure executions, which are major contributors to wall-clock time of the program, required for the simulation of impermeability.

C.4. Storage of Velocity Field

Fortran program created section 3.4 can simulate diffusion in a random packing of core-shell particles by approximating the diffusion event as random motion of a large amount of tracers. In the latter sections 3.5 and 3.6, flow of water through the periodic random packing was simulated to obtain a velocity field in the system and it was explained that random-walk models can be coupled with a velocity field to simulate dispersion. The Fortran program for diffusion is modified and added new features to convert it for dispersion. There are three main modification steps.

1. Reading and storing the velocity field obtained from a different platform.
2. An innate trilinear interpolation mechanism for the stored discrete velocity field.
3. Changing the structure of random-walk loop in the program.

COMSOL Multiphysics creates data tables of selected dependent variables calculated at the nodes of a user defined regular grid. Velocity field inside the exact packing used in diffusion model was extracted from the results of fluid flow simulation done using COMSOL, by defining a regular grid of 101 nodes in each axis, equally separated from each other by a length equal to 1/100 of the cubic periodic cell length. The data table contained velocity field components at an ordered sequence of nodes, output file was re-organized into a certain format by using Octave so that the format can be used in Fortran code to properly read the file.

These data can be written on a four dimensional array, first three dimensions of which holds normalized node coordinates while velocity components are held by the 4th dimension of the array. Following code opens the file containing data, allocates memory to rank 4 `VF` array and assigns read data to array elements.

```
GRID=101
ALLOCATE(VF(GRID,GRID,GRID,3))

DO I=1,GRID
  DO J=1,GRID
    DO K=1,GRID

      READ(10,100) VF(I,J,K,1),VF(I,J,K,2),VF(I,J,K,3)
```

```
    ENDDO
  ENDDO
ENDDO
```

If the velocity field is obtained for a system in Stokes Flow limits ($Re \leq 0.1$), then the velocity field is eligible for linear scaling. If the original velocity field read from data file has an average Peclet Number Pe_0, then another velocity profile corresponding to and arbitrary Pe can be calculated by the following code.

```
PE= Pe
PE0=Pe0
DO I=1,101
  DO J=1,101
    DO K=1,101
      DO L=1,3
        VF(I,J,K,L)=VF(I,J,K,L)*(PE/PE0)
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

Of course, the Reynold Number for the superficial velocity city value corresponding to chosen Pe must also not exceed Stokes Flow limits.

C.5. Tri-linear Interpolation of Velocity Vectors

Velocity vectors at the points other the the nodes are calculated by trilinear interpolation. A dedicated subroutine was written for this purpose. A rank 1 array with 3 elements, `VEL` was declared for holding the interpolated vector components. The subroutine takes 5 arguments which are local tracer position, velocity field, periodic cell length, size of first three dimensions in velocity field array and `VEL` respectively and returns interpolated velocity components to the array `VEL`.

For a given local position of a tracer in the main periodic cell, components of a scaled position ($\mathbf{X}_{\boldsymbol{L}}^{Scaled}$) with respect to regular grid spacing used in data extraction ($L_{ns} = L_{pc}/100$) can be calculated by the Equation 56.

$$X_{L,i}^{scaled} = \mathrm{X}_{L,i}/L_{ns} \tag{56}$$

Then, the cubic unit cell in which the interpolation must be carried out (interpolation cell), is defined by the positions of the closest 8 velocity vectors to the scaled local position. Position ($\mathbf{P}_{\boldsymbol{E}}$) components of the interpolation cell edge with smallest position components is then found by the Equation 57. Note, the components will be integers in the interval $1 \leq \mathrm{X}_{E,i} \leq 101$

$$\mathrm{P}_{E,i} = \Phi\left(X_{L,i}^{scaled}\right) + 1 \tag{57}$$

If the edges of interpolation cell is numbered as seen in Figure C.1, then position points of all edges can be found by the Equations (58) to (65).

$$\mathrm{P}_E{}^1 = (\mathrm{P}_{E,1}, \mathrm{P}_{E,2}, \mathrm{P}_{E,3}) \tag{58}$$

$$\mathrm{P}_E{}^2 = (\mathrm{P}_{E,1} + 1, \mathrm{P}_{E,2}, \mathrm{P}_{E,3}) \tag{59}$$

$$\mathrm{P}_E{}^3 = (\mathrm{P}_{E,1}, \mathrm{P}_{E,2}, \mathrm{P}_{E,3} + 1) \tag{60}$$

$$\mathrm{P}_E{}^4 = (\mathrm{P}_{E,1} + 1, \mathrm{P}_{E,2}, \mathrm{P}_{E,3} + 1) \tag{61}$$

$$\mathrm{P}_E{}^5 = (\mathrm{P}_{E1}, \mathrm{P}_{E,2} + 1, \mathrm{P}_{E,3}) \tag{62}$$

$$P_E{}^6 = (P_{E,1} + 1, P_{E,2} + 1, P_{E,3}) \quad (63)$$

$$P_E{}^7 = (P_{E,1}, P_{E,2} + 1, P_{E,3} + 1) \quad (64)$$

$$P_E{}^8 = (P_{E,1} + 1, P_{E,2} + 1, P_{E,3} + 1) \quad (65)$$

After the edges of interpolation cell is calculated, velocity components at the corresponding edge points are called from VF array and assigned to each edge.

Since the interpolation cell is a unit cube, a normalized tracer position inside the interpolation cell ($\mathbf{X_{Ln}}$) can be defined as having the components:

$$X_{Ln,i} = \frac{X_{L,i} - \Phi\left(X_{L,i}^{scaled}\right).L_{ns}}{L_{ns}} = X_{L,i}^{scaled} - \Phi\left(X_{L,i}^{scaled}\right) \quad (66)$$

Normalized tracer position can be used as the interpolation point in a unit interpolation cell cornered at the origin and occupying the edge of first octant. It is equivalent to interpolating velocity components in the cell defined by points calculated in Equations (67) to (68) but these points are just used for locating velocity components in VF array. Additionally, using normalized tracer position simplifies interpolation equations by getting rid of intersection terms in the line equations.

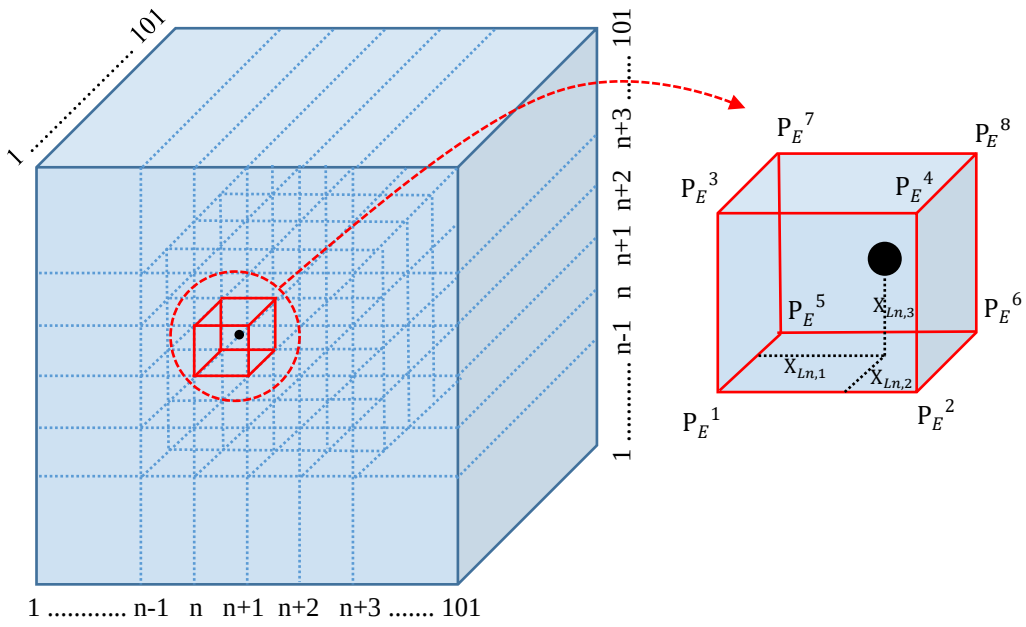


Figure C.1: A basic visual representation of VF array (large blue cube on left) and an interpolation cell (small unit cube high-lighted with red) for a tracer (black dot) with indicated local position. Edge of the interpolation cell is at (n, n+1, 1), where P_E^1 is placed at. Magnified view of the interpolation cell on the right shows the arangement of numbered velocity vectors. Normalized position of the tracer inside the interpolation is illustrated by black dotted lines.

An algorithm that uses the equations given in this section is presented below.

1. Read local tracer position, velocity field, periodic cell length and amount of nodes in one dimension of the cubic velocity field array.
2. Determine the position of reference edge of interpolation cell by Equations (69) and (70).
3. Determine normalized tracer position in the interpolation cell by using Equation (71).
4. Locate velocity vectors in VF using the location of interpolation cell calculated by Equations (72) to (73) and assign the first velocity component to corresponding edges.
5. Interpolate.

6. Repeat steps 4 and 5 for remaining two velocity components.

Following Fortran subroutine is an implementation of the interpolation algorithm. Actual interpolation is carried out inside the do loop individually for each velocity vector component. Neighboring points along lines parallel to x-axis is interpolated first, reducing the system to bilinear. Then second interpolation is done along z-axis. Finally, remaining two points are interpolated along y-axis to obtain velocity components at the local position $\mathbf{X}_L$ of tracer.

```
SUBROUTINE GIMME_VELOCITY(LOCALP,FIELD,LENGTH,GRID,VEL)
IMPLICIT NONE

!INTERPOLATION ALGORITHM STEP 1
INTEGER, INTENT(IN) :: GRID
DOUBLE PRECISION, INTENT(IN) :: FIELD(GRID,GRID,GRID,3)
DOUBLE PRECISION, INTENT(IN) :: LOCALP(3),LENGTH
DOUBLE PRECISION, INTENT(OUT) :: VEL(3)
DOUBLE PRECISION :: XUNIT,YUNIT,ZUNIT,MAIN(8),FIRST(4),SECOND(2)
INTEGER :: EDGE(3),DUMMY

!INTERPOLATION ALGORITHM STEP 2
EDGE(1)=INT(LOCALP(1)/(LENGTH/100))+1
EDGE(2)=INT(LOCALP(2)/(LENGTH/100))+1
EDGE(3)=INT(LOCALP(3)/(LENGTH/100))+1

!INTERPOLATION ALGORITHM STEP 3
XUNIT=MOD(LOCALP(1),(LENGTH/100))/(LENGTH/100)
YUNIT=MOD(LOCALP(2),(LENGTH/100))/(LENGTH/100)
ZUNIT=MOD(LOCALP(3),(LENGTH/100))/(LENGTH/100)

DO DUMMY=1,3
  !INTERPOLATION ALGORITHM STEP 4
  MAIN(1)=FIELD(EDGE(1),EDGE(2),EDGE(3),DUMMY)
  MAIN(2)=FIELD(EDGE(1)+1,EDGE(2),EDGE(3),DUMMY)
  MAIN(3)=FIELD(EDGE(1),EDGE(2),EDGE(3)+1,DUMMY)
  MAIN(4)=FIELD(EDGE(1)+1,EDGE(2),EDGE(3)+1,DUMMY)
  MAIN(5)=FIELD(EDGE(1),EDGE(2)+1,EDGE(3),DUMMY)
  MAIN(6)=FIELD(EDGE(1)+1,EDGE(2)+1,EDGE(3),DUMMY)
  MAIN(7)=FIELD(EDGE(1),EDGE(2)+1,EDGE(3)+1,DUMMY)
  MAIN(8)=FIELD(EDGE(1)+1,EDGE(2)+1,EDGE(3)+1,DUMMY)

  !INTERPOLATION ALGORITHM STEP 5 (REDUCES TO BILINEAR)
  FIRST(1)=MAIN(1)+((MAIN(2)-MAIN(1))*XUNIT)
  FIRST(2)=MAIN(3)+((MAIN(4)-MAIN(3))*XUNIT)
  FIRST(3)=MAIN(5)+((MAIN(6)-MAIN(5))*XUNIT)
  FIRST(4)=MAIN(7)+((MAIN(8)-MAIN(7))*XUNIT)

  !INTERPOLATION ALGORITHM STEP 5 (REDUCES TO LINEAR)
  SECOND(1)=FIRST(1)+((FIRST(2)-FIRST(1))*ZUNIT)
  SECOND(2)=FIRST(3)+((FIRST(4)-FIRST(3))*ZUNIT)

  !INTERPOLATION ALGORITHM STEP 5
```

```
  VEL(DUMMY)=SECOND(1)+((SECOND(2)-SECOND(1))*YUNIT)
ENDDO

END SUBROUTINE GIMME_VELOCITY
```

C.6. Adaptation of Diffusion Program to Simulate Dispersion

Equation (15) given in the section 3.1.1 was modified into the Equation (74) to include a convective displacement term alongside the random displacement. Convective displacement by definition is the distance a tracer travels in a time interval Δt according to the velocity ($\boldsymbol{\vartheta}$) of the fluid at the current position of the tracer.

$$\mathbf{X}(t+\Delta t) = \mathbf{X}(t) + \boldsymbol{\xi}\Delta l + \Delta t\boldsymbol{\vartheta}\left(X_1(t), X_2(t), X_3(t)\right) \tag{75}$$

Velocity field obtained from the fluid flow simulation done in COMSOL was transferred to `VF` array. The array contains velocity components in SI units. This array is converted to contain components of net displacement vectors to be used in Equation (75) by multiplying all of its elements by Δt. This modification does not affect trilinear interpolation subroutine since `VF` still contains vector components.

```
      DO I=1,GRID
            DO J=1,GRID
                  DO K=1,GRID
                        DO L=1,3
                        VF(I,J,K,L)=VF(I,J,K,L)*DT
                        ENDDO
                  ENDDO
            ENDDO
      ENDDO
```

With the changes made to Equation (15) and addition of interpolation subroutine, the last code fragment given in section 3.4.3, which calculates random-walk of tracers in a random jammed packing of core-shell particles, needs 3 main alterations.

1. Equation (15) must be changed to Equation (75).

2. Apart from the local positions $\mathbf{X_L}(t + \Delta t)$ needed for collision control, local positions of tracers before displacement, $\mathbf{X_L}(t)$, must also be calculated prior to the execution of interpolation subroutine since Equation (75) depends on it.
3. Extracted data changes to cover only longitudinal displacement, since it is the only point of interest.

A rank 1 array `VELOCITY` with size 3 is declared, given to interpolation subroutine as argument, and is holding interpolated net displacement vectors due to fluid flow. After the modifications on diffusion code, dispersion code is completed and given below.

```
CALL INIT_RANDOM_SEED()

DO I=1,NP

  CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)

  DO J=1,NS
      !ADDITIONAL CALCULATION OF LOCAL POSITION
      PLOCAL(1)=MODULO(POSITIONOLD(I,1),RRATIO)
      PLOCAL(2)=MODULO(POSITIONOLD(I,2),RRATIO)
      PLOCAL(3)=MODULO(POSITIONOLD(I,3),RRATIO)

      CALL GIMME_VELOCITY(PLOCAL,VF,RRATIO,GRID,VELOCITY)

    !ALTERATION OF EQUATION (15) INTO EQUATION (76)
    POSITIONNEW(I,1)=POSITIONOLD(I,1)+NUMBERSX(J)*DX+VELOCITY(1)
    POSITIONNEW(I,2)=POSITIONOLD(I,2)+NUMBERSY(J)*DX+VELOCITY(2)
    POSITIONNEW(I,3)=POSITIONOLD(I,3)+NUMBERSZ(J)*DX+VELOCITY(3)

    PLOCAL(1)=MODULO(POSITIONNEW(I,1),RRATIO)
    PLOCAL(2)=MODULO(POSITIONNEW(I,2),RRATIO)
    PLOCAL(3)=MODULO(POSITIONNEW(I,3),RRATIO)

    DO K=1,NOSRJP

      IF ((((PLOCAL(1)-RJP(K,1))**2)+((PLOCAL(2)-RJP(K,2))**2)&
      &+((PLOCAL(3)-RJP(K,3))**2)).LE.(PR**2)) THEN

      DO L=1,NOL

        IF ((((PLOCAL(1)-RJP(K,1))**2)+((PLOCAL(2)-RJP(K,2))**&
        &2) +((PLOCAL(3)-RJP(K,3))**2)) .LE. (RSOI(L)**2)) THEN

        DO M=1,SC(L)

          IF     ((((PLOCAL(1)-CSPRJP(K,L,M,1))**2)+((PLOCAL(2)-CS&
          &PRJP(K,L,M,2))**2) +((PLOCAL(3)-CSPRJP(K,L,M,3))**2&
          &)).LE.(CSPRJP(K,L,M,4)**2)) THEN
```

```
          POSITIONNEW(I,1)=POSITIONOLD(I,1)
          POSITIONNEW(I,2)=POSITIONOLD(I,2)
          POSITIONNEW(I,3)=POSITIONOLD(I,3)

          GOTO 200

          ENDIF
        ENDDO
        ENDIF
      ENDDO
      ENDIF

200 IF (MOD(J,B).EQ.0) THEN
                          !ALTERATION TO DATA EXTRACTION
      DISPLACEMENT(I,J/B)=ABS(POSITIONNEW(I,3)-POSITIONI(I,3))
    ENDIF

    POSITIONOLD(I,1)=POSITIONNEW(I,1)
    POSITIONOLD(I,2)=POSITIONNEW(I,2)
    POSITIONOLD(I,3)=POSITIONNEW(I,3)

  ENDDO

ENDDO
```

Collected longitudinal displacement data taken at equally distant times, is treated to calculate variance of longitudinal displacements of all tracers in the tracer ensemble. Two new real strings `VARIANCE` and `MEANDISP` are declared to hold intermediate values during the calculations, where the first also holds the final variance value. Following code fragment carries out data treatment and writes time dependent variance data to a file.

```
OPEN (#, FILE="LDISP VAR.TXT", STATUS="UNKNOWN", ACTION="WRITE")

DO I=1,AODP
  SUMDISP=0.0

  DO J=1,NP
    SUMDISP=SUMDISP+ DISPLACEMENT (J,I)
  ENDDO

  MEANDISP=SUMDISP/NP
  VARIANCE=0.0

  DO J=1,NP
    VARIANCE=VARIANCE+(((DISPLACEMENT(J,I)-MEANDISP)&
    &**2)/NP)
  ENDDO

  WRITE(#,'format') VARIANCE
```

```
ENDDO
```

C.7. Parallelization of Diffusion and Dispersion Programs

Parallelization of Fortran programs was done by using OpenMP. It is an open source extension for open source GFortran compiler. Making use of *pragmas* otherwise invisible to the compiler, OpenMP forces other available processor threads to execute the program in parallel with each other. Depending on the fraction of the code that is parallelizable, wall-clock time of the program is significantly improved.

OpenMP pragmas are in comment form and only recognized by the compiler if the directive "-fopenmp" is added to compilation directives. At the start of parallel region, "!$omp parallel" pragma is used. Parallel region is terminated at the line "!$omp end parallel" appears. Do loop iterations in the program can be shared between available threads by "!omp do" and "!omp end do" placed at the start and end of the loop respectively. If not specified, load is distributed to each available thread equally. In the diffusion and dispersion programs, only the loops taking care of random-walk iterations were placed inside a parallel region by using combined directives "!$omp parallel do" and "!omp end parallel do" since the other sections of the program does not have a significant contribution to wall-clock times. If required, certain sections of the parallel region can be marked with "!$omp critical" and "!omp end critical" pragmas. In the critical region, procedures are executed in series.

One thing that is really problematic in parallel computing is pseudo-random number generation. RNG subroutines readily available in the compilers use a seed to generate a pseudo-random number, then previous pseudo-random number is used as seed to generate the next random number in the sequence. Entire sequence of these random numbers are actually generated based on a function and predictable for a given initial seed, but since they are uniformly distributed they can be treated as random numbers, hence they are pseudo-random numbers.

During parallel execution of the program, an individual thread gets a subset of the entire sequence of pseudo-numbers that are collectively generated by all processor threads. However these subsets of numbers might not be uniformly distributed. If each individual thread is wanted to use a uniformly distributed sequence of pseudo random numbers, there is only two solutions to the problem.

1. Use a dedicated pseudo-random number generator for each thread
2. Do not allow parallel execution during pseudo-random number generation.

First option might be a subject to mathematics or computer engineering research but it is beyond the interests of even this study, considering heavy usage of Fortran. However, the second option required much less knowledge in mathematics and computer engineering and a feasible option for proper parallelization of the model.

Fortran programs written for Diffusion and Dispersion models already pre-generates and stores all required random numbers required for a tracer to complete its random motion. Recall, the arrays `NUMBERSX`, `NUMBERSY` and `NUMBERSZ` are of size total amount of number steps ,`NS`, and they are assigned values +1 or -1 at random in a single line of code by the subroutine `NUMBERS`.

```
!$omp critical
  CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)
!$omp end critical
```

If the RNG line is protected by a critical region, very simply defined by the code given above, any thread that tries to execute the `NUMBERS` subroutine must wait until the other thread finishes generating random numbers for the tracer it is occupied with. Effectively, this is no different than the entire non-parallellized programs since every tracer uses a complete sequence of random numbers to take random steps.

Another important factor that needs to be considered for parallelization procedure is the variables that is re-assigned throughout the parallel region in the program. Boundary dummies for do loops as well as read-only variables previously defined and used in the parallel region are safe. However any other user-defined variable may cause problems in the parallel region. Common variables in both programs that may cause

problems are PLOCAL, NUMBERSX, NUMBERSY and NUMBERSZ while VELOCITY was introduced only for dispersion program. These strings must be marked as 'private' at the beginning of the parallel region so that each thread independently calculate new values for these variables throughout the execution. Marking is done by simply attaching "private([string1, string2, ..])" directive at the end of the starting pragma. After the parallelization, main iteration loop looks like the following.

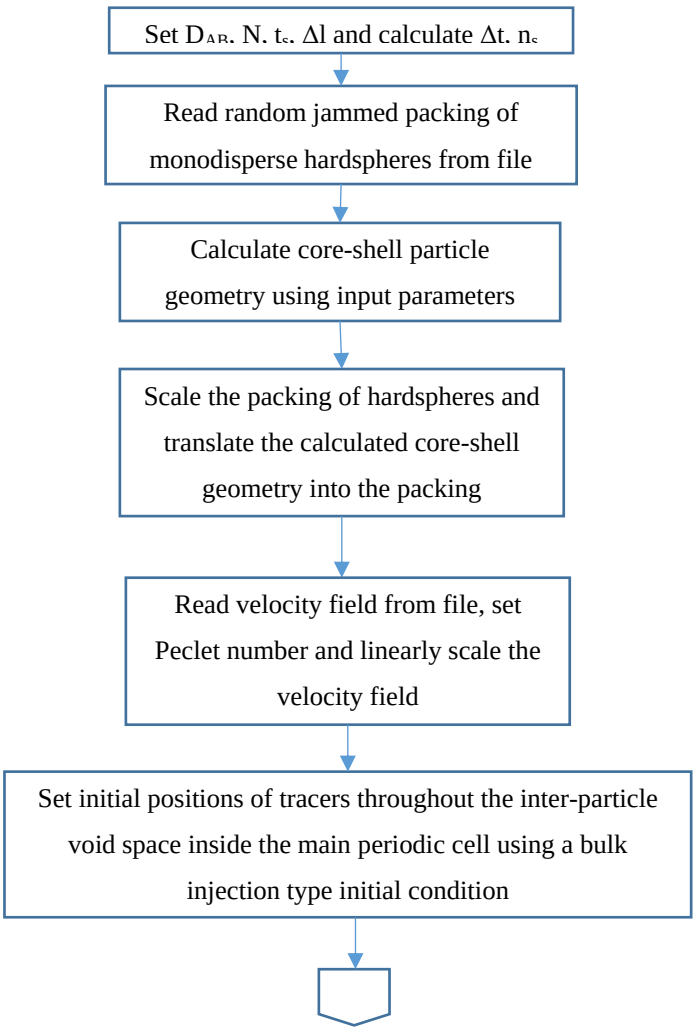
```
!$omp parallel do private(plocal, numbersx, numbersy, numbersz,
velocity)
DO I=1,NP

  !$omp critical
    CALL NUMBERS(NUMBERSX,NUMBERSY,NUMBERSZ)
  !$omp end critical
  .
  .!SECOND DO BLOCK HERE
  .
ENDDO
!$omp end parallel do
```

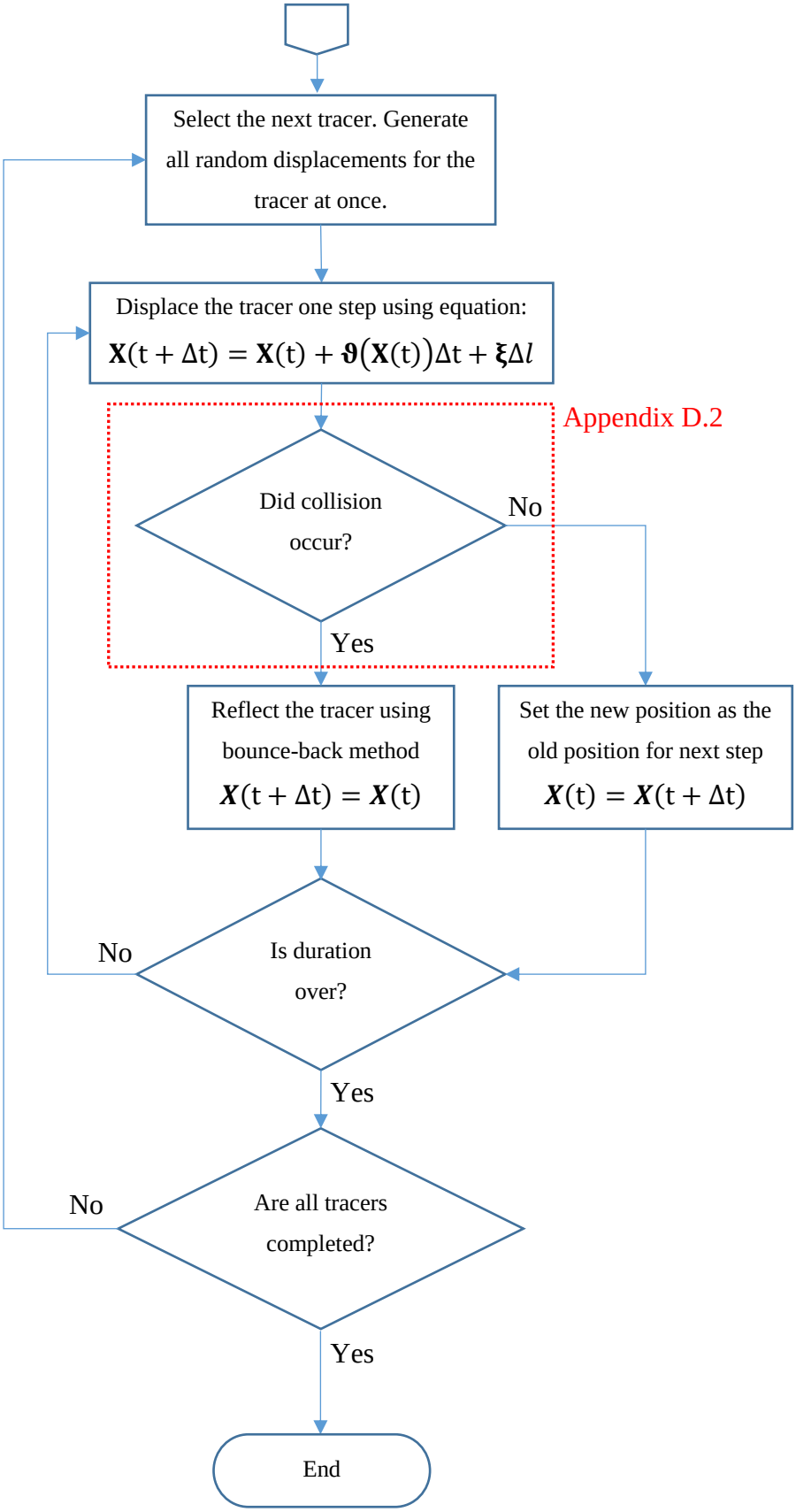
Note that, POSITIONOLD and POSITIONNEW arrays as well as DISPLACEMENT, also assigned new values in the parallel region. However the new values are not assigned more than once to a single element in these arrays. Therefore they do not cause any inconveniency and not included in the private list.

APPENDIX D

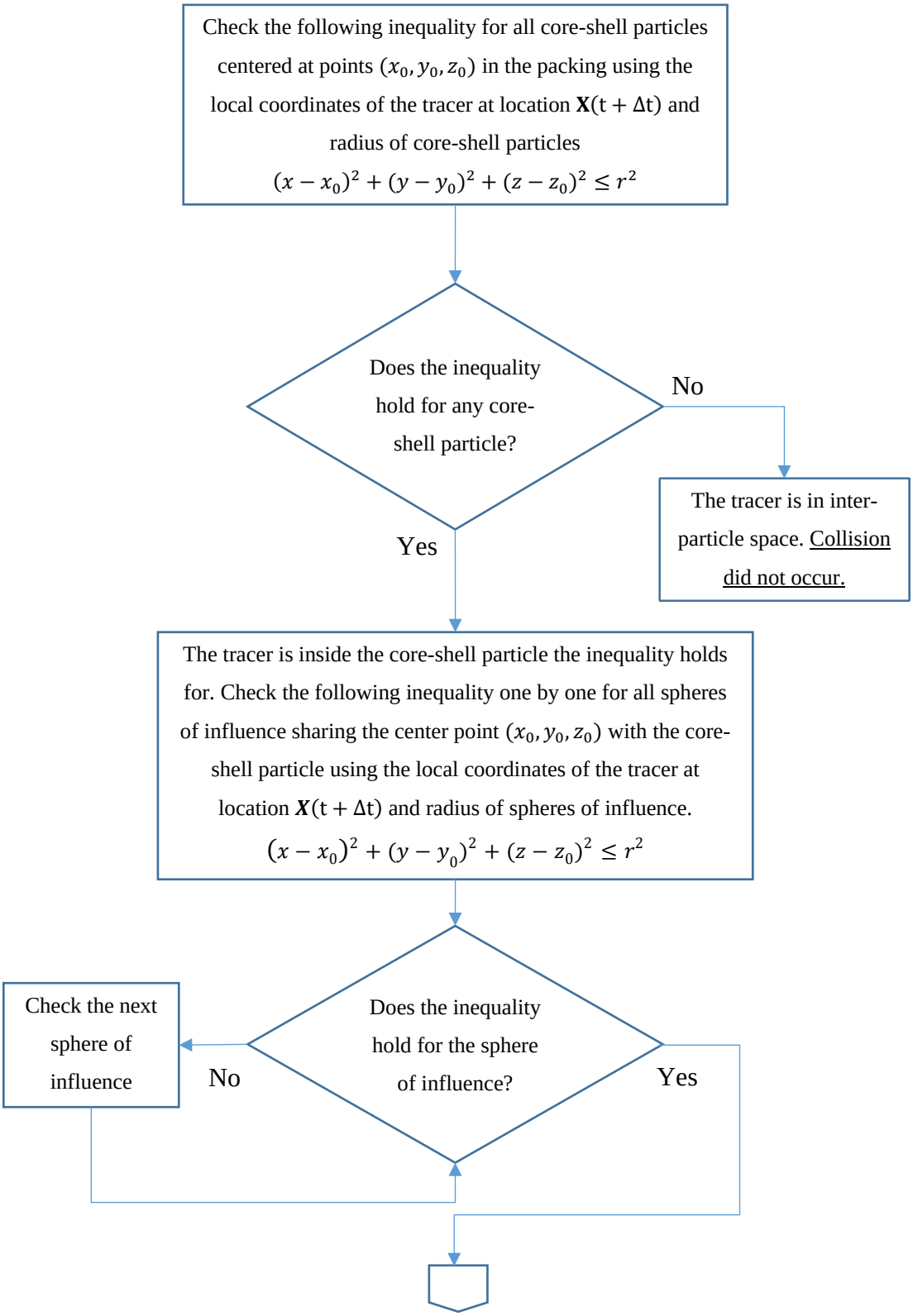
FLOWCHARTS

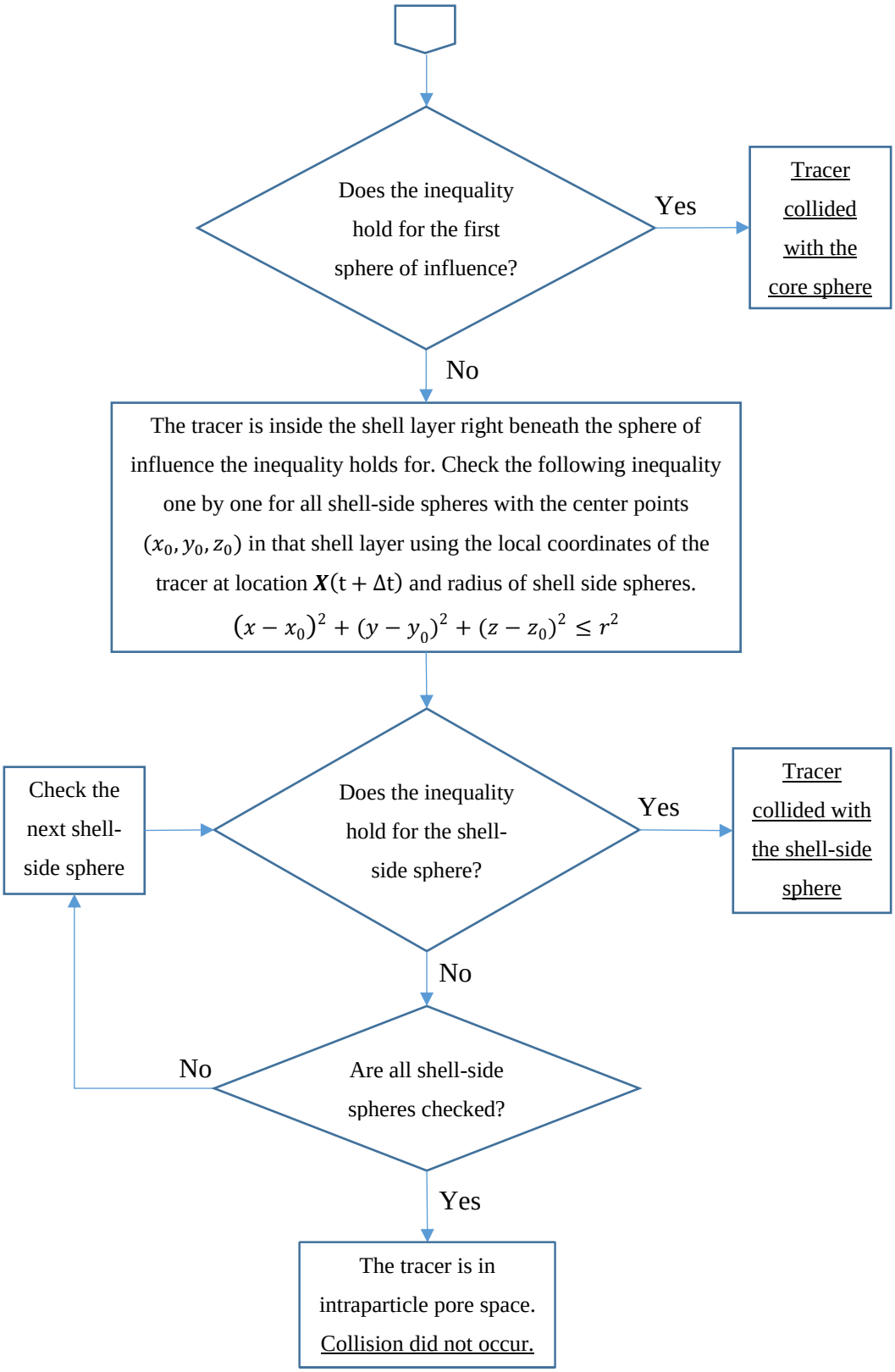
D.1. Diffusion/Dispersion Algorithm




Select the next tracer. Generate all random displacements for the tracer at once.
Displace the tracer one step using equation:
$\mathbf{X}(t+\Delta t)=\mathbf{X}(t)+\boldsymbol{\vartheta}(\mathbf{X}(t))\Delta t+\boldsymbol{\xi}\Delta l$
Appendix D.2
Did collision occur?
No
Yes
Reflect the tracer using bounce-back method
$\boldsymbol{X}(t+\Delta t)=\boldsymbol{X}(t)$
Set the new position as the old position for next step
$\boldsymbol{X}(t)=\boldsymbol{X}(t+\Delta t)$
Is duration over?
No
Yes
Are all tracers completed?
No
Yes
End

D.2. Collision Control Algorithm




Does the inequality hold for the first sphere of influence?
Yes
Tracer collided with the core sphere
No
The tracer is inside the shell layer right beneath the sphere of influence the inequality holds for. Check the following inequality one by one for all shell-side spheres with the center points (x_0, y_0, z_0) in that shell layer using the local coordinates of the tracer at location $\boldsymbol{X}(t + \Delta t)$ and radius of shell side spheres.
$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \leq r^2$
Does the inequality hold for the shell-side sphere?
Yes
Tracer collided with the shell-side sphere
Check the next shell-side sphere
No
Are all shell-side spheres checked?
No
Yes
The tracer is in intraparticle pore space. Collision did not occur.

D.3. Overall Work Flowchart