Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemistry

Mina Sørensen Bratvold

The permeability of porous media as studied by NEMD

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Signe Kjelstrup
Co-supervisor: Michael Tobias Rauter and Olav Galteland
June 2021

NTNU
Norwegian University of
Science and Technology

Mina Sørensen Bratvold

The permeability of porous media as studied by NEMD

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Signe Kjelstrup
Co-supervisor: Michael Tobias Rauter and Olav Galteland
June 2021

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemistry

Abstract

The overall goal of the project is to contribute towards establishing a non-equilibrium thermodynamic theory for the nano-scale. More specifically, state variables of confined media are defined, the knowledge about driving forces is expanded and a method to determine the permeability is designed. Nano-porous media is here represented as a face centered cubic lattice of spherical grain particles. The nano-porous medium contains fluid, and an applied pressure difference causes fluid to flow through the system. The system is studied using Non-equilibrium molecular dynamics (NEMD) simulations. To build trust in the input and post-processing scripts, two simpler systems are studied initially, an evaporating liquid and a slit pore where the width is varied. For the slit pore, the results from the molecular dynamics simulations are compared to the solution for planar Poiseuille flow. The deviation from planar Poiseuille flow is found to decrease with increasing slit pore width, and for the widest slit pores the deviation is less than 10%. The viscosity has to be calculated to allow for comparison with the planar Poiseuille flow equations. The viscosity is determined by applying the SLLOD algorithm, performing simulations at different shear rates and extrapolating the data to zero shear rate.

The main part of the project is extending and expanding the work of Galteland *et. al.*, by calculating the pressure in nano-porous media, and designing a method to determine the permeability [1]. The permeability is determined from the gradient in integral pressure in the porous medium at non-equilibrium conditions. Four different values of the lattice constant are tested, and confinement effects are observed for the smallest lattice constants. To obtain the integral pressure, the total compressional energy must be determined. This requires knowledge about the geometry of the lattice, as well as the integral rock pressure and surface tension as functions of fluid pressure. To obtain these functions, equilibrium simulations at different fluid densities are carried out. Comparing the different lattice constants, the permeability divided by viscosity is found to increase with increasing lattice constant and is highly dependent on porosity. The permeability calculated for a lattice constant of $a = 20$ is about 3.5 times larger than the permeability given by the Kozeny-Carman equation.

Sammendrag

Det overordnede målet med prosjektet er å bidra til å etablere en beskrivelse av irreversibel termodynamikk for nano-skalaen. Mer spesifikt defineres tilstandsvariabler for fluid som er innesluttet i porøse medier, kunnskapen om drivkrefter utvides og en metode for å bestemme permeabiliteten utformes. Nano-porøse medier er her representert som et kubisk flatesentrert gitter av sfæriske partikler. Det nano-porøse mediet inneholder væske, og en påført trykkforskjell får væske til å strømme gjennom systemet. Systemet studeres ved hjelp av ikke-likevekt molekylærdynamikk-simuleringer (NEMD). For å bygge tillit til input- og etterbehandlingsskriptene, studeres to enklere systemer innledningsvis, en væske som fordamper og en spaltepore der bredden varieres. For spalteporen sammenlignes resultatene fra molekylærdynamikk-simuleringene med løsningen for plan Poiseuille-strømning. Avviket fra plan Poiseuille-strømning avtar med økende bredde på spalteporene, og for de bredeste spalteporene er avviket mindre enn 10 %. Viskositeten må beregnes for å muliggjøre sammenligning med ligningen for plan Poiseuille-strømning. Viskositeten bestemmes ved å bruke SLLOD-algoritmen, utføre simuleringer med forskjellige skjærhastigheter og ekstrapolere dataene til null skjærhastighet.

Hoveddelen av prosjektet er å forlenge og utvide arbeidet til Galteland *et. al.*, for beregning av trykket i nano-porøse medier og utforme en metode for å bestemme permeabilitet [1]. Permeabiliteten bestemmes fra gradienten i integraltrykket i det porøse mediet ved ikke-likevektsbetingelser. Fire ulike verdier av gitterkonstanten blir testet, og inneslutningseffekter blir observert for de minste gitterkonstantene. For å finne integraltrykket, må den totale kompresjonsenergien bestemmes. Dette krever kunnskap om geometrien til gitteret, samt det integrerte trykket i det faste porøse mediet og overflatespenningen som funksjoner av væsketrykk. For å oppnå disse funksjonene brukes likevektssimuleringer ved forskjellige væsketettheter. Ved sammenligning av de ulike gitterkonstantene, er permeabilitet delt på viskositet funnet å øke med økende gitterkonstant og avhenger i stor grad av porøsitet. Permeabiliteten beregnet for en gitterkonstant på $a = 20$ er omtrent 3,5 ganger større enn permeabiliteten gitt av Kozeny-Carman-ligningen.

Preface

This report is the result of a master project related to the course *TKJ4900 Chemistry, Master's Thesis* at the Norwegian University of Science and Technology (NTNU). The aim of the course is to gain in-depth knowledge within a specific field, by gathering information from literature, as well as planning and carrying out a research project using scientific methods. The project work has been within the field of non-equilibrium thermodynamics, and has been performed in collaboration with PoreLab, a Norwegian Center of Excellence created in 2017.

For computational resources, I would like to thank UNINETT Sigma2-the National Infrastructure for High Performance Computing and Data Storage in Norway.

I would like to express my vast gratitude to my main supervisor Signe Kjelstrup for valuable guidance, for sharing her knowledge and providing insightful feedback throughout the whole project work. I also want to thank my co-supervisors Michael Rauter and Olav Galteland, as well as Dick Bedeaux for providing support and feedback during the project work. I have appreciated your interest in the project, and your willingness to always help at short notice. Your feedback and help is much appreciated. The co-supervisors have also contributed by providing useful scripts for running simulations and post-processing the results.

I also want to thank the other employees at PoreLab that have engaged in discussions. Being a part of a research group has been a valuable experience.

I want to thank my family and friends for the support and understanding you have provided throughout my degree and this project work. I highly value your encouragement. You are always there for me, and for that I am extremely grateful.

List of Figures

List of Tables

Contents

1 Introduction

Many systems include flow through porous media, both in nature and the industry [11]. Biological tissues, blood vessels, bones, soils and rocks are some examples of such systems in nature. For industrial systems, porous media transport is central in the fields of petroleum engineering, hydrology and chemical engineering [12].

A porous medium is a solid which contains void spaces called pores. The void spaces can be connected or unconnected, and can be dispersed within the solid in a regular or random manner [13]. There are two characteristic sizes describing porous media, the size of a grain and the distance between the surfaces of two grains [1]. Different types of porous media exist with pore sizes down to the nano-meter scale. Nano-porous media have attached significant attention because of their excellent porous properties and have become central for applications in the fields of chromatography, membrane separation and catalysis [14][15].

Because of their central role in a variety of fields, describing flow in porous media is of great importance. This involves the calculation of thermodynamic variables and driving forces. The pressure gradient is important, as it acts as a driving force for flow. There are also other relevant driving forces for porous media transport, for example thermal driving forces [16]. However, the effect of other driving forces are less studied for porous media than the effect of the pressure gradient [16]. Darcy's law has been extensively used to describe fluid flow in porous media, relating the instantaneous fluid flux to the pressure difference over a given distance [17][18]. A variable that appears in Darcy's law is the permeability, which describes the ease of flow. Knowledge of the permeability of porous media is therefore crucial to understanding fluid flow in various materials [19].

The widespread application of nano-porous materials has lead to a growing interest in the accurate description of the thermodynamic behavior of fluids confined in nano-pores [15]. The pressure is an important thermodynamic variable also at the nano-scale, as it is needed for describing the flow rate, diffusion coefficient, and swelling of the nano-porous material [15]. It is known that thermodynamic properties change upon confinement, so calculation of thermodynamical variables and driving forces is not straightforward for nano-porous media [20]. The presence of curved surfaces and fluid confinements causes difficulties for calculating the microscopic pressure tensor and pressure gradient [1]. Another problem is that the calculation of the microscopic pressure tensor yields different results depending on the chosen scale. This is problematic, as the scale that the hydrodynamic equations refer to is not well-defined for nano-porous and micro-porous media [1]. In general, challenges arise since traditional thermodynamical laws and concepts may not be applicable on the nano-scale [15].

Classical porous media models suffer from inconsistencies as variables and parameters are not rigorously defined and hence cannot be consistently measured [21]. Some of the problems have been solved by developing averaging theory. The method relies on introducing thermodynamics into a system entropy inequality. However, this method has also led to inconsistencies and ill-defined variables. Miller *et. al.*. established a framework called the thermodynamically constrained averaging theory (TCAT) approach [21]. It can be used to address complex multiphase, multispecies porous media. However, the group identified significant unresolved problems with the framework, for example that it results in non-traditional closure relations.

Hill formulated that the thermodynamic equations must be adjusted for smallness for nano-porous media [1] [22]. Small system effects must be considered when the surface energy of the system cannot be neglected in relation to the bulk energy [23]. An alternative formulation is that a system is small when the total energy of two small systems combined is not equal to the separate energies. This means that thermodynamical properties, such as internal energy, entropy and component masses, are non-extensive, or in other words not proportional to system volume. Hill showed that two pressures are needed in order to describe the confinement in nano-porous media; the differential and integral pressure [22]. The differential pressure is the variable that we normally understand as pressure on the macroscopic-level, whereas the integral pressure is defined in terms of the work done by adding one small system of constant volume to the remaining ones, keeping the temperature constant [1] [22].

Erdős *et. al.*. studied the differential and integral pressures at the nano-scale and investigated the effect of several factors which contribute to the confinement in nano-porous media [15]. In particular, the group investigated the effect of pore geometry, fluid-wall interactions and differential pressure of the bulk fluid phase. The group showed that at the nano-scale, the differential and integral pressures are not the same.

Based on the distinction of the differential and integral pressure, Galteland *et. al.*. presented a new way of computing the pressure in a nano-porous medium [1]. Using Hill's thermodynamics for small systems, the group computed the pressure of a single phase fluid in a porous medium confined in a regular lattice of spherical particles. This description represents a simple model of a porous medium, which is in reality a complex system. A central element in the derivations of Galteland *et. al.*., was the application of a representative elementary volume (REV), that contains a statistically representative collection of pores [1]. Macroscale properties are then defined through the integral of microscale properties over a REV [24]. The concepts behind and the properties of the REV are discussed by many authors, eg. Bear [25], De Marsily [26] and Hassanizadeh and Gray [27].

Galteland *et. al.*. found that for a spherical rock particle of radius R, the integral and differential rock pressures, $\hat{p^r}$ and p^r respectively, are related by $\hat{p}^r - p^r = \gamma/R$, where γ is the surface tension [1]. In addition to a model consisting of a single spherical grain, they looked at a case with a FCC-lattice of spherical grains. While the study was done for two cases, the group argued that the chosen approach can be used in general for the description of porous media. For example, the group claimed that the method can be applied to a random distribution of spherical grains, but that the REV will then need to be larger in order to include a statistically representative collection of pores [1].

The permeability is important for understanding flow in porous media. In Darcy's law, the permeability k links the fluid flow to the pressure gradient over the porous medium, according to [18]:

$$q = \frac{Q}{A} = -\frac{k}{\mu}\frac{\Delta P}{L'} \tag{1.1}$$

Here, the fluid flux q is given by the volumetric flow rate Q per unit cross-sectional area A of the porous medium. ΔP is the pressure drop, L' is the length of the sample, and μ is the viscosity of the fluid [18]. The validity of Darcy's law requires certain assumptions to be met. The flow must be isothermal, laminar single-phase flow, with constant fluid viscosity, and no rock-fluid interaction [17]. Several studies in the literature

have suggested that the application of Darcy's law is not always valid [17]. Velasco *et. al.*. found that conventional Darcy's law and relative permeability concepts need to be adapted to account for fluid-wall interactions that are relevant at the nano-scale [28].

The permeability is affected by several factors, which should be understood for a better insight into flow through porous media. It is known that permeability is dependent on characteristics of the pore structure, such as porosity, tortuosity and interaction between fluid and solid [12] [19]. The relationship between pore structure and permeability has often been modeled by ([19], [29], [30], [31], [32], [33], [34])

$$k = c\Phi r^2 \tag{1.2}$$

where the permeability k is dependent on the porosity Φ, pore radius r and a geometric factor c accounting for the shape, connectivity, aspect ratio of pores and tortuosity of the pores. Different models use different geometric factors c. Following Equation (1.2), the permeability scales quadratically with the pore radius. As a porous medium typically contains pores of different sizes, a representative radius must be chosen. Nishiyama *et. al.*. found that the permeability correlates well with a critical pore radius, where the critical pore radius is defined as the radius of the largest sphere that can freely pass through the porous medium [19].

Kozeny contributed to relating the permeability to porosity through well-defined parameters of the pore structure [12],

$$k \propto \tau \frac{\Phi^3}{(1-\Phi)^2} d_{\mathrm{w}}^2 \tag{1.3}$$

τ is the tortuosity and d_w is an effective grain size. The equation was derived based on the assumption that the porous medium can be considered as a bundle of streamtubes. Later, Carman modified the equation by multiplying with the tortuosity [12],

$$k \propto \tau^2 \frac{\Phi^3}{(1-\Phi)^2} d_{\mathrm{w}}^2 \tag{1.4}$$

He argued that one must scale with the factor τ when linking the microscopic fluid velocity to the flow per unit cross-sectional area for the porous medium.

Complex pore structures make permeability calculations challenging for porous media [19]. Numerous studies have been reported where permeability of porous media is investigated in molecular dynamics simulations. An example is the work of Velasco *et. al.*., who carried out a molecular dynamics study where they investigated the permeability in different liquids confined in the nano-pores of pseudo-porous rock [28]. As describing transport of fluid in nano-porous media is not straightforward, a simpler system which may act as a starting point of analysis is the transport of fluid through a slit pore. Such a system has been studied previously by Todd *et. al.*. ([35]), Lim *et. al.*. ([36]) and Travis *et. al.*. ([37]), among others. Travis *et. al.*. performed non-equilibrium molecular dynamics (NEMD) simulations of simple Lennard-Jones fluids undergoing planar Poiseuille flow in a slit pore which was only a few molecular diameters in width [37]. They found that Newton's law breaks down for very narrow pores, as the shear viscosity exhibits singularities.

The main part of this project is continuing the work of Galteland et. al., by calculating the pressure in nano-porous media of variable lattice constant, and designing a method to

determine the permeability [1]. The permeability will be determined from the gradient in integral pressure through the system. The integral pressure will be computed locally as described by Galteland *et. al.* [1]. An aim is to determine if the new way to compute the permeability can be allocated to known models, such as the Kozeny-Carman equation. To compare to the Kozeny-Carman equation, the shear viscosities must be calculated.

As a gradual build-up to studying nano-porous media, two simpler systems will be considered as part of the thesis. In this way, the complexity of the systems is gradually increased. The two simpler systems are a slit pore and a box containing liquid that is evaporating. The results of these studies can be compared to data and models found in the literature (see [7] and [2]). For expanding on and extending the work done by Galteland and coworkers, the main tool will be non-equilibrium molecular dynamics (NEMD) simulations [1]. By first studying the two simpler systems, one can validate the system representation in `LAMMPS` and the methods for post-processing the data before moving on to the nano-porous system. This is advantageous, as the new method for calculating the pressure and permeability in nano-porous media is harder to predict.

To summarize, this project will help an ongoing effort to define state variables of confined media, expand the knowledge about driving forces and design a method to determine the permeability. The overall goal of the project is to contribute towards establishing a non-equilibrium thermodynamic theory for the nano-scale.

The report includes relevant theory, given in Section 2, and a description of applied methods in Section 3. Different case studies are studied as part of the project. These are defined in Section 4, and the associated results are presented in Section 5 and discussed in Section 6. Limitations of the work are pointed out in Section 6.5 and suggestions for further work are given in Section 6.6. At last, the main conclusions are given in Section 7.

2 Theory

The following sections will define thermodynamic variables and driving forces for nano-porous media.

2.1 Thermodynamical variables for the REV

When Galteland *et. al.* developed an approach for calculating the pressure inside a nano-porous medium, the representative volume element was central. Thermodynamic extensive variables on the macro-scale could be obtained by integrating over a representative elementary volume (REV) [38]. A representative elementary volume is a volume element with the following properties [1]:

- The size of the REV is large compared to the pore size and small compared to the size of the porous medium.
- The REV contains a statistically representative collection of pores. Galteland *et. al.* argued that the REV in general must be larger as the heterogeneity of the porous medium increases [1].

When all grains are identical spheres positioned on a FCC-lattice, a properly chosen layer covering half the unit cell can be a valid choice of the REV [1]. However, a full unit cell is also a valid choice, which is more general. Then, the two half unit cells do not have to be equal.

The REV can generally contain a set of phases, interfaces and contact lines [1]. The value of each of the REV-variables is obtained as a sum of contributions from each phase, interface and three-phase contact line present [16]. The following notation is used for surface area and line length:

- $\Omega^{\alpha\beta,REV}$: Surface area between phases $\alpha\beta$ in the REV.
- $\Lambda^{\alpha\beta\delta,REV}$: Contact line length between phases $\alpha\beta\delta$ in the REV.

A basis set of macro-scale variables of the REV is made up of $(U^{REV}, S^{REV}, V^{REV}, M^{REV,i})$. Here, U is the internal energy, S is the entropy, V is the volume and M^i is the mass of component i. The basis set applies to the whole REV. A basic assumption is that the REV set of basis variables are Euler homogeneous functions of degree one [16]. This means that a REV of double size has double the energy, entropy, mass, surface areas and line lengths. It also implies that one temperature T, one pressure p and one chemical potential μ_i' per component can be defined for the REV [16]. The intensive variables T, p, and μ_i' are not averages of the corresponding variables on the pore-scale, but are instead derived from the total internal energy [38].

2.2 The pressure in a nano-porous medium

The grand potential of the REV is given as

$$\Upsilon^{\mathrm{REV}} \equiv -k_B T \ln Z_g \equiv -pV^{\mathrm{REV}}, \tag{2.1}$$

where Z_g is the grand partition function and k_B is the Boltzmann constant.

The grand potential of the REV is obtained as a sum of contributions from all phases, surfaces and contact lines contained within the REV according to [16]:

$$\Upsilon^{\mathrm{REV}} = \sum_{\alpha=1}^{m} \Upsilon^{\alpha,\mathrm{REV}} + \sum_{\alpha>\beta=1}^{m} \Upsilon^{\alpha\beta,\mathrm{REV}} + \sum_{\alpha>\beta>\delta=1}^{m} \Upsilon^{\alpha\beta\delta,\mathrm{REV}} \tag{2.2}$$

Since the grand potential of the REV is equal to minus the product sum of pressure and volume, it follows that the pressure of the REV can be expressed as

$$p = \frac{1}{V^{\mathrm{REV}}} \left(\sum_{\alpha=1}^{m} p^{\alpha} V^{\alpha,\mathrm{REV}} - \sum_{\alpha>\beta=1}^{m} \gamma^{\alpha\beta} \Omega^{\alpha\beta,\mathrm{REV}} - \sum_{\alpha>\beta>\delta=1}^{m} \gamma^{\alpha\beta\delta} \Lambda^{\alpha\beta\delta,\mathrm{REV}} \right) \tag{2.3}$$

Superscripts α, β and δ are used for three different phases. V^{REV} is the volume of the REV, p^{α} and $V^{\alpha,\mathrm{REV}}$ are the pressure and volume of the α-phase, $\gamma^{\alpha\beta}$ is the surface tension, $\Omega^{\alpha\beta,\mathrm{REV}}$ is the surface area, $\gamma^{\alpha\beta\delta}$ is the line tension and $\Lambda^{\alpha\beta\gamma,\mathrm{REV}}$ is the line length.

Let us now consider a system which contains a single fluid f in a micro-porous medium r. The product of pressure and volume is given by

$$pV = p^f V^f + p^r V^r - \gamma^{fr} \Omega^{fr} \tag{2.4}$$

where p and V are the pressure and volume of the REV, p^f and V^f are the pressure and volume of the fluid in the REV, p^r and V^r are the pressure and volume in the grains of the REV, and γ^{fr} and Ω^{fr} are the the surface tension and surface area between the fluid and the grain.

For small systems, Hill defined two different pressures, the integral pressure $\hat{p}$ and the differential pressure p. For a system with volume V, the pressures are related by [1]

$$p(V) = \frac{\partial \hat{p}(V) V}{\partial V} = \hat{p}(V) + V \frac{\partial \hat{p}(V)}{\partial V} \tag{2.5}$$

The integral and differential pressures are linked to different types of mechanical work on an ensemble of small systems. The differential pressure times the change in the small system volume is the work done on the surroundings by the volume change [1]. This work is the same for small and large systems. The integral pressure times the volume per replica is the work done by adding one small system of constant volume to the other ones, keeping the temperature constant [1]. This work is special for small systems.

The differential pressure is the variable that we normally understand as pressure on the macroscopic level. It is only when the integral pressure $\hat{p}$ depends on V that the pressures are different. This is not the case for large systems, so the two pressures are equal in the thermodynamic limit [15].

Equation (2.4) is not applicable to nano-porous systems. In such systems, the relation is written in terms of integral pressure instead of differential pressure:

$$\hat{p} V = \hat{p}^f V^f + \hat{p}^r V^r - \hat{\gamma}^{fr} \Omega^{fr} \tag{2.6}$$

where $\hat{p}^f$ and $\hat{p}^r$ are the integral pressures of the sub-volumes V^f and V^r, whereas $\hat{\gamma}^{fr}$ is the integral surface tension. The integral pressure and the integral surface tension normally depend on the system size.

For a nano-porous system consisting of a single spherical grain with radius R confined by a single phase fluid, Galteland *et. al.* found the fluid and grain pressures to be related by [1]

$$p^r - p^f = \frac{2\gamma^{fr}}{R} \tag{2.7}$$

The above is called Young-Laplace's law. The integral and differential pressures of the spherical phase r were related by

$$\hat{p}^r - p^r = \frac{\gamma^{fr}}{R} \tag{2.8}$$

For a face-centered cubic lattice of spherical grains confined by a single phase fluid, the pressures were again found to satisfy Young-Laplace's law, according to [1]

$$p_i^r - p^f = \frac{2\gamma_i^{fr}}{R_i} \tag{2.9}$$

The index i refers to a specific grain in the FCC-lattice.

2.3 Integral pressure from chemical potential

In this section, an alternative approach will be presented for obtaining the integral pressure in a nano-porous medium at equilibrium. The integral pressure will be related to the chemical potential and geometrical properties of the system.

Let us consider a FCC-lattice unit cell with two types of particles, fluid particles denoted f and grain particles denoted r. The Hill-Gibbs equation for a grand canonical ensemble of the FCC-lattice unit cell, filled with fluid f, becomes [23]

$$U_t = TS_t + N'_{f,t}\mu'_f + N'_{r,t}\mu'_r - \hat{p}V\mathcal{N} \tag{2.10}$$

where μ' is the chemical potential and $\mathcal{N}$ is the number of replicas. Introducing the relation between the number of replicas and the mole number N', leads to the Hill-Gibbs-Duhem equation on the form

$$d(\hat{p}V^{REV}) = SdT + pdV^{REV} + N'_f d\mu'_f + N'_r d\mu'_r \tag{2.11}$$

Let us keep T and V^{REV} constant. The volume of the REV is equal to the volume of a unit cell. As a consequence, Equation (2.11) can be simplified to

$$d\hat{p} = \rho_f d\mu'_f + \rho_r d\mu'_r \tag{2.12}$$

In Equation (2.12), two of the variables can be varied freely. When changing the chemical potential of the fluid in the reservoir next to the FCC-lattice, the chemical potential of the rock will change according to

$$\frac{N'_f}{V^{REV}}d\mu'_f + \frac{N'_r}{V^{REV}}d\mu'_r = \frac{N'_f}{V_f}d\mu'_f \tag{2.13}$$

By comparing Equation (2.12) and Equation (2.13), an expression for the integral pressure is obtained:

$$\hat{p} = \int \frac{N'_f}{V_f}d\mu'_f \tag{2.14}$$

Equation (2.14) is equivalent to

$$\hat{p} = \int \frac{N^{f\prime}}{V^{REV}} + \frac{N^{f\prime}(\frac{V^{REV}-V^f}{V^f})}{V^{REV}} d\mu'_f \tag{2.15}$$

Furthermore, Equation (2.12) leads to

$$N'_r d\mu'_r = N'_f(\frac{V_f}{V^{REV}} - 1)d\mu'_f \tag{2.16}$$

and by introducing the porosity Φ, this can be written as

$$\rho_r d\mu'_r = \rho_f \frac{1-\Phi}{\Phi} d\mu'_f \tag{2.17}$$

This gives us an expression for the chemical potential of the fluid particles. Once the chemical potential of fluid and the geometrical properties of the system are known, the integral pressure can be determined from Equation (2.15).

2.4 Driving forces in porous media

Any variation in saturation between the REVs along the x-axis will lead to a driving force [16]. It is difficult to measure the pressure inside the REV. However, the pressure in the fluid phases adjacent to the porous medium can be determined more easily. The total pressure of the REV is given by

$$p = \bar{p} - \bar{p}^c \tag{2.18}$$

where $\bar{p}^c$ is the surface-averaged contributions to the pressure, whereas $\bar{p}$ is the volume-averaged contributions to the pressure from the homogeneous phases. The total pressure difference of the system is then given by

$$\Delta P = \Delta\bar{p} - \Delta\bar{p}^c \tag{2.19}$$

2.5 Constitutative equation for isothermal single fluid

The fundamental basis of non-equilibrium thermodynamics lies in the formulation of entropy production σ' as a product sum of conjugate fluxes J'_i and forces X_i for a thermodynamic system [39],

$$\sigma' = \sum_i J'_i X_i \geq 0 \tag{2.20}$$

The entropy production in a porous medium can be formulated as

$$\sigma' = J_u \frac{\partial}{\partial x}\left(\frac{1}{T}\right) - \sum_{i=1}^{n} J_i \frac{\partial}{\partial x}\left(\frac{\mu'_i}{T}\right) = J'_q \frac{\partial}{\partial x}\left(\frac{1}{T}\right) - \frac{1}{T}\sum_{i=1}^{n} J_i \frac{\partial}{\partial x}\mu'_{i,T} \tag{2.21}$$

J_u is internal energy flux, J_i is mass flux of component i, and J'_q is measurable heat flux. The derivation is given in Appendix A.3.

Close to equilibrium, there is a linear relationship between all fluxes and forces,

$$J_i = \sum_j^n L_{ij} X_j \tag{2.22}$$

where L_{ij} are called phenomenological coefficients and describe the coupling between fluxes and forces. Based on the formulation of the entropy production in Equation (2.21), constitutative equations can be derived [16].

For an isothermal single fluid f flowing inside a porous medium, the entropy production has one term, $-J_V \frac{1}{T}\Delta P$, where J_V is the volume flux. The derivation is given in Appendix A.3.1. It follows from Equation (2.22) that the volume flux can be written as

$$J_V = -L_{pp}\frac{1}{T}\Delta P \tag{2.23}$$

where we call L_{pp} a phenomenological coefficient or transport coefficient. By including the constant temperature in the transport coefficient, we get

$$J_V = -L_p \Delta P \tag{2.24}$$

with the new transport coefficient $L_p = L_{pp}/T$. This coefficient is then the permeability. The permeability is normally a function of the state variables pressure and temperature. In the hydrodynamic regime, it is also a function of viscosity η, so $L_p = L_p(p, T, \eta)$ [16]. Equation (2.24) tells us that to determine the permeability, the volume flux should be measured as a function of the pressure difference.

2.6 Permeability

2.6.1 Darcy's law

Darcy's law describes slow and viscous flow of fluid through a porous medium [40]. It is a proportionality relationship between the instantaneous volume flow rate through a porous medium, the viscosity of the fluid, and the pressure drop over a given distance [18]:

$$Q = -\frac{k}{\mu}\frac{A\Delta P}{L'} \tag{2.25}$$

Q is the volume flow rate in units of volume per time, k is the intrinsic permeability of the porous medium, A is the cross-sectional area of the flow, ΔP is the total pressure drop and L' is the length over which the pressure drop takes place. μ is the viscosity, and it will be described in more detail in Section 2.8. The negative sign is included to account for the flow of fluid from high to low pressure. Dividing both sides of the equation with the cross-sectional area leads to

$$q = \frac{-k}{\mu}\frac{\Delta P}{L'} \tag{2.26}$$

q is the fluid flux in units [m/s], given by

$$q = \frac{\text{Volumetric flow}}{\text{Cross-sectional area}} \tag{2.27}$$

q is related to the fluid velocity v via the porosity Φ:

$$q = v\Phi \tag{2.28}$$

Darcy's law is only valid for slow, viscous flow. The relation can typically be applied successfully to flow with a Reynolds number less than 1, which is classified as laminar flow. The Reynolds number is given by [41]

$$\mathrm{Re}_p = \frac{V_0 D_p \rho}{\mu} \tag{2.29}$$

where ρ is density, D_p is the diameter of the spherical bed particles and V_0 is the superficial velocity.

The assumptions of Darcy's law in porous media are [17]:

- Laminar single-phase flow.
- Isothermal conditions.
- Constant fluid viscosity.
- No rock-fluid interaction.

Tight porous media typically contain a large amount of narrow pores ranging from a few nano-meters to a few micro-meters. In the confined space, traditional Darcy's law is not sufficient for the analysis of fluid transport [42].

2.6.2 Kozeny-Carman equation

The Kozeny-Carman equation relates the permeability of a porous medium to the porosity, through well-defined parameters of the pore structure. More specifically, the permeability is described by an effective hydraulic pore radius, the fluctuation in local hydraulic pore radii, the length of streamlines and the fractional volume conducting flow [12]. Each of these quantities are represented by a parameter. Firstly, the effective hydraulic pore radius is related to a characteristic hydraulic length. Secondly, the fluctuation in local hydraulic radii is related to a constriction factor. Next, the length of streamlines is characterized by a tortuosity. Lastly, the fractional volume conducting flow from inlet to outlet is described by an effective porosity.

The resulting Kozeny-Carman equation relates the permeability k to the porosity through

$$k = c_0 \tau^2 r_h^2 \Phi = c_0 \tau^2 \frac{\Phi^3}{S_0^2} \tag{2.30}$$

c_0 is a coefficient called Kozeny's constant, τ is the tortuosity, $r_h = \Phi / S_0$ is the mean hydraulic radius, and S_0 is the specific surface area with respect to a unit volume of porous medium. The porosity is given by

$$\Phi = \frac{\text{Volume of pores}}{\text{Total volume}} \tag{2.31}$$

The hydraulic radius is chosen as the effective pore radius of the porous medium. Being a purely geometric length, the effect on permeability from pore size variation or connectivity is neglected. This choice of effective pore radius is not unique. Other length scales have in fact been proposed to be more suitable for permeability description [12].

For a porous medium, the tortuosity represents the effect of the flow deviating on a microscopic level from the direction of the applied piezometric head difference [12]. This effect is reflected by the length of the microscopic streamlines, which motivates the following definition of tortuosity: $\tau = \Delta s / s_e$. Here, Δs is the length of the porous medium in the direction of applied piezometric head difference, whereas s_e is the effective streamline length. Tortuosity is described in more detail in Section 2.9.

The streamlines are affected by the shape of the pore channels. Streamlines converge and diverge when pore channels are constricting or expanding [12]. The permeability is

affected as a consequence of the variable fluid velocity along the streamlines, and this effect is represented by the constriction factor.

For Equation (2.30), Carman reported that a value of $c_0\tau^2 = \frac{1}{5}$ fits well with experimental data [25]. Inserting this value and the expression $r_h = \Phi/S_0$ for the mean hydraulic radius, gives

$$k = \frac{\Phi^3}{5S_0^2} \tag{2.32}$$

The equation can alternatively be written as

$$k = \frac{\Phi^3}{5(S')^2(1-\Phi)^2} \tag{2.33}$$

where S' is the specific surface area with respect to a unit volume of solid. Equation (2.32) and Equation (2.33) are equivalent.

The Kozeny-Carman equation is of approximate validity, and should therefore be applied with caution [18]. The formulation of the Kozeny-Carman equation is based on the following assumptions:

- The fluid flow is assumed to be laminar, creeping flow [18].
- The porous medium is treated as a bundle of capillaries of equal length. The velocity components normal to the tubes' axes are neglected. In reality these components will be different from zero because of the converging and diverging nature of the flow in the tubes [25].

For high porosities, the Kozeny-Carman equation will not be the best choice for describing how the permeability depends on porosity, according to Dullien [18]. These systems are better described by what he calls *flow around submerged objects*. Another limitation pointed out by Dullien is that in reality, the permeability also depends on the size distribution and the topographical arrangement of capillaries. This is not considered in the Kozeny-Carman equation, as for a fixed $c_0\tau^2$, the permeability is uniquely defined by Φ and S_0.

2.7 Planar Poiseuille flow

As a simplification of a nano-porous medium, we consider planar Poiseuille flow between two infinitely long parallel plates. This can be used as a model system for the case of flow in a slit pore of variable width. Let the parallel plates have a fixed distance h in between. The system is shown in Figure 2.1. Both plates are at rest and the flow is caused by a pressure gradient dp/dx in the direction x parallel to the plates. The flow is incompressible, laminar and steady-state.

A derivation of the velocity of the flow is given by Brennen [2]. The derivation is based on the following assumptions:

- The only non-zero component of velocity is u_x.
- The velocity and pressure are independent of time (steady-state).
- The velocity $u_x(y)$ is a function only of y.

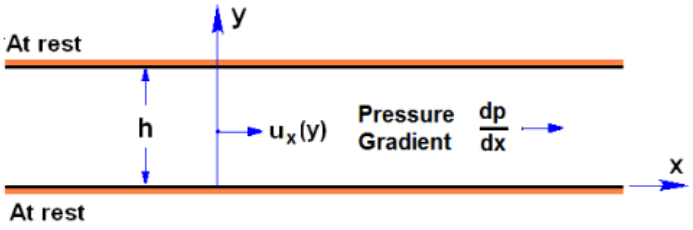


Figure 2.1: Planar Poiseuille flow between two infinitely long plates separated by a distance h [2].

- No-slip conditions apply at the lower and upper walls, which means that the velocity is zero there.

Following Brennen, the velocity in the x-direction for planar Poiseuille flow through a slit pore can be expressed as

$$u_x = \frac{1}{\mu}\left(-\frac{dp}{dx}\right)\frac{y}{2}(h-y) \tag{2.34}$$

where dp/dx is the pressure gradient in the x-direction. The volumetric flow rate Q^* per unit depth normal to the plane of the flow is

$$Q^* = \int_0^h u_x dy = \frac{h^3}{12\mu}\left(-\frac{dp}{dx}\right). \tag{2.35}$$

2.8 Viscosity

The shear viscosity η is a measure of how easily a fluid transmits momentum in a direction perpendicular to the direction of velocity or momentum flow [43]. Viscosity has units of $[Pa \cdot s]$. Let J_m be the momentum flux in units of momentum per area and time. Next, we introduce ∇v_{stream} as the spatial gradient of the fluid velocity normal to the momentum flow. These quantities are then related via the viscosity, according to [43]

$$J_m = -\eta \nabla (v_{\text{stream}}) \tag{2.36}$$

Shear viscosity can be determined by a variety of methods [44]. It can be obtained from equilibrium simulations by considering pressure or momentum fluctuations. It can also be obtained by non-equilibrium methods that make use of steady-state shear. In that case, one has the choice between using periodic shear flow or sliding boundary conditions. The commonly used SLLOD algorithm is an example of the latter [44].

In the SLLOD algorithm, Couette flow is introduced by shearing the top xz-face of the simulation box with a velocity U in the x-direction as shown in Figure 2.2 . The gradient in the velocity in the x-direction is related to the shear stress via

$$\eta \frac{\partial \mathbf{u}_x}{\partial y} = -P_{xy} \tag{2.37}$$

where η is the shear viscosity and $\tau' = -P_{xy}$ is the shear stress. The shear viscosity at equilibrium, η_s, is computed by extrapolating the shear viscosity η to zero shear rate.

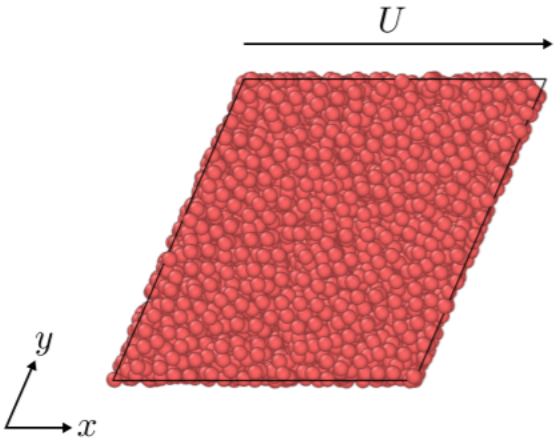


Figure 2.2: In the SLLOD algorithm, one imposes a Couette flow by shearing the top face with a constant velocity U [3]. The red spheres are fluid particles.

An assumption is that the fluid is Newtonian. Furthermore, by applying the SLLOD algorithm, we ensure that the generated heat is removed.

Using the SLLOD algorithm requires a modification of the equation of motion, and it is done in a non-Hamiltonian way [44]. When shearing in the x-direction with the gradient in the y-direction, the SLLOD equations of motion are expressed as

$$\begin{aligned} \frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} &= \mathbf{v}_i + s^* r_{i,z} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\ \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} &= \frac{1}{m_i}\mathbf{f}_i - s^* v_{i,z} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \end{aligned} \tag{2.38}$$

where i is the particle index, and a second index z indicates the vector component. s^* is the shear rate, $\mathbf{r}$ is the position, t is the time, $\mathbf{v}$ is the velocity, m is the mass and $\mathbf{f}$ is the force. The periodic boundary conditions of the system must also be modified, which can be obtained by using a continuously deforming triclinic unit-cell.

The SLLOD approach has some disadvantages, as explained by Hess [44]. The equations of motion produce an overall rotation in the system, and the viscosity does not correspond to the viscosity obtained from an experiment [44]. In the SLLOD algorithm, a linear velocity profile is imposed on the atomic level. However, a linear velocity profile does not necessarily exist in an experiment. Instead, external forces induce a Couette flow on the macroscopic level. As a consequence, the viscosities will in general not be equal for complex liquids. They will however be the same for a simple liquid [44].

2.9 Tortuosity

Tortuosity is one of the parameters that characterize the packing structure [45]. It is defined as the ratio of the average length of the actual fluid flow paths through the packing and the packing height. Carman included tortuosity in his permeability model to account for the sinuosity of the flow paths.

A variety of different values and models for tortuosity have been reported in the literature. Mota et al. reported tortuosity values in the range 1.47–1.53 for packing of spheres [46]. Gunn has suggested tortuosity values of 1.4 for spheres, 1.93 for cylinders and 1.8 for hollow cylinders [47]. Others reported values that vary in the range of 1.25 to 1.79 [25].

A correlation that is applicable to packed beds is $\tau = \frac{\pi}{2}$ [48] [49]. The correlation suggests that the tortuosity does not depend on the size of the packing particles [45]. One should be aware that using this correlation does not necessarily give a precise description of the tortuosity. Sobieski and Lipiński argued that the different mathematical formulas found for tortuosity in the literature give different results for the same data, and that it is impossible to indicate which equation should be used for a specific case [49].

3 Computational methods

3.1 General information

For this thesis, results have been generated using Non-equilibrium molecular dynamics simulations in `LAMMPS`. Molecular dynamics is a commonly used technique for simulating the dynamics of chemical systems [50].

Calculations were performed partly on the supercomputer Sauron and partly on the supercomputer Saga [51]. Saga was provided by UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway, project number NN9229K.

The data has been post-processed using scripts in `MATLAB` [52] and `Python` [53]. `OVITO` has allowed visualisation of the systems. It is a software for visualization of scientific data and for analyzing molecular and other particle-based simulation models [54].

In this section, the use of computational chemistry, and more specifically non-equilibrium molecular dynamics simulations, will be motivated. The section will also contain description of central terms, and calculation procedures related to the molecular dynamics simulations.

3.2 Computational chemistry

Modeling is a tool to obtain fundamental understanding, and is often an important substitute of experiments. Experiments are often expensive and time-consuming, and they may involve hazardous substances. Modeling in the form of computational chemistry can be used to investigate the origins of chemical phenomena and examine molecular properties [55]. It can show how central variables depend on the physical environment, for example temperature, pressure, concentration and the presence of solvent. It is also useful for developing ideas that can help improve certain processes. Despite the many advantages, computational chemistry also has its drawbacks. The systems and timescales that can be studied are small, and the simulations represent a virtual reality.

3.3 Molecular dynamics

Molecular dynamics is a versatile tool for simulating the dynamics of chemical systems [50]. The purpose is to mimic the true physical movements of atoms and molecules. The movements are determined by numerically solving the Newton's equations of motion [50]. Small time steps (typically 1 fs) and a time-step integrator algorithm are used. The forces between the particles and the potential energy are defined by molecular mechanics force fields. In particular, the forces are determined by taking the gradient of a classical force field.

The aim of molecular dynamics is to:

- Obtain thermodynamic properties correctly.
- Get short time dynamics correct.
- Get long time dynamics correct on average.

In molecular dynamics, positions and velocities are updated in each time step according to an algorithm. En example of an algorithm is the Euler integrator, given by [56]:

1. Initialize positions $\mathbf{R}(t = 0)$ and velocities $\mathbf{V}$(t=0).

   Start iterations

2. Get forces $\mathbf{F} = -\nabla V'(\mathbf{R})$, where V' is the potential energy.
3. Update positions and velocities.

$$\mathbf{R}(t + \Delta t) = \mathbf{R}(t) + \Delta t \mathbf{V}(t) \tag{3.1a}$$

$$\mathbf{V}(t + \Delta t) = \mathbf{V}(t) + \Delta t \frac{\mathbf{F}(t)}{m} \tag{3.1b}$$

4. $t = t + \Delta t$

   End iterations

A problem with the Euler integrator is that it is unstable [56]. In general, good algorithms must be time-reversible and area-preserving (symplectic) [57]. The Verlet, Velocity Verlet and Leap-Frog algorithms are examples of integrators that have these properties [57] [58]. To check whether an operator is area-preserving, one considers the Jacobian determinant. In 2D, for an operation T: $\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix}$ the Jacobian matrix is defined as [59]:

$$\mathbf{J} = \begin{pmatrix} \frac{dx'}{dx} & \frac{dx'}{dy} \\ \frac{dy'}{dx} & \frac{dy'}{dy} \end{pmatrix} \tag{3.2}$$

If the absolute value of the Jacobian determinant is 1, the transformation is area-preserving. Furthermore, an algorithm is time-reversible if one obtains $(\mathbf{R}_n, \mathbf{V}_n)$ when using $(\mathbf{R}_{n+1}, \mathbf{V}_{n+1})$ as initial conditions and moving backwards by one time step.

3.4 Non-equilibrium molecular dynamics

Equilibrium molecular dynamics has been generalized to simulate non-equilibrium systems [60]. This is obtained by adding sources of thermodynamic heat and work. The generalization is based on microscopic mechanical definitions of macroscopic thermodynamic and hydrodynamic variables such as temperature and stress.

Like equilibrium molecular dynamics, non-equilibrium molecular dynamics (NEMD) is based on time-reversible equations of motion [61]. NEMD provides a consistent microscopic basis for the irreversible macroscopic second law of thermodynamics.

3.5 Thermostats

In molecular dynamics, ensemble averages are often calculated, which are thermodynamic averages. Possible ensembles are [62]:

- NVT (Canonical ensemble): The number of particles, volume and temperature are kept constant.
- NVE (Microcanonical ensemble): The number of particles, volume and energy are kept constant.
- NPT: The number of particles, pressure and temperature are kept constant.
- $\mu' VT$ (Gibbs ensemble or Grand canonical ensemble): The chemical potential, volume and temperature are kept constant.

In molecular dynamics simulations, one studies the evolution of a given set of particles which move in a volume V according to Newton's laws [63]. This represents a microcanonical (NVE) ensemble, as the energy is conserved. On the other hand, to mimic experiments, one must often perform simulations at constant temperature instead. This corresponds to a canonical (NVT) ensemble. The temperature control is achieved by coupling the system to a thermostat. The thermostat acts as a thermal bath, which scales the velocities so that the system approaches T_{bath}. Typically, velocities are rescaled based on random procedures (e.g. Berendsen, Bussi-Parinello thermostat), or deterministically (e.g. Nose-Hoover thermostat). Another approach is to randomly regenerate velocities (e.g. Andersen thermostat).

Not all numerical methods developed to simulate equilibrium systems can be successfully adapted to out-of-equilibrium systems. This is indeed true for thermostats, and a thermostat can in fact induce artificial effects on a system. Thermostats often display good agreement with rheology experiments, but the performance rapidly degrades beyond weak dissipation and small shear rates [63]. Ruiz-Franco *et.al* showed that choosing the right thermostat and parameters requires careful evaluation of temperature, density and velocity profiles [63]. They studied a Lennard-Jones fluid under steady shear flow and considered three different thermostats: Langevin, Dissipative particle dynamics (DPD) and Bussi-Donadio-Parrinello (BDP) thermostats. The group studied a wide choice of parameters, and found that a poor choice of the thermostat parameters can negatively affect the dynamic response of the system under shear, and thereby give a physical behaviour which is far from reality [63]. During the last years, many different thermostats have been developed, with the purpose of reducing side effects due to coupling and reproduce the phenomena observed in experiments more accurately. Many thermostats that exhibit a good temperature control do not correctly reproduce hydrodynamics. Reproduction of hydrodynamics require local momentum conservation and Galilean invariance [63].

3.5.1 Langevin thermostat

The Langevin thermostat gives ergodicity in all possible cases [63]. Dissipative and noise forces are added to the Hamiltonian to include the effective behaviour of the solvent [63]. The equations of motion for the Langevin dynamics are [63]

$$m_i \dot{\mathbf{r}}_i = \mathbf{v}_i, \quad m_i \ddot{\mathbf{r}}_i = \sum_{j(\neq i)} \mathbf{F}_{ij}^C + \mathbf{F}_i^R + \mathbf{F}_i^D \tag{3.3}$$

The first term, $\mathbf{F}_{ij}^C$, is the conservative pairwise force, the second term, $\mathbf{F}_i^R$, is the random force due to the thermal motion of the bath particles, and the last term, $\mathbf{F}_i^D$, is the dissipative (drag) force. A problem with the Langevin thermostat is that it does not

reproduce hydrodynamics. This is due to the fact that it is not Galilean invariant, and does not locally conserve momentum [63].

3.5.2 Dissipative particle dynamics

The dissipative particle dynamics (DPD) thermostat represents a modification of the Langevin thermostat, and is better at reproducing hydrodynamic effects [63]. DPD satisfies the fluctuation-dissipation theorem [63]. The friction and noise terms are pairwise and act over all pairs of neighbouring particles, i.e. at the local level. DPD locally conserves momentum as all forces act between pairs of particles. The Galilean invariance is also conserved, as the drag force acts on the relative velocity.

The thermostat has three parameters which determines its transport properties: the exponent s', the cut-off r_c' and the friction constant ξ [63]. The combined effect of the three DPD-thermostat parameters is to tune the viscosity of the system in a finer way with respect to Langevin dynamics. The value of r_c strongly affects the computational cost. This is because it controls the number of pairs of particles that enter into the thermostatting procedure.

Rovigatti *et. al.* concluded that the DPD thermostat was the best in terms of stability, realism and computational efficiency [63]. They found that the common choice of setting $s' = 0.5$ or 1 gave good results overall. Choosing an optimal value of r_c is however not straightforward. There is not a physical motivation to choose a certain value, and the optimal value is in general independent of the other parameters. Making a good choice requires an understanding of the system that is studied. Too large values of the cut-off radius might give unphysical behaviours such as anomalies and inhomogeneities in the velocity and density profiles. The group argued that the cut-off radius should be between the first maximum and the first minimum of the radial distribution function, and that a value smaller than but close to the first minimum of the radial distribution function of the system, is a good starting point [63].

3.5.3 Nose-Hoover thermostat

The Nose-Hoover thermostat is the simplest time-reversible scheme that gives ergodicity for the one-dimensional harmonic oscillator [64]. It is applicable to both equilibrium and non-equilibrium many-body simulations.

The Nose-Hoover thermostat introduces a fictitious dynamical variable, which has the physical meaning of a friction. The friction slows down or accelerates particles until the temperature has reached the desired value [65].

The Nose-Hoover thermostat is used for the molecular dynamics simulations in this project. It is chosen over the Langevin thermostat due to the problems with the Langevin thermostat that are pointed out in section 3.5.1. It is chosen over the Dissipative particle dynamics thermostat due to its complexity, and the challenges related to determining suitable parameter values.

3.6 LAMMPS

`LAMMPS` is a code for molecular dynamics with a focus on modeling materials [66]. It is an acronym for *Large-scale Atomic/Molecular Massively Parallel Simulator*. `LAMMPS` can

be used for modeling solid-state materials (eg. metals and semiconductors), soft matter (eg. biomolecules and polymers) and coarse-grained or mesoscopic systems.

There are several advantages to `LAMMPS`, some being that it [67]:

- Is versatile.
- Has good parallel performance.
- Is easy to extend.
- Is well documented.
- Has an active and supportive user community.

3.6.1 Pressure calculation in LAMMPS

In molecular dynamics simulations, the system is given by a set of particles with a given mass. Each particle is further characterized by its position and velocity. In the systems considered for this thesis, the flow is driven by a pressure gradient. The calculation of pressure is therefore central. The pressure tensor of the system, P_{IJ}, consists of a kinetic contribution, which is the first term in Equation (3.4) and a virial contribution, which is the second term in Equation (3.4). The kinetic energy term is related to temperature, whereas the virial consists of contributions from interatomic interactions [68]. The virial is in general computed for all pairwise, 2-body, 3-body, 4-body, many-body, and long-range interactions [68].

$$P_{IJ} = \frac{\sum_k^N m_k v_{k_I} v_{k_J}}{V} + \frac{\sum_k^{N'} r_{k_I} f_{k_J}}{V} \tag{3.4}$$

In the equation above, m_k is the mass of particle k. v_{kI} is the velocity of particle k in direction I, and similarly v_{kJ} is the velocity of particle k in direction J. r_i and f_i are the position and force vectors for particle i.

The procedure for calculating pressure in `LAMMPS` starts by dividing the system into chunks. Each atom in the system is assigned to a single chunk, based on the spatial bin that the atom belongs to. For each of the chunks, one can calculate the kinetic and virial contributions separately, and then find the pressure as sum of the contributions. The kinetic contribution of the pressure tensor is defined as

$$P_{IJ}^{kin} V = \sum_k^N m_k v_{kI} v_{kJ} \tag{3.5}$$

where the sum is taken over all particles in a specific chunk in direction I and J. N is the number of atoms in the chunk.

Now assume that we have a system with a mass flux present. The kinetic contribution is associated with random fluctuations in velocity, but when there is a mass flux present, the velocities in the flow direction may dominate. Therefore, one should correct for the center of mass velocity for each chunk. This is obtained by subtracting the time-averaged center of mass velocity from the kinetic contribution.

With the correction of the center of mass velocity, $v_{c,I}$, in the particular direction, the kinetic contribution can be written as

$$P_{II}^{kin}V = \sum_{k}^{N} m_k(v_{kI} - v_{c,I})(v_{kI} - v_{c,I}) \tag{3.6}$$

which can be rearranged to

$$P_{II}^{kin}V = \sum_{k}^{N} v_{kI}^2 - 2v_{c,I}\sum_{k}^{N} v_{kI} + Nv_{c,I}^2 \tag{3.7}$$

3.6.2 *Method of planes* in LAMMPS

Todd *et. al.* developed a general statistical mechanical technique for calculating the pressure tensor of an atomic fluid [35]. It is a simple and efficient technique that is based on the continuity equations of hydrodynamics [35]. The derivation avoids the mathematically awkward Taylor series expansion which is used in the Irving-Kirkwood derivation. The group applied the method to the case of planar Poiseuille flow through a narrow slit pore, and found the technique to be efficient. Todd *et. al.* arrived at a *method of planes* (MOP) for calculating the pressure tensor. It was shown to be more accurate and computationally efficient than the Irving-Kirkwood method, for calculating variations of the pressure tensor across a slit pore [35].

In `LAMMPS` the *method of planes* can be achieved by using `compute stress/mop` command [69]. The kinetic and configurational part of the stress tensor are given by Equation (21) and Equation (16) in [35] [69].

3.6.3 Viscosity calculation in LAMMPS

The viscosity can be calculated by performing a non-equilibrium molecular dynamics (NEMD) simulation by shearing the simulation box via the `fix deform` command and using the `fix nvt/sllod` command to thermostat the fluid via the SLLOD equations of motion (see Section 2.8) [43].

For calculating the viscosity, Equation (2.36) and Equation (2.37) are relevant. The velocity profile setup in the fluid can be monitored by the `fix ave/chunk` command, which determines ∇v_{stream}. By comparing Equation (2.36) and Equation (2.37), it is seen that the momentum flux J_m is equal to the off-diagonal component of the pressure tensor, which can be calculated by the `compute pressure` command.

3.6.4 Velocity calculation in LAMMPS

For calculating a 2D velocity profile in `LAMMPS`, the system is first divided into chunks in 2D. The velocity profile setup in the fluid is monitored by the `fix ave/chunk` command.

3.7 Mass flux and mass current

The mass flux and mass current are determined from velocity values in the direction of flow. They are calculated for each chunk along the flow direction. Let there be a flow of

fluid in the z-direction. The mass flux J in a chunk can be calculated by

$$J = \frac{m \cdot N \cdot \overline{v_z}}{V} = \frac{m \sum_{i=k}^{N} v_{z,i}}{V} \tag{3.8}$$

where $v_{z,i}$ is the z-component of velocity for particle i, and $\overline{v_z}$ is the average z-component of velocity. The mass current in a chunk can be calculated by

$$J_c = \frac{m \sum_{i=k}^{N} v_{z,i}}{dj} \tag{3.9}$$

where dj is the length of the chunk.

3.8 Lennard-Jones/spline potential

The Lennard-Jones potential describes the potential energy of the interaction between two non-bonding atoms or molecules based on the distance between them [70]. The potential energy accounts for the attractive and repulsive forces. The Lennard-Jones potential is a simple model capable of describing many real systems [7]. When used in computer simulations, the intermolecular potential is truncated, typically at a distance of 2.5-5 molecular diameters. Although the truncation has little effect on the system's structure, to fully represent the Lennard-Jones system's thermodynamical properties, the contributions from the long-range tail must be included by using tail corrections [7]. Furthermore, some properties, such as surface tension, are especially sensitive to truncation and shifting of the potential.

A convenient alternative to the Lennard-Jones potential, is the Lennard-Jones/spline potential. It is a Lennard-Jones potential that is truncated in a unique way, so that both the pair potential and the force continuously approach zero at r_c [7]. In that way, one avoids the need for further specification and risk of ambiguity in how the potential is used in simulation. The pair potential of the Lennard-Jones/spline model is [7]

$$u(r) = \begin{cases} 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] & \text{for } r < r_s \\ a^* \left(r - r_c\right)^2 + b \left(r - r_c\right)^3 & \text{for } r_s < r < r_c \\ 0 & \text{for } r > r_c \end{cases} \tag{3.10}$$

ϵ and σ are the usual Lennard-Jones parameters, whereas r_s is the inflection point of the Lennard-Jones potential [7]. a^*, b and r_c are determined such that the potential and its derivative are continuous at r_s and r_c. This means that the force is zero at r_c and that the delta-function contribution to the force in the Lennard-Jones potential at the cut-off is avoided. As the Lennard-Jones/spline model has short range, it gives shorter simulation times when compared to the Lennard-Jones model [7]. Although the Lennard-Jones/spline model has essentially the same structural features as the Lennard-Jones-model, the thermodynamical properties are different due to the shorter range of the potential [7].

3.9 Periodic boundary conditions

Studying bulk properties in molecular dynamics simulations require the use of periodic boundary conditions. The principle is to replicate all atoms in the computational cell throughout the space to form an infinite lattice [4].

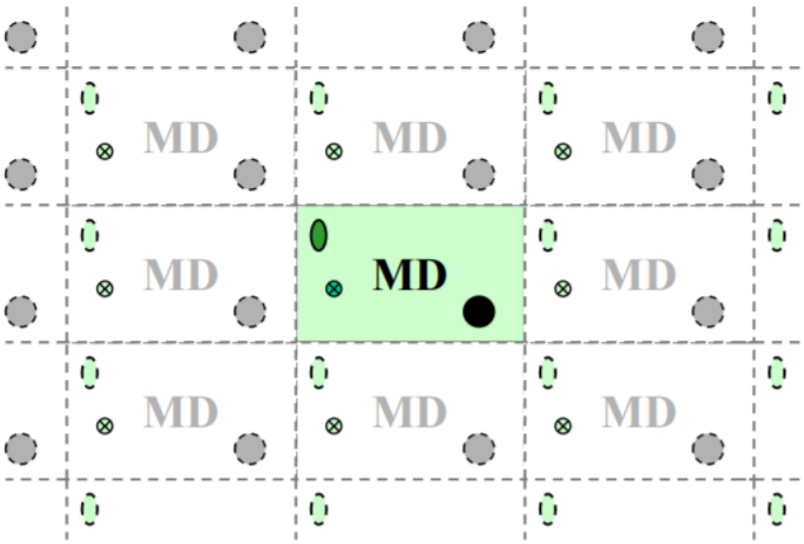


Figure 3.1: Illustration of the concept of periodic boundary conditions [4]. A computational cell (in green) is replicated to form an infinite lattice. The particles in the computational cell do not only interact with each other, but also with their periodic replicas.

Let the atoms in the computational cell have positions $\overrightarrow{r_i}$. Then the periodic boundary conditions produce mirror images of the atoms at positions defined by

$$\overrightarrow{r_i^{\text{image}}} = \overrightarrow{r_i} + l^* \overrightarrow{a} + m^* \overrightarrow{b} + n^* \overrightarrow{c} \tag{3.11}$$

$\overrightarrow{a}$, $\overrightarrow{b}$ and $\overrightarrow{c}$ are vectors that correspond to the edges of the box, whereas l^*, m^* and n^* are any integers from $-\infty$ to ∞.

Using periodic boundary conditions imply that particles can interact across the boundary [71]. Each particle in the computational cell is in fact interacting with their images in the adjacent boxes, as indicated by Figure 3.1 [4]. Periodic boundary conditions also imply that atoms can exit one end of the box and re-enter at the other end [71].

3.10 Reflective particle method

Systems where there is flow induced by a pressure difference is studied as part of the thesis. A method is therefore needed to generate a pressure difference over a system. Li *et. al.* proposed a method to induce Poiseuille flow in a molecular dynamics simulation by introducing a partially reflective membrane [5]. The reflecting particle method (RPM) drives the fluid flow while conserving particle number and total energy.

Figure 3.2 shows the principle of the reflective particle method [5]. One introduces a fictitious membrane, which acts as a filter. If a fluid atom crosses $x = 0$ from left to right ($x = Lx^-$ to $x = 0^+$) , it can pass freely. In other words, there is a 100% chance of the particle passing through. However, if the particle crosses from right to left, there is a probability p of the particle being reflected and probability $1 - p$ of the particle passing through. Apart from this, the membrane has no other effect on the atoms [5]. In other words, atoms interact with each other across the membrane in the same manner as

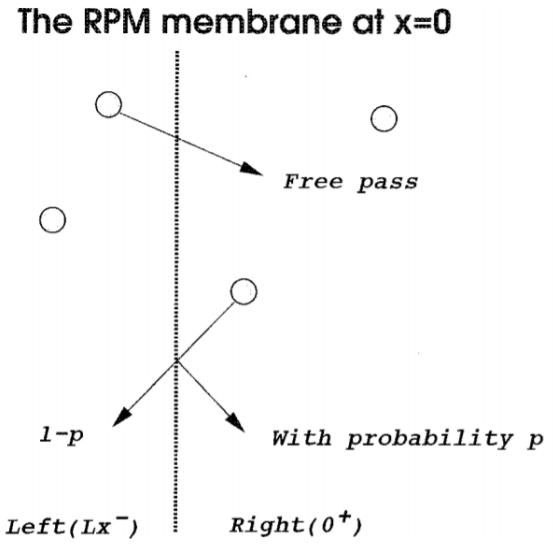


Figure 3.2: The reflective particle method [5].

anywhere else in the fluid. The number of particles and the total energy of the system are conserved quantities, due to the fact that no particles or energy are injected into the system. This means that steady state flow is possible. In contrast to what may happen for methods where random particles are inserted, the particles will not be found in energetically unfavorable positions [5].

3.11 Reduced units

The idea of dimensionless reduced units, is expressing all quantities in terms of the parameters ϵ, m and σ that are associated with a molecular interaction potential [10]. σ is a size parameter, ϵ is an energy parameter of the potential, and m is the mass of particle. The Lennard-Jones/spline potential is an example of a molecular interaction potential, and it is a convenient choice for molecular dynamics simulations. The parameter values for Argon-like particles are given in Table 3.2.

There are several advantages associated with using reduced units. The equations are simplified, it is easy to scale the results, and the computed values are closer to unity. Reduced units are often abbreviated and indicated by an asterisk, as shown in the table below.

Property	Symbol	Reduced form
Length	r^*	r/σ
Time	t^*	$t\sqrt{\epsilon/(m\sigma^2)}$
Temperature	T^*	k_BT/ϵ
Force	F^*	$F\sigma/\epsilon$
Internal energy	U^*	U/ϵ
Pressure	P^*	$p\sigma^3/\epsilon$
Density	ρ^*	$\rho\sigma^3$
Surface tension	γ^*	$\gamma\sigma^2/\epsilon$

Table 3.1: Dimensionless reduced units [8] [9].

Physical quantity	Symbol	Value for Argon
Length	σ	$3\,4 \cdot 10^{10}$m
Energy	ϵ	$1\,65 \cdot 10^{21}$J
Mass	m	$6\,69 \cdot 10^{26}$kg

Table 3.2: Parameter values for Argon particles which are used in the definition of dimensionless reduced units [10].

3.12 Simulation error

There are two types of errors that can occur in simulations, systematic errors and statistical errors. Possible sources of systematic error are imperfectness of force field, the approximations that are applied, invalid assumptions, and numeric precision. On the other hand, statistical errors are due to the fact that the simulations are finite and that the data is often correlated [72].

Systematic errors can only be known if accurate experimental data or results from a more accurate method are available and if the statistical errors are lower then the systematic ones. Statistical errors can be computed using the *block averaging method*. The procedure starts by dividing the data into a number of subgroups of a length such that one expects the sub-averages to be randomly distributed [72]. Then the error in the mean of the complete series can be estimated from the standard deviation of the mean of those sub-averages. A block averaging procedure is used as part of the thesis, for determining uncertainty in mass flux, mass current, pressure and density.

4 Systems and case studies

4.1 Overview

Several systems have been considered as part of the project. The system complexity has been gradually increased, based on the argument that there is no point in looking at a complex system if one cannot successfully apply the relevant tools to a simple system. The goal was to arrive at a representation of a porous medium in terms of a lattice of grain particles with fluid flowing through, and design a method to determine the permeability from the gradient in integral pressure.

The first system considered was a box containing evaporating liquid. The purpose of this case study was to get familiar with using `LAMMPS`, and testing that the post-processing modules were working and producing reasonable results. There existed available data that could act as a reference for checking the validity of the post-processing scripts (see [7]). As a next step, a slit pore was considered, where a pressure gradient causes flow of fluid through the system. This system is closer to the desired porous media system, in the sense that there is a fluid flow caused by an applied pressure difference. However, the slit pore does not contain any grain particles, and is therefore not a porous medium. For the slit pore case, the results in terms of volumetric flow and velocity profiles were compared to the solutions for planar Poiseuille flow.

Finally, the actual representation of the porous media was considered. The porous medium was represented as a lattice of spherical grains, as in the work of Galteland *et. al.* [1]. Different geometries were considered for the porous media, in terms of the lattice constant. The lattice constant dictates the distance between the surfaces of two grains, and is one of the characteristic sizes of porous media.

An overview of the cases that have been considered follows:

- *Two phase box*
- *Slit pore*
  - Slit pore base case.
  - Slit pore with variable pressure difference and slit pore width.
  - Viscosity from Non-Equilibrium Molecular Dynamics (NEMD).
  - *Method of planes* (MOP) for slit pore.
- *Nano-porous media*
  - Porous medium base case with lattice constant $a = 20$.
  - Different approaches for calculating integral pressure.
  - Porous medium with lattice constant $a = 25$.
  - Porous medium with lattice constant $a = 30$.
  - Porous medium with lattice constant $a = 40$.

The different cases will be described in detail in the following sections.

4.2 Slit pore

Consider a slit pore of solid walls with fluid passing through. The slit pore has a width h in the y-direction and a length L in the z-direction, which is the direction of flow. The system is of infinite length in the x-direction. The system is studied with molecular dynamics simulations in `LAMMPS`. We define a simulation box that is a cuboid of dimensions L_x, L_y and L_z, which satisfy $L_x = L_y < L_z$. The simulation box is shown in Figure 4.1. To represent that the system is of infinite length in the x-direction, periodic boundary conditions are used. The system contains fluid particles and solid particles, that are placed on FCC-lattices. There are two bulk fluid phases, one to the left and one to the right of the slit pore. The bulk phases span the entire xy-plane.

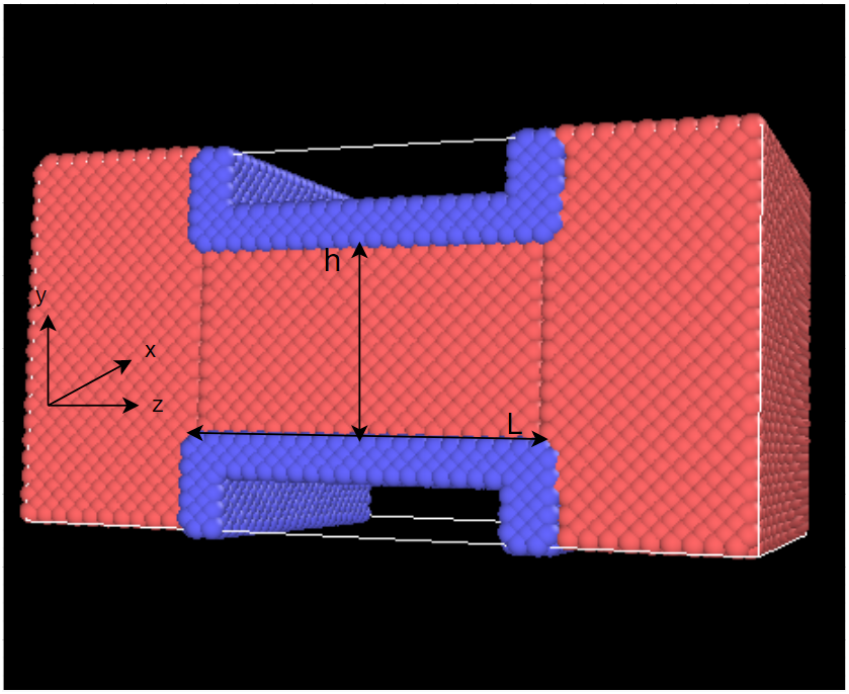


Figure 4.1: Slit pore of width h in the y-direction and length L in the z-direction. The system is of infinite length in the x-direction. The red spheres are fluid particles, whereas the blue spheres are solid particles that make up the walls of the slit pore. To the left and to the right of the slit pore, there are bulk fluid phases.

A pressure difference is created over the slit pore by applying the reflective particle method. The reflective boundary probabilities are chosen in such a way that the pressure in the left bulk phase is higher than the pressure in the right bulk phase. The pressure gradient leads to a flow of fluid through the slit pore from left to right.

The interactions between the particles are modeled using the Lennard-Jones/spline potential. Parameters are needed for the interaction between fluid/fluid, solid/solid and solid/fluid. For the calculation of different properties, the simulation box is divided into chunks in the direction of flow, which is the z-direction.

4.2.1 Slit pore, base case

As a reference for the slit pore, a base case is simulated, with the parameter values listed below. Reduced units are used (see Section 3.11). The lengths given correspond to the

number of unit cells in a given direction. The length of each unit cell is 1.74.

- Fluid density: $D = 0.75$.
- Width of slit pore: $h = 10$.
- Left reflective boundary probability: 0.05.
- Right reflective boundary probability: 0.01.
- Length of simulation box in z-direction: $L_z = 40$.
- Length of simulation box in positive x- and y-direction: $W = 10$.
- Density of solid: $D_w = 1.05$.
- Number of chunks in the z-direction: 40.
- Number of unit cells in each chunk in the z-direction: 1.
- Reduced temperature T^*=0.90.
- Initial velocity of fluid particles: 0.90.
- Lennard-Jones/spline variables of fluid-fluid: $\epsilon_{11} = 1$, $\sigma_{11} = 1$ and $\alpha_{11} = 1$.
- Lennard-Jones/spline variables of fluid-solid: $\epsilon_{12} = 1$, $\sigma_{12} = 1$ and $\alpha_{12} = 1$.
- Lennard-Jones/spline variables of solid-solid: $\epsilon_{22} = 1$, $\sigma_{22} = 1$ and $\alpha_{22} = 1$.
- Mass of fluid particles: 1.
- Mass of solid particles: 1.
- Number of time steps for equilibration: 500 000.
- Number of time steps after equilibration: 2 000 000.
- Time step size: 0.002.

Isothermal conditions are chosen to purely investigate the pressure as the driving force, and thereby exclude possible transport due to gradients in temperature.

4.2.2 Slit pore with variable pressure difference and slit pore width

When determining permeability, the relation between volumetric flow and pressure difference must be known. We want to investigate how this relation is affected by the slit pore width. A set of simulations with different slit widths and pressure differences is therefore carried out. The pressure difference is calculated as the difference between the average pressure of the left bulk phase and the average pressure of the right bulk phase. Initially, when using different slit pore widths and the same value for other parameters, it was found that not only did the pressure difference vary, but also the average pressure of the left and right bulk phases. This was a problem, as in order to compare the different slit pore widths, they should have the same average pressure. Therefore, new simulations were carried out, where particles were deleted in a trial-and-error manner, until approximately the same average pressure was obtained.

The following parameter values are used:

- Slit pore width: $h = 2(4.69), 4(7.81), 6(12.49), 8(15.62), 10(18.74), 14(24.99)$ and $16(28.11)$. The notation $c_1(c_2)$ means a number c_1 of unit cells set for the molecular dynamics simulation and a value of c_2 being the actual width of the slit pore. The actual width is here taken as the distance in lattice units from the center of the outer wall particle of the top wall to the center of the outer wall particle at the bottom wall.
- Average pressure of the left and right bulk phases: $P_{avg} = 0.7$.
- Left reflective boundary probabilities: 0.05, 0.06, 0.08 and 0.10. By varying this probability, different pressure differences over the slit pore are created.
- Right reflective boundary probability: 0.01.
- Number of time steps for equilibration: 500 000.
- Number of time steps after equilibration: 2 000 000 or 4 000 000, depending on the slit pore width.

The parameters that are not listed have the same values as those used for the slit pore base case described in the previous section.

4.2.3 Viscosity from NEMD

It is desired to compare the results of volumetric flow vs. ΔP with the solution for planar Poiseuille flow. In order to do so, viscosity must be calculated. Viscosity is dependent on pressure, and the viscosity is calculated at the pressure P_{avg}. The shear viscosity is plotted as a function of shear rate, which should be a constant function for Newtonian fluids. The curve is extrapolated to zero shear rate to obtain the shear viscosity at equilibrium.

4.2.4 Pressure calculation with the *method of planes*

Up to this point, the pressure difference has been calculated as the difference between the average pressure of the left bulk phase and the average pressure of the right bulk phase. However, this is not an unique way of determining the pressure difference within the slit pore. In this section a different approach will be defined and tested. We remember that the pressure tensor consists of a kinetic contribution and a configurational contribution. Here, we calculate the kinetic contribution as before, using the division into chunks and correcting for the center of mass velocity in each chunk. Before, the configurational contribution was also calculated as a sum of the contributions from each chunk. Now, the configurational contribution is instead calculated using the *method of planes* (MOP), as described in Section 3.6.2. Each chunk is divided into 10 planes. Since the length of each chunk is equal to the lattice spacing, this means that the thickness of each plane is 1/10 of the lattice spacing. The pressure difference is calculated by multiplying the gradient in pressure within the slit pore with the length of the slit pore.

4.3 Nano-porous media

4.3.1 System description

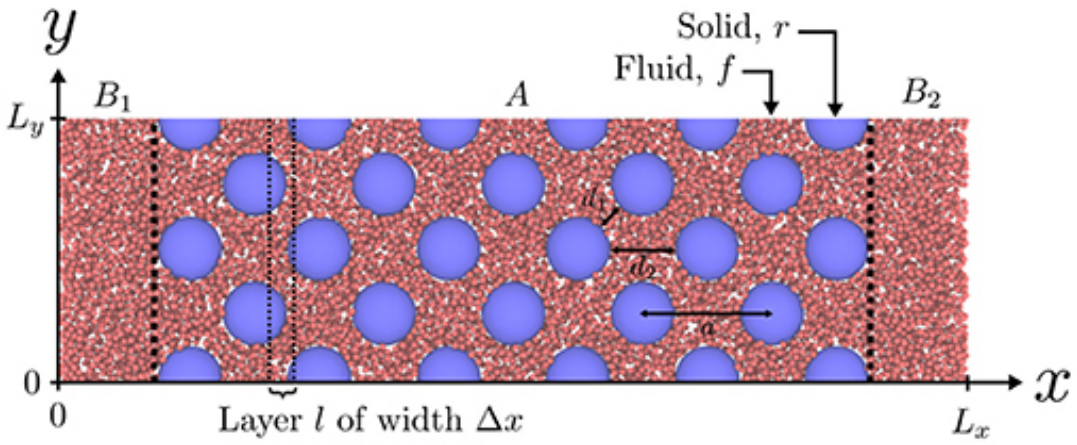


Figure 4.2: A slice of the simulation box for the NEMD-simulations of nano-porous media [1]. The blue particles are grain particles, whereas the red particles are fluid particles. Region A consists of a FCC-lattice of spherical grain particles with fluid in between, whereas regions B_1 and B_2 contain only fluid particles.

We now move on to representing a porous medium. The representation of the porous medium, introduced by Galteland *et. al.*, contains identical spherical grain particles placed on a FCC-lattice [1]. Figure 4.2 gives an illustration of the system. The pores are filled with fluid, represented with the red particles in the figure. At non-equilibrium, the fluid flows through the porous medium, from left to right, as a result of an applied pressure difference.

Some important characteristics of the porous medium are the shortest distances between the surfaces of neighbouring solid spheres, given by d_1 and d_2 in the figure. For a lattice constant a, these are given by

$$d_1 = \frac{1}{2}(\sqrt{2}a - 4R) \tag{4.1}$$

and

$$d_2 = a - 2R \tag{4.2}$$

where R is the radius of the solid spheres.

Properties of the system can be obtained using molecular dynamics simulations in `LAMMPS`. For that purpose, we use a 3D-simulation box with lengths L_x, L_y and L_z, that satisfies $L_x > L_y = L_z$. Simulations at equilibrium, as well as out of equilibrium, are carried out. For the equilibrium simulations, periodic boundary conditions are used. For non-equilibrium simulations, reflecting particle boundaries are used to generate a pressure difference along the x-direction. Particles moving from right to left pass the periodic boundary at $x = 0$ and $x = L_x$ with a probability (1-α_p) and get reflected with a probability α_p. Particles moving from left to right pass freely through the boundary. A large α_p gives a high pressure difference.

The particles present interact with the Lennard-Jones/spline potential. Each particle type has a hard core diameter R_{ii} and a soft core diameter σ_{ii}. The fluid particles have hard and soft core diameters R_{ff} and σ_{ff}, whereas the solid particles have hard and soft core diameters R_{rr} and σ_{rr}. The hard core and soft core diameters for the fluid-grain

pairs are given by the Lorentz mixing rule [1];

$$R_{fr} = \frac{1}{2}(R_{ff} + R_{rr}) \tag{4.3}$$

and

$$\sigma_{fr} = \frac{1}{2}(\sigma_{ff} + \sigma_{rr}) \tag{4.4}$$

A central part of the calculations, is the definition of a representative elementary volume, REV. We let the REV be a unit cell. This is a proper choice as all REVs, except for those at the boundaries, are then identical. Each REV is made up of layers (bins) that we call l (see Figure 4.2). These layers are thinner than the REV itself. The definition of the REV implies that $p^{REV} = p$. At equilibrium, the pressure of the REVs in the bulk liquid phases is constant and equal to the fluid pressure p^f. As the system is at equilibrium, p^{REV} must be the same for all REVs in the system. We are interested in the pressure in the porous region A, and start by looking at the compressional energy pV^{REV}. This is given as a sum of all contributions from the layers that make up the REV, according to [1]

$$pV^{\mathrm{REV}} = \sum_{l \in \mathrm{REV}} \hat{p}_l V_l = \sum_{l \in \mathrm{REV}} p_l^f V_l^f + \hat{p}^r \sum_{l \in \mathrm{REV}} V_l^r - \gamma^{fr} \sum_{l \in \mathrm{REV}} \Omega_l^{fr} \tag{4.5}$$

The first term is the contribution from the fluid f, the second term is related to the solid particles r and the last term is a contribution from the interface between solid and fluid.

First, the terms related to the geometry are determined, that is the volume of fluid V_l^f, the volume of rock particles V_l^r and the surface area Ω_l^{fr}. The fluid pressure is also obtained as part of the NEMD simulation in `LAMMPS`. Next, the values of $\hat{p}^r$ and γ^{fr} are fitted so that $p^{REV} = p$ is everywhere the same, as it should be at equilibrium. All terms are then known, and one can calculate $\hat{p}_l V_l$ of each of the layers l using

$$\hat{p}_l V_l = p_l^f V_l^f + \hat{p}^r V_l^r - \gamma^{fr} \Omega_l^{fr} \tag{4.6}$$

We call pV^{REV} the compressional energy of the REV. In later sections, the compressional energy is shown in different stages, similar to what was done by Galteland *et. al.* [1]. The stages are:

1. Bulk fluid contribution: $p_l^f V_l^f$.
2. Bulk fluid and grain contribution: $p_l^f V_l^f + \hat{p}^r V_l^r$.
3. Total compressional energy: $p_l^f V_l^f + \hat{p}^r V_l^r - \gamma^{fr} \Omega^{fr}$.

Molecular dynamics simulations are carried out in the canonical ensemble using the Nose-Hover thermostat. The overall procedure for the porous medium is as follows:

1. Along the x-axis, the simulation box is divided into n rectangular bins of size Δx, L_y, L_z where $\Delta x = L_x/n$. The volume of each bin is $V_l = \Delta x L_y L_z$. For each bin, calculate the volume of fluid, volume of rock, total volume and the surface area of solid particles.
2. Carry out simulations at equilibrium (zero pressure gradient). For each equilibrium simulation, determine the surface tension γ^{fr} and the integral rock pressure $\hat{p}^r$ by fitting such that the compressional energy is the same in each REV.

3. Obtain γ^{fr} and $\hat{p}^r$ as functions of fluid pressure from the data points for different equilibrium simulations. The different fluid pressures are generated by running equilibrium simulations at different fluid densities.
4. Simulate the non-equilibrium case (with pressure gradient present). Use the information of $\gamma^{fr}(p^f)$ and $\hat{p}^r(p^f)$ that was obtained from the equilibrium simulations. To compute a REV variable away from equilibrium, compute the average over the layers that make up the REV. Moving one layer down the gradient, the procedure is repeated.
5. Compute and plot different contributions to the compressional energy for the non-equilibrium case.

4.3.2 NEMD simulations

The procedure for the molecular dynamics simulations for the porous medium is specified in an *input file* and a *restart file*. The following is a guide to the specifications of these files, and the non-equilibrium simulation is chosen as example:

Input file

1. Define a FCC-lattice of points in space. It consists of a unit cell of basis atoms that is replicated infinitely many times.
2. Define parameters for the geometry of the 3D simulation box consisting of fluid and solid particles.
3. Define densities of solid and fluid.
4. Define probabilites related to the right and left reflective particle boundaries. This will dictate the pressure gradient in the system.
5. Define Lennard-Jones/spline variables for:
   - Fluid-fluid: ϵ_{11}, σ_{11} and α_{11}.
   - Fluid-solid: ϵ_{12}, σ_{12} and α_{12}.
   - Solid-solid: ϵ_{22}, σ_{22} and α_{22}.
6. Define and create regions for the simulation box, solid walls and fluid.
7. Create solid atoms with specified density in the solid region. Equivalently, create fluid atoms with specified density in the fluid region.
8. Assign the fluid and solid atoms to two separate groups, one group for fluid and one group for solid.
9. Set the mass of fluid and solid atoms.
10. Define the Lennard-Jones/spline potentials between the atoms.
11. Delete solid and fluid atoms that overlap, because this would give an infinite potential.
12. Create a neighbour list and select how it should be updated.

13. The solid particles are assumed to be stationary, ie. do not update their positions or velocities during the NEMD simulations.
14. Set the initial velocity of the atoms equal to the temperature.
15. Select the desired output and the frequency of the logging.
16. Choose to keep the number of particles, volume and temperature constant (NVT) and use the Nose-Hoover thermostat.
17. Set the time step size and the number of time steps to run the simulation for.

    Restart file

18. Divide the box into bins (chunks) in the x-direction.
19. Use reflective membrane boundaries. Set probabilities of particles being reflected at the right and left edge of the simulation box.
20. Calculate the temperature of the fluid based on the total kinetic energy.
21. Compute the per-atom stress tensor for each fluid atom.
22. Compute the virial, which is the configurational part of the pressure. It contains the interactions between atoms.
23. Compute square velocities.
24. Compute the x-, y- and z- components of the center of mass velocity for each chunk. This is done by `LAMMPS` by summing $mass \cdot velocity$ for each atom in the chunk and dividing the sum by the total mass of the chunk.
25. Compute the 2D-velocity profile, by creating chunks in two directions.
26. Choose to keep the number of particles, volume and temperature constant (NVT) and use the Nose-Hoover thermostat.
27. Choose the number and size of time steps to use for the simulation.

4.3.3 Porous medium base case with lattice constant a=20

A case study with a radius of $R = 5.0$ and a lattice constant of $a = 20$ is now investigated. This is taken as the base case for the porous medium simulations. The parameter values that are used, given in reduced units, are:

- Reduced temperature: $T^* = 2.0$.
- Lennard-Jones parameters:
  - Fluid-fluid interactions: $\epsilon_{11} = 1.0$, $\sigma_{11} = 1.0$, $\alpha_{11} = 1.0$ and $R_{11} = 0$.
  - Fluid-solid interactions: $\epsilon_{12} = 1.0$, $\sigma_{12} = 5.5$, $\alpha_{12} = 1.0$ and $R_{12} = 4.5$.
  - Solid-solid interactions: $\epsilon_{22} = 1.0$, $\sigma_{22} =$, $\alpha_{22} = 1.0$ and $R_{22} = 9.0$.
- Radius of solid particles: $R = 5.0$.
- Lattice constant: $a = 20$.

- Dimensions of simulation box: $140 \cdot 20 \cdot 20$ (each value is given as a number of unit cells).
- Fluid density: 0.5.
- Density of solid: $4/a^3$.
- Length of chunks in the x-direction: 1 unit cell.
- Initial velocity of fluid particles: 2.0.
- Mass of fluid particles: 1.
- Mass of solid particles: 1.
- Number of time steps for equilibration: 500 000.
- Number of time steps after equilibration: 1 000 000.
- Time step size: 0.002.

For non-equilibrium simulation:

- Left reflective boundary probability: 1.

4.3.4 Different approaches for calculating integral pressure

A comparison will be made between the integral pressure obtained for the base case using the approach described by Galteland *et. al.* [1], and the integral pressure obtained by integrating Hill-Gibbs-Duhems equation (see Section 2.3). The results for the latter are obtained by Varughese [6].

4.3.5 Porous medium with lattice constant a=25

Next, a case with lattice constant $a = 25$ is investigated. The dimensions of the simulation box are $175 \cdot 25 \cdot 25$ and the number of time steps after equilibration is 4 000 000. The other parameter values are the same as those given in Section 4.3.3 for the base case.

4.3.6 Porous medium with lattice constant a=30

A case with a lattice constant of $a = 30$ is also simulated. The dimensions of the simulation box are $210 \cdot 30 \cdot 30$ and the number of time steps after equilibration is 4 000 000. Again, the other parameter values are the same as those given in Section 4.3.3 for the base case.

4.3.7 Porous medium with lattice constant a=40

Lastly, a porous medium with a lattice constant of $a = 40$ is studied. The dimensions of the simulation box are $280 \cdot 40 \cdot 40$ and the number of time steps after equilibration is 4 000 000. Again, the other parameter values are the same as those given in Section 4.3.3 for the base case.

4.4 Two phase box

Consider a box which is a cuboid with dimensions L_x, L_y and L_z. The dimensions satisfy $L_x = L_y < L_z$. The box contains liquid which is evaporating at a constant temperature. The time-evolution of the system can be studied with molecular dynamics simulations in `LAMMPS`. A simulation box must then be defined, and here we let the simulation box coincide with the cuboid with dimensions L_x, L_y and L_z. We initialize the system by letting half of the simulation box be filled with liquid with a given density at a specified temperature. The particles are given an initial velocity equal to the initial temperature in order to first melt the crystal. After equilibration, the desired temperature is set, and the molecular dynamics simulations are run for 1 000 000 time steps. We use an ensemble with constant volume, number of particles and temperature. The particle interactions are represented with a Lennard-Jones/spline potential.

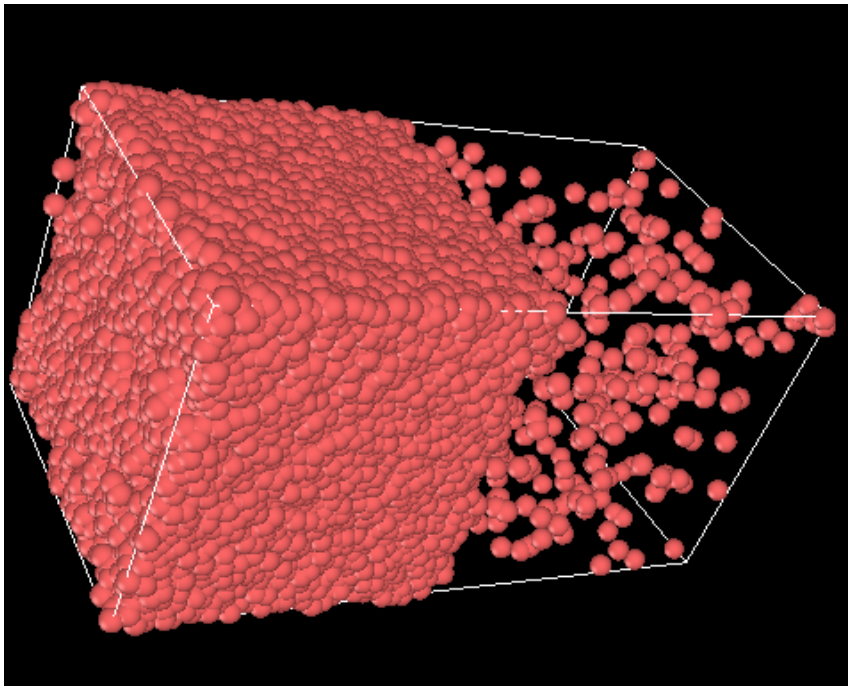


Figure 4.3: Evaporation box with dimensions L_x, L_y and L_z, which satisfy $L_x = L_y < L_z$. The red spheres are fluid particles. The region to the left is liquid, whereas the region to the right is vapor.

The procedure is repeated with different temperatures. For each temperature, the pressure, density of gas and density of liquid are calculated. The results can then be compared to the data for coexisting gas and liquid provided by Hafskjold et. al [7].

The parameter values that are used in the simulations are listed below. Reduced units are used (see Section 3.11). The lengths given correspond to the number of unit cells in a given direction.

- $L_x = L_y = 20$ and $L_z = 40$.
- Lattice density: 0.75.
- Mass of particles: 1.
- Coefficients for the Lennard-Jones/spline potential: ϵ_{11}=1, σ_{11}=1, $\alpha_{11} = 1$, where

the index 1 refers to the fluid particles.

- Time step size: 0.002.
- Number of time steps for equilibration: 500 000.
- Temperature for equilibration: 0.675.
- Temperatures for evaporation: 0.55, 0.57, 0.60, 0.62, 0.64, 0.65, 0.67, 0.70, 0.72, 0.75, 0.77, 0.80, 0.82 and 0.85.

5 Results

5.1 Slit pore

5.1.1 Slit pore base case

We want to compare the results for the slit pore case with the equations for planar Poiseuille flow, and for that purpose we should make sure that we first understand the results for a reference case. The reference is here taken as the base case defined in Section 4.2.1 and the results for the base case are shown in this section. First, the mass current and mass flux are shown in Figure 5.1 and Figure 5.2.

The presence of a mass flux is caused by the gradient in pressure, and the fluid pressure is therefore plotted in Figure 5.3. The fluid pressure has a kinetic contribution and a configurational contribution, and the latter is plotted in Figure 5.4. Furthermore, Figure 5.5 shows the fluid density. Lastly, the velocity is shown in Figure 5.6 and Figure 5.7. The first figure shows a 2D-velocity profile, whereas the last figure shows the velocity in z-direction along the y-axis in the middle of the slit pore.

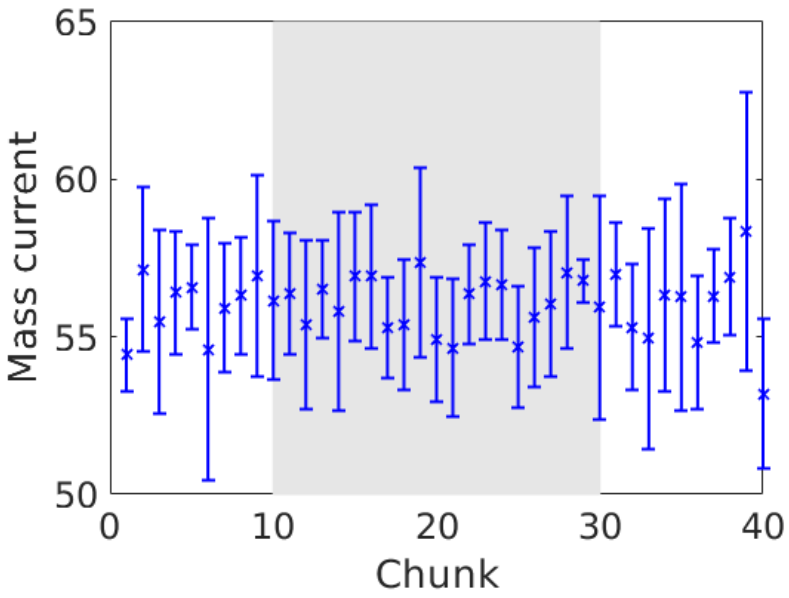


Figure 5.1: Mass current plotted for the 40 chunks in z-direction for the slit pore base case defined in Section 4.2.1. The grey region represents the actual slit pore, i. e. the narrow region created by solid wall particles. Reduced units are used (see Section 3.11).

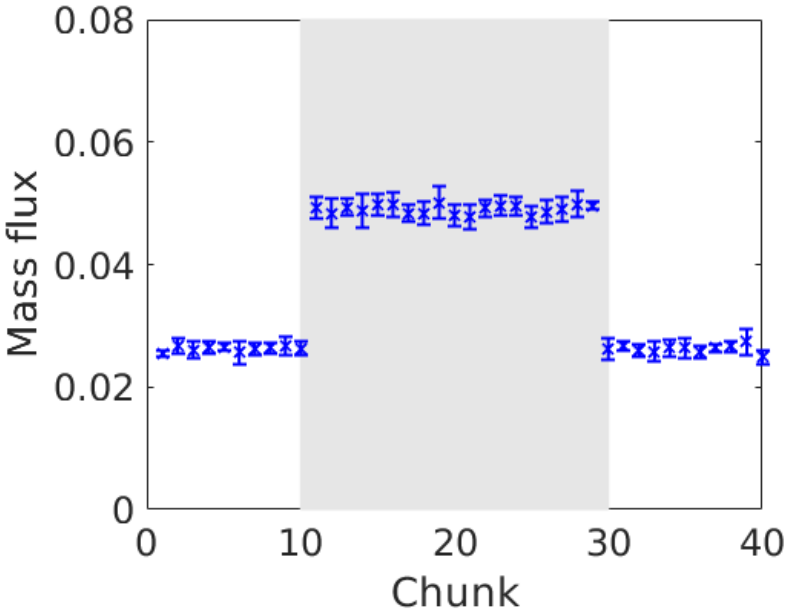


Figure 5.2: Mass flux plotted for the 40 chunks in z-direction for the slit pore base case defined in Section 4.2.1. The grey region represents the actual slit pore, i. e. the narrow region created by solid wall particles. Reduced units are used (see Section 3.11).

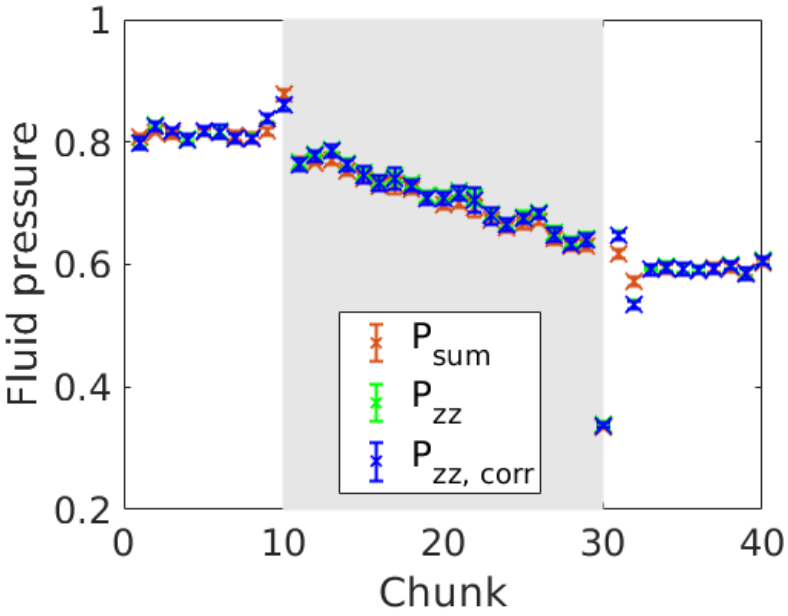


Figure 5.3: Fluid pressure corrected for the center of mass velocity is plotted for the 40 chunks in z-direction for the slit pore base case defined in Section 4.2.1. The z-component of the stress tensor is plotted with and without the correction for center of mass velocity. The pressure is the overall stress tensor. The grey region represents the actual slit pore, i. e. the narrow region created by solid wall particles. Reduced units are used (see Section 3.11).

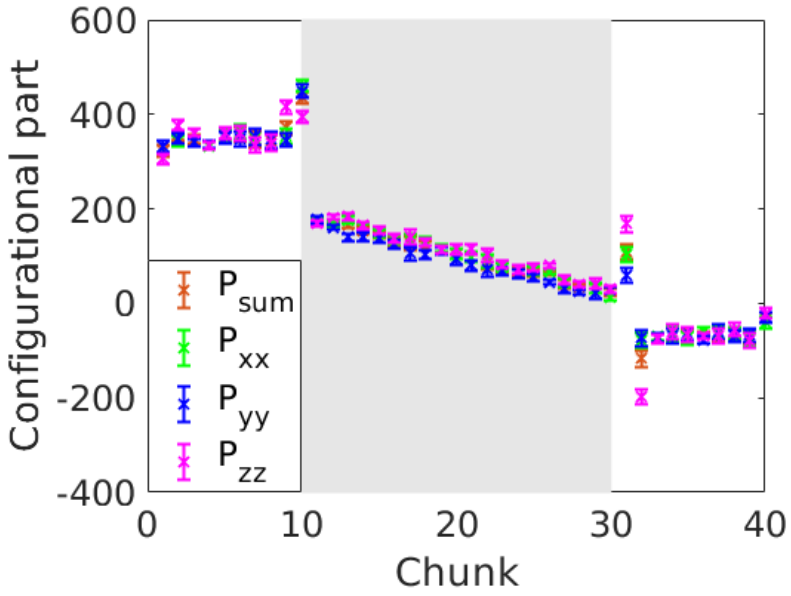


Figure 5.4: The configurational part of the pressure of the 40 chunks in z-direction for the slit pore base case defined in Section 4.2.1. The configurational components of the stress tensor in x-, y- and z-direction are plotted. The grey region represents the actual slit pore, i. e. the narrow region created by solid wall particles. Reduced units are used (see Section 3.11).

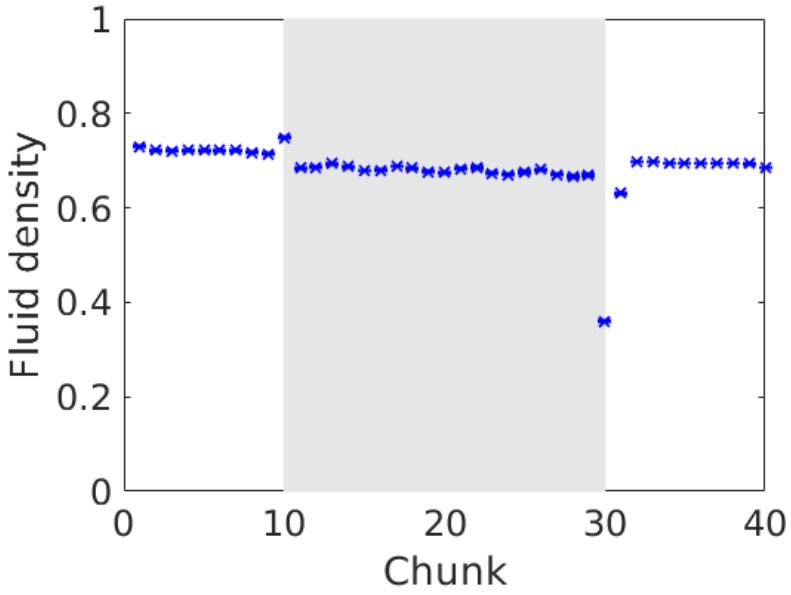


Figure 5.5: Fluid density is plotted for the 40 different chunks in z-direction for the slit pore base case defined in Section 4.2.1. The grey region represents the actual slit pore, i. e. the narrow region created by solid wall particles. Reduced units are used (see Section 3.11).

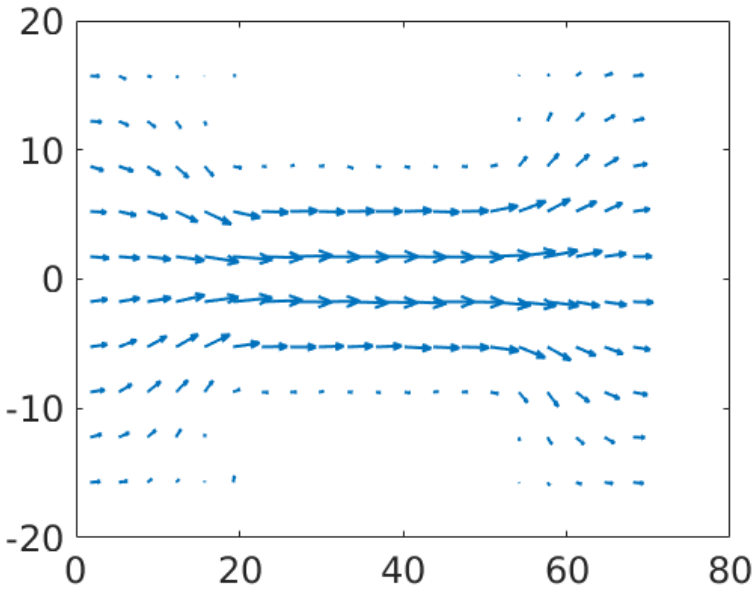


Figure 5.6: A 2D-velocity profile in the yz-plane for the slit pore base case defined in Section 4.2.1. Reduced units are used (see Section 3.11).

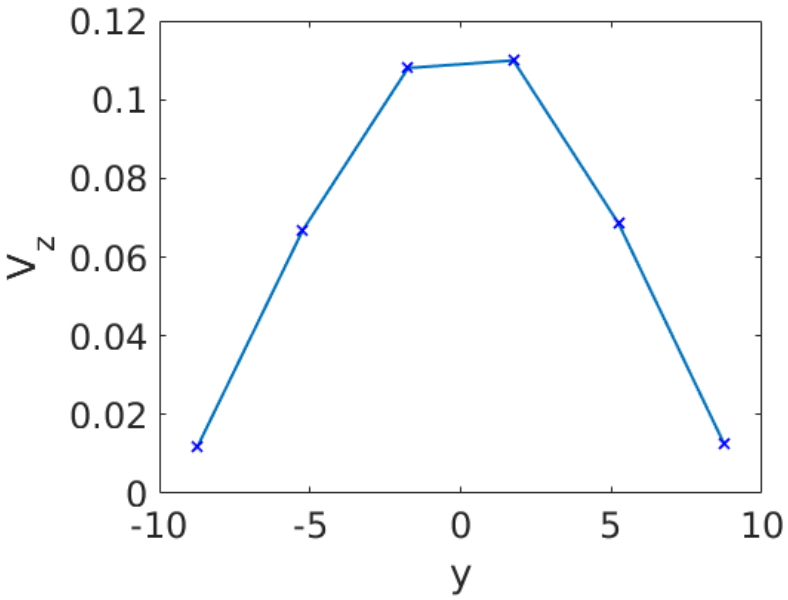


Figure 5.7: Fluid velocity in the z-direction as a function of the y-coordinate in the middle of the slit pore for the base case defined in Section 4.2.1. Reduced units are used (see Section 3.11).

5.1.2 Slit pore with variable pressure difference and width

For planar Poiseuille flow, Equation (2.35) relates the volumetric flow rate to the pressure difference. We want to see if the slit pore results can be understood in comparison to this equation. This means that we should generate different pressure differences over the slit pore. Equation (2.35) also includes the slit width h, so it is also relevant to vary this parameter, and see if it affects how well the results agree.

In Figure 5.8, the volumetric flow per unit depth normal to the plane of flow is plotted against pressure difference for seven different values of slit pore width. The slit pore widths are measured from the center of the outer wall particle at the upper wall to the center of the outer wall particle at the lower wall. The NEMD data is indicated with crosses in the figure and the linear fit is extrapolated to zero pressure difference. The corresponding relations of volumetric flow vs. ΔP from the equation for planar Poiseuille flow are also indicated. The slopes of the lines correspond to the permeability divided by viscosity, denoted *permeability/viscosity* in this report. The determination of viscosity is needed to obtain the solution for planar Poiseuille flow, so we will look at the viscosity in more detail in Section 5.1.3.

As the slopes of the curves for volumetric flow vs. ΔP correspond to *permeability/viscosity*, it is interesting to investigate the slopes further. The quantity is plotted in Figure 5.9, where the NEMD data are given as crosses, and the planar Poiseuille flow data are given as circles. The relative percentage deviation from the corresponding planar Poiseuille flow data is indicated in Figure 5.10.

To investigate how the assumptions of planar Poiseuille flow are met for different slit pore widths, a comparison is made between slit pore widths $h = 28.11$ and $h = 12.49$. Figure 5.11, Figure 5.12 and Figure 8.13 show the effect of slit pore width in terms of velocity and density.

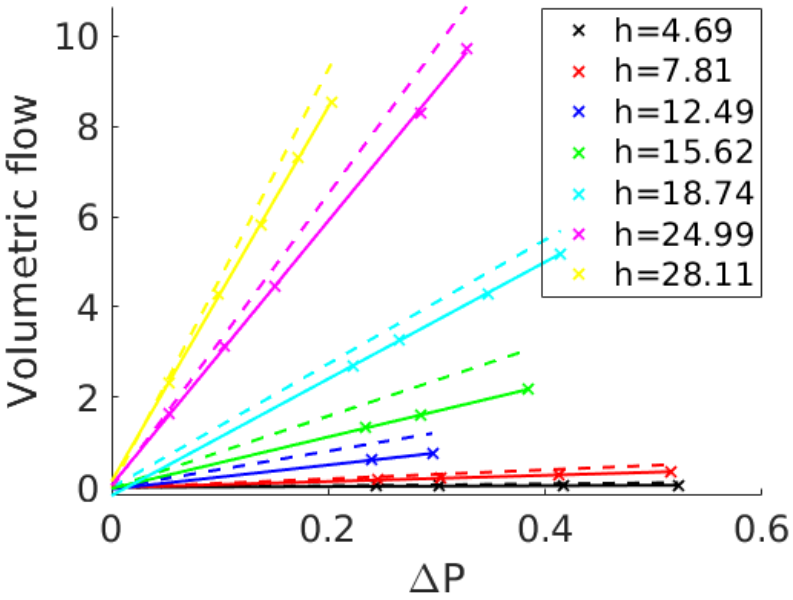


Figure 5.8: Volumetric flow plotted against pressure difference for seven different values of slit pore width h. The crosses represent data from NEMD simulations. For each slit pore width, a linear fit is made and the line is extrapolated to zero. The dotted lines with corresponding colors represent the solutions for planar Poiseuille flow. Reduced units are used (see Section 3.11).

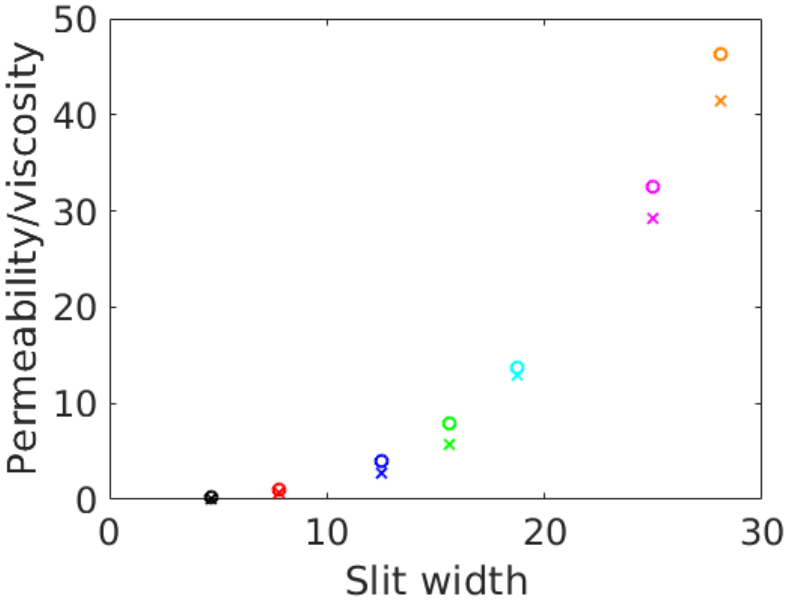


Figure 5.9: Permeability divided by viscosity, which corresponds to the slope of volumetric flow vs. ΔP for the different lines in Figure 5.8, is plotted. The data points from the NEMD simulations are given as crosses and the data points for the planar Poiseuille flow are given as circles with the corresponding colors.

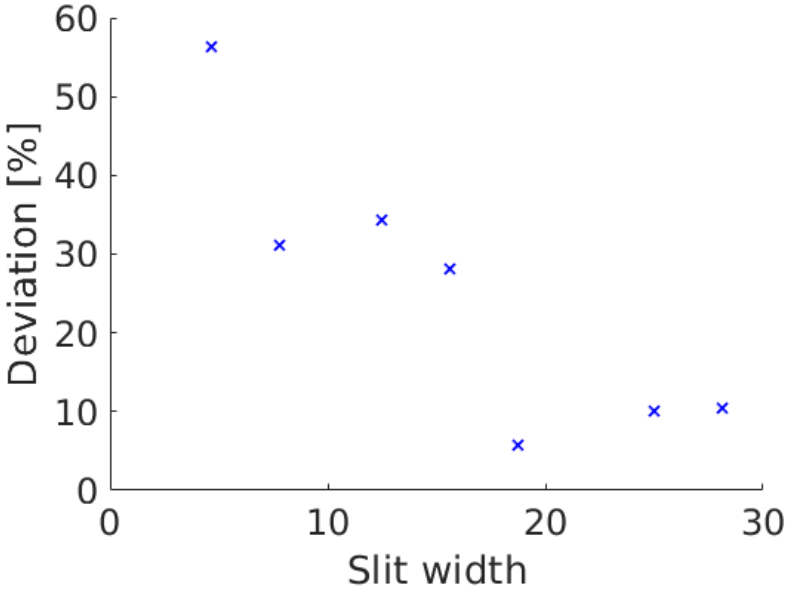
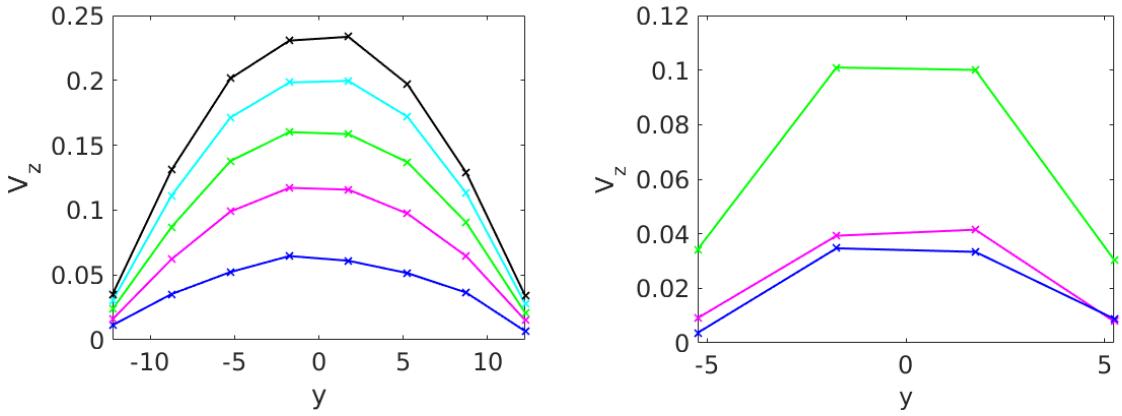


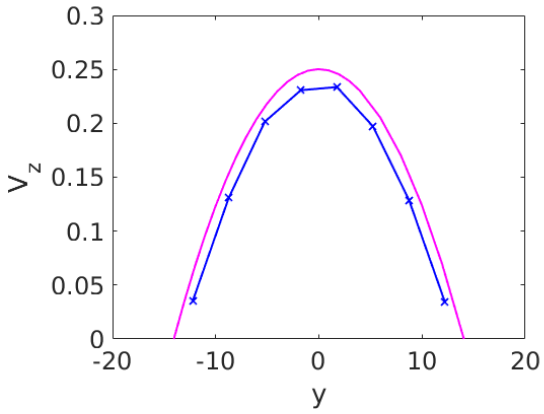
Figure 5.10: Percentage relative deviation of *permeability/viscosity* for the NEMD data from the corresponding slope for planar Poiseuille flow in Figure 5.9.



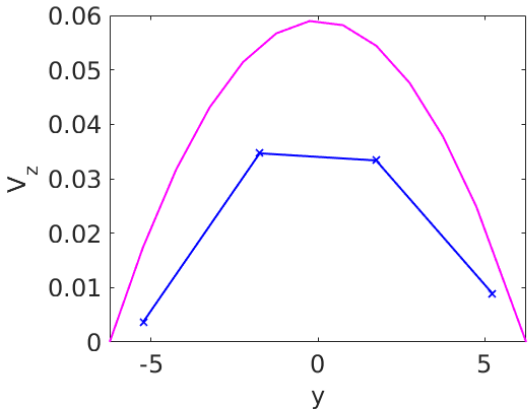
(a) The slit pore width is $h = 28.11$.

(b) The slit pore width is $h = 12.49$.

Figure 5.11: Fluid velocity in the z-direction plotted as a function of the y-coordinate in the middle of the slit pore. The different profiles in the same figure are obtained for different pressure differences over the slit pore. The pressure difference increases from the bottom to the top curve. Reduced units are used (see Section 3.11).

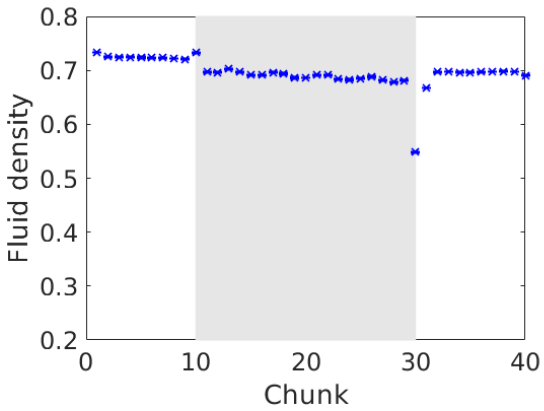


(a) The slit pore width is $h = 28.11$.

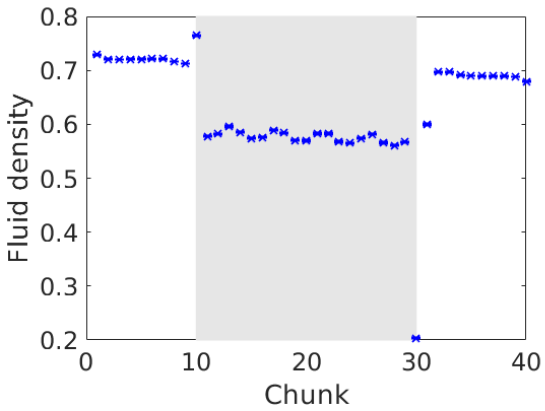


(b) The slit pore width is $h = 12.49$.

Figure 5.12: Fluid velocity in the z-direction plotted as a function of the y-coordinate in the middle of the slit pore. The solution for planar Poiseuille flow is included in pink for comparison. Reduced units are used (see Section 3.11).



(a) The slit pore width is $h = 28.11$.



(b) The slit pore width is $h = 12.49$.

Figure 5.13: Fluid density plotted for the 40 chunks in z-direction. The grey region represents the actual slit pore, i. e. the narrow region created by solid wall particles. Reduced units are used (see Section 3.11).

5.1.3 Viscosity from NEMD

For obtaining the solutions for planar Poiseuille flow in Figure 5.8, the viscosity μ has to be known. In Figure 5.14, the shear viscosity is plotted as a function of shear rate, and the data is extrapolated to zero shear rate to find the shear viscosity at equilibrium. Viscosity is dependent on temperature and pressure [73]. The density is chosen so that it results in a pressure equal to the average pressure of the left and right bulk phases. A viscosity value of $\mu = 1.14$ is obtained. The sensitivity of the solution for planar Poiseuille flow with respect to the viscosity is investigated with Figure 5.15. In that figure, the permeability divided by viscosity is plotted as a function of slit pore width for planar Poiseuille flow using seven different values of viscosity. The viscosity values range from $\mu = 0.94$ to $\mu = 1.54$.

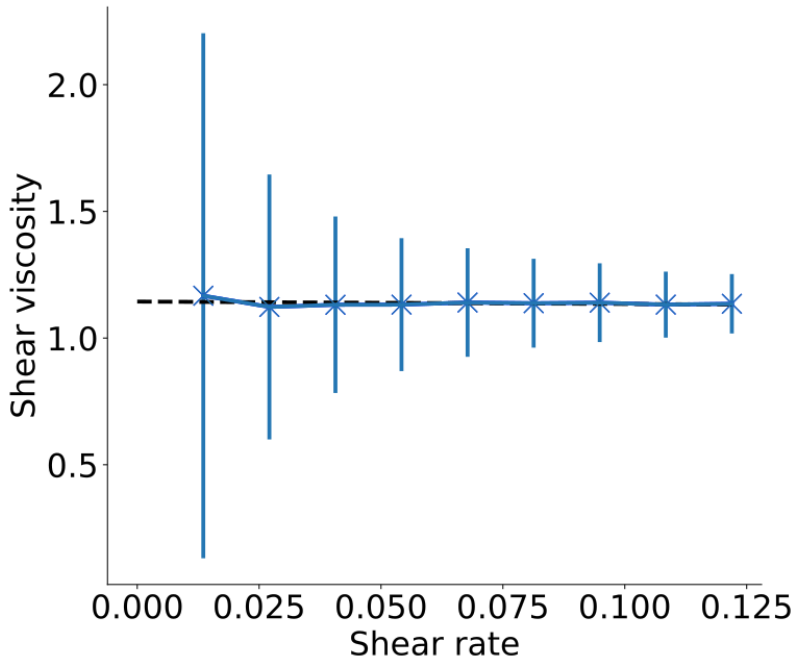


Figure 5.14: Shear viscosity plotted against shear rate. The density is chosen so that it results in a pressure equal to P_{avg}, which is the average pressure of the left and right bulk phases. The data is extrapolated to zero shear rate to find the shear viscosity at equilibrium. The uncertainty in each data point is indicated by the vertical lines. Reduced units are used (see Section 3.11).

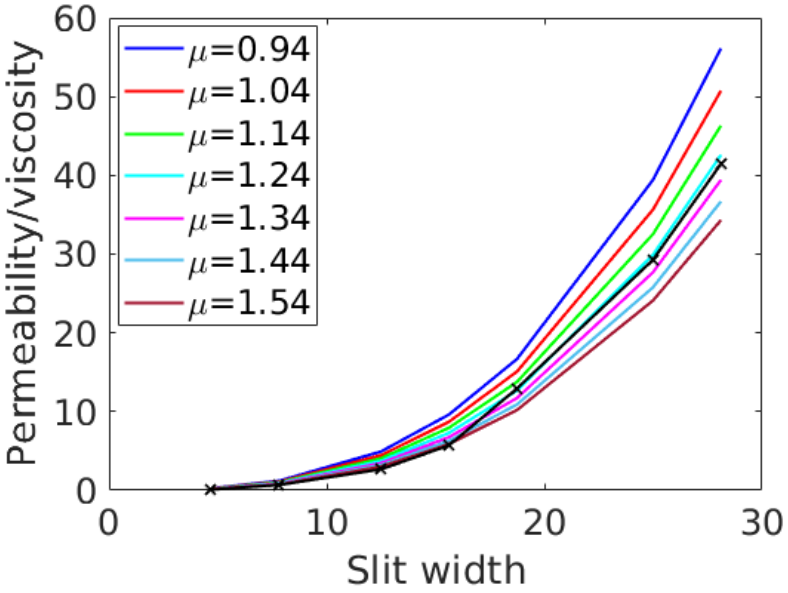


Figure 5.15: Impact of the viscosity value on *permeability/viscosity* for planar Poiseuille flow. The black data points indicate the results obtained from the NEMD simulations.

5.1.4 Pressure calculation with the *method of planes*

Appendix A.4.1 give the results that are obtained when using the *method of planes* for calculating the configurational part of the pressure. The main observations are summarized in the following list:

- Using the *method of planes* gives lower values of *permeability/viscosity* compared to the previous approach.
- The general trend is that the percentage relative deviation from planar Poiseuille flow decreases as the slit pore width increases.
- For all values of slit pore width, the *method of planes* resulted in larger deviation from planar Poiseuille flow than the previous approach.

5.2 Porous medium base case with lattice constant a=20

The porous medium base case will act as a reference for the later case studies where the lattice constant is varied. The main goal is to design a method for calculating the permeability of nano-porous media, but this will first require an understanding of a reference case. The parameters for the base case are chosen in such a way that a direct comparison can be made with the results obtained by Galteland *et. al.* for a lattice constant of $a = 20$ [1]. If the results agree, this will make us trust the input and post-processing scripts more. In this section, results for the porous medium base case defined in Section 4.3.3 will be presented. The lattice constant is $a = 20$.

5.2.1 Equilibrium

Following the procedure given in 4.3.1, the first step is to calculate properties related to the geometry of the system. The volume of grain V^r, the fluid volume V^f and the surface area Ω^{fr} are plotted in Figure 5.16. The next step is to determine the values of γ^{fr} and $\hat{p}^r$ that give a constant compressional energy through the system at equilibrium. Figure 5.16 shows the compressional energy obtained for one specific equilibrium simulation, corresponding to one specific value of fluid density.

To later analyse the non-equilibrium case, we need to know how the integral rock pressure $\hat{p}^r$ and the surface tension γ^{fr} depend on the fluid pressure. For this purpose, Figure 5.17 and Figure 5.18 are created. The blue data points are obtained by running equilibrium simulations at different fluid densities, that correspond to different fluid pressures. For each equilibrium simulation, one determines the values of $\hat{p}^r$ and γ^{fr} that give the same compressional energy in all REVs. The data points for different fluid pressures fall on straight lines and functions $\gamma^{fr}(p^f)$ and $\hat{p}^r(p^f)$ are obtained by linear regression. In Figure 5.19 and Figure 5.20, these functions are compared to the functions obtained by Galteland *et. al.* for lattice constant $a = 20$ [1].

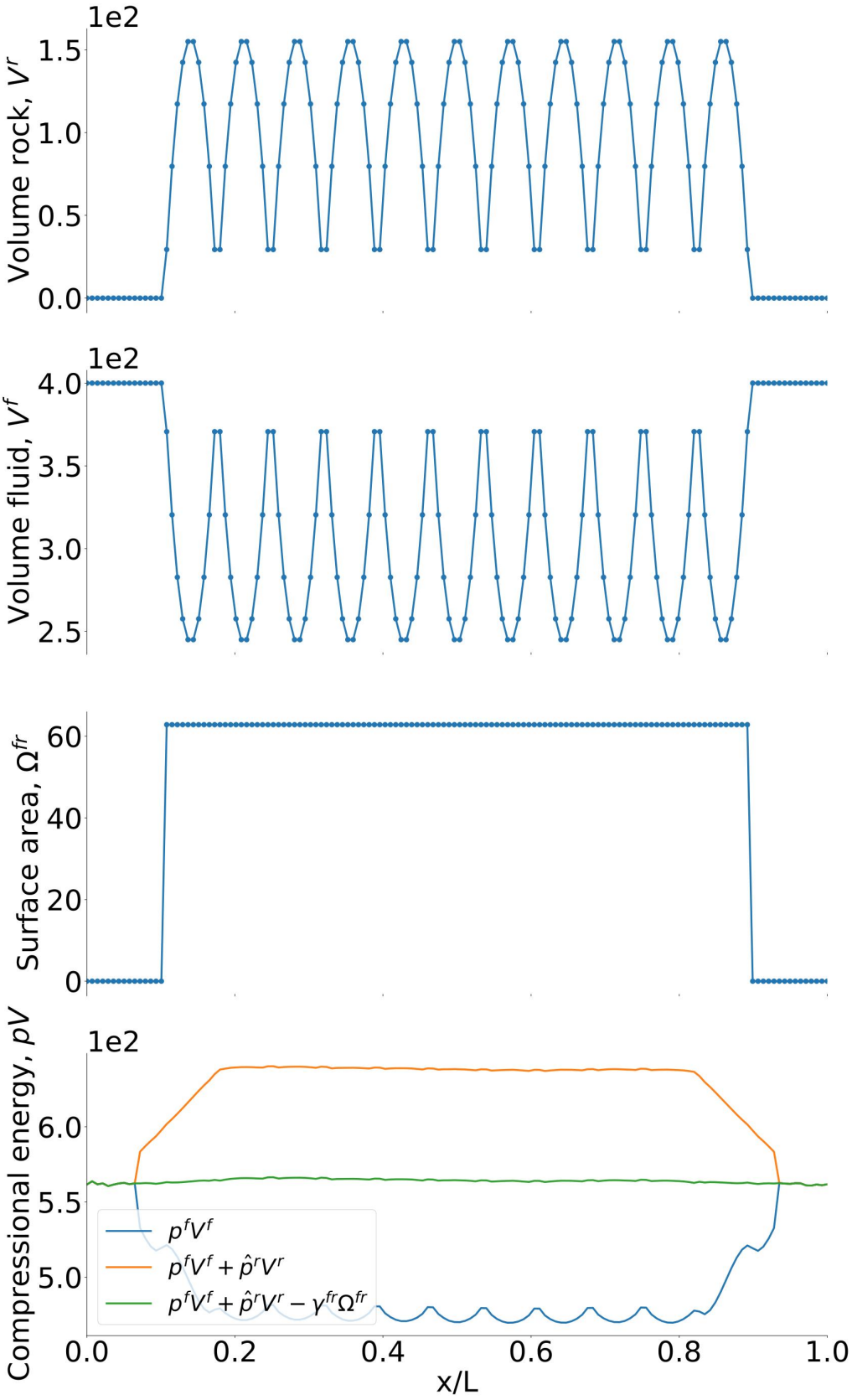


Figure 5.16: Volume of grain, V^r, volume of fluid, V^f, and surface area, Ω^{fr}, for the porous medium base case defined in Section 4.3.3 with lattice constant $a = 20$. The compressional energy smoothed over the REVs at equilibrium is also plotted at three different stages.

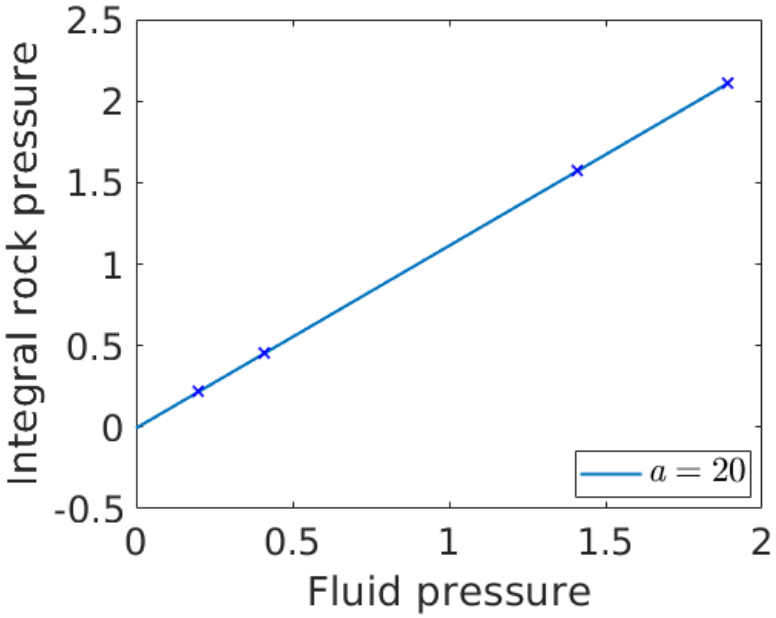


Figure 5.17: Integral rock pressure, $\hat{p}^r$, as a function of fluid pressure for the porous medium base case defined in Section 4.3.3 with lattice constant $a = 20$. The different fluid pressures are generated by running equilibrium simulations at different densities. The fluid densities used are 0.1, 0.2, 0.5 and 0.55. Reduced units are used (see Section 3.11).

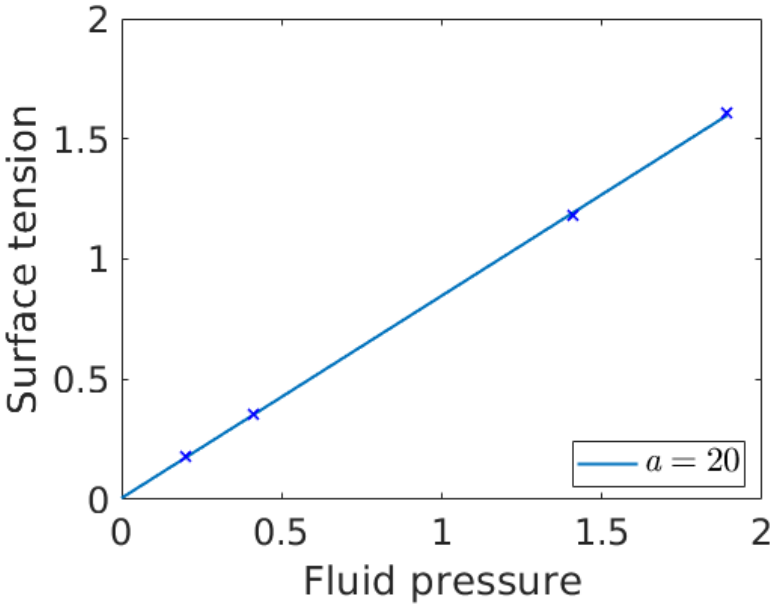


Figure 5.18: Surface tension, γ^{fr}, as a function of fluid pressure for the porous medium base case defined in Section 4.3.3 with lattice constant $a = 20$. The different fluid pressures are generated by running equilibrium simulations at different densities. The fluid densities used are 0.1, 0.2, 0.5 and 0.55. Reduced units are used (see Section 3.11).

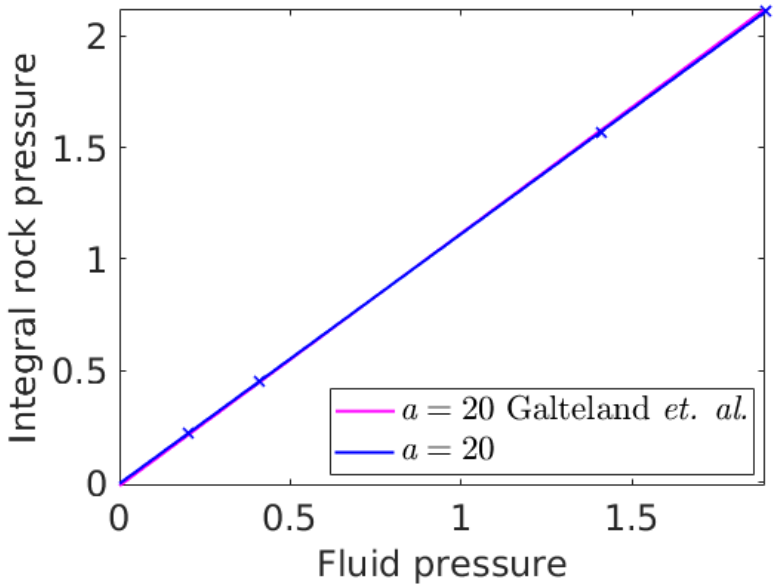


Figure 5.19: Integral rock pressure, $\hat{p}^r$, as a function of fluid pressure. The blue data points and fit correspond to Figure 5.17, whereas the pink line is the function obtained by Galteland *et. al.* for the corresponding case with lattice constant $a = 20$ [1]. Reduced units are used.

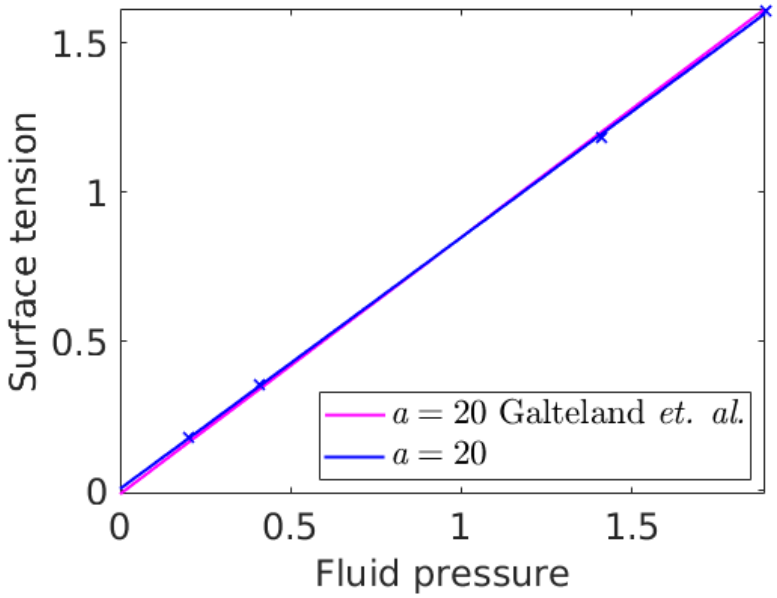


Figure 5.20: Surface tension, γ^{fr}, as a function of fluid pressure. The blue data points and fit correspond to Figure 5.18, whereas the pink line is the function obtained by Galteland *et. al.* for the corresponding case with lattice constant $a = 20$ [1]. Reduced units are used (see Section 3.11).

5.2.2 Non-equilibrium

To study the driving forces in nano-porous media, we next look at the non-equilibrium case. The geometry of the system is the same as in the previous section, so the data for V^f, V^r and Ω^{fr} in Figure 5.16 still apply. However, the applied pressure difference makes the fluid pressure different at non-equilibrium. In Figure 5.21, the different contributions to the fluid pressure tensor are plotted for each chunk. The total fluid pressure in each chunk is also plotted (the pink graph). Based on the known quantities V^f, V^r, Ω^{fr} and p^f, and the functions $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$ that were determined from equilibrium simulations, the compressional energy can be calculated. In Figure 5.22a the compressional energy is plotted at three different stages. In Figure 5.22b the compressional energy is smoothed over the REVs. By dividing the compressional energy on V^{REV}, the integral pressure can be obtained. The integral pressure is plotted in Figure 5.22c, where it is smoothed over the REVs. The slope of the curve is related to the permeability of the porous medium. The orange line in Figure 5.22c is obtained by linear regression using the data points in the porous region. From the slope we determine a gradient in integral pressure equal to $d\hat{p}/dl = -2.514 \pm 0.008$ in the middle part of the porous region.

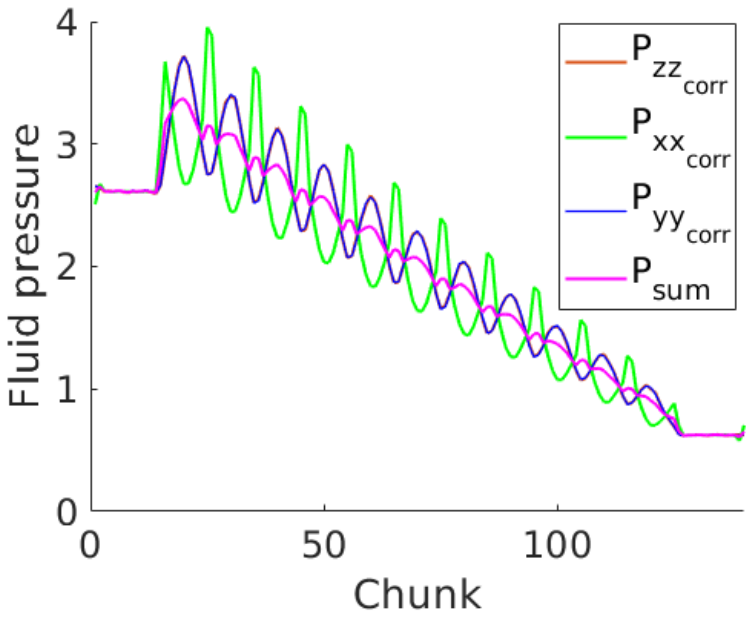


Figure 5.21: Fluid pressure at non-equilibrium plotted for the porous medium base case defined in Section 4.3.3 with lattice constant $a = 20$.

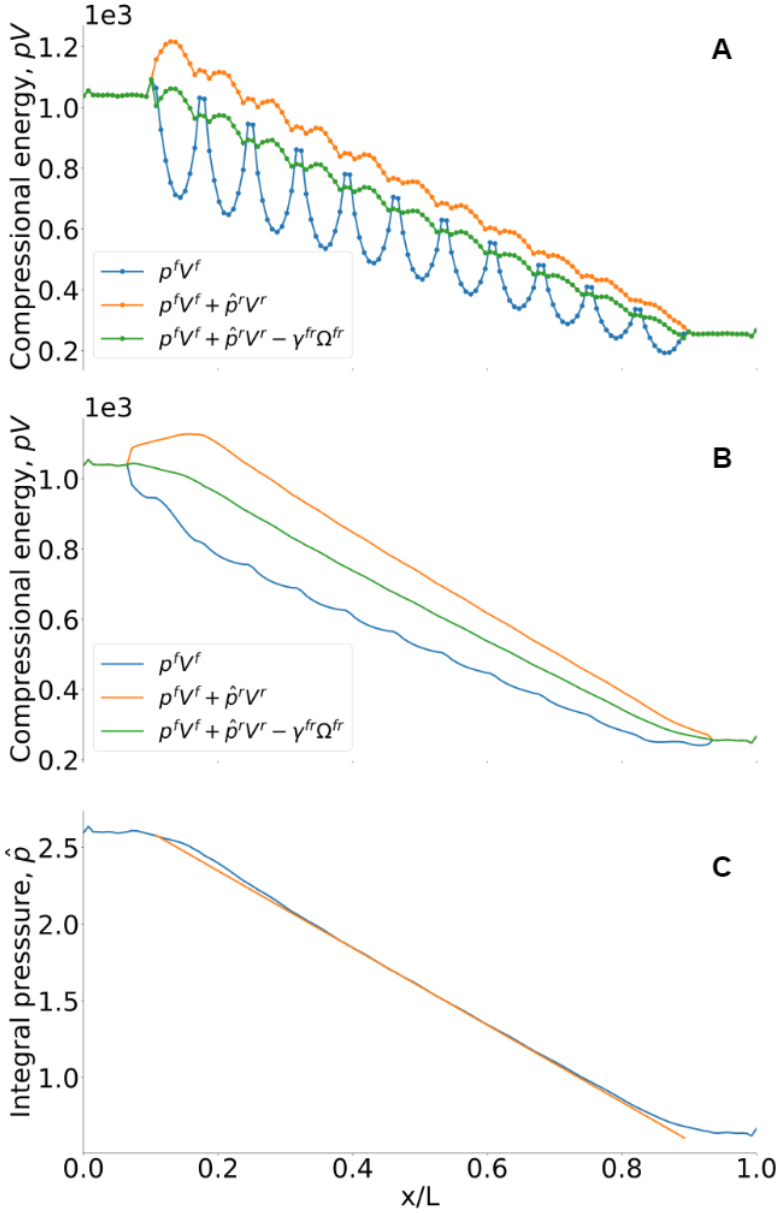


Figure 5.22: Results for the non-equilibrium case defined in Section 4.3.3 with lattice constant $a = 20$. a) Compressional energy in each bin, plotted at three different stages. b) Compressional energy smoothed over the REVs, plotted at three different stages. c) Integral pressure smoothed over the REVs. The orange line is obtained by linear regression for the porous region, and is used to determine the gradient in integral pressure, $d\hat{p}/dl$, in the middle part of the porous region.

5.2.3 Permeability

We aim to establish a method for calculating the permeability in nano-porous media. In this section, the permeability will be calculated from the gradient in integral pressure, in contrast to the overall pressure gradient that is used in relations such as Darcy's law.

In Darcy's law, the permeability is related to the pressure gradient $\Delta P/L'$ by

$$k = -\frac{\mu q}{\Delta P/L'} \tag{5.1}$$

However, instead of using the pressure gradient $\Delta P/L'$, we will instead relate the permeability of the porous medium to the gradient in integral pressure, $d\hat{p}/dx$. Equation (5.1) is then replaced with the following equation:

$$k = -\frac{\mu q}{d\hat{p}/dx} \tag{5.2}$$

We may rewrite the fluid flux q as:

$$\begin{aligned} q &= \frac{\text{Volumetric flow}}{\text{Cross-sectional area}} \\ &= \frac{\text{Mass flux} \cdot \text{Cross-sectional area}}{\text{Density}} \frac{1}{\text{Cross-sectional area}} \\ &= \frac{\text{Mass flux}}{\text{Density}} \end{aligned} \tag{5.3}$$

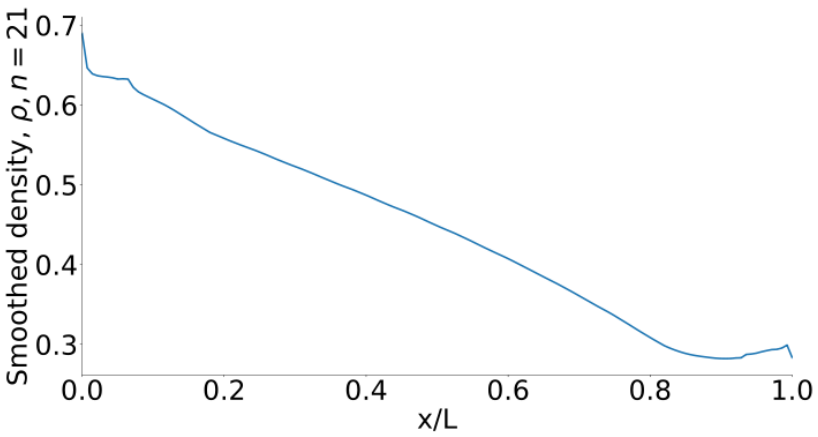


Figure 5.23: Fluid density smoothed over the REVs, for the case defined in Section 4.3.3 with lattice constant $a = 20$. $n = 21$ is the number of bins that are used for smoothing the profile. Reduced units are used (see Section 3.11).

The density varies through the system, as seen from Figure 5.23. Consequently, the fluid flux varies through the system as a function of the density. The permeability is therefore not constant through the system.

To obtain the permeability from Equation (5.2), the porosity, viscosity and q must be determined in addition to $d\hat{p}/dx$. The porosity is calculated from

$$\text{Porosity} = \frac{\text{Volume of fluid}}{\text{Total volume}} \tag{5.4}$$

where the volumes refer to the porous region. The viscosity is dependent on temperature and pressure. The viscosity is determined for the temperature $T^* = 2.0$ and the average pressure of the left and right bulk phases. In Figure 5.24, the shear viscosity is plotted as a function of shear rate, and the dynamic viscosity is found to be $\mu = 0.87$ by extrapolating the data to zero shear rate. The local fluid flux is taken as the mass flux divided by density in each chunk in the porous region, and is plotted in Figure 5.25.

The permeability in Equation (5.2) is calculated from the following data:

- Viscosity $\mu = 0.87$.
- Length of system: $L = 140$.
- $d\hat{p}/dx = -2.514/140 = -0.018$.
- Porosity, $\Phi = 0.74$.

In Figure 5.26, the permeability is plotted and in Figure 5.27 the permeability divided by viscosity is plotted. The last plot is made to allow comparison between different lattice constants in later sections. For other values of the lattice constant, the viscosity value is not calculated, so therefore the *permeability/viscosity* is plotted instead of the permeability on its own.

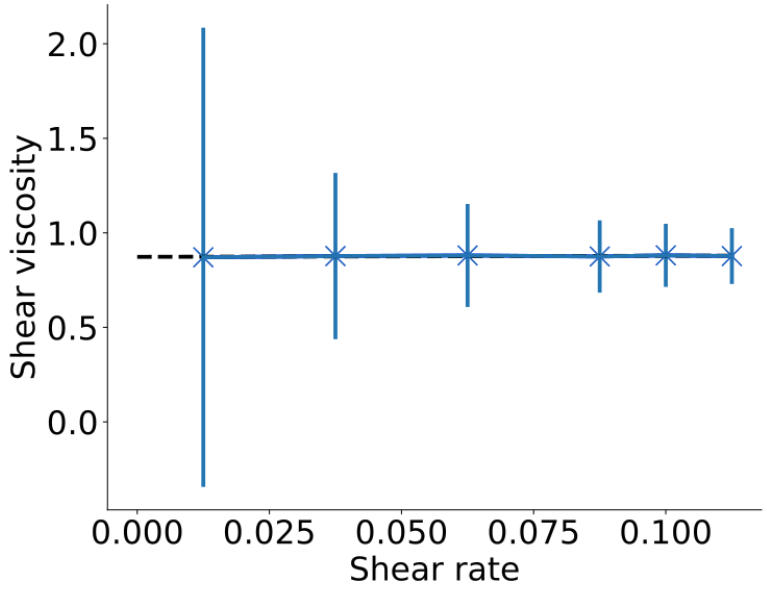


Figure 5.24: Shear viscosity plotted against shear rate. The dynamic viscosity is found by extrapolating to zero shear rate. Reduced units are used (see Section 3.11).

From the data simulations, the fluid flux can be calculated locally in each chunk along the x-axis. However, calculating the permeability based on a fluid flux that varies through the system, is not practical for experiments. Therefore, an alternative calculation of permeability will also be given where the mass flux and average density in the porous region is used.

$$k = -\frac{\mu \cdot \frac{\text{mass flux}}{\text{average density}}}{d\hat{p}/dx} \tag{5.5}$$

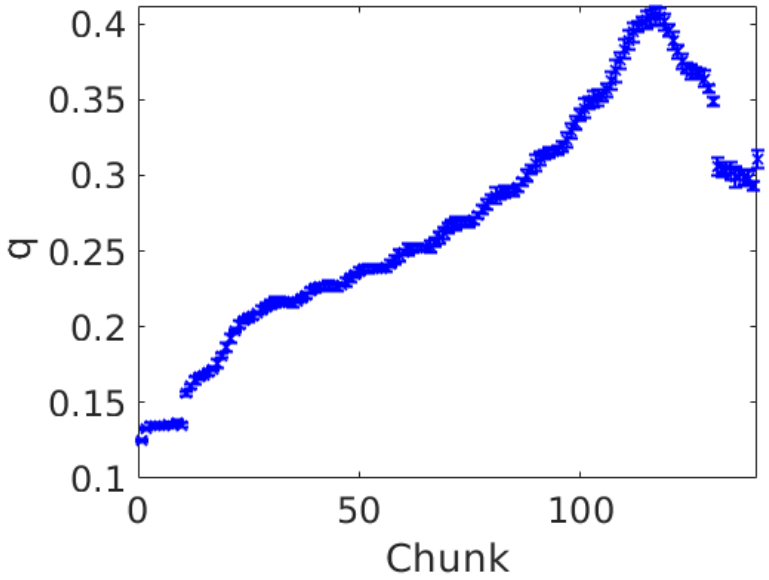


Figure 5.25: Fluid flux q in each chunk at non-equilibrium for the porous medium base case described in Section 4.3.3 with lattice constant $a = 20$. Reduced units are used (see Section 3.11).

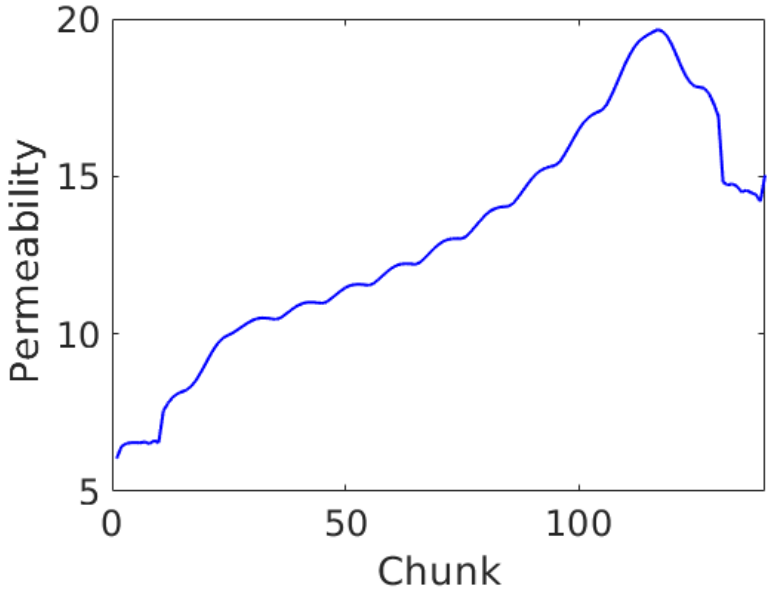


Figure 5.26: The permeability through the system at non-equilibrium for the base case described in Section 4.3.3 with lattice constant $a = 20$. The permeability varies as a function of q. Reduced units are used (see Section 3.11).

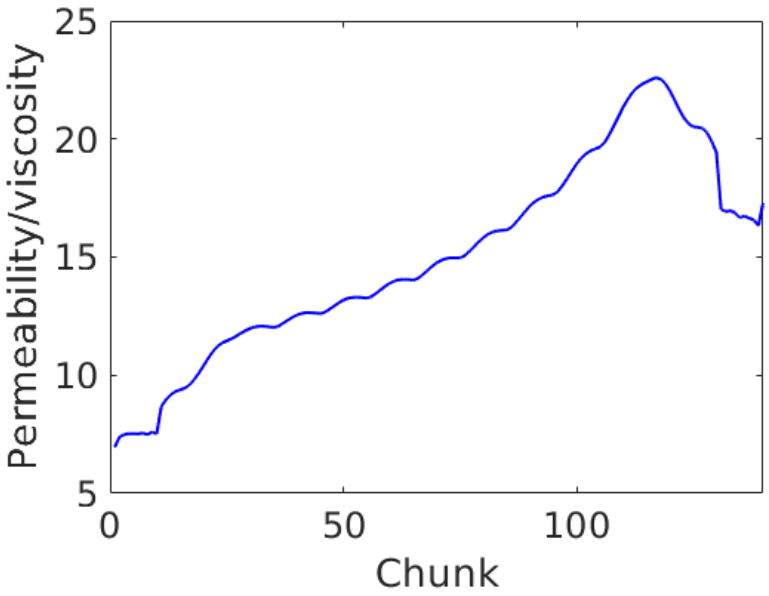


Figure 5.27: The permeability divided by viscosity through the system at non-equilibrium for the porous medium base case described in Section 4.3.3 with lattice constant $a = 20$. Reduced units are used (see Section 3.11).

From Equation (5.5), we obtain:

$$k = -\frac{\mu \cdot \frac{\text{mass flux}}{\text{average density}}}{d\hat{p}/dx} = 12.9 \tag{5.6}$$

with a mass flux of 0.12 (see Figure 5.28) and an average density of 0.45 in the porous region.

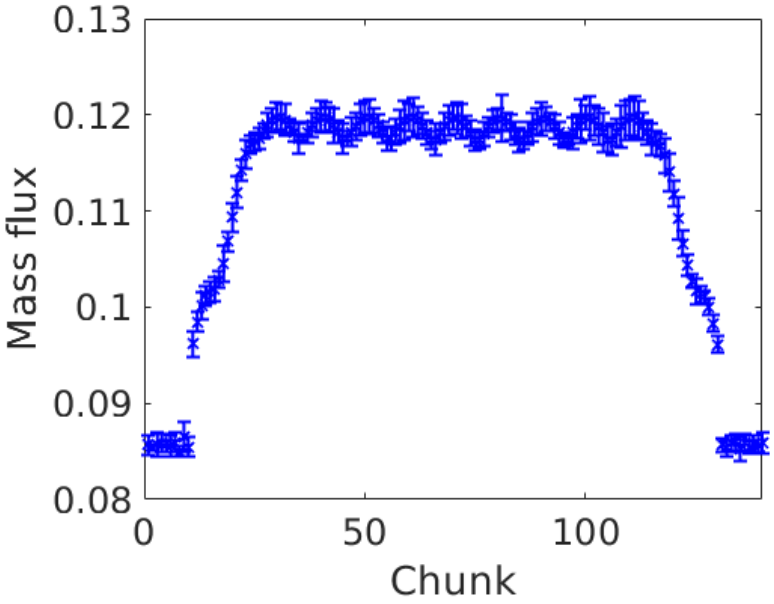


Figure 5.28: Mass flux in each chunk along the z-axis for the porous medium base case defined in Section 4.3.3 with lattice constant $a = 20$. Reduced units are used (see Section 3.11).

5.2.4 Kozeny-Carman equation

We want to compare the permeability obtained in Equation (5.6) to the permeability obtained by the Kozeny-Carman equation. The latter can be used to calculate the permeability of porous media, but should be applied with caution, as explained in Section 2.6.2. The permeability is calculated from values of porosity Φ and surface area S_0:

- Specific surface area, $S_0 = 0.16$.
- Porosity, $\Phi = 0.74$.

The specific surface area is calculated as

$$\text{Specific surface area} = \frac{\text{Surface area of solid spheres}}{\text{Volume of porous region}} \tag{5.7}$$

Using Equation (2.32), the permeability becomes

$$k = \frac{\Phi^3}{5S_0^2} = 3.7 \tag{5.8}$$

This means that the permeability calculated from Equation (5.6) is 3.5 times larger than the permeability calculated from the Kozeny-Carman equation.

The Kozeny-Carman equation is valid for Reynolds numbers up to 1.0, so the Reynolds number should be computed [41]. Inserting the superficial velocity $V_0 = 0.14$, the particle diamter $D_p = 10$, the average density $\rho = 0.45$ and the viscosity $\mu = 0.87$ gives

$$\text{Re}_p = \frac{V_o D_p \rho}{\mu} = 0.72. \tag{5.9}$$

5.2.5 Different approaches for calculating integral pressure

It is of interest to compare the two different approaches for calculating the integral pressure that were presented in Section 4.3.4. In Figure 5.29 the integral pressure is plotted as a function of chemical potential, using both methods. The results obtained by integrating Hill-Gibbs-Duhems equation has been obtained by Varughese [6].

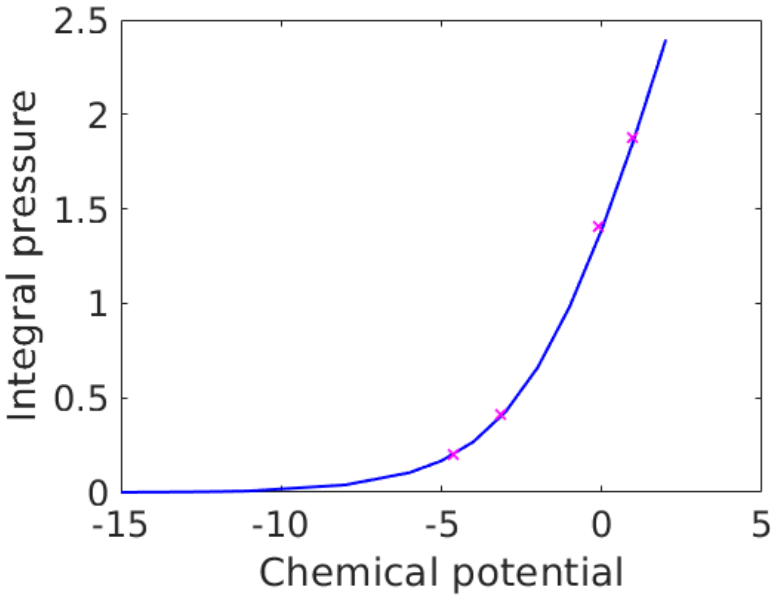


Figure 5.29: The integral pressure as function of chemical potential, obtained by integrating Hill-Gibbs-Duhems equation, is shown as a blue line [6]. The crosses correspond to the NEMD results obtained at equilibrium for the case described in Section 4.3.3 with lattice constant $a = 20$. Reduced units are used (see Section 3.11).

5.3 Porous media with variable lattice constant

In this section, the effect of varying the lattice constant is summarized. By comparing different lattice constants, we want to study confinement effects and the effect on permeability. For detailed results for lattice constants $a = 25$, $a = 30$ and $a = 40$, see Appendix A.5, Appendix A.6 and Appendix A.7. Figure 5.30 and Figure 5.31 give γ^{fr} and $\hat{p}^r$ as functions of fluid pressure for the different values of the lattice constant. In Figure 5.32, the different lattice constants are compared in terms of integral pressure. The slopes of the dotted lines, which are obtained by linear regression within the porous regions, are:

- $a = 20$: $d\hat{p}/dl = -2.514 \pm 0.008$
- $a = 25$: $d\hat{p}/dl = -1.776 \pm 0.004$
- $a = 30$: $d\hat{p}/dl = -1.335 \pm 0.002$.
- $a = 40$: $d\hat{p}/dl = -0.896 \pm 0.001$.

From these data, the permeability divided by viscosity is calculated from Equation (5.5) for all lattice constants. The results are summarized in Table 5.1. In the table, the porosity is given for each of the cases. In the Kozeny-Carman equation (see Equation (2.33)), the permeability is related to the porosity in terms of $\Phi^3/(1-\Phi)^2$. This is the motivation for plotting the obtained values for *permeability/viscosity* against $\Phi^3/(1-\Phi)^2$ in Figure 5.33.

It is known that the diffusion coefficient in small systems can vary with $1/L$ where L is the length of the simulation box [74]. In our system, the length of the simulation box is directly related to the lattice constant. With this in mind, it is interesting to check how the *permeability/viscosity* scales with the inverse lattice constant. Figure 5.34 is made for that purpose.

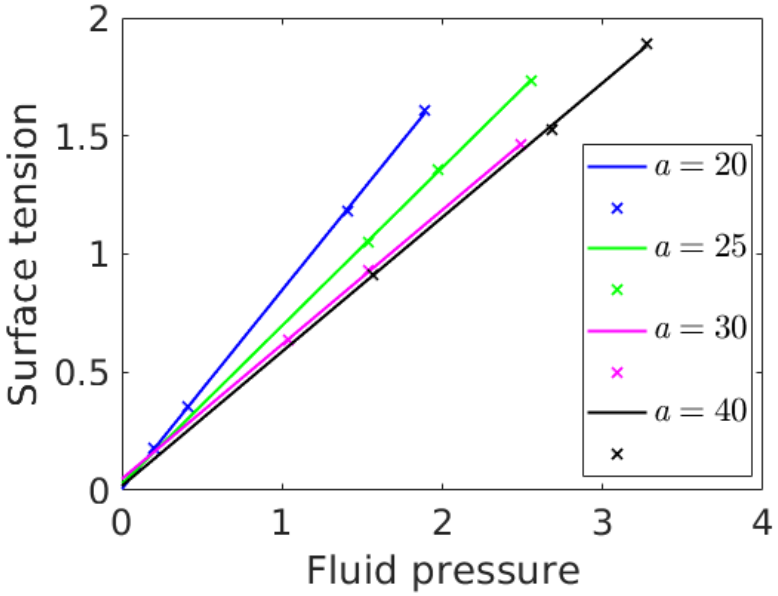


Figure 5.30: Surface tension, γ^{fr}, as function of fluid pressure for porous media with four different lattice constants a. Reduced units are used (see Section 3.11).

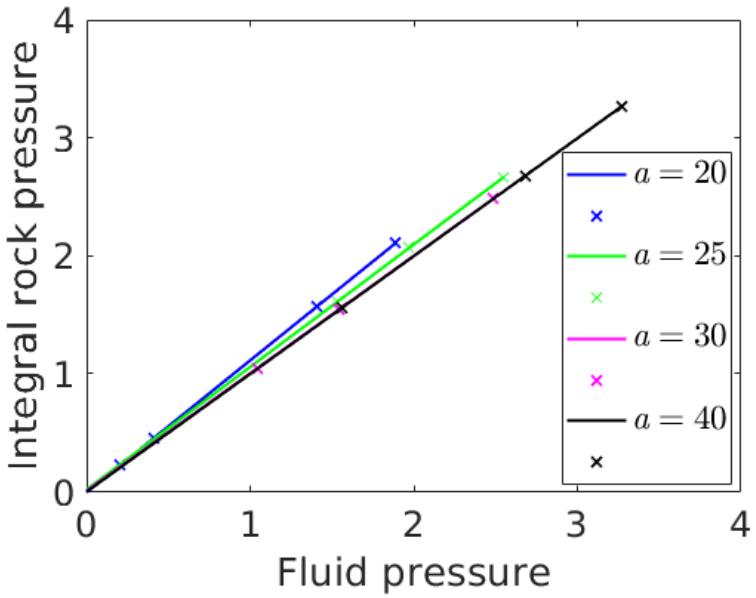


Figure 5.31: Integral rock pressure, $\hat{p}^r$, as function of fluid pressure for porous media with four different lattice constants a. Reduced units are used (see Section 3.11).

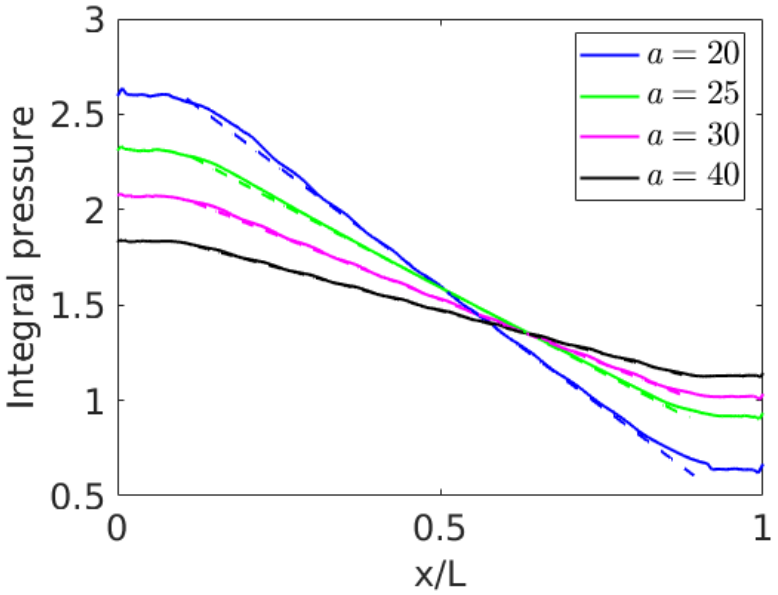


Figure 5.32: Integral pressure plotted for porous media with four different lattice constants a. The dotted lines are obtained by linear regression within the porous region.

Lattice constant a	Porosity	Average density	Permeability/viscosity	Permeability
20	0.74	0.45	14.8	12.9 (3.7)
25	0.89	0.48	34.1	-
30	0.94	0.49	66.7	-
40	0.97	0.49	179.2	-

Table 5.1: Permeability divided by viscosity calculated for porous media with different lattice constants a. The porous medium is represented as a FCC-lattice of solid spherical particles with radius $R = 5.0$, and the temperature is $T^* = 2.0$. The particles are interacting with a Lennard-Jones/spline potential in NEMD simulations. The porosity and average density in the porous media are indicated. For lattice constant $a = 20$, the permeability is determined and can be compared to the permeability obtained by the Kozeny-Carman equation, which is given in parentheses.

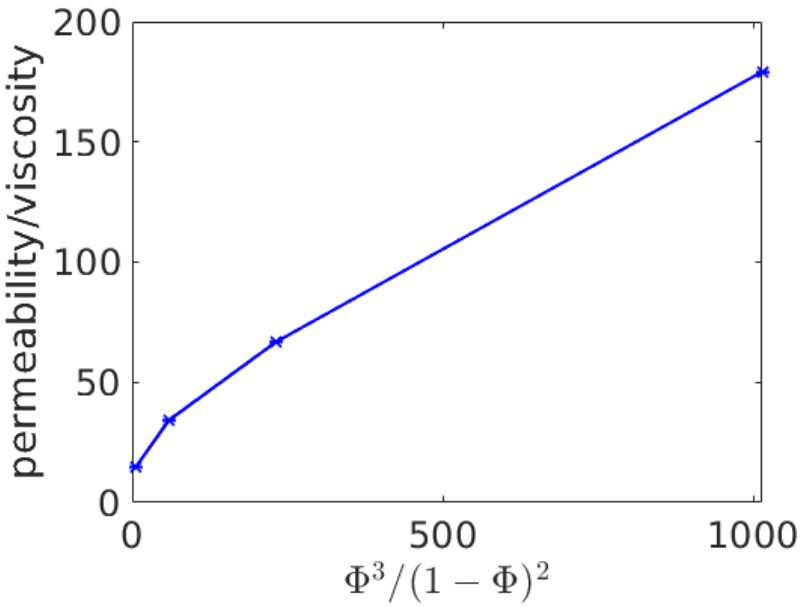


Figure 5.33: *Permeability/viscosity* plotted against $\Phi^3/(1-\Phi)^2$, where Φ is the porosity. The four data points correspond to four different lattice constants (see Table 5.1).

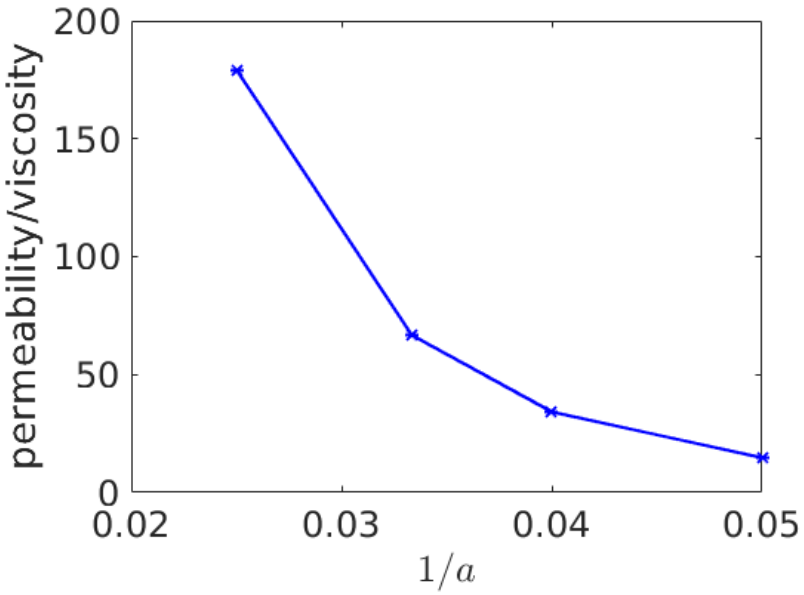


Figure 5.34: *Permeability/viscosity* plotted against $1/a$, where a is the lattice constant. The four data points correspond to $a = 20$, $a = 25$, $a = 30$ and $a = 40$.

5.4 Two-phase box

In Table 5.2, the results for the case study described in Section 4.4 with an evaporating liquid are given. Each simulation was carried out at a different temperature, and the obtained results for pressure, density of gas and density of liquid are provided in the table. The data are visualized in Figure 5.35. The temperature is plotted as a function of density for the purpose of comparing with the results obtained by Hafskjold *et. al.*, shown in Table A.4 [7].

T^*	P^*	n^*_{gas}	n^*_{liquid}
0.55	0.0021	0.0039	0.8040
0.57	0.0031	0.0056	0.7940
0.60	0.0046	0.0084	0.7780
0.62	0.0059	0.0105	0.7667
0.64	0.0079	0.0137	0.7550
0.65	0.0089	0.0158	0.7489
0.67	0.0113	0.0196	0.7362
0.70	0.0156	0.0275	0.7157
0.72	0.019	0.0334	0.7005
0.75	0.0252	0.0452	0.6754
0.77	0.0305	0.0561	0.6563
0.80	0.0399	0.0739	0.6243
0.82	0.0467	0.0960	0.5964
0.85	0.0599	0.1386	0.5445

Table 5.2: Data obtained by molecular dynamics simulations for the case defined in Section 4.4, for an evaporating liquid contained in a box. The columns give from left to right the reduced temperature, the reduced pressure, the reduced density of gas and the reduced density of liquid.

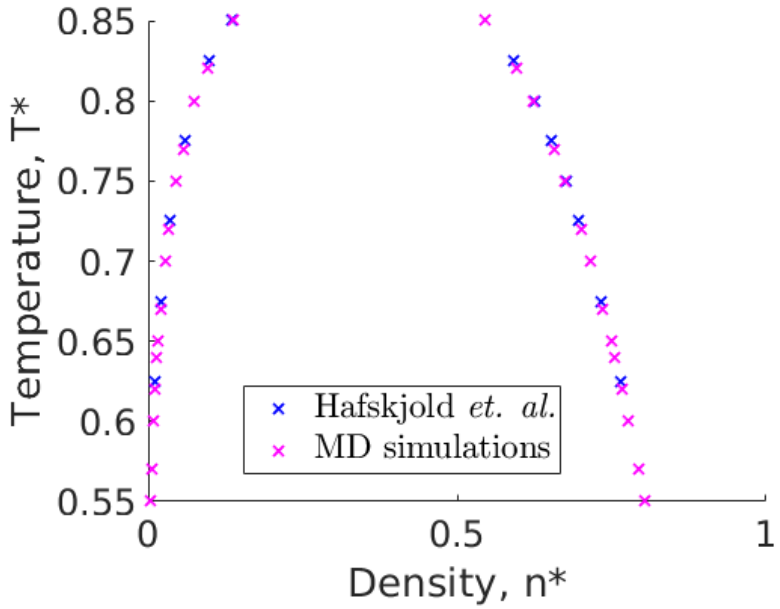


Figure 5.35: Reduced temperature plotted as a function of reduced density for the case defined in Section 4.4, for an evaporating liquid. The pink data points are obtained by molecular dynamics simulations and are given in Table 5.2, whereas the blue data points are taken from Table A.4 and are results obtained by Hafskjold et. al [7].

6 Discussion

6.1 Slit pore

6.1.1 Slit pore, base case

The slit pore results will be interpreted in relation to the equations for planar Poiseuille flow. However, the slit pore base case, which acts as a reference for the other slit pore cases, will first be analysed. We start by considering the mass current and mass flux. The mass current in the system should be constant, and indeed there is no evident change in mass current through the system in Figure 5.1 . However, the error bars are relatively large. In contrast to the mass current, the mass flux changes in a clear manner through the system. There is a jump in mass flux at the entrance and exit of the slit pore, and the mass flux is larger in the slit pore than in the bulk fluids. This is because of the smaller cross-sectional area in the slit pore, compared to the bulk phases. The mass flux is approximately equal in the two bulk phases, where the cross-sectional area is the same.

Looking at Figure 5.3, the general trend is that the pressure decreases along the system, and there is a pressure difference over the slit pore. We notice that there is a sudden jump in pressure at the entrance and exit of the slit pore. The partitioning into chunks is not perfect, so the chunks at these specific positions include both bulk phase and slit pore. This means that the volume calculation is not well-defined in these points. Furthermore, the jumps may also be due to entrance effects. Figure 5.6 shows that the velocity profile is close to constant in the middle part of the slit pore. However, at the entrance and outlet of the slit pore, this is not the case. This means that we do not have fully-developed laminar flow in the entire slit pore. Consequently, properties such as pressure, density and mass flux may be affected. In Figure 5.3, a comparison is made between the pressure component in z-direction, with and without the correction for the center of mass velocity (P_{zz} and $P_{zz,corr}$). We see that the correction has a small effect.

The pressure tensor has a kinetic part and a configurational part. The latter is plotted in Figure 5.4. As for the total fluid pressure, the configurational part also decreases from left to right through the system. Again, we notice that there are sudden jumps at the entrance and exit of the slit pore.

From Figure 5.5 it is seen that the bulk phase at the left has the highest fluid density, which is as expected due to the pressure being highest there. When entering the slit pore, there is a drop in fluid density, and when leaving the slit pore and entering the right bulk phase, there is an increase in fluid density. Again, we notice that there is a sudden jump in fluid density at the entrance and exit of the slit pore. We also notice a wavy behaviour of the fluid density within the slit pore. It could be related to the structure of the solid wall in terms of an FCC-lattice of solid particles. This could cause fluid particles to be located in between solid wall particles, affecting the fluid density.

Figure 5.7 shows that the velocity profile in the middle of the slit pore approaches a parabola. One of the assumptions for planar Poiseuille flow is no-slip conditions at the walls. Figure 5.7 shows that the velocity is almost zero close to the solid walls of the slit pore. Due to the resolution of points, there are no points directly at the slit pore wall where $y = \pm h$.

There are however several assumptions of planar Poiseuille flow that are not satisfied for

the slit pore case. As mentioned, we do not have fully-developed laminar flow in the entire slit pore, as seen from Figure 5.6. Another difference is that in planar Poiseuille flow, the flow is located between infinitely long parallel plates. We use solid particles to form the walls instead, and the walls are consequently not completely flat. Furthermore, our simulations were done with bulk liquid phases to the left and right of the slit pore. We therefore see an entrance effect, where u_x is not the only non-zero component of velocity at the edges of the slit pore.

The parameter α_{12} which is specified for the Lennard-Jones/spline potential, dictates the interactions between fluid and solid particles. To allow for comparison with planar Poiseuille flow, one should choose a value that gives no-slip condition. We have seen that the no-slip condition is approximately fulfilled for the base case with $\alpha_{12} = 1$, which justifies the chosen value of this parameter. Another value which should be justified is the density of the solid. The solid density affects the roughness of the solid. A high roughness would result in the no-slip condition not being satisfied. The density should therefore be chosen so that the roughness is not too high, and here a value of $D_w = 1.05$ was used.

6.1.2 Slit pore with variable pressure difference and slit pore width

In this section, we will discuss the agreement between the planar Poiseuille equation and the NEMD results for slit pores with variable width.

In Figure 5.8, the NEMD data for each value of slit pore width approximately falls on a straight line. Figure 5.8 also shows that the larger the slit pore width, the larger the volumetric flow for a given pressure difference. Comparing the two lines of equal color, we observe that the volumetric flow rates obtained from NEMD simulations deviate from the solution for planar Poiseuille flow. Figure 5.9 shows that the NEMD simulations give values of permeability divided by viscosity that are smaller than the corresponding solution for planar Poiseuille flow. There are several possible explanations. As explained in the previous section, not all assumptions for planar Poiseuille flow are fullfilled. Another possible cause is uncertainty in the viscosity value, as the viscosity is used to establish the graphs for planar Poiseuille flow in Figure 5.8. The sensitivity with respect to the viscosity will therefore be discussed in the next section. A third possible explanation for the deviation, is our definition of the slit pore width, which is not unique. One could for example instead take the slit pore width as the distance between the edges of the outer wall particles of the upper and lower walls. One would then obtain lower values of the slit pore widths, thereby shifting the dotted lines in Figure 5.8 downwards.

Furthermore, it is seen from Figure 5.10 that the percentage deviation in general decreases as the slit pore width increases. For the largest values of slit pore width, the deviation is around 10%. The better agreement for the larger slit pore widths can be justified based on the assumptions for the planar Poiseuille flow equation. The interactions between the solid and fluid particles are not accounted for in planar Poiseuille flow. For a large slit pore, the number of fluid-solid interactions are small compared to the number of fluid-fluid interactions inside the slit pore. However, as the slit pore width decreases, these effects become more and more significant. Figure 5.11, Figure 5.12 and Figure 8.13 will give more insight into the effect of the slit pore width, in terms of agreement with planar Poiseuille flow.

Figure 5.11a gives the fluid velocity v_z along the y-direction in the middle of the slit pore for a large slit width. The velocity profiles approaches a parabolic shape for all different pressure differences. The end points are not at zero velocity, due to the resolution causing no existence of points directly at the slit pore walls. In Figure 5.11b, velocity profiles for a more narrow slit pore are shown. The velocity profiles are now further away from a parabolic shape, but we must remember that the resolution is such that the graphs are constructed from fewer data points for this slit pore width.

In Figure 5.12, velocity profiles for the two different slit pore widths are plotted in comparison with the corresponding solutions for planar Poiseuille flow. We see that for the largest slit width, the velocity profile is close to that of planar Poiseuille flow. However, for the smaller slit width, the velocity profile is significantly different from the corresponding planar Poiseuille velocity profile. This is consistent with the observation that the deviation in Figure 5.10 is decreasing with increasing slit pore width.

The large and smaller slit pore widths are also compared in terms of density. The fluid density profiles are given in Figure 8.13. We observe that the fluid density is not constant in the slit pore, meaning the fluid is not incompressible. Incompressible fluid is one of the assumptions for planar Poiseuille flow, so this may be another cause of the deviation in Figure 5.8. Another aspect to notice in Figure 8.13 is the wavy behaviour of the density profile within the slit pore, which also appeared for the slit pore base case.

We have now seen that the results can be understood in terms of the equations for planar Poiseuille flow for the largest slit pores considered. The deviations for the smaller slit pores are understood in terms of the assumptions forming the basis for the planar Poiseuille flow equations.

6.1.3 Viscosity from NEMD

As the viscosity is required for the calculation of *permeability/viscosity* for planar Poiseuille flow, we discuss the determination of viscosity from NEMD in this section.

It is seen from Figure 5.14 that the shear viscosity data approximately fall on a straight horizontal line. For Newtonian fluids, the viscosity is independent of the shear rate, and the data should therefore fall on a straight line [75]. We observe that the uncertainty is large for the low values of shear rate, and smaller for larger values of shear rate. As there is a significant uncertainty associated with the viscosity, it is interesting to look at the sensitivity of the solution for planar Poiseuille flow with respect to the viscosity. We then look at Figure 5.15. The figure shows that the effect of the viscosity is significant, as the curves for the different viscosity values are noticeably different. A higher viscosity value leads to lower values of *permeability/viscosity*. Figure 5.15 shows that increasing the viscosity value would lead to the NEMD data deviating less from the solution for planar Poiseuille flow, as the circles in Figure 5.9 would then be shifted downwards. Correspondingly, a lower viscosity value would cause the NEMD data to deviate more from the solution for planar Poiseuille flow.

One should be aware that the viscosity was determined for a pressure equal to the average of the left and right bulk pressures in the slit pore. However, we know that the pressure inside the slit pore deviates from these pressure values, which makes it more difficult to determine a precise value to use for the viscosity.

6.1.4 Pressure calculation with the *method of planes*

Two different approaches have been used for the calculation of the configurational part of the pressure tensor. Firstly, the contribution was calculated by taking averages of the defined chunks, which resulted in Figure 5.8, Figure 5.9 and Figure 5.10. Secondly, the configurational contribution was calculated using the *method of planes*, which gave Figure A.1, Figure A.2 and Figure A.3.

In the approach with the *method of planes*, we use the actual gradient in pressure within the slit pore, in contrast to the first approach where the pressure difference is calculated from information only in the bulk phases. It is therefore unexpected that the second approach deviates more from the solution of the planar Poiseuille flow. A possible explanation is the uncertainty in the viscosity value. Changing the viscosity value of the viscosity would shift the solution for planar Poiseuille flow (see Figure 5.15), and this could result in the MOP-approach giving results closer to the solution for planar Poiseuille flow. Using another definition of the slit pore width would also shift the solution for planar Poiseuille flow, possible causing the MOP-approach to give results closer to the planar Poiseuille equation.

6.2 Porous medium base case with lattice constant a=20

The main new contribution of this thesis is relating the permeability of nano-porous media to the gradient in integral pressure. However, we must first understand the results for the porous media base case. The base case results will be compared to those obtained by Galteland *et. al.* with lattice constant $a = 20$. Galteland *et. al.* calculated the compressional energy at equilibrium and non-equilibrium for FCC-lattices of solid particles with lattice constants $a = 20$ and $a = 30$. The calculation of integral pressure follows from the compressional energy by dividing by the volume of REV, and is considered in this thesis. From the integral pressure gradient follows the determination of permeability, which is new here.

The two different approaches for obtaining integral pressure, the one presented by Galteland *et. al.* and the one where Hill-Gibbs-Duhems equation is integrated, will also be compared.

6.2.1 Equilibrium

The geometry of the system is represented in terms of profiles for volume and surface area of the solid particles and volume of the fluid particles. The repetitive patterns in Figure 5.16 are due to the lattice structure with equal unit cells. The surface area is constant in the porous region due to that the layers of solid particles are perfectly aligned. Comparing the figures to the corresponding ones in the work of Galteland *et. al.* [1], the graphs have the same shape, but the numerical values are about four times smaller. The reason is not intuitive, as apparently the same system with the same parameters has been studied. There is therefore a need to justify the numerical values in Figure 5.16. For the porous media base case, a unit cell in the FCC-lattice is divided into 20 chunks. Each of these chunks are represented by a point in Figure 5.16. From the surface area graph, we get that the total surface area in a unit cell is $\#\ chunks \cdot surface\ area\ in\ each\ chunk = 20 \cdot 62.83 = 1256.6$. In a FCC-lattice there are 4 lattice points per unit cell, and therefore there are 4 solid particles per unit cell in Figure 4.2. As the particles have a radius of

$R = 5.0$, they have a total surface area of $\#\ particles \cdot surface\ area\ of\ each\ particle = 4 \cdot 4\pi R^2 = 1256.6$. This means that the surface area graph in Figure 5.16 is justified.

Similarly, we take a look at the grain volume. From the grain volume graph in Figure 5.16, we get that the volume of grain in a unit cell corresponds to the sum of the values in two subsequent peaks, containing 20 points in total. This gives a value of $\#\ peaks\ in\ a\ unit\ cell \cdot factor\ due\ to\ symmetry \cdot sum\ of\ volume\ in\ half\ a\ peak = 2 \cdot 2 \cdot (155.0 + 142.4 + 117.3 + 79.6 + 29.3) = 2094.4$. Each peak has two symmetric halves, which is the reason for the factor 2. The total volume of 4 spheres of radius R is $\#\ particles \cdot volume\ of\ each\ particle = 4 \cdot \frac{4}{3}\pi R^3 = 2094.4$, which means that the plot of V^r is also justified. The reason for the different numerical values in the work of Galteland *et. al.* is unknown to this author [1].

Figure 5.16 also shows the compressional energy for the equilibrium case, averaged over the REVs. The total compressional energy is constant through the system. This is because of how we construct the parameters for the equilibrium case. The values of $\hat{p}^r$ and γ^{fr} are fitted so that this is indeed satisfied. The total compressional energy is 4-5 times smaller than the compressional energy obtained by Galteland *et. al.* [1]. They are different because the different contributions to the compressional energy are dependent on the geometrical properties V^r, V^f and Ω^{fr} [1].

To determine the compressional energy of the non-equilibrium case, we need $\hat{p}^r$ and γ^{fr} as functions of fluid pressure. To obtain such functions, we run equilibrium simulations at different fluid densities, which corresponds to different values of fluid pressure. The results are shown in Figure 5.17 and Figure 5.18. As the fluid pressure increases, $\hat{p}^r$ and γ^{fr} both increase. We see that the data points for different equilibrium simulations fall on straight lines. From Figure 5.19 and Figure 5.20, it is seen that the functions $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$ are almost equal to those obtained by Galteland *et. al.* The relative deviation for the slopes is 2% for $\gamma^{fr}(p^f)$ and 1% for $\hat{p}^r(p^f)$. These functions are obtained from curve fitting, using the known values of V^f, V^r and Ω^{fr}. As the geometrical quantities were found to have different values in this work compared to the work of Galteland *et. al.*, it is interesting to see that the functions $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$ are similar nevertheless.

6.2.2 Non-equilibrium

The volume and surface area of the solid particles, and the volume of fluid is the same at non-equilibrium as for the equilibrium case (see Figure 5.16). The compressional energy shown in Figure 5.22a has a similar shape to the corresponding figure obtained by Galteland *et. al.* [1]. Again, the numerical values are different. The total compressional energy obtained by Galteland *et. al.* ranged from around 760 in the left bulk phase to around 50 in the right bulk phase. In Figure 5.22a, the total compressional energy ranges from around 1050 to around 250. We see from Figure 5.22b that the smooth profile for the total compressional energy is close to linear within the porous region. The integral pressure profile is obtained by dividing the compressional energy on V^{REV}. As a consequence, the integral pressure profile is also close to linear within the porous region, as seen from Figure 5.22c. The bulk contribution $p^f V^f$ is the dominating part of the compressional energy. The integral pressure therefore decreases from left to right because the fluid pressure decreases from left to right. The slope of the curve for integral pressure is related to the permeability of the system, which we will investigate in the next section.

6.2.3 Permeability and Kozeny-Carman

We see from Figure 5.26 that the permeability based on Equation (5.2) varies through the system, and increases from left to right. This is due to the variation in fluid flux q shown in Figure 5.25. The permeability is around three times larger at the right edge of the porous medium than at the left edge. The large variation in permeability is related to the large change in fluid density as shown in Figure 5.23.

When calculating the permeability from Equation (5.5), the fluid flux q is represented by the mass flux divided by average density. We have seen that there are large variations in density for the base case, so the consequence of using the average density should be studied further. It might be a good idea to reduce the pressure difference, to create a more uniform density profile, so that the deviation from the average density is smaller throughout the system.

The permeability obtained from Equation (5.6) is about 3.5 times larger the the permeability obtained from the Kozeny-Carman equation. The permeability from the Kozeny-Carman equation is fully determined by the porosity Φ and the surface area S^0. The calculation of Reynolds number in Equation (5.9), gives a number which is in the validity range for the Kozeny-Carman equation. However, we must remember that the Kozeny-Carman equation is of approximate validity. Several effects are neglected, as pointed out in Section 2.6.2. There is also some uncertainty related to Kozeny's constant, as different values are reported in the literature.

6.2.4 Different approaches for calculating integral pressure

Figure 5.29 shows good agreement between the two different approaches for calculating the integral pressure for a lattice constant $a = 20$. This is promising and makes us trust both approaches more. If the two approaches yield similar results, one may choose the simplest approach for obtaining the integral pressure at equilibrium.

6.3 Porous media with variable lattice constant

In this section, the effect of changing the lattice constant will be analysed. By changing the lattice constant, the distances between the solid particles are varied, thereby affecting the degree of confinement in the porous media systems. The values of lattice constant that are compared to the base case are $a = 25$, $a = 30$ and $a = 40$. The lattice constant $a = 30$ was also studied by Galteland *et. al.*, but the calculation of permeability from the integral pressure gradient is a new aspect here [1].

6.3.1 Equilibrium

There is a clear difference in shape when comparing the profiles for surface area for $a = 25$, $a = 30$ and $a = 40$ to the base case (see Figure 5.16, Figure A.5, Figure A.12 and Figure A.21). For $a = 25$, $a = 30$ and $a = 40$, the layers with solid spheres are not perfectly aligned, as they were for the base case. This means that there are layers containing only fluid within the porous region, and in those layers the surface area is zero. Still, we observe that the profiles for V^r, V^f and Ω^{fr} have a periodic behavior as expected for all values of the lattice constant. This is due to the periodic nature of the FCC-lattice.

In Figure A.15 and Figure A.16, a comparison is made between lattice constants $a = 20$ and $a = 30$ in terms of the functions $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$. We see that using a different lattice constant affects the dependency of $\hat{p}^r$ and γ^{fr} on fluid pressure. The same effect was observed by Galteland *et. al.* [1]. They found that the pressure inside grains in a FCC-lattice and the surface tension depends on the distances between surfaces of the spheres [1]. It was suggested that the difference is due to the disjoining pressure. In Figure 5.30 and Figure 5.31, $\gamma^{fr}(p^f)$ and $\hat{p}^r(p^f)$ are plotted for the four different values of the lattice constant. It is seen that for the largest value of lattice constant, the curves are similar, whereas the difference is larger for smaller lattice constants. As the lattice constant decreases, confinement effects become more important. The surface tension has a curvature dependency, and for the smallest lattice constants, the dependency on the curvature is more dominating than for the larger lattice constants.

The compressional energy at equilibrium smoothed over the REVs is shown at the bottom of Figure 5.16, Figure A.5, Figure A.12 and Figure A.21. A common trait is that the total compressional energy is constant in the porous region due to how the values of γ^{fr} and $\hat{p}^r$ are fitted. The compressional energy in each REV increases with increasing lattice constant.

6.3.2 Non-equilibrium

From the profiles of compressional energy at non-equilibrium (see figure 5.22, figure A.8, figure A.17, and figure A.24), we observe that the fluid makes the largest contribution to the compressional energy. When smoothed over the REVs, the total compressional energy gives profiles which are close to linear within the porous regions. However, at the edges there is some curvature. At the edges, the porous medium is not surrounded by layers of porous medium, but is instead adjacent to a fluid layer. This produces an entrance effect. We see from the mass flux profiles (see Figure 5.28, Figure A.11, Figure A.20 and Figure A.27) that there are transition regions at the start of the porous region, where the mass flux increases from the left bulk phase to the porous region. Similarly, the mass flux decreases in a transition region at the left, from the left of the porous region to the left bulk phase. The change in mass flux can also be explained by the partitioning into chunks, which is not perfect at the transition between bulk and porous regions.

However, because of the linearity of the profiles of compressional energy in the middle part of the porous regions, the integral pressure profiles in Figure 5.22c, Figure A.8c, Figure A.17c and Figure A.24c are also close to linear in the middle of the porous regions. This means that constant slopes in these regions can be determined. Comparing the slopes in figure 5.32, it is seen that as the lattice constant increases, the slope $d\hat{p}/dl$ decreases in absolute value. Another observation is that as the lattice constant decreases, the deviation from the linear fit increases at the end of the porous region. This means that for a low lattice constant, the gradient in integral pressure is not the same at the edges of the porous region as in the center.

6.3.3 Permeability

Comparing the profiles of permeability divided by viscosity in Figure A.9, Figure A.18 and Figure A.25 to the profile we saw for the base case (see Figure 5.27), we observe that the profiles have similar shapes. This is due to the similar profiles for fluid flux (see Figure 5.25, Figure A.10, Figure A.19 and Figure A.26). However, the numerical values

are different in these profiles, and the trend is that the *permeability/viscosity* increases as the lattice constant increases. Looking at Equation (5.2), we can understand why this happens. As the lattice constant increases:

- The porosity Φ increases.
- The fluid flux q increases.
- The slope $d\hat{p}/dl$ decreases in absolute value.

All of these effects contribute towards making the *permeability/viscosity* larger as the lattice constant increases.

In Table 5.1, the *permeability/viscosity* calculated from Equation (5.5) is given for the four different lattice constants. The calculation involves the average density, and we see that the average density is between 0.45-0.49 for the four cases considered. Permeability is known to be dependent on porosity, so the corresponding porosity values are also given in the table. The data show that a small change in porosity has a large impact on the *permeability/viscosity*. For an average density of 0.49, the *permeability/viscosity* is almost three times as large for a porosity of 0.97 than for a porosity of 0.94. In the Kozeny-Carman equation, the permeability is related to the porosity in terms of $\Phi^3/(1-\Phi)^2$. It is therefore interesting to plot the values for *permeability/viscosity* in Table 5.1 against $\Phi^3/(1-\Phi)^2$. A linear relationship would indicate that the permeability is proportional to $\Phi^3/(1-\Phi)^2$. We must remember that we are plotting *permeability/viscosity*, not the permeability, as the viscosity has only been determined for lattice constant $a = 20$. The viscosity is in general dependent on temperature and pressure. However, as the temperature is the same for all systems, and the fluid density is similar, the viscosity is expected to be similar for the four cases. The graph in Figure 5.33 is not linear, but it is not that far away. The Kozeny-Carman equation assumes that the porous medium can be treated as a bundle of capillaries of equal length, that the velocity components normal to the tubes' axes can be neglected and that the fluid flow is laminar. That the graph in Figure 5.33 is not that far from linear, suggests that these assumptions approximately hold. Dullien argued that for high values of porosity, the Kozeny-Carman equation is not the best choice for describing flow in porous media [18]. The porosity for $a = 40$ is as high at $\Phi = 0.97$, so this could be one possible explanation for the deviation from linearity. Another explanation could be the fact that the average density is not equal for all four cases, causing the viscosity to not be equal for all four cases.

Another way of investigating how the system dimension affects the permeability, is plotting the *permeability/viscosity* against the reverse lattice constant. This is done in Figure 5.34, which shows that the *permeability/viscosity* is highly dependent on the lattice constant. We do not find a linear relationship here, in contrast to what has been observed when relating the diffusion coefficient in small systems to $1/L$ where L is the length of the simulation box [74].

6.4 Two phase box

For the case of the evaporating liquid, Table 5.2 shows that the higher the temperature, the higher the pressure. For a given value of temperature, there are two corresponding values of density in Table 5.2 and Figure 5.35. The two density values in Figure 5.35 represent the density of gas (left point) and density of liquid (right point). The density of

gas increases as the temperature increases, whereas the density of liquid decreases with increasing temperature. This is the expected behaviour, as a higher temperature results in more evaporation, turning more liquid into gas, and thereby affecting the densities. It is seen from Figure 5.35 that the molecular dynamics simulation data fall on the same curve as the data obtained by Hafskjold *et. al.* [7]. The reproduction of the data obtained by Hafskjold *et. al.* is an indication that the post-processing scripts are valid.

6.5 Limitations

A discussion of the limitations associated with the work presented will follow next.

For the nano-porous media model, representing the porous medium by a FCC-lattice is a simplification. In reality, porous media have in general complex structures, with a variety of pore sizes and shapes. Here, we assume that all pores are of the same size and shape.

When calculating the permeability of the nano-porous medium, we have represented the fluid flux q with the *mass flux/average density*. The variation in density in the system is large, so using the average density might not give a precise description.

The assumptions for planar Poiseuille flow are not fully satisfied for the slit pore. The fluid is not fully incompressible and we do not have fully developed flow in the slit pore. In addition, we do not have flow between two infinitely long parallel plates, and entrance effects appear as a consequence. We therefore do not expect the results for the slit pore to fully agree with those for the planar Poiseuille flow. Still, we expect the results to agree more for the wide slit pores compared to the more narrow ones, as has been pointed out already.

For the slit pore, we have assumed that the viscosity is constant within the system. For narrow slit pores, Wu *et. al.* argued that the viscosity is not constant [76]. They argued that the viscosity is indeed a function of wall properties and nano-pore dimension. We have seen that the analytical solution for the planar Poiseuille flow is sensitive to the viscosity value. When adding the fact that the viscosity has a large associated uncertainty, it is evident that the viscosity is a cause of error that should be considered.

The Kozeny-Carman equation is of approximative validity, and has several limitations. In reality, the permeability will not be fully determined by the porosity and the specific surface area, as explained in Section 2.6.2. We must also remember that different values for the Kozeny constant is reported in the literature, and that the value used here may therefore not give a precise description.

6.6 Further work

The work can be expanded and extended in several ways. Suggestions are listed below:

- Nano-porous media
  - It would be interesting to run simulations where other parameters are varied, and study the effect on the driving forces of the porous media. Examples of parameters that could be varied are the radius of solid particles and the temperature.

  - Instead of using a FCC-lattice, one could run simulations with other types of lattices, for example cubic lattices, and see if the approach can be applied successfully to other lattices.
  - One could run simulations with random distributions of solid particles. Galteland *et. al.* suggested that the REV must then be larger, to contain a representative collection of pores [1].
  - The non-equilibrium simulations were carried out for a single pressure difference over the system. It would be interesting to study systems with other applied pressure differences, by varying the reflective membrane properties. In particular, one could simulate systems with lower pressure differences, to produce a more uniform density profile in the system, and thereby a more uniform permeability through the system.

- Slit pore
  - The dissipative particle dynamics (DPD) thermostat is not applied here due to its complexity. However, this could be an interesting future work, as Ruiz-Franco *et. al.* concluded that it was the best in terms of stability, realism and computational efficiency when they compared the Langevin, Bussi-Donadio-Parrinello and Dissipative particle dynamics thermostats [63].

7 Conclusion

The new approach for calculating the pressure in a nano-porous medium, developed by Galteland *et. al.*, has been applied to porous media with different lattice constants [1]. The main method has been Non-equilibrium molecular dynamics (NEMD) simulations. The goal was to help an ongoing effort to define state variables of confined media, expand the knowledge about driving forces and design a method to determine the permeability. For a nano-porous medium, two pressures are required to describe the system, the integral and differential pressure. Here, the permeability has been related to the gradient in integral pressure in the porous medium. The porous medium has been represented as a face centered cubic lattice of grain particles, and an applied pressure difference causes fluid to flow through the system. The application of a representative elementary volume (REV) has been central. The integral pressure has been obtained from the total compressional energy of the REV, which has contributions from the fluid, the solid grain particles and the interfaces between these.

When calculating the permeability from the local fluid flux, the permeability varies through the system. The permeability has also been calculated based on the mass flux and average density of the porous region, as an alternative to the local fluid flux. For both approaches, the permeability was found to increase with increasing lattice constant. The calculated permeability for a lattice constant $a = 20$ was 3.5 times larger than the permeability calculated from the Kozeny-Carman equation, which is an established equation for calculating the permeability of porous media. However, similar to what is seen for the Kozeny-Carman equation, the permeability was found to highly depend on porosity.

Comparing different lattice constants, confinement effects have been observed. For $a = 30$ and $a = 40$, the integral rock pressure and surface tension as functions of fluid pressure were almost equal. However, when reducing the lattice constant, the integral rock pressure and surface tension become dependent on the system size. The same effect was observed by Galteland *et. al.* [1].

The integral pressure at equilibrium can also be determined by integrating Hill-Gibbs-Duhems equation, which is a procedure used by Varughese [6]. Comparing results for the porous medium with lattice constant $a = 20$, the two different approaches for calculating the integral pressure have shown good agreement.

To build trust in the input and post-processing scripts, two simpler systems were studied before considering the porous medium system. These were a two-phase box containing evaporating liquid and a slit pore where the width was varied. For the evaporation box, the results of Hafskjold *et. al.* were reproduced [7]. The NEMD results for the slit pore were compared to the known solution for planar Poiseuille flow. The conditions for planar Poiseuille flow were not fully satisfied, but still a meaningful comparison could be made. As the slit width increased, the results showed better agreement, and for the widest slit pores the deviation was less than 10%.

References

[1] Galteland, O., Bedeaux, D., Hafskjold, B. & Kjelstrup, S. Pressures Inside a Nano-Porous medium. The Case of a Single Phase Fluid. *Frontiers in Physics* **7** (2019).

[2] Brennen, C. E. Couette and planar Poiseuille flow. `http://brennen.caltech.edu/fluidbook/basicfluiddynamics/Navierstokesexactsolutions/couetteflow.pdf`.

[3] Galteland, O. Shear viscosity of a low density Lennard-jones/spline fluid (2020).

[4] Boundary.ppt. `http://people.virginia.edu/~lz2n/mse627/notes/Boundary.pdf`.

[5] Li, J., Liao, D. & Yip, S. Coupling continuum to molecular-dynamics simulation: Reflecting particle method and the field estimator. *Physical Review E* **57** (1998).

[6] Varughese, K. (2021). Personal communication.

[7] Hafskjold, B. *et al.* Thermodynamic properties of the 3D Lennard-Jones/spline model (2019).

[8] Lautenschlaeger, M. & Hasse, H. Thermal and caloric properties of fluids from non-equilibrium molecular dynamics simulations using the two-gradient method. *The Journal of Chemical Physics* **149**, 244106 (2018).

[9] Allen, M. P. & Tildesley, D. J. *Computer Simulation of Liquids: Second Edition* (Oxford University Press, 2017).

[10] Jin, Z.-H. Lennard–Jones Potential and Reduced units. `https://cms.sjtu.edu.cn/gs/doc/MD2019/LJ.pdf` (2018).

[11] Eberhard, U. *et al.* Determination of the Effective Viscosity of Non-newtonian Fluids Flowing Through Porous Media. *Frontiers in Physics* **7** (2019).

[12] Berg, C. Permeability Description by Characteristic Length, Tortuosity, Constriction and Porosity. *Transport in Porous Media* **103** (2014).

[13] Heinemann, Z. E. *Fluid Flow in Porous Media*, vol. 1 (2005).

[14] Okadaz, K. & MacKenzie, K. J. *Nanomaterials, From research to application* (2006).

[15] Erdős, M. *et al.* Gibbs Ensemble Monte Carlo Simulation of Fluids in Confinement: Relation between the Differential and Integral Pressures. *Nanomaterials* **10**, 293 (2020).

[16] Kjelstrup, S., Bedeaux, D., Hansen, A., Hafskjold, B. & Galteland, O. Non-isothermal Transport of Multi-phase Fluids in Porous Media. Constitutive Equations. *Frontiers in Physics* **6** (2019).

[17] Dejam, M., Hassanzadeh, H. & Chen, Z. Pre-Darcy Flow in Porous Media. *Water Resources Reserach* **53** (2017).

[18] Dullien, F. *Porous media, Fluid Transport and Pore Structure* (Academic Press, 1992). 2nd edition.

[19] Nishiyama, N. & Yokoyama, T. Permeability of porous media: Role of the critical pore size: Critical Pore Size-Permeability Relation. *Journal of Geophysical Research: Solid Earth* **122** (2017).

[20] Rauter, M. *et al.* Two-phase Equilibrium Conditions in Nanopores. *Nanomaterials* **10**, 608 (2020).

[21] Miller, C. & Gray, W. Thermodynamically Constrained Averaging Theory Approach for Modeling Flow and Transport Phenomena in Porous Medium Systems: 4. Species Transport Fundamentals. *Advances in water resources* **31**, 577–597 (2008).

[22] Hill, T. L. Thermodynamics of Small Systems. *The Journal of Chemical Physics* **36** (1962).

[23] Bedeaux, D., Kjelstrup, S. & Schnell, S. *Nanothermodynamics: General Theory* (2020).

[24] Gray, W. G. & Hassanizadeh, S. M. Macroscale continuum mechanics for multiphase porous-media flow including phases, interfaces, common lines and common points. *Advances in Water Resources* **21**, 261–281 (1998).

[25] Bear, J. *dynamics of fluids in porous media* (American Elsevier, 1972).

[26] De Marsily, G. *Quantitative Hydrogeology, Groundwater Hydrology for engineers* (Academic, Orlando, 1986).

[27] Hassanizadeh, S. M. & Gray, W. G. General conservation equations for multi-phase systems: 1. Averaging procedure. *Adv. Water Resources,* **2**, 131 (1979).

[28] Velasco, R., Pathak, M., Panja, P. & Deo, M. What Happens to Permeability at the Nanoscale? A Molecular Dynamics Simulation Study (2017).

[29] Costa, A. Permeability-porosity relationship: A reexamination of the Kozeny-Carman equation based on a fractal pore-space geometry assumption. *Geophys. Res. Lett* **33** (2006).

[30] Martys, N. & Garbocz, E. J. Length scales relating the fluid permeability and electrical conductivity in random two-dimensional model porous media. *Phys. Rev. B* **46**, 6080–6090 (1992).

[31] Guéguen, Y. & Dienes, J. Transport properties of rocks from statistics and percolation. *Math. Geol* **21**, 1–13 (1989).

[32] Walsh, J. B. & Brace, W. F. The effect of pressure on porosity and the transport properties of rock. *J. Geophys. Res* **89**, 9425–9431 (1984).

[33] Sarout, J. Impact of pore space topology on permeability, cut-off frequencies and validity of wave propagation theories. *Geophys. J. Int* **189**, 481–492 (2012).

[34] Guéguen, Y. & Palciauskas, V. Introduction to the Physics of Rocks. *Princeton Univ. Press, Princeton, N. J.* (1994).

[35] Todd, B., Evans, D. & Daivis, P. Pressure tensor for inhomogeneous fluids. *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics* **52**, 1627–1638 (1995).

[36] Lim, Y. & Bhatia, S. Simulation of methane permeability in carbon slit pores. *Journal of Membrane Science* **369**, 319–328 (2011).

[37] Travis, K. & Gubbins, K. Poiseuille flow of Lennard-Jones fluids in narrow slit pores. *The Journal of Chemical Physics* **112**, 1984–1994 (2000).

[38] Kjelstrup, S., Bedeaux, D., Hansen, A., Hafskjold, B. & Galteland, O. Non-isothermal transport of multi-phase fluids in porous media. The entropy production (2018).

[39] Kjelstrup, S., Bedeaux, D., Johannessen, E. & Gross, J. *Non-Equilibrium Thermodynamics for Engineers* (World Scientific, 2017).

[40] Atangana, A. *Principle of Groundwater Flow*, 15–47 (2018).

[41] Kruczek, B. *Carman–Kozeny Equation*, 306–308 (2016).

[42] Feng, D., Wu, K., Wang, X., Li, J. & Li, X. Modeling the confined fluid flow in micro-nanoporous media under geological temperature and pressure. *International Journal of Heat and Mass Transfer* **145**, 118758 (2019).

[43] Calculate viscosity. `https://lammps.sandia.gov/doc/Howto_viscosity.html`.

[44] Hess, B. Determining the shear viscosity of model liquids from molecular dynamics simulations. *The Journal of Chemical Physics* **116**, 209–217 (2002).

[45] Lanfrey, P.-Y., Kuzeljevic, Z. & Duduković, M. Tortuosity model for fixed beds randomly packed with identical particles. *Chemical Engineering Science* **65**, 1891–1896 (2010).

[46] Mota, M., Teixeira, J. & Yelshin, A. Tortuosity in Bioseparations and its Application to Food Processes (1998).

[47] Gunn, D. J. On axial dispersion in fixed beds. *Chemical Engineering and Processing: Process Intensification* **32**, 333–338 (1993).

[48] Bartell, F. E. & Osterhof, H. J. The Pore Size of Compressed Carbon and Silica Membranes. *J. Phys. Chem* **32**, 1553–1571 (1928).

[49] Sobieski, W. & Lipiński, S. The analysis of the relations between porosity and tortuosity in granular beds. *Technical Sciences* **20**, 75–85 (2017).

[50] Molecular dynamics. `https://www.nature.com/subjects/molecular-dynamics`.

[51] Saga. `https://documentation.sigma2.no/hpc_machines/saga.html`.

[52] Matlab. `https://se.mathworks.com/products/matlab.html`.

[53] Python. `https://www.python.org/`.

[54] About ovito. `https://www.ovito.org/docs/current/introduction.html`.

[55] Sperger, T., Sanhueza, I. & Schoenebeck, F. Computation and experiment: A Powerful Combination to Understand and Predict Reactivities. *Accounts of chemical research* **49** (2016).

[56] Schiller, U. D. An overview of integration schemes for molecular dynamics simulations. https://www2.mpip-mainz.mpg.de/~andrienk/journal_club/integrators.pdf (2008).

[57] The Leapfrog Integrator. http://www.physics.drexel.edu/~steve/Courses/Comp_Phys/Integrators/leapfrog/.

[58] Martini, A. Short Course on Molecular Dynamics Simulation. https://nanohub.org/resources/7575/download/Martini_L3_IntegrationAlgorithms.pdf.

[59] Jacobian. https://mathworld.wolfram.com/Jacobian.html.

[60] Hoover, W. G. Nonequilibrium molecular dynamics: the first 25 years. *Physica A* **194**, 450–461 (1993).

[61] Hoover, W. G. & Hoover, C. G. Nonequilibrium molecular dynamics. *Condensed Matter Physics* **8**, 247–260 (2005).

[62] Different ensembles in molecular dynamics simulations. https://faculty.uml.edu//vbarsegov/teaching/bioinformatics/lectures/MDEnsemblesModified.pdf.

[63] Ruiz-Franco, J., Rovigatti, L. & Zaccarelli, E. On the effect of the thermostat in non-equilibrium molecular dynamics simulations (2018).

[64] Hoover, W. G. & Holian, B. L. Kinetic moments method for the canonical ensemble distribution. *Physics Letters A* **211**, 253–257 (1996).

[65] Thermostats in molecular dynamics. https://www2.ph.ed.ac.uk/~dmarendu/MVP/MVP03.pdf.

[66] Lammps molecular dynamics simulator. https://lammps.sandia.gov/.

[67] Plimpton, S. Lammps features and Capabilities. https://lammps.sandia.gov/tutorials/italy14/italy_overview_Mar14.pdf (2014).

[68] compute pressure command. https://lammps.sandia.gov/doc/compute_pressure.html.

[69] compute stress/mop command. https://lammps.sandia.gov/doc/compute_stress_mop.html.

[70] Lennard-Jones Potential. https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Physical_Properties_of_Matter/Atomic_and_Molecular_Properties/Intermolecular_Forces/Specific_Interactions/Lennard-Jones_Potential.

[71] boundary command. https://lammps.sandia.gov/doc/boundary.html.

[72] Morales, J. J., Nuevo, M. J. & Rull, L. F. Statistical error methods in computer simulations. *Journal of Computational Physics* **89**, Pages 432–438 (1990).

[73] Schmelzer, J., Zanotto, E. & Fokin, V. Pressure dependence of viscosity. *The Journal of chemical physics* **122**, 074511 (2005).

[74] Celebi, A., Jamali, S. H., Bardow, A., Vlugt, T. & Moultos, O. Finite-size effects of diffusion coefficients computed from molecular dynamics: a review of what we have learned so far. *Molecular Simulation* 1–15 (2020).

[75] George, H. F. & Qureshi, F. Newton's law of Viscosity, Newtonian and Non-Newtonian Fluids. *Encyclopedia of Tribology* (2013).

[76] Wu, K. *et al.* Wettability effect on nanoconfined water flow. *Proceedings of the National Academy of Sciences* **114**, 201612608 (2017).

A Appendix

A.1 Symbol list

Greek symbols

Symbol	Description
α_p	Probability of reflection
γ	Surface tension
ϵ	Lennard-Jones energy parameter
η	Shear viscosity
η_s	Shear viscosity at equilibrium
Λ	Length of contact line
μ	Dynamic viscosity
μ_i'	Chemical potential of component i
μ'^c	Change in chemical potential by changing the composition
ξ	Friction constant in DPD
ρ_i	Density
σ'	Entropy production
σ	Lennard-Jones size parameter
σ_{ii}	Soft-core diameter
τ	Tortuosity
$\tau' = -P_{xy}$	Shear stress
Υ	Grand potential
Φ	Porosity
Ω	Surface or interface area
Ω^{fr}	Surface area of grain particles

Latin symbols

Symbol	Description
a	Lattice constant
a^*	Lennard-Jones parameter
A	Cross-sectional area of flow
b	Lennard-Jones parameter
c	Geometric factor
c_0	Kozeny's constant
d	Pore length
D	Fluid density
D_w	Density of solid
d_w	Effective grain size
dt	Time interval
dj	Length of chunk
D_p	Diameter of spherical bed particles
d_1	Horizontal shortest distance between spheres
d_2	Diagonal shortest distance between spheres
E	Energy

$\mathbf{F}_{ij}^{C}$	Conservative pairwise force
$\mathbf{F}_{i}^{R}$	Random force
$\mathbf{F}_{i}^{D}$	Dissipative (drag) force
$\mathbf{f}$	Force
$\mathbf{F}(t)$	Force at time t
G	Gibbs free energy
H_i	Partial specific enthalpy of i
h	Slit pore width
$\mathbf{J}$	Jacobian matrix
J_i'	General flux
J_i	Mass flux of component i
J_c	Mass current
J_u	Internal energy flux
J_q'	Measurable heat flux
J_V	Volume flux
J_s	Entropy flux
J_m	Momentum flux
k	Permeability
k_B	Boltzmann constant
L	System length
L_x	Length of simulation box in x-direction
L_y	Length of simulation box in y-direction
L_z	Length of simulation box in z-direction
L_{ij}	Phenomenological coefficient
L_{pp}	Transport coefficient
L_p	Permeability
L'	Length over which a pressure drop takes place
M^i	Mass of component i
m	Mass
N	Number of particles
N'	Mole number
n	Number of bins
n^*	Reduced density
n_{gas}^*	Reduced density of gas
n_{liquid}^*	Reduced density of liquid
$\mathcal{N}$	Number of replicas
ΔP	Pressure difference
P_{avg}	Average pressure of left and right bulk phases
dp/dx	Pressure gradient in x-direction
p	Differential pressure
$\hat{p}$	Integral pressure
$\hat{p}^r$	Integral rock pressure
p^r	Differential rock pressure
p^f	Fluid pressure
$\bar{p}$	Volume-averaged pressure
$\bar{p}^c$	Surface-averaged pressure
$d\hat{p}/dl$ and $d\hat{p}/dx$	Gradient in integral pressure
P_{IJ}	Pressure tensor

P_{IJ}^{kin}	Kinetic contribution to pressure tensor
P_{xx}	x-component of stress tensor without COM-correction
P_{yy}	y-component of stress tensor without COM-correction
P_{zz}	z-component of stress tensor without COM-correction
$P_{xx,corr}$	x-component of stress tensor with COM-correction
$P_{yy,corr}$	y-component of stress tensor with COM-correction
$P_{zz,corr}$	z-component of stress tensor with COM-correction
P_{sum}	Overall stress tensor
P^*	Reduced pressure
pV	Compressional energy
q	Fluid flux given by volumetric flow rate per unit cross sectional area
Q	Volumetric flow rate
Q^*	Volumetric flow rate per unit depth normal to the plane of flow
$\bar{r}$	Average pore radius
r	Pore radius
r_c'	Cut-off in DPD
r_c	Lennard-Jones/spine parameter
r_s	Inflection point of Lennard-Jones potential
$\mathbf{r}$	Position
$\mathbf{R}(t)$	Position at time t
R	Radius of solid particles
Re_p	Reynold-number
$\overrightarrow{r_i}$	Position of atom in computational cell
$\overrightarrow{r_i^{\text{image}}}$	Mirror image position
r_h	Mean hydraulic radius
R_{ii}	Hard-core diameter
S	Entropy
s	Entropy density
s^*	Shear rate
S_i	Partial specific entropy of component i
s'	Exponent in DPD
S^0	Specific surface area with respect to a unit volume of porous medium
S'	Specific surface area with respect to a unit volume of solid
Δs	Length of porous medium in direction of applied piezometric head difference
s_e	Effective streamline length
T	Temperature
T^*	Reduced temperature
T^{bath}	Temperature of thermal bath
t	Time
U	Internal energy
U'	Constant velocity for SLLOD
u	Internal energy density
u_x	x-component of velocity
V	Volume
$\mathbf{V(t)}$	Velocity at time t
V'	Potential energy
V^r	Volume of grain
V^f	Volume of fluid

V_i	Partial specific volume
v	Fluid velocity
v_{kI}	Velocity of particle k in direction I
$v_{c,I}$	Correction for COM
V_0	Superficial velocity
∇v_{stream}	Spatial gradient of the fluid velocity normal to the momentum flow
W	Length of simulation box in positive x- and y-direction
x	Coordinate axis
X_i	General thermodynamic driving force
y	Coordinate axis
Z_g	Grand partition function
z	Coordinate axis

Subscripts and superscripts

Symbol	Description
α	phase
β	interface
δ	contact line
f	fluid
l	layer
r	rock
t	evaluated at constant temperature
REV	representative elementary volume

A.2 Coexisting gas and liquid

T^*	P^*	n^*_{gas}	n^*_{liquid}
0.55	0.0021	0.0040	0.8041
0.60	0.0046	0.0083	0.7780
0.65	0.0089	0.0154	0.7490
0.70	0.0157	0.0270	0.7158
0.75	0.0256	0.0450	0.6760
0.80	0.0399	0.0753	0.6250
0.85	0.0597	0.1356	0.5449

Table A.4: Data for coexisting gas and liquid provided by Hafskjold *et. al.* [7]. The columns give from left to right the reduced temperature, the reduced pressure, the reduced density of gas and the reduced density of liquid.

Table A.4 gives data for coexisting gas and liquid provided by Hafskjold *et. al.* [7].

A.3 Entropy production in a porous medium

Let us consider a representative elementary volume (REV) of a porous medium. It contains the actual porous medium m' and $m' - 1$ fluid components. Balance equations for mass and internal energy are given by [38]

$$\frac{\partial \rho_i}{\partial t} = -\frac{\partial}{\partial x} J_i \tag{A.1}$$

$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x} J_u = -\frac{\partial}{\partial x} \left[J'_q + \sum_{i=1}^{n} J_i H_i \right] \tag{A.2}$$

where ρ is the density, t is the time, J_i is the component flux, u is the specific internal energy, J_u is the internal energy flux, J'_q is the measurable heat flux and H_i is the enthalpy. The index i refers to component i. As component m' is not moving, it is a convenient frame of reference for the fluxes.

On the macro-scale, the entropy balance equation is

$$\frac{\partial s}{\partial t} = -\frac{\partial}{\partial x} J_s + \sigma' \tag{A.3}$$

where J_s is the entropy flux, s is the specific entropy and σ' is the entropy production. The entropy production is a sum of contributions within the REV. An expression for the entropy production is derived by combining the balance equations with Gibbs equation. It is assumed that the Gibbs equation is valid for the REV also when transport takes place. However, the REV must have a minimum size for the Gibbs' equation to be valid. The Gibbs equation keeps its form during a time interval dt, giving

$$\frac{\partial s}{\partial t} = \frac{1}{T} \frac{\partial u}{\partial t} - \frac{1}{T} \sum_{i=1}^{n} \mu'_i \frac{\partial \rho_i}{\partial t} \tag{A.4}$$

Here, T is the temperature and μ'_i is the chemical potential of component i.

By inserting the balance equations for mass and energy (Equation (A.1) and Equation (A.2)) into Equation (A.4) and comparing the result to the entropy balance in Equation (A.3), it is seen that the entropy flux is

$$J_s = \frac{1}{T} J'_q + \sum_{i=1}^{n} J_i S_i \tag{A.5}$$

The entropy flux consists of the sensible heat flux over the temperature and the sum of the specific entropies carried by the components, S^i. The entropy production can be formulated as

$$\sigma' = J_u \frac{\partial}{\partial x} \left(\frac{1}{T} \right) - \sum_{i=1}^{n} J_i \frac{\partial}{\partial x} \left(\frac{\mu'_i}{T} \right) = J'_q \frac{\partial}{\partial x} \left(\frac{1}{T} \right) - \frac{1}{T} \sum_{i=1}^{n} J_i \frac{\partial}{\partial x} \mu'_{i,T} \tag{A.6}$$

where $\mu'_{i,T}$ is the chemical potential of component i evaluated at constant temperature. The two formulations for the entropy production are equivalent, but the last expression is preferred for analysis of experiments. The entropy production defines the independent thermodynamic driving forces and their conjugate fluxes.

A.3.1 Entropy production for isothermal single fluid in porous media

Often the volume flow is measured rather than the component flows. It is therefore convenient to introduce the volume flow into the entropy production expression. Let us consider a system where we have a single isothermal fluid f flowing through a porous medium. The entropy production in Equation (A.6) then becomes

$$\sigma' = -\frac{1}{T}\left(J_f \frac{\partial \mu'_{f,T}}{\partial x}\right) \tag{A.7}$$

where J_f is the mass flux of the fluid. The volume flux J_V is related to the mass flux by

$$J_V = J_f V_f \tag{A.8}$$

where V_f is the volume of fluid.

Next, we want to rewrite the chemical potential term. The chemical potential of component i is given by the derivative of Gibbs free energy G with respect to the mass of the component [16]

$$\mu'_i \equiv \left(\frac{\partial G}{\partial M_i}\right)_{T,p} \tag{A.9}$$

This means that the total differential of the chemical potential is

$$d\mu'_i = -S_i dT + V_i dp + \sum_{j=1}^{k} \mu'^c_{i,j} dM_j \equiv -S_i dT + V_i dp + d\mu'^c_i \tag{A.10}$$

where S_i and V_i are the entropy and volume of component i. μ'^c_i is the change in chemical potential by changing the composition of the medium. $S_i = -\left(\partial \mu'_i / \partial T\right)_{p,M_j}$, $V_i = \left(\partial \mu'_i / \partial p\right)_{T,M_j}$ and $\mu'^c_{i,j} = \left(\partial \mu'_i / \partial M_j\right)_{p,T,M_k}$ are partial specific quantities.

When the system is isothermal and the composition is uniform, Equation (A.10) simplifies to

$$d\mu'_i = V_i dp \tag{A.11}$$

Inserting Equation (A.8) and Equation (A.11) into Equation (A.7), the entropy production simplifies to

$$\sigma' = -\frac{1}{T}\left(J_V \frac{\partial p}{\partial x}\right) \tag{A.12}$$

A.4 Slit pore

A.4.1 Pressure calculation with the *method of planes*

The three first figures shown in this section correspond to Figure 5.8, Figure 5.9 and Figure 5.10 in Section 5.1.2, but the configurational part of the pressure tensor is now calculated with the *method of planes*. Figure A.1 shows the volumetric flow plotted against pressure difference over the slit pore, using the *method of planes* for calculating the configurational part of the pressure tensor. The slopes correspond to the permeability divided by viscosity, and this quantity is plotted for each of the slit pore widths in Figure A.2. The results from the NEMD simulations are related to the solution for planar Poiseuille flow as before, and the percentage relative deviation is plotted as a function of slit pore width in Figure A.3.

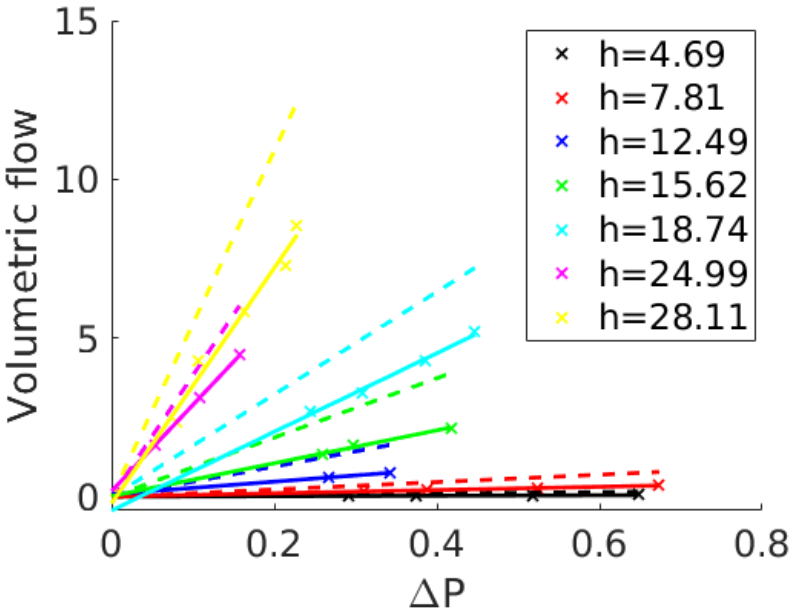


Figure A.1: Volumetric flow plotted against pressure difference over the slit pore, using the approach described in Section 3.6.2 for the pressure calculation. The *method of planes* was used for calculation of the configurational contribution to the pressure tensor. Different values of slit pore width h are tested. The crosses represent data from NEMD simulations. For each slit pore width, a linear fit is made, and the line is extrapolated to zero. The dotted lines with corresponding colors represent the solutions for planar Poiseuille flow. Reduced units are used (see Section 3.11).

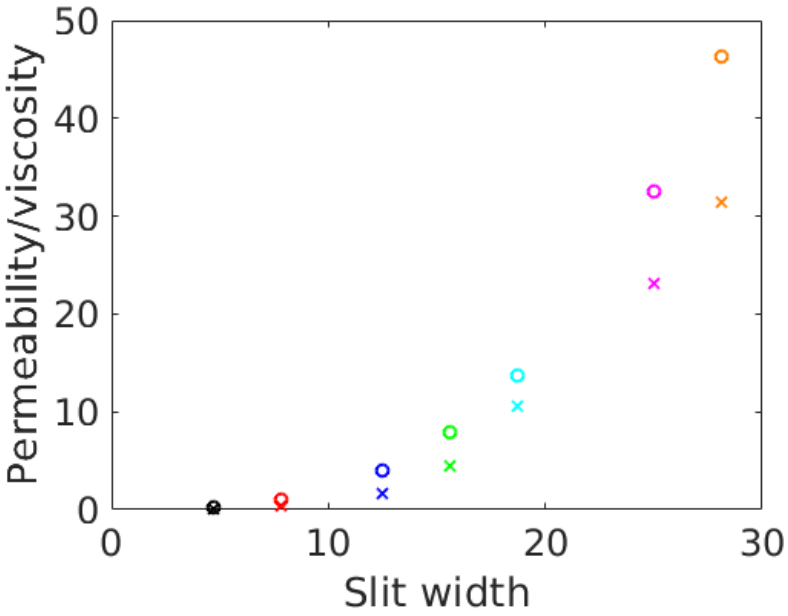


Figure A.2: *Permeability/viscosity* for different values of slit pore width, using the approach described in Section 3.6.2 for the pressure calculation. The *method of planes* was used for calculation of the configurational contribution to the pressure tensor. The data points for the NEMD data are given as crosses and the data points for the planar Poiseuille flow are given as circles with the corresponding colors.

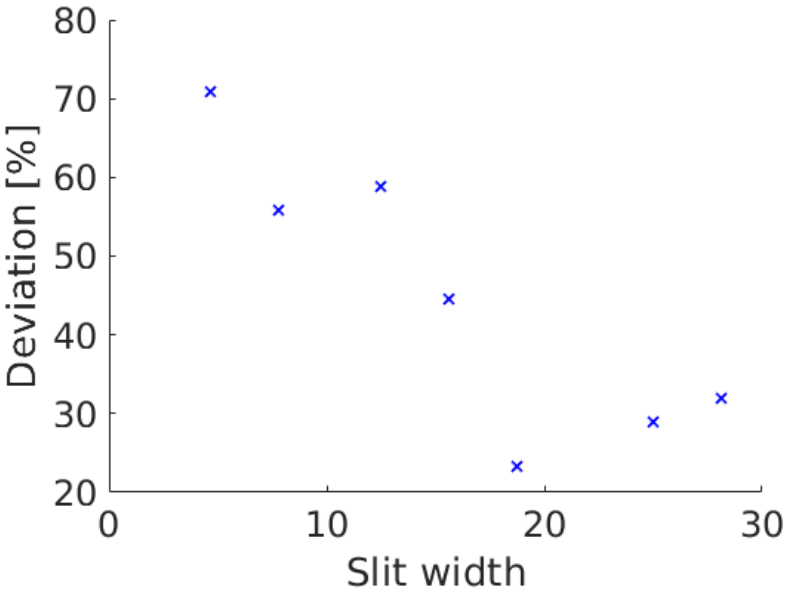


Figure A.3: Percentage relative deviation from the corresponding solution for planar Poiseuille flow, using the approach described in Section 3.6.2 for the pressure calculation. The *method of planes* was used for calculation of the configurational contribution to the pressure tensor.

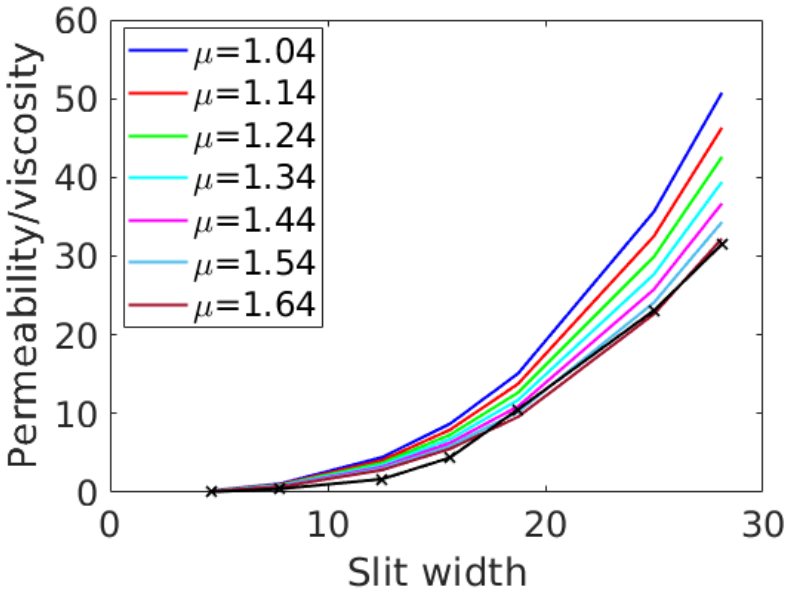


Figure A.4: Impact of the viscosity value on *permeability/viscosity* for planar Poiseuille flow. The black data points indicate the results obtained from the NEMD simulations using the *method of planes* for calculating the configurational contribution to the pressure tensor.

A.5 Porous medium with lattice constant a=25

A.5.1 Equilibrium

Figure A.5 shows the volume of grain V^r, the volume of fluid V^f and the surface area Ω^{fr} for a lattice constant $a = 25$. These properties are used to determine the values of γ^{fr} and $\hat{p}^r$ that give equal compressional energy in all REVs. The compressional energy can then be plotted, as is done in Figure A.5. To obtain the compressional energy at non-equilibrium, one must determine the integral rock pressure $\hat{p}^r$ and the surface tension γ^{fr} as functions of fluid pressure. In Figure A.6 and Figure A.7, the surface tension and integral rock pressure are obtained for three different fluid densities, corresponding to different fluid pressures. Linear regression is performed to obtain $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$.

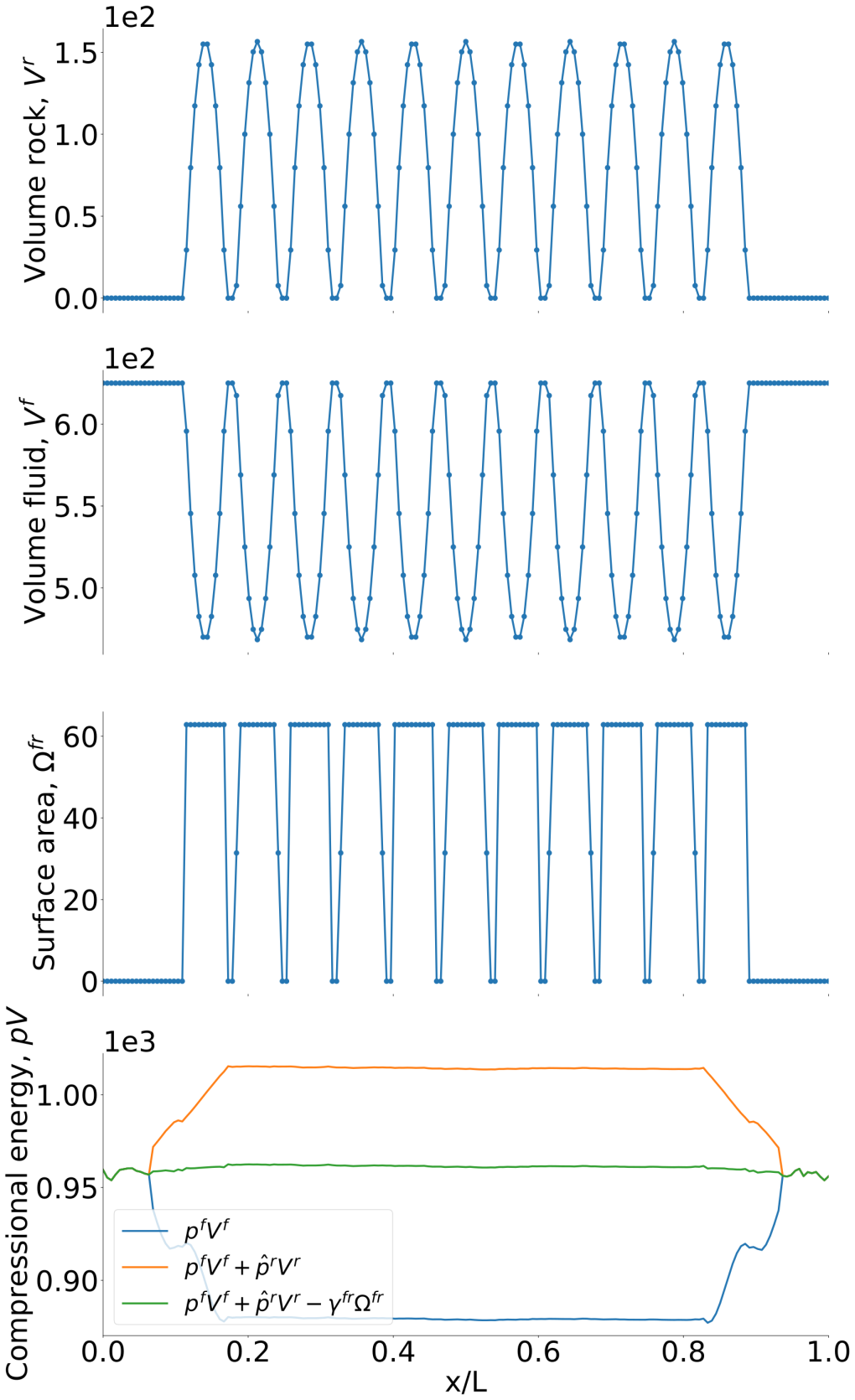


Figure A.5: Volume of grain, V^r, volume of fluid, V^f, and surface area, Ω^{fr}, for the case defined in Section 4.3.5 with lattice constant $a = 25$. The compressional energy smoothed over the REVs at equilibrium is also plotted at three different stages.

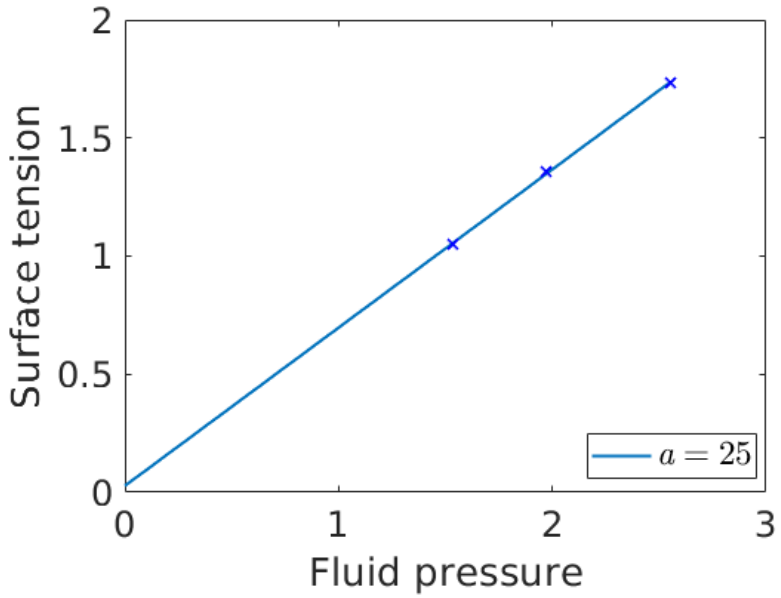


Figure A.6: Surface tension, γ^{fr} as a function of fluid pressure for the case defined in Section 4.3.5 with lattice constant $a = 25$. The different fluid pressures are generated using equilibrium simulations at different densities. The fluid densities used are 0.5, 0.55 and 0.6. Reduced units are used (see Section 3.11).

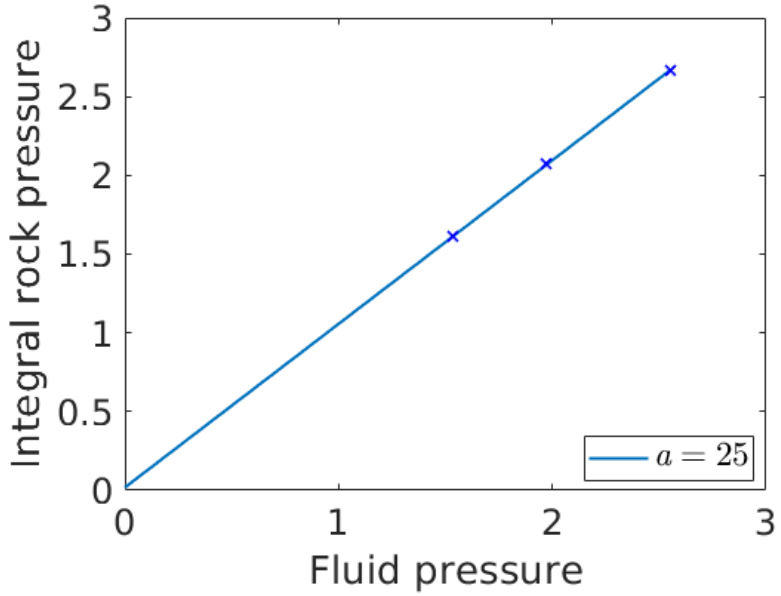


Figure A.7: Integral rock pressure, $\hat{p}^r$, as a function of fluid pressure for the case defined in Section 4.3.5 with lattice constant $a = 25$. The different fluid pressures are generated using equilibrium simulations at different densities. The fluid densities used are 0.5, 0.55 and 0.6. Reduced units are used (see Section 3.11).

A.5.2 Non-equilibrium

The compressional energy at non-equilibrium is plotted at three different stages in Figure A.8a. The profile is smoothed over the REVs and plotted in Figure A.8b. From the profile of total compressional energy in Figure A.8b, the integral pressure can be obtained by dividing on the volume of the REV. In Figure A.8c, the integral pressure smoothed over the REVs is plotted.

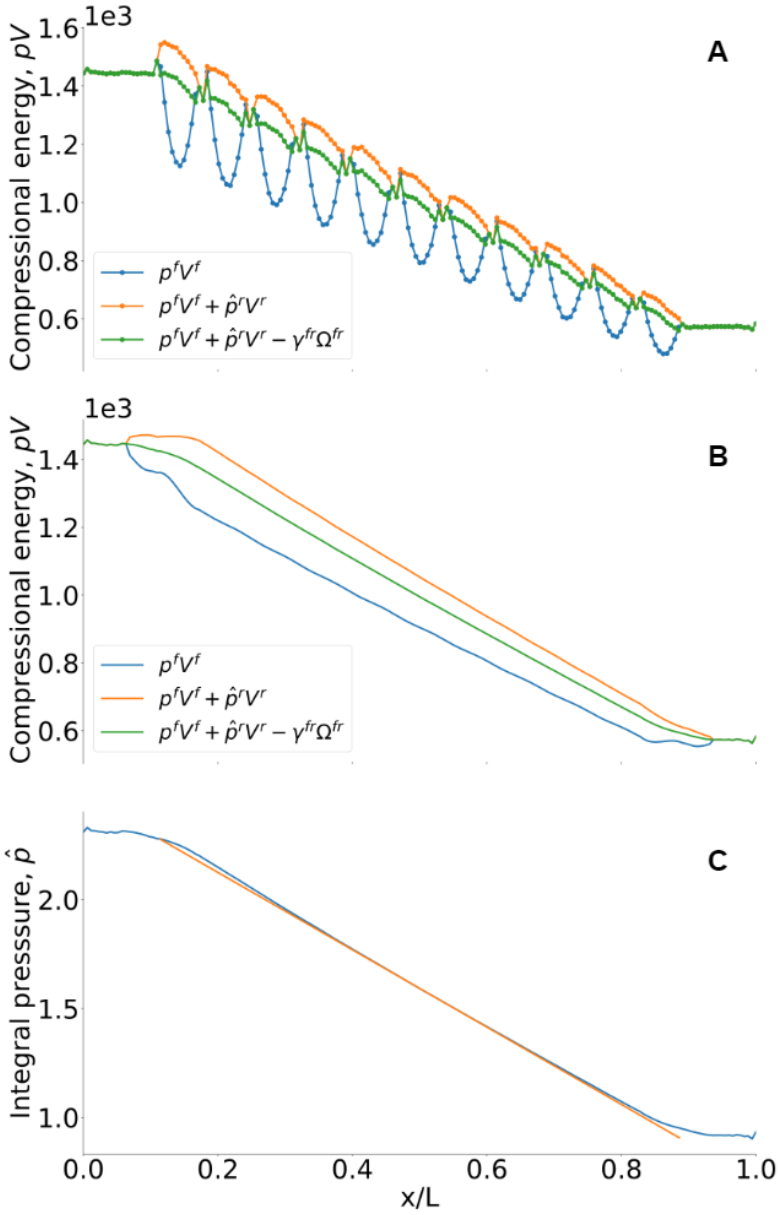


Figure A.8: Results for the non-equilibrium case defined in Section 4.3.5 with lattice constant $a = 25$. a) Compressional energy in each bin, plotted at three different stages. b) Compressional energy smoothed over the REVs, plotted at three different stages. c) Integral pressure smoothed over the REVs. The orange line is obtained by linear regression for the porous region, and is used to determine the gradient in integral pressure, $d\hat{p}/dl$, in the middle part of the porous region.

A.5.3 Permeability

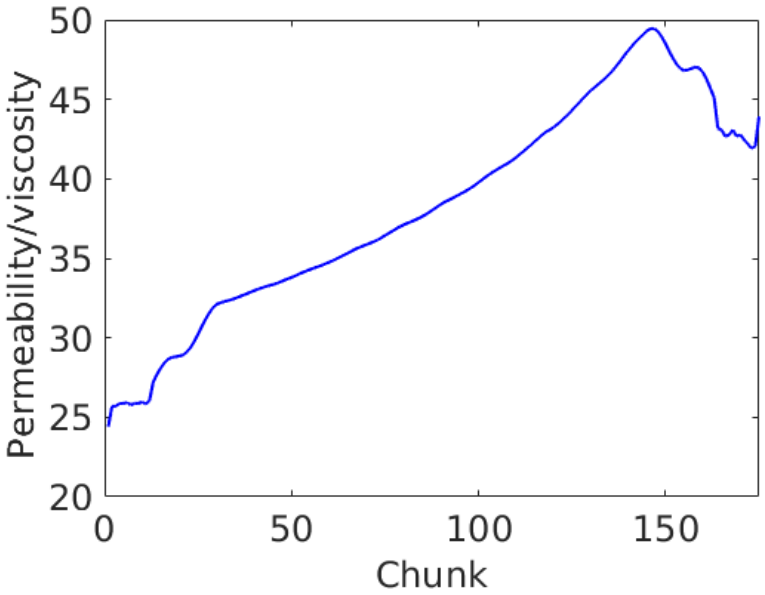


Figure A.9: The permeability divided by the viscosity through the system at non-equilibrium for the case defined in Section 4.3.5 with lattice constant $a = 25$. Reduced units are used (see Section 3.11).

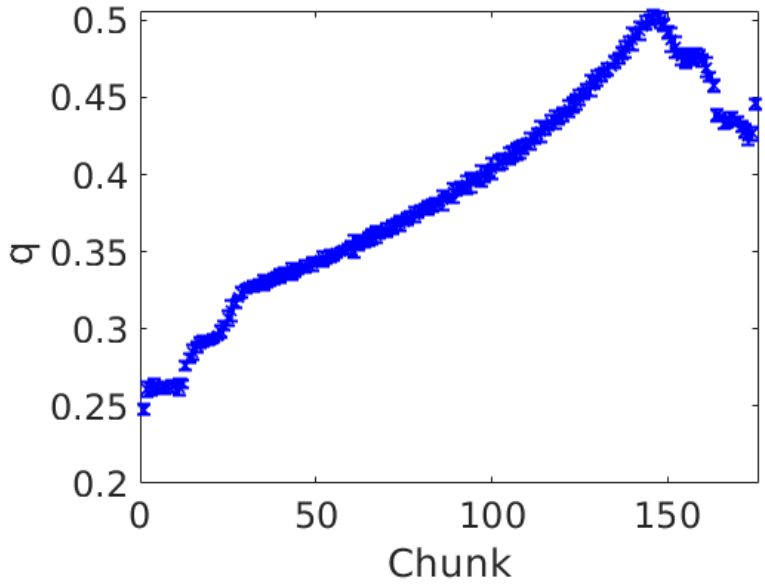


Figure A.10: Fluid flux q in each chunk for the non-equilibrium case defined in Section 4.3.5 with lattice constant $a = 25$. Reduced units are used (see Section 3.11).

In Figure A.9, the permeability divided by the viscosity is plotted through the system at non-equilibrium, using Equation (5.2). The slope of the orange line in Figure A.8c is $d\hat{p}/dl = -1.776 \pm 0.004$. The porosity is calculated by Equation (5.4) and is $\Phi = 0.89$. The only parameter that varies through the system in Equation (5.2) is q (see Figure A.10.)

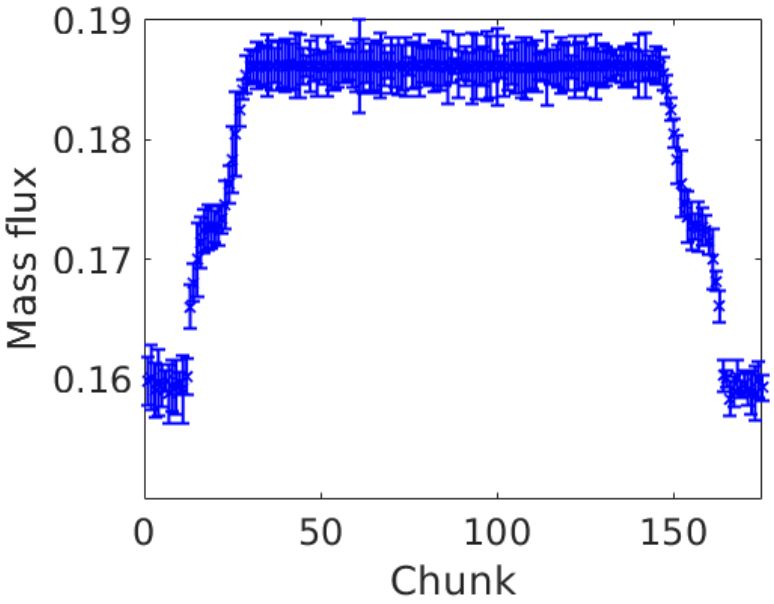


Figure A.11: Mass flux in each chunk for the case defined in Section 4.3.5 with lattice constant $a = 25$. Reduced units are used (see Section 3.11).

From Equation (5.5), we obtain the *permeability/viscosity*:

$$\frac{k}{\mu} = -\frac{\frac{\text{mass flux}}{\text{average density}}}{d\hat{p}/dx} = 34.1 \tag{A.13}$$

with a mass flux of 0.19 (see Figure A.11) and an average density of 0.48.

A.6 Porous medium with lattice constant a=30

A.6.1 Equilibrium

Figure A.12 shows the volume of grain V^r, the volume of fluid V^f and the surface area Ω^{fr} for a lattice constant $a = 30$. As before, these properties are used to determine γ^{fr} and $\hat{p}^r$. Then, the compressional energy can be plotted, as is done in Figure A.12. To obtain the compressional energy at non-equilibrium, one must determine the integral rock pressure $\hat{p}^r$ and the surface tension γ^{fr} as functions of fluid pressure. In Figure A.14 and Figure A.13, the surface tension and integral rock pressure are obtained for three different fluid densities, corresponding to different fluid pressures. Linear regression is performed to obtain $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$.

In Figure A.15 and Figure A.16, a comparison is made between $a = 20$ and $a = 30$, for the surface tension and integral rock pressure as functions of the fluid pressure.

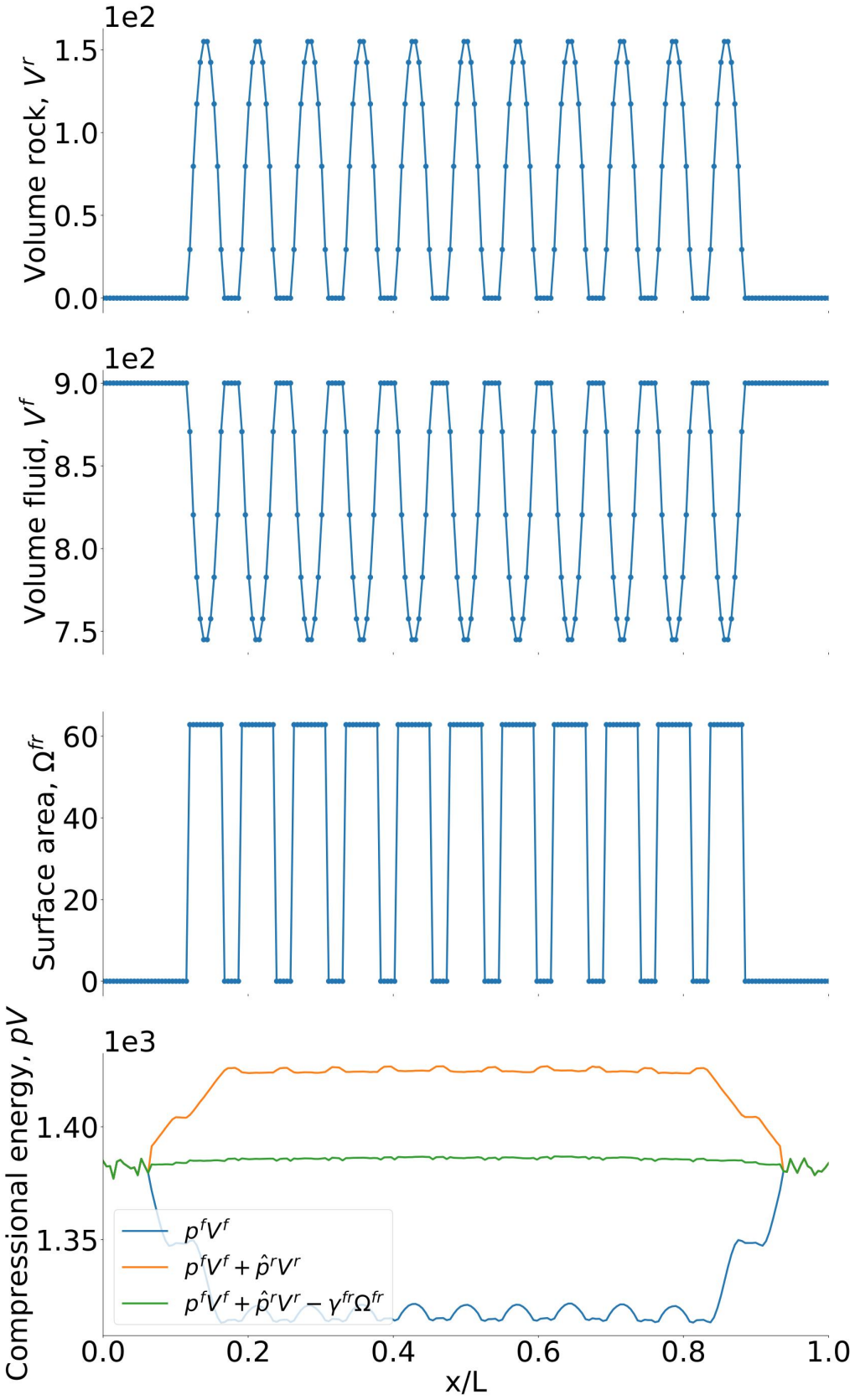


Figure A.12: Volume of grain, V^r, volume of fluid, V^f, and surface area, Ω^{fr}, for the case defined in Section 4.3.6 with lattice constant $a = 30$. The compressional energy smoothed over the REVs at equilibrium is also plotted at three different stages.

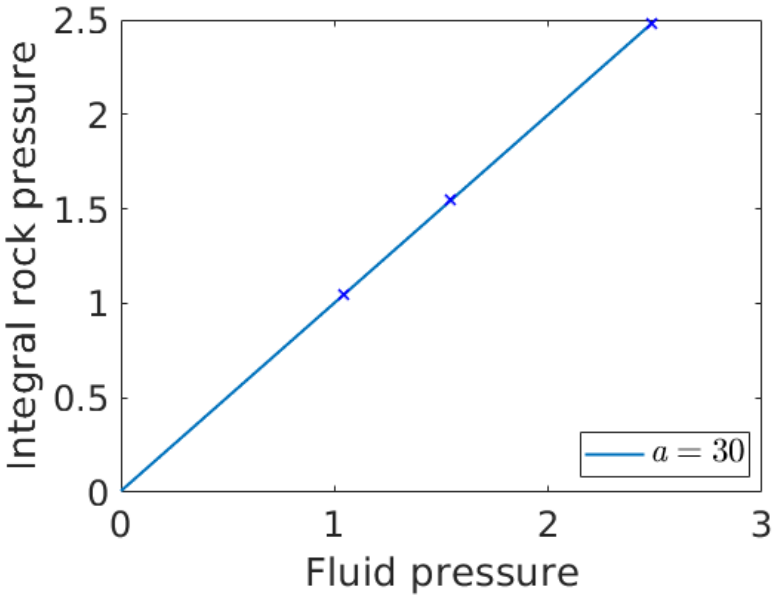


Figure A.13: Integral rock pressure, $\hat{p}^r$, as a function of fluid pressure for the case defined in Section 4.3.6 with lattice constant $a = 30$. The different fluid pressures are generated from equilibrium simulations at different densities. The points correspond to fluid densities 0.4, 0.5 and 0.6. Reduced units are used (see Section 3.11).

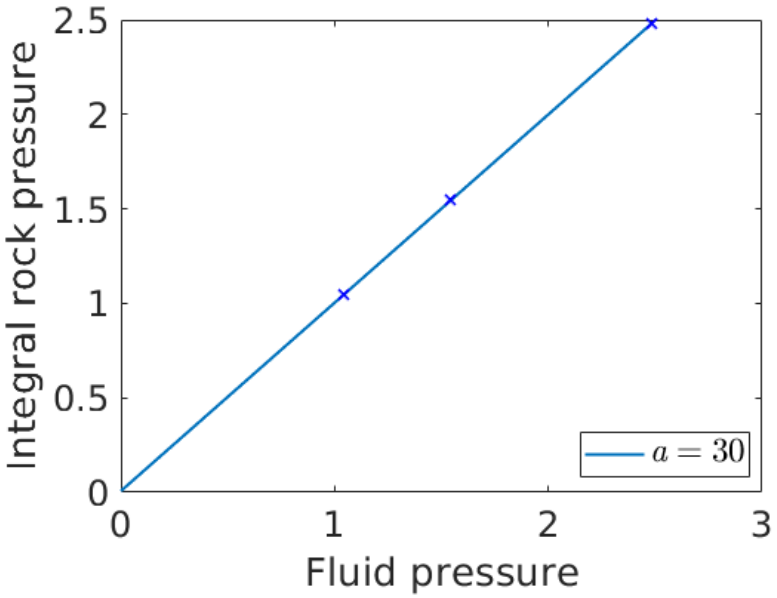


Figure A.14: Surface tension, γ^{fr}, as a function of fluid pressure for the case defined in Section 4.3.6 with lattice constant $a = 30$. The different fluid pressures are generated from equilibrium simulations at different densities. The points correspond to fluid densities 0.4, 0.5 and 0.6. Reduced units are used (see Section 3.11).

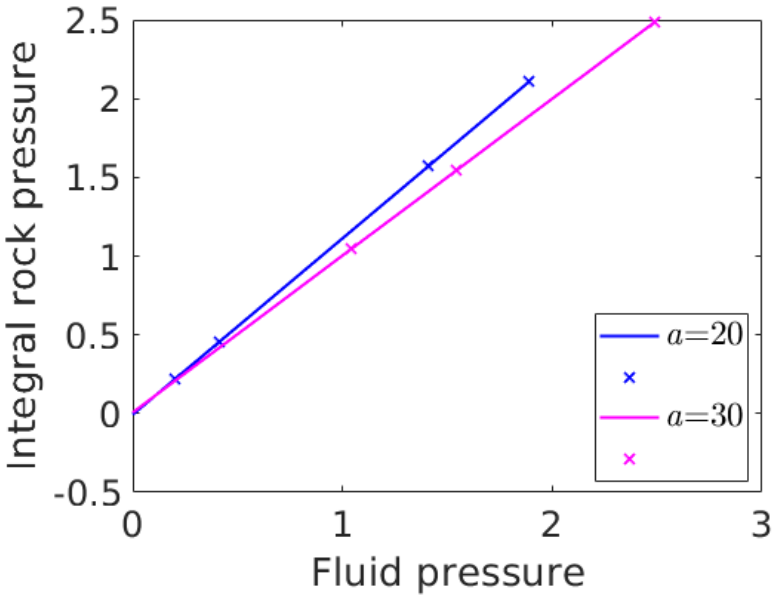


Figure A.15: Integral rock pressure, $\hat{p}^r$, as function of fluid pressure for porous media with lattice constants $a = 20$ and $a = 30$.

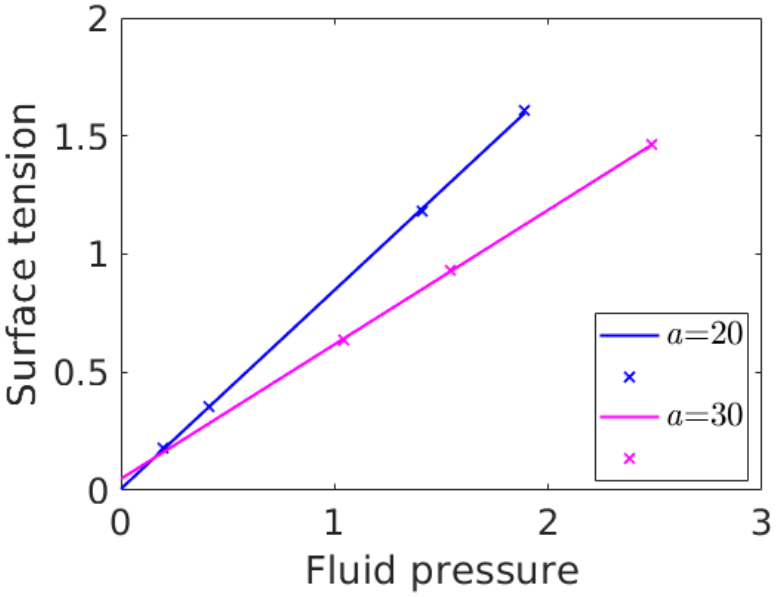


Figure A.16: Surface tension, γ^{fr}, as function of fluid pressure for porous media with lattice constants $a = 20$ and $a = 30$.

A.6.2 Non-equilibrium

The compressional energy at non-equilibrium is plotted at three different stages in Figure A.17a. The profile is smoothed over the REVs and plotted in Figure A.17b at three different stages. From the profile for total compressional energy in Figure A.17b, the integral pressure can be obtained by dividing on the volume of the REV. The integral pressure smoothed over the REVs is plotted in Figure A.17c.

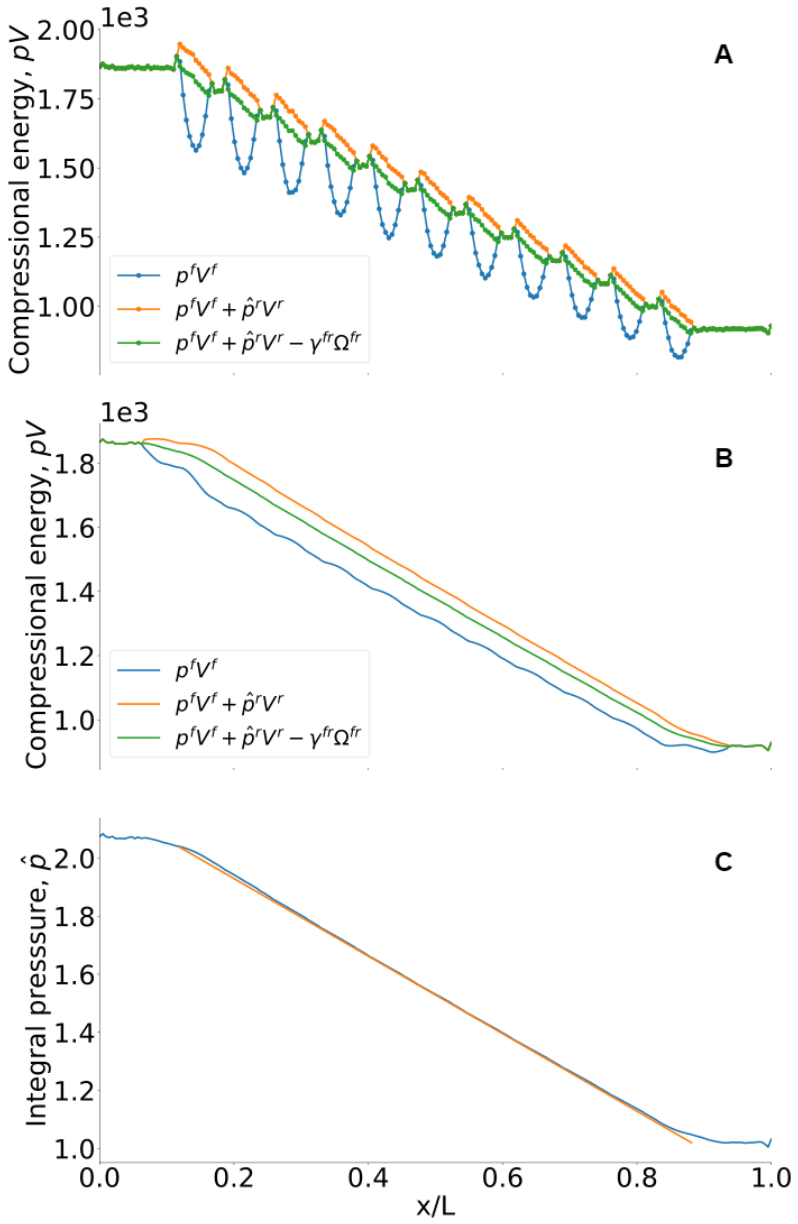


Figure A.17: Results for the non-equilibrium case defined in Section 4.3.6 with lattice constant $a = 30$. a) Compressional energy in each bin, plotted at three different stages. b) Compressional energy smoothed over the REVs, plotted at three different stages. c) Integral pressure smoothed over the REVs. The orange line is obtained by linear regression for the porous region, and is used to determine the gradient in integral pressure, $d\hat{p}/dl$, in the middle part of the porous region.

A.6.3 Permeability

In Figure A.18, the permeability divided by the viscosity is plotted through the system at non-equilibrium, using Equation (5.2). The slope of the orange line in Figure A.17c is $d\hat{p}/dl = -1.334 \pm 0.002$. The porosity is calculated by Equation (5.4) and is $\Phi = 0.94$. The only parameter that varies through the system in Equation (5.2) is the q (see Figure A.19.)

From Equation (5.5), the following *permeability/viscosity* is obtained:

$$\frac{k}{\mu} = -\frac{\frac{\text{mass flux}}{\text{average density}}}{d\hat{p}/dx} = 66.7 \tag{A.14}$$

with a mass flux of 0.24 (see Figure A.20) and an average density of 0.49.

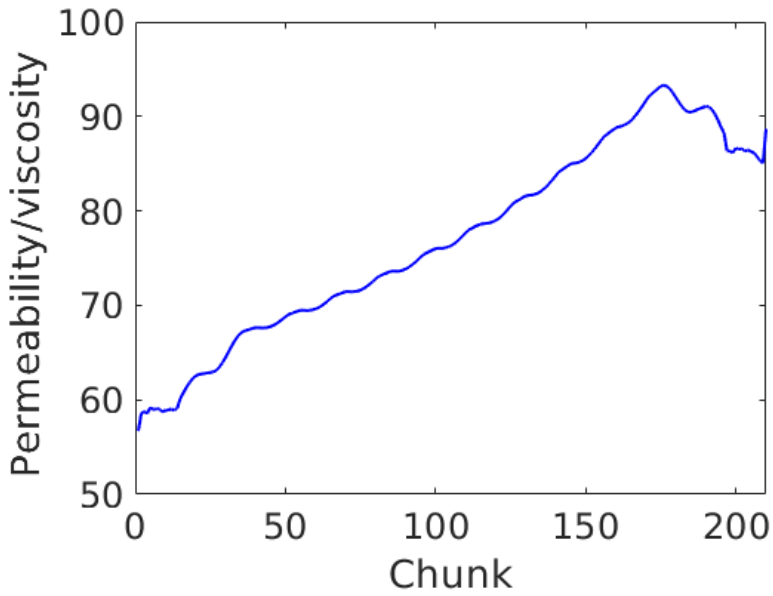


Figure A.18: The permeability divided by the viscosity through the system at non-equilibrium for the case defined in Section 4.3.6 with lattice constant $a = 30$. Reduced units are used (see Section 3.11).

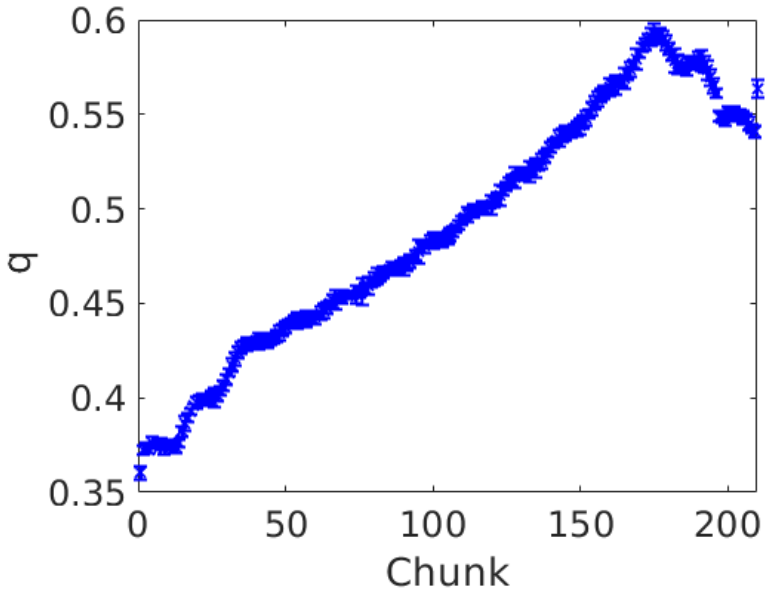


Figure A.19: Fluid flux q in each chunk for the non-equilibrium case defined in Section 4.3.6 with lattice constant $a = 30$. Reduced units are used (see Section 3.11).

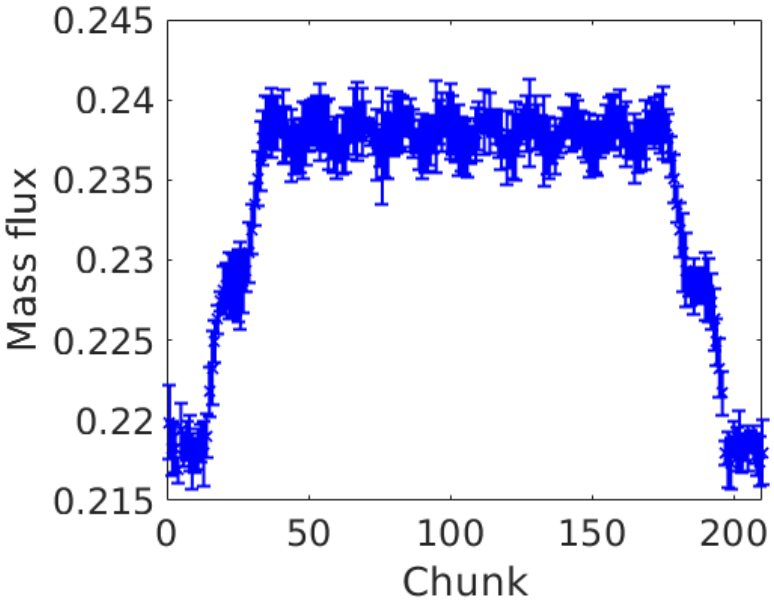


Figure A.20: Mass flux in each chunk for the case defined in Section 4.3.6 with lattice constant $a = 30$. Reduced units are used (see Section 3.11).

A.7 Porous medium with lattice constant a=40

A.7.1 Equilibrium

Figure A.21 shows the volume of grain V^r, the volume of fluid V^f and the surface area Ω^{fr} for a lattice constant $a = 40$. As before, these properties are used to determine γ^{fr} and $\hat{p}^r$. Then, the compressional energy can be plotted, as is done in Figure A.21.

To obtain the compressional energy at non-equilibrium, one must determine the integral rock pressure $\hat{p}^r$ and the surface tension γ^{fr} as functions of fluid pressure. In Figure A.22 and Figure A.23, the surface tension and integral rock pressure are obtained for three different fluid densities, corresponding to different fluid pressures. Linear regression is performed to obtain $\hat{p}^r(p^f)$ and $\gamma^{fr}(p^f)$.

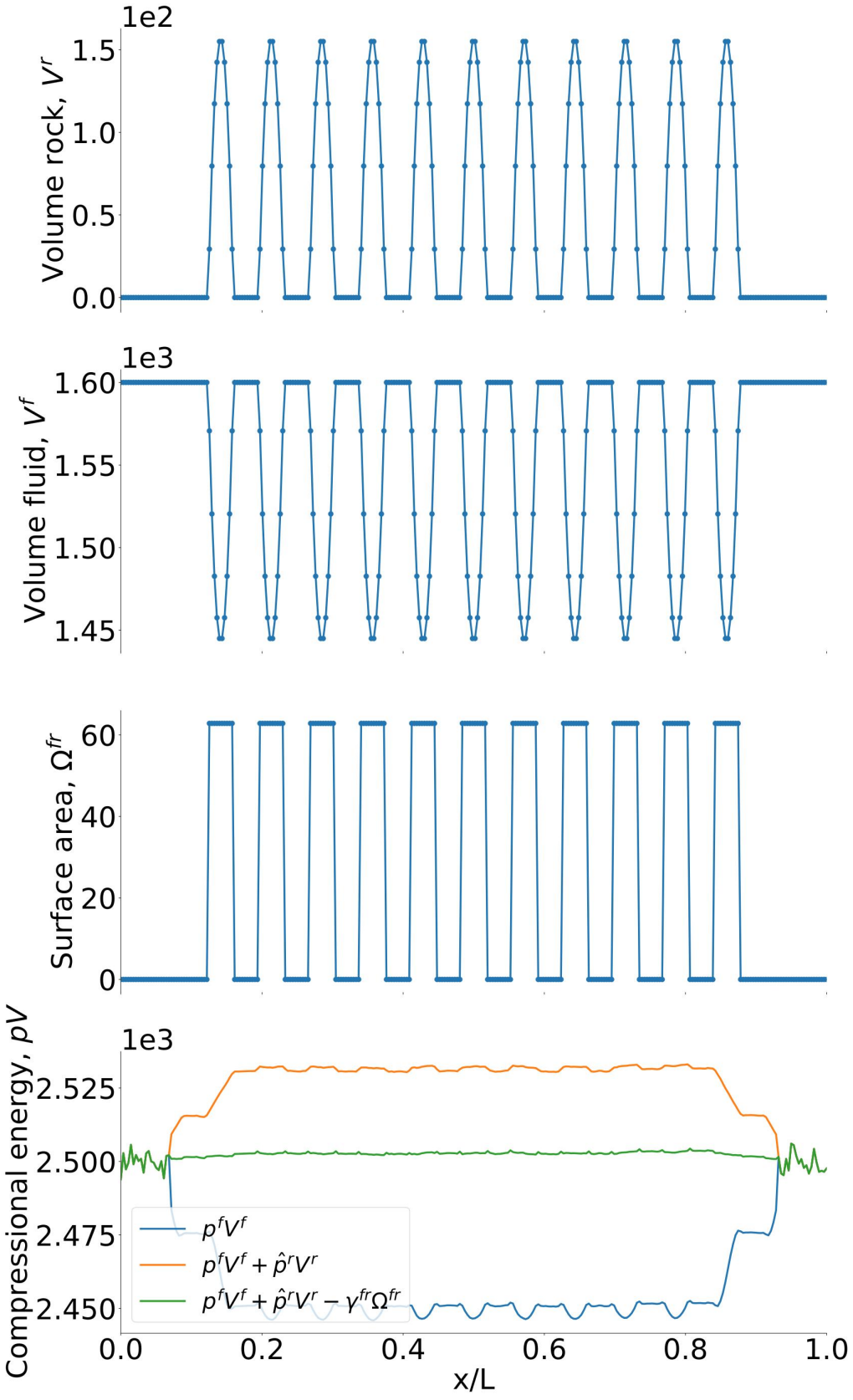


Figure A.21: Volume of grain, V^r, volume of fluid, V^f, and surface area, Ω^{fr}, for the case defined in Section 4.3.7 with lattice constant $a = 40$. The compressional energy smoothed over the REVs at equilibrium is also plotted at three different stages.

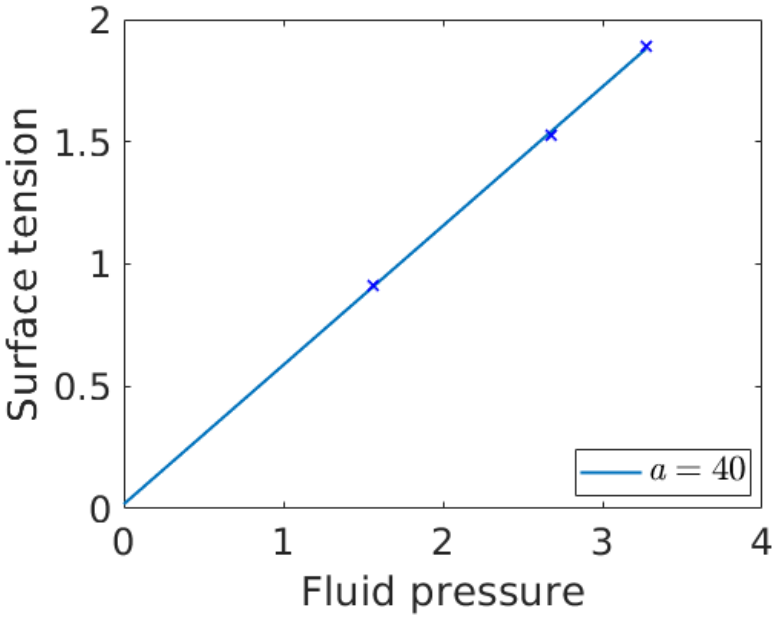


Figure A.22: Surface tension, γ^{fr}, as a function of fluid pressure for the case defined in Section 4.3.7 with lattice constant $a = 40$. The different fluid pressures are generated from equilibrium simulations at different densities. The points correspond to fluid densities 0.5, 0.61 and 0.65. Reduced units are used (see Section 3.11).

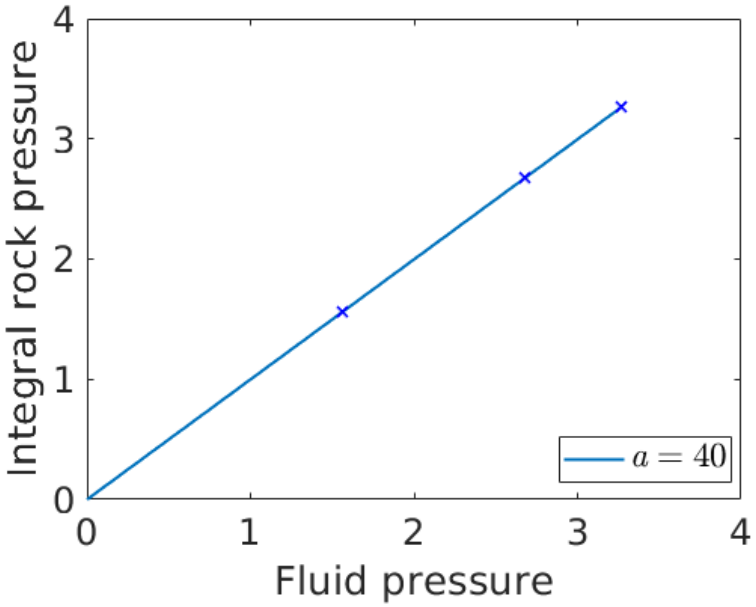


Figure A.23: Integral rock pressure, $\hat{p}^r$, as a function of fluid pressure for the base case defined in Section 4.3.7 with lattice constant $a = 40$. The different fluid pressures are generated from equilibrium simulations at different densities. The fluid densities used are 0.5, 0.61 and 0.65. Reduced units are used (see Section 3.11).

A.7.2 Non-equilibrium

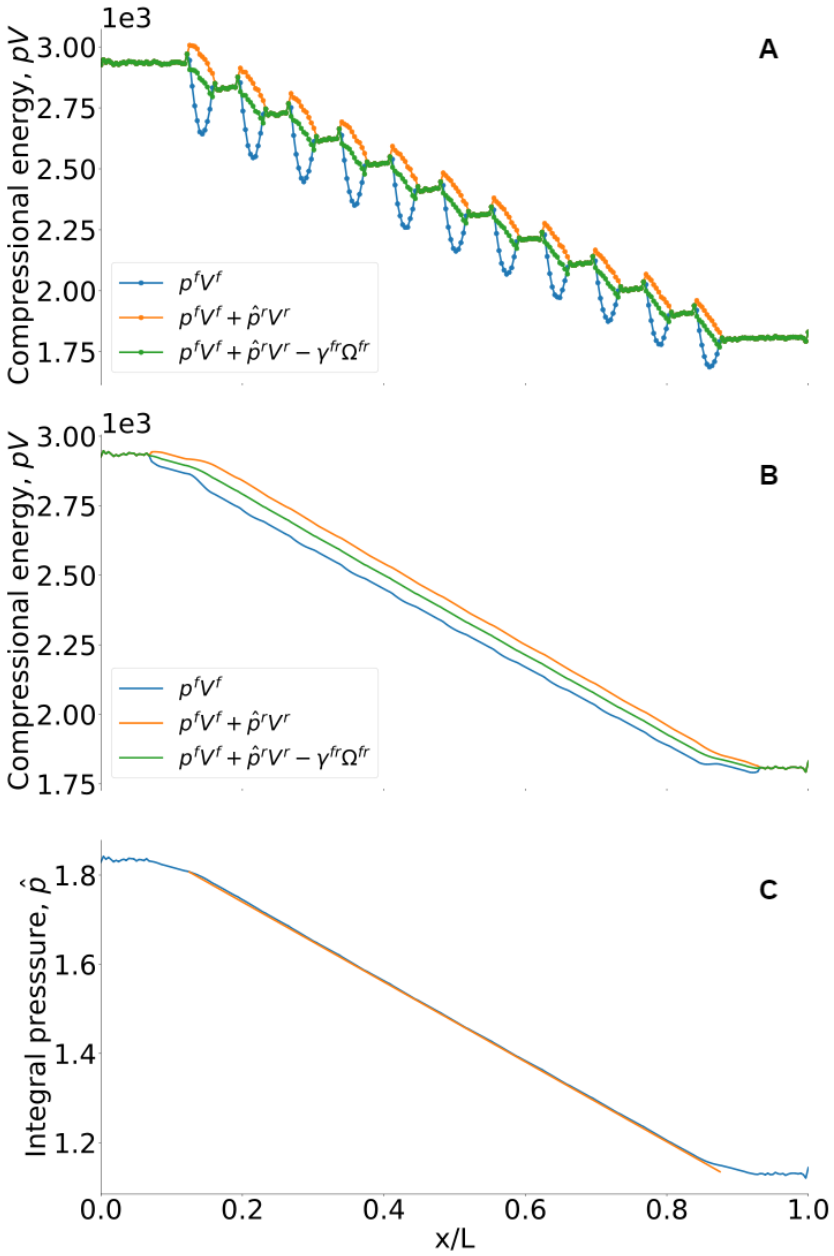


Figure A.24: Results for the non-equilibrium case defined in Section 4.3.5 with lattice constant $a = 40$. a) Compressional energy in each bin, plotted at three different stages. b) Compressional energy smoothed over the REVs, plotted at three different stages. c) Integral pressure smoothed over the REVs. The orange line is obtained by linear regression for the porous region, and is used to determine the gradient in integral pressure, $d\hat{p}/dl$, in the middle part of the porous region.

The compressional energy at non-equilibrium is plotted at three different stages in Figure A.24a. The profile is smoothed over the REVs and plotted in Figure A.24b at three different stages. From the profile for total compressional energy in Figure A.24b, the integral pressure can be obtained by dividing on the volume of the REV. The integral pressure smoothed over the REVs is plotted in Figure A.24c.

A.7.3 Permeability

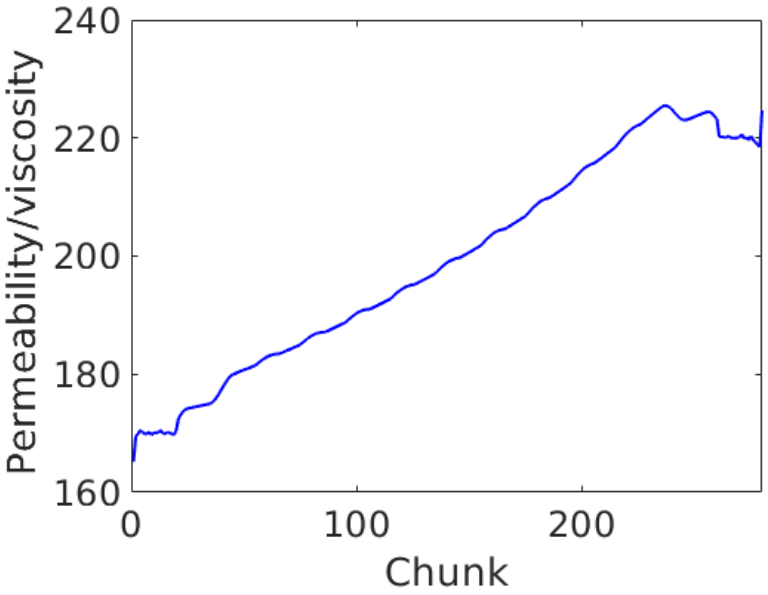


Figure A.25: The permeability divided by the viscosity through the system at non-equilibrium for the case defined in Section 4.3.7 with lattice constant $a = 40$. Reduced units are used (see Section 3.11).

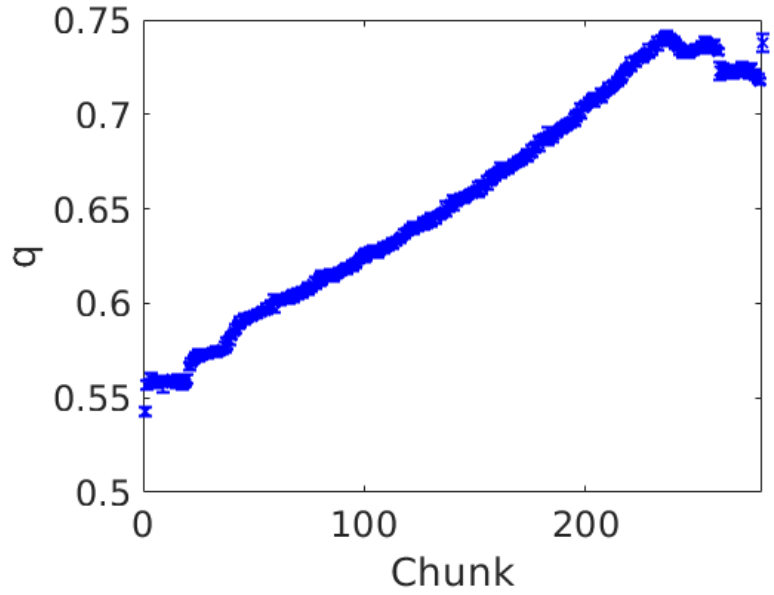


Figure A.26: Fluid flux q in each chunk at non-equilibrium for the case defined in Section 4.3.7 with lattice constant $a = 40$. Reduced units are used (see Section 3.11).

In Figure A.25, the permeability divided by the viscosity is plotted through the system at non-equilibrium, using Equation (5.2). The slope of the orange line in Figure A.24c is $d\hat{p}/dl = -0.896 \pm 0.001$. The porosity is calculated by Equation (5.4) and is $\Phi = 0.97$. The only parameter that varies through the system in Equation (5.2) is q (see Figure A.26).

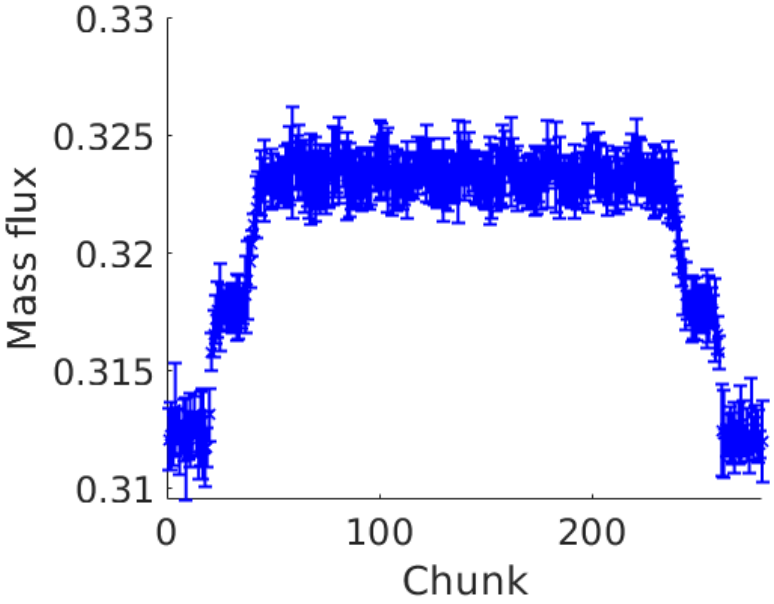


Figure A.27: Mass flux in each chunk for the case defined in Section 4.3.7 with lattice constant $a = 40$. Reduced units are used (see Section 3.11).

From Equation (5.5), we obtain a *permeability/viscosity* equal to:

$$\frac{k}{\mu} = -\frac{\frac{\text{mass flux}}{\text{average density}}}{d\hat{p}/dx} = 179.2 \tag{A.15}$$

with a mass flux of 0.32 (see Figure A.27) and an average density of 0.49.

A.8 Input-scripts and post-processing scripts

A.8.1 Slit pore, input script for generating restart-file

```
##################################################
############### Initialization ###################
##################################################

lattice                 fcc 0.75


variable L equal 40
# Length of the box (in z-direction)

variable W equal 10
# Length of the box in positive x and y direction

variable D equal 0.75
# density of fluid

variable Dw equal 1.05
# density of solid

variable ch equal 1
# Number of units cells in each chunk

variable N  equal 87287
# seed

variable ref1  equal 0.01
# Probability related to the right reflective particle boundary

variable ref2  equal 0.05
# Probability related to the left reflective particle boundary

# Lennard-Jones-spline Variables
variable eps11 equal 1.0
variable sig11 equal 1.0
variable alp11 equal 1.0


variable eps12 equal 1.0
variable sig12 equal 1.0
variable alp12 equal 1.0



variable eps22 equal 1.0
variable sig22 equal 1.0
```

```
variable alp22 equal 1.0



#################################################
######### Regions/Groups/Potentials#########
#################################################


region                  box block -${W} ${W} -${W} ${W} 0 ${L}
#Define region for simulation box

region                  solid block -${W} ${W} -${W} ${W} 10 30
#Define region for solid bock

region                  del_cen block -${W} ${W} -5 5 9 31
#Define region for deleting the middle of the simulation box

region                  cen block -${W} ${W} -5 5 11 29
#Define region for filling in the middle of the simulation box
   with fluid

region                  del_bot block -${W} ${W} -${W} -7 12 28
#Define region for deleting bottom part of solid block (too
   speed up calculations)

region                  del_top block -${W} ${W} 7 ${W} 12 28
#Define region for deleting top part of solid block (too speed
   up calculations)

region                  liq1 block -${W} ${W} -${W} ${W} 0 10
#Define region for liquid at the left of the solid

region                  liq2 block -${W} ${W} -${W} ${W} 30 40
#Define region for liquid at the right of the solid

region                  del_liq_l block -2 2 -2 2 2 4
region                  del_liq_r block -2 2 -2 2 38 39

create_box              2 box
#Create simulation box

#   Create Wall
lattice                 fcc ${Dw}
#Create fcc lattice with density Dw for solid particles

create_atoms            2 region solid
```

```
  #Create atoms in the solid block region

delete_atoms            region del_cen
#Delete solid atoms in the center

delete_atoms            region del_bot
#Delete solid atoms at the bottom

delete_atoms            region del_top
#Delete solid atoms at the top

#   Create Liquid
lattice                 fcc $D
#Create fcc lattice with density D for the liquid particles

create_atoms            1 region liq1
#Create liquid atoms in the liquid region to the left

create_atoms            1 region liq2
 #Create liquid atoms in the liquid region to the right

create_atoms            1 region cen
#Create liquid atoms in the center

delete_atoms            region del_liq_l
delete_atoms            region del_liq_r



# Group fluid and pore atoms
group                   fluid type 1
#Group fluid atoms

group                   p2 type 2
#Group solid atoms


mass                    1 1.0
#Set mass of fluid atoms to 1

mass                    2 1.0
#Set mass of solid atoms to 1


# Define the Lennard-Jones potentials between the atoms

pair_style              lj/spline
pair_coeff              1 1 ${eps11} ${sig11} ${alp11}  0 0.0
```

```
#Set the coefficients for fluid-fluid

pair_coeff              1 2 ${eps12} ${sig12} ${alp12}  0 0.0
#Set the coefficients for fluid-solid

pair_coeff              2 2 ${eps22} ${sig22} ${alp22}  0 0.0
#Set the coefficients for solid-solid

delete_atoms            overlap 1.0 fluid p2 #Delete the fluid
   and solid atoms that overlap.


################################################
######## Neighbors/Computation Balance #######
################################################

neighbor                0.3 bin
neigh_modify            every 20 delay 0 check no
# Updating Neighbor List

neigh_modify            exclude type 2 2
#The solid particles are stationary. Their positions and
   velocities are not updated during the simulation.

# Optimizing Computation per processor

fix                     balance fluid balance 1000 1.15 shift xy
    20 1.15


##############################
######## Computation #######
##############################

velocity                fluid create 0.90 $N
#Set the initial velocity of the atoms equal to the temperature




#########################################
########## Positions ###################
#########################################


dump                    dump all custom 100000 tmp.dump id type
```

```
x y z


#################################
########## Run ##################
#################################

fix                     1 fluid nvt temp 0.90 0.90 0.02
#Keep number of particles, volume and temperature constant.

timestep                0.002
#Set the timestep size to 0.002

run                     500000
#Run the simulations for 500000 timesteps


# Write restart file
write_restart           restart.lg
```

A.8.2 Slit pore, input script

```
################################################
############## Initialization ##################
################################################

read_restart ../restart.lg

lattice                 fcc 0.75

variable L equal 40
variable W equal 10
variable D equal 0.75
variable Dw equal 1.05
variable ch equal 1
variable N  equal 87287
variable ref1  equal 0.01
variable ref2  equal 0.05

# Lennard-Jones-spline Variables
variable eps11 equal 1.0
variable sig11 equal 1.0
variable alp11 equal 1.0



variable eps12 equal 1.0
variable sig12 equal 1.0
```

```
variable alp12 equal 1.0


variable eps22 equal 1.0
variable sig22 equal 1.0
variable alp22 equal 1.0



################################################
######### Regions/Groups/Potentials#########
################################################


region                  box block -${W} ${W} -${W} ${W} 0 ${L}




# Group fluid and pore atoms
group                   fluid type 1
group                   p2 type 2


mass                    1 1.0
mass                    2 1.0



# Define the Lennard-Jones potentials between the atoms

pair_style              lj/spline
pair_coeff              1 1 ${eps11} ${sig11} ${alp11}  0 0.0
pair_coeff              1 2 ${eps12} ${sig12} ${alp12}  0 0.0
pair_coeff              2 2 ${eps22} ${sig22} ${alp22}  0 0.0



################################################
######## Neighbors/Computation Balance #######
################################################

neighbor                0.3 bin
neigh_modify            every 20 delay 0 check no
neigh_modify            exclude type 2 2

# Optimizing Computation per processor
```

```
fix                     balance fluid balance 1000 1.15 shift xy
    20 1.15


################################################
############### Prepare Chunks #################
################################################


# Dividing box into bins(chunks)
compute                 chunk_f fluid chunk/atom bin/1d z lower
   ${ch}
#Create chunks in the z-direction. In z-direction, each chunk
   contains one unit cell.

#######################################
######## Reflective Membrane ######
#######################################

fix                     reflect1 fluid wall/rpm zhi EDGE ${ref1}
     units box
#Create reflective particle boundary at the rigth edge of the
   simulation box.

fix                     reflect2 fluid wall/rpm zlo EDGE ${ref2}
     units box
#Create reflective particle boundary at the left edge of the
   simulation box.


################################################
############## Compute / Dump Temperature ##
################################################

### Temperature with COM ###
compute                 temp_fluid fluid temp

fix                     dump_t_f fluid ave/chunk 1000 1 1000
   chunk_f &
                        temp file dump_t_f.out norm none



################################################
############## Compute / Dump PRESSURE ######
################################################
```

```
### Pressure without COM ###

compute                 press fluid stress/atom temp_fluid
#Computes per-atom stress tensor for each atom in a group.
   Pressure is the same as the stress-tensor for the entire
   system.



fix                     dump_p_f fluid ave/chunk 1000 1 1000
   chunk_f &
                        c_press[*] file dump_p_fluid.out norm
                           none


### Pressure with COM correction ###

# Configurational Part #


compute                 press_fvir fluid stress/atom NULL virial
#The virial is the configurational part of the pressure. It is
   due to interactions between particles.


fix                     dump_p_vir fluid ave/chunk 1000 1 1000
   chunk_f &
                        c_press_fvir[*] file dump_p_vir.out norm
                            none


# Square Velocity #

variable                sq_vx atom vx*vx
variable                sq_vy atom vy*vy
variable                sq_vz atom vz*vz

fix                     dump_sq fluid ave/chunk 1000 1 1000
   chunk_f &
                        v_sq_vx[*] v_sq_vy[*] v_sq_vz[*] file
                           sq_v.out norm none

# Center of Mass Velocity #

compute                 com fluid vcm/chunk chunk_f
#Define a computation that calculates the center-of-mass
   velocity for multiple chunks of atoms.
```

```
fix                     com_ave fluid ave/time 1000 1 1000 c_com
   [*] file com.out mode vector


##############################
########  Velocity ########
##############################

fix                     dump_vel fluid ave/chunk 1000 1 1000
   chunk_f &
                        vx vy vz file dump_vel.out norm none


#########################################
############## 2D Velocity Field ######
#########################################

compute                 chunk_2d_zy fluid chunk/atom bin/2d z
   lower 2.0 y lower 2.0


compute                 chunk_2d_zx fluid chunk/atom bin/2d z
   lower 2.0 x lower 2.0



fix                     dump_vel_zx fluid ave/chunk 1000 1 1000
   chunk_2d_zx &
                        vx vy vz file vel_zx.out norm none


fix                     dump_vel_zy fluid ave/chunk 1000 1 1000
   chunk_2d_zy &
                        vx vy vz file vel_zy.out norm none

#########################################
########## Positions ################
#########################################

### write output file ###
dump                    dump all custom 200000 tmp.dump id type
   x y z



##############################
########## Run ###############
```

```
################################

fix                       1 fluid nvt temp 0.90 0.90 0.02
timestep                  0.002
run                       2000000


# Write restart file
write_restart             restart.lg
```

A.8.3 Slit pore, post-processing script for mass and density

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% Mass %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
M = dlmread('../restart_out/dump_vel.out',' ',3,0);
chunk = M(1,2);
A_f = zeros(chunk);


uc_t = 1.74716;
ch = 1;

dx = ch*uc_t; %chunk thickness
W = 10*uc_t;

%% Volume of each chunk

%Left bulk phase
for u = 1:10
      dx_tot = dx * u;

      A_f(u) = 2*W*2*W;
      V_f(u) =  A_f(u) * dx;

end

%Slit pore
for u = 11:29
      dx_tot = dx * u;
      A_f(u) = 2*W*2*9.37076;
      V_f(u) =  A_f(u) * dx;

end

%Right bulk phase
for u = 30:40
```

```
        dx_tot = dx * u;

        A_f(u) = 2*W*2*W;
        V_f(u) =  A_f(u) * dx;

end



%% Block Averaging Procedure
st = 1;
en = 10;


for k = st:en

s_tart = 0.00 + (k-1)*0.1;
e_nd = 0.00 + k*0.1;


n = 1;
s = 1;
l = size(M,1);
s_steady = 0;
part = l/(chunk+1);


mass_steady = zeros(chunk);
curr_steady = zeros(chunk);

N_total_steady = zeros(chunk);
volflow_steady=zeros(chunk);


while s <= part


    for i = 1:chunk


            if s >= s_tart*part && s < e_nd*part

            mass_steady(i) = mass_steady(i) + M(n+i,8)/V_f(i);
            curr_steady(i) = curr_steady(i) + M(n+i,8);

            volflow_steady(i)=volflow_steady(i)+M(n+i, 8)*A_f(i)*
               A_f(i)/M(n+i, 5);
```

```
            N_total_steady(i) = N_total_steady(i)  + M(n+i,5)/V_f(
                i);

              if i == chunk
              s_steady = s_steady + 1;
              end
            end


    end

        n = n + chunk +1;
        i = 1;
        s = s+1;



end


mass_steady_v(1:chunk,k) = mass_steady(1:chunk,1)/s_steady;
Number_F_v(1:chunk,k) = N_total_steady(1:chunk)/s_steady;
curr_steady_v(1:chunk,k) = curr_steady(1:chunk,1)/s_steady;
volflow_steady_v(1:chunk, k)=curr_steady(1:chunk,1)/s_steady;

end


for i = 1:chunk
    mass(i) = mean(mass_steady_v(i,5:en));
    Num(i) = mean(Number_F_v(i,5:en));
    curr(i) = mean(curr_steady_v(i,5:en));
    volflow(i)=mean(curr_steady_v(i,5:en));

    mass_std(i) = std(mass_steady_v(i,5:en));
    Num_std(i) = std(Number_F_v(i,5:en));
    curr_std(i) = std(curr_steady_v(i,5:en));
    volflow_std(i)=std(curr_steady_v(i,5:en));
end



%% Plotting

x = linspace(1,chunk,chunk);


figure(1);
```

```
rectangle('Position', [10 0 20 0.08], 'EdgeColor',[.9 .9 .9], '
    FaceColor', [.9 .9 .9])
hold on
errorbar(x,mass(1,:),mass_std(1,:),'vertical','x','color','blue
    ','LineWidth',1.5);
xlabel('Chunk','fontsize', 18);
ylabel('Mass flux','fontsize', 18);
box
set(gca,'FontSize',18)
hold off

figure(2);
rectangle('Position', [10 50 20 15], 'EdgeColor',[.9 .9 .9], '
    FaceColor', [.9 .9 .9])
hold on
errorbar(x,curr(1,:),curr_std(1,:),'vertical','x','color','blue
    ','LineWidth',1.5);
xlabel('Chunk','fontsize', 18);
ylabel('Mass current','fontsize', 18);
box
set(gca,'FontSize',18)
hold off


figure(3);
rectangle('Position', [10 0 20 1], 'EdgeColor',[.9 .9 .9], '
    FaceColor', [.9 .9 .9])
hold on
errorbar(x,Num(1,:),Num_std(1,:),'vertical','x','color','blue','
    LineWidth',1.5);
xlabel('Chunk', 'fontsize', 18);
ylabel('Fluid density', 'fontsize', 18);
box
set(gca,'FontSize',18)
hold off


slit_pore_mass=mass(1, 12:28);
mass_flux_slitpore=mean(slit_pore_mass)
std_mass=std(slit_pore_mass)
Q=mean(volflow(1, 12:28))
std_Q=std(volflow(1, 12:28))
mass_left=mass(1, 2:10);
mass_right=mass(1, 30:39);
mass_bulk=(mean(mass_left)+mean(mass_right))/2
```

A.8.4 Slit pore, post-processing script for pressure

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%  PRESS %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all

P_f = dlmread(sprintf('../restart_out/dump_p_fluid.out',1),'
    ',3,0);
P_c = dlmread(sprintf('../restart_out/dump_p_vir.out',1),'
    ',3,0);
SQV = dlmread('../restart_out/sq_v.out','  ',3,0);
COM = dlmread('../restart_out/com.out','  ',3,0);
M = dlmread('../restart_out/dump_vel.out','  ',3,0);


%% Add here the obtained center of mass velocity of the first
    run
comx_s
    =[-0.000841977472499998;-0.00247912859300000;-0.00216857371169000;

-0.00345033738600000;-0.00368481507880000;-0.00498350404300000;
-0.00493610153480000;-0.00262584159863000;-0.00245722753360000;
-0.000418514288189999;0.000100975940300000;-3.58626853000019e
    -05;
-0.000429194938239999;-0.000306703727199999;0.00104789230711000;
-0.00163243138100000;-0.00198992369004000;-0.00235051701120000;
0.000194537305400001;-0.00277965946025000;-0.00117342116330000;
-0.00110650076620000;-0.00203835234334000;0.000855322499399999;
0.000854326022900000;0.00180478429980000;0.00199046271170000;
0.00112102648360990;0.00150552782960000;0.00237341075500000;
0.00249303186520000;0.00272917758580000;0.00113189349860000;
0.000258297046400000;0.00143992420550000;0.00100410939400000;
0.000715356813200000;-0.00121754851184800;-2.64912628999990e-05;
-0.000355739849900002]

comy_s
    =[0.000641526616420000;0.00223623610500000;0.000497581526399999;

0.00131193280230000;0.00184977061390000;0.000190403312725998;
0.00184243141850000;0.000384107771700000;0.000665134029969999;
-1.89347987000001e
    -05;-0.000234730901330001;0.000859769157200001;
-0.00241131342390000;0.000371820925000001;0.00142937210980000;
0.000444135345500001;-0.000389465769600000;0.00114012557240000;
3.78720117899988e-05;0.000537275802000002;-0.000330470278500000;
-0.000335707853600000;-0.00362521917820000;-0.000276149461100000;

-0.00232032840687000;-0.000310429217800000;-0.000926280177000000;
```

```
0.00122054021180000;-0.00137006156440000;0.000202092459000000;
0.000232507242099998;-0.000694498195080000;-0.000593944292600000;

-0.00163139502170000;-0.000496034857100001;-0.00137643709260000;
-0.000105488907000000;6.88062306700004e
    -05;-0.000591650348960001;
-0.00106514156684000]

comz_s
    =[0.0347505353620000;0.0370297429336000;0.0360053720963000;
0.0370061855270000;0.0369013298360000;0.0359891963510000;
0.0365040507900999;0.0365754285005000;0.0372035341561000;
0.0353576013370000;0.0724462380490000;0.0704826364580000;
0.0715579377300000;0.0718646893336000;0.0731630622330000;
0.0730348074940000;0.0706202067140000;0.0705295869730000;
0.0747277858490000;0.0709363792105000;0.0702163429420001;
0.0721470777000001;0.0742375056426000;0.0736890942100000;
0.0702790216357001;0.0716341871359999;0.0740855132930001;
0.0760092544420001;0.0744231669670000;0.0714772711075000;
.0422435126930000;0.0371497465620000;0.0375023322037000;
0.0376907377572000;0.0382451755795000;0.0371909402500000;0
.0382524802090000;0.0379659250984000;0.0399909290710000;
0.0367827729683000]




chunk = P_f(1,2);
A_f = zeros(chunk);


uc_t = 1.74716;
ch = 1;

dx = ch*uc_t; %chunk thickness
W = 10*uc_t;




%% Volume of each chunk


for u = 1:10
      dx_tot = dx * u;

      A_f(u) = 2*W*2*W;
      V_f(u) =  A_f(u) * dx;
```

```
end

for u = 11:29
      dx_tot = dx * u;
      A_f(u) = 2*W*2*9.37076;
      V_f(u) =  A_f(u) * dx;

end

for u = 30:40
      dx_tot = dx * u;

      A_f(u) = 2*W*2*W;
      V_f(u) =  A_f(u) * dx;

end



%% Block Averaging Procedure
st = 1;
en = 10;


for k = st:en

s_tart = 0.00 + (k-1)*0.1;
e_nd = 0.00 + k*0.1;

np = 1;
sp = 1;
lp = size(P_f,1);
p_steady = 0;
partp = lp/(chunk+1);

press_total_f = zeros(chunk);
press_steady_f = zeros(chunk);
press_steady_fxx = zeros(chunk);
press_steady_fyy = zeros(chunk);
press_steady_fzz = zeros(chunk);



press_steady_fc = zeros(chunk);
press_steady_fxxc = zeros(chunk);
press_steady_fyyc = zeros(chunk);
```

```
press_steady_fzzc = zeros(chunk);


N_total = zeros(chunk);
N_total_steady = zeros(chunk);


sq_x = zeros(chunk);
sq_y = zeros(chunk);
sq_z = zeros(chunk);

sq_x_corr = zeros(chunk);
sq_y_corr = zeros(chunk);
sq_z_corr = zeros(chunk);


while sp <= partp


    for ip = 1:chunk
        P_sum = -((P_f(np+ip,6)+P_f(np+ip,7)+P_f(np+ip,8))/(3*
            V_f(ip)));
        P_xx = - P_f(np+ip,6)/V_f(ip);
        P_yy = - P_f(np+ip,7)/V_f(ip);
        P_zz = - P_f(np+ip,8)/V_f(ip);

        P_sumc = -((P_c(np+ip,6)+P_c(np+ip,7)+P_c(np+ip,8))/(3))
            ;
        P_xxc = - P_c(np+ip,6);
        P_yyc = - P_c(np+ip,7);
        P_zzc = - P_c(np+ip,8);

        N_new = P_f(np+ip,5)/V_f(ip);



          if sp >= s_tart*partp && sp < e_nd*partp
           press_steady_f(ip) = press_steady_f(ip) + P_sum;
           press_steady_fxx(ip) = press_steady_fxx(ip) + P_xx;
           press_steady_fyy(ip) = press_steady_fyy(ip) + P_yy;
           press_steady_fzz(ip) = press_steady_fzz(ip) + P_zz;

           press_steady_fc(ip) = press_steady_fc(ip) + P_sumc;
           press_steady_fxxc(ip) = press_steady_fxxc(ip) + P_xxc
              ;
           press_steady_fyyc(ip) = press_steady_fyyc(ip) + P_yyc
              ;
           press_steady_fzzc(ip) = press_steady_fzzc(ip) + P_zzc
```

```
            ;

        sq_x(ip) = sq_x(ip) + SQV(np+ip,6);
        sq_y(ip) = sq_y(ip) + SQV(np+ip,7);
        sq_z(ip) = sq_z(ip) + SQV(np+ip,8);

        sq_x_corr(ip) = sq_x_corr(ip) + SQV(np+ip,6) - 2.*
            comx_s(ip,1).*M(np+ip,6) + P_c(np+ip,5).*comx_s(ip
            ,1).^2;
        sq_y_corr(ip) = sq_y_corr(ip) + SQV(np+ip,7)  - 2.*
            comy_s(ip,1).*M(np+ip,7) + P_c(np+ip,5).*comy_s(ip
            ,1).^2;
        sq_z_corr(ip) = sq_z_corr(ip) + SQV(np+ip,8)  - 2.*
            comz_s(ip,1).*M(np+ip,8) + P_c(np+ip,5).*comz_s(ip
            ,1).^2;


        px_own(ip) = sq_x(ip) + press_steady_fxxc(ip);
        py_own(ip) = sq_y(ip) + press_steady_fyyc(ip);
        pz_own(ip) = sq_z(ip) + press_steady_fzzc(ip);

        px_own_corr (ip) = (sq_x_corr(ip) + press_steady_fxxc
            (ip))/V_f(ip);
        py_own_corr (ip) = (sq_y_corr(ip) + press_steady_fyyc
            (ip))/V_f(ip);
        pz_own_corr (ip) = (sq_z_corr(ip) + press_steady_fzzc
            (ip))/V_f(ip);

        N_total_steady(ip) = N_total_steady(ip) + N_new;

         if ip == chunk
         p_steady = p_steady + 1;
         end
       end


    end

        np = np + chunk +1;
        ip = 1;
        sp = sp+1;

      % mass(sp) = mean(mass_new(12:18));

end


press_steady_f_v(1:chunk,k) = press_steady_f(1:chunk,1)/p_steady
```

```
    ;
press_steady_fxx_v(1:chunk,k) = press_steady_fxx(1:chunk,1)/
    p_steady;
press_steady_fyy_v(1:chunk,k) = press_steady_fyy(1:chunk,1)/
    p_steady;
press_steady_fzz_v(1:chunk,k) = press_steady_fzz(1:chunk,1)/
    p_steady;


press_steady_f_vc(1:chunk,k) = press_steady_fc(1:chunk,1)/
    p_steady;
press_steady_fxx_vc(1:chunk,k) = press_steady_fxxc(1:chunk,1)/
    p_steady;
press_steady_fyy_vc(1:chunk,k) = press_steady_fyyc(1:chunk,1)/
    p_steady;
press_steady_fzz_vc(1:chunk,k) = press_steady_fzzc(1:chunk,1)/
    p_steady;

sq_fxx_vc(1:chunk,k) = sq_x(1:chunk,1)/p_steady;
sq_fyy_vc(1:chunk,k) = sq_y(1:chunk,1)/p_steady;
sq_fzz_vc(1:chunk,k) = sq_z(1:chunk,1)/p_steady;

px_own_v(1:chunk,k) = px_own(1,1:chunk)/p_steady;
py_own_v(1:chunk,k) = py_own(1,1:chunk)/p_steady;
pz_own_v(1:chunk,k) = pz_own(1,1:chunk)/p_steady;

px_own_corr_v(1:chunk,k) = px_own_corr(1,1:chunk)/p_steady;
py_own_corr_v(1:chunk,k) = py_own_corr(1,1:chunk)/p_steady;
pz_own_corr_v(1:chunk,k) = pz_own_corr(1,1:chunk)/p_steady;

Number_F_v(1:chunk,k) = N_total_steady(1:chunk)/p_steady;



end


for i = 1:chunk
    p_sum(i) = mean(press_steady_f_v(i,5:en));
    p_xx(i) = mean(press_steady_fxx_v(i,5:en));
    p_yy(i) = mean(press_steady_fyy_v(i,5:en));
    p_zz(i) = mean(press_steady_fzz_v(i,5:en));
    Num(i) = mean(Number_F_v(i,5:en));

    po_xx(i) = mean(px_own_v(i,5:en));
    po_yy(i) = mean(py_own_v(i,5:en));
    po_zz(i) = mean(pz_own_v(i,5:en));
```

```
    po_xx_corr(i) = mean(px_own_corr_v(i,5:en));
    po_yy_corr(i) = mean(py_own_corr_v(i,5:en));
    po_zz_corr(i) = mean(pz_own_corr_v(i,5:en));

    p_sumc(i) = mean(press_steady_f_vc(i,5:en));
    p_xxc(i) = mean(press_steady_fxx_vc(i,5:en));
    p_yyc(i) = mean(press_steady_fyy_vc(i,5:en));
    p_zzc(i) = mean(press_steady_fzz_vc(i,5:en));

    sq_xxc(i) = mean(sq_fxx_vc(i,5:en));
    sq_yyc(i) = mean(sq_fyy_vc(i,5:en));
    sq_zzc(i) = mean(sq_fzz_vc(i,5:en));


    p_sum_std(i) = std(press_steady_f_v(i,5:en));
    p_xx_std(i) = std(press_steady_fxx_v(i,5:en));
    p_yy_std(i)  = std(press_steady_fyy_v(i,5:en));
    p_zz_std(i)  = std(press_steady_fzz_v(i,5:en));
    Num_std(i) = std(Number_F_v(i,5:en));


    p_sum_stdc(i) = std(press_steady_f_vc(i,5:en));
    p_xx_stdc(i) = std(press_steady_fxx_vc(i,5:en));
    p_yy_stdc(i)  = std(press_steady_fyy_vc(i,5:en));
    p_zz_stdc(i)  = std(press_steady_fzz_vc(i,5:en));


    sq_xx_stdc(i) = std(sq_fxx_vc(i,5:en));
    sq_yy_stdc(i)  = std(sq_fyy_vc(i,5:en));
    sq_zz_stdc(i)  = std(sq_fzz_vc(i,5:en));

    po_xx_std(i) = std(px_own_v(i,5:en));
    po_yy_std(i) = std(py_own_v(i,5:en));
    po_zz_std(i) = std(pz_own_v(i,5:en));

    po_xx_corr_std(i) = std(px_own_corr_v(i,5:en));
    po_yy_corr_std(i) = std(py_own_corr_v(i,5:en));
    po_zz_corr_std(i) = std(pz_own_corr_v(i,5:en));

end


%% Plotting

x = linspace(1,chunk,chunk);
```

```
figure(6);
rectangle('Position', [10 0.2 20 0.8], 'EdgeColor',[.9 .9 .9], '
   FaceColor', [.9 .9 .9])
hold on
errorbar(x,p_sum(1,:),p_sum_std(1,:),'vertical','x','color
   ',[0.8500, 0.3250, 0.0980],'LineWidth',1.5,'MarkerSize',10);
errorbar(x,p_zz(1,:),p_sum_std(1,:),'vertical','x','color','
   green','LineWidth',1.5,'MarkerSize',10);
errorbar(x,po_zz_corr(1,:),po_zz_corr_std(1,:),'vertical','x','
   color','blue','LineWidth',1.5,'MarkerSize',10);

xlabel('Chunk', 'fontsize', 18);
ylabel('Fluid pressure', 'fontsize', 18);
legend('P_{sum}','P_{zz}','P_{zz, corr}', 'fontsize', 17)
set(gca,'FontSize',18)
box
hold off

figure(7);
rectangle('Position', [10 0.3 20 0.7], 'EdgeColor',[.9 .9 .9], '
   FaceColor', [.9 .9 .9])
hold on
errorbar(x,Num(1,:),Num_std(1,:),'vertical','x','color',[0.8500,
    0.3250, 0.0980],'LineWidth',1.5);
title('Fluid Density');
xlabel('Chunk');
ylabel('Density');
grid on
box
hold off


figure(8);
rectangle('Position', [10 -400 20 1000], 'EdgeColor',[.9 .9 .9],
    'FaceColor', [.9 .9 .9])
hold on
errorbar(x,p_sumc(1,:),p_sum_stdc(1,:),'vertical','x','color
   ',[0.8500, 0.3250, 0.0980],'LineWidth',1.5);
errorbar(x,p_xxc(1,:),p_sum_stdc(1,:),'vertical','x','color','
   green','LineWidth',1.5);
errorbar(x,p_yyc(1,:),p_sum_stdc(1,:),'vertical','x','color','
   blue','LineWidth',1.5);
errorbar(x,p_zzc(1,:),p_sum_stdc(1,:),'vertical','x','color','
   magenta','LineWidth',1.5);
xlabel('Chunk', 'fontsize', 18);
ylabel('Configurational part', 'fontsize', 18);
legend('P_{sum}','P_{xx}','P_{yy}','P_{zz}', 'fontsize', 17 )
```

```
set(gca,'FontSize',18)
box
hold off

p_left= (po_zz_corr(1,2)+po_zz_corr(1,3)+po_zz_corr(1,4)+
    po_zz_corr(1,5)+po_zz_corr(1,6)+po_zz_corr(1,7)+po_zz_corr
    (1,8))/7
p_right=(po_zz_corr(1,33)+po_zz_corr(1,34)+po_zz_corr(1,35)+
    po_zz_corr(1,36)+po_zz_corr(1,37)+po_zz_corr(1,38)+po_zz_corr
    (1,39))/7
deltaP=p_left-p_right
Pavg=(p_left+p_right)/2
```

A.8.5 Slit pore, post-processing script for velocity

```
clear all

%% Load
V_zy = dlmread(sprintf('../restart_out/vel_zy.out'),' ',3,0); %
    Region shortened in x direction

V_zx = dlmread(sprintf('../restart_out/vel_zx.out'),' ',3,0); %
    Region shortened in y direction


%% General
chunk = V_zy(1,2);
lp = size(V_zy,1);
partp = lp/(chunk+1);
sp = 1;
np = 1;
p_steady = 0;

% Volume
d = 2*1.74741;
W = 13*1.74741;

V = d*d*W;


Vz_zy = zeros(partp);
Vy_zy =  zeros(partp);


Vz_zx = zeros(partp);
Vx_zx =  zeros(partp);
Vy_zx =  zeros(partp);

Vz_zy_steady = zeros(partp);
```

```
Vy_zy_steady =  zeros(partp);
Vx_zy_steady = zeros(partp);

Vz_zx_steady = zeros(partp);
Vx_zx_steady =  zeros(partp);
Vy_zx_steady =  zeros(partp);

while sp <= partp



for ip = 1:chunk
%
      Vz_zy(1,ip) =  Vz_zy(1,ip) + V_zy(np+ip,9);
      Vy_zy(1,ip) =  Vy_zy(1,ip) + V_zy(np+ip,8);

      Vz_zx(1,ip) =  Vz_zx(1,ip) + V_zx(np+ip,9);
      Vy_zx(1,ip) =  Vy_zx(1,ip) + V_zx(np+ip,8);


      if sp >= 0.05*partp

      if V_zy(np+ip,6) > 0
      Vz_zy_steady(1,ip) =  Vz_zy_steady(1,ip) + V_zy(np+ip,9)./
         V_zy(np+ip,6); %
      Vy_zy_steady(1,ip) =  Vy_zy_steady(1,ip) + V_zy(np+ip,8)./
         V_zy(np+ip,6); %
      Vx_zy_steady(1,ip) =  Vx_zy_steady(1,ip) + V_zy(np+ip,7)./
         V_zy(np+ip,6); %
      end


       if V_zy(np+ip,6) == 0
      Vx_zy_steady(1,ip) =  Vx_zy_steady(1,ip) + V_zy(np+ip,7);
      Vz_zy_steady(1,ip) =  Vz_zy_steady(1,ip) + V_zy(np+ip,9);
      Vy_zy_steady(1,ip) =  Vy_zy_steady(1,ip) + V_zy(np+ip,8);
       end

         if V_zx(np+ip,6) > 0
      Vz_zx_steady(1,ip) =  Vz_zx_steady(1,ip) + V_zx(np+ip,9)./
         V_zx(np+ip,6); %
      Vy_zx_steady(1,ip) =  Vy_zx_steady(1,ip) + V_zx(np+ip,8)./
         V_zx(np+ip,6); %
      Vx_zx_steady(1,ip) =  Vx_zx_steady(1,ip) + V_zx(np+ip,7)./
         V_zx(np+ip,6); %
      end
```

```
      if V_zx(np+ip,6) == 0
     Vx_zx_steady(1,ip) =  Vx_zx_steady(1,ip) + V_zx(np+ip,7);
     Vz_zx_steady(1,ip) =  Vz_zx_steady(1,ip) + V_zx(np+ip,9);
     Vy_zx_steady(1,ip) =  Vy_zx_steady(1,ip) + V_zx(np+ip,8);
     end




      if ip == chunk
            p_steady = p_steady + 1;
     end
    end


end

        np = np + chunk +1;
        ip = 1;
        sp = sp+1;

end

Vz_zy_steady =  Vz_zy_steady./(p_steady);
Vy_zy_steady =  Vy_zy_steady./(p_steady);
Vx_zy_steady =  Vx_zy_steady./(p_steady);

Vz_zx_steady =  Vz_zx_steady./(p_steady);
Vx_zx_steady =  Vx_zx_steady./(p_steady);
Vy_zx_steady =  Vy_zx_steady./(p_steady);



Y2(1:200) = V_zy(2:201,5);
Z2(1:200) = V_zy(2:201,4);


figure(10);
quiver(Z2(1:200),Y2(1:200),Vz_zy_steady(1,1:200),Vy_zy_steady
   (1,1:200),1,'linewidth', 1.5 );
set(gca,'FontSize',18)
hold off


X1(1:200) = V_zx(2:201,5);
Z1(1:200) = V_zx(2:201,4);
```

```
figure(11);
quiver(Z1(1:200),X1(1:200),Vz_zx_steady(1,1:200),Vx_zx_steady
    (1,1:200),1, 'linewidth', 1.5);
set(gca,'FontSize',18)
hold off


%Plot the velocity profile for a cross-section of the slit pore
%Choose a cross-section in the middle of the slit pore
%Pick out component 103-108 from the vector Vz_zy_steady
z=Z1(100);   %The z-value in the middle of the slit pore
plot(Y2(103:108), Vz_zy_steady(1, 103:108), 'linewidth', 1.5);
hold on
scatter(Y2(103:108), Vz_zy_steady(1, 103:108), 'blue', 'marker',
     'x','linewidth', 1.5);
Y2(103:108)
Vz_zy_steady(1, 103:108)
xlabel('y', 'Fontsize', 18);
ylabel('V_z', 'FontSize', 18);
set(gca,'FontSize',18)
```

A.8.6 Slit pore, post-processing script for temperature

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%  Temperature %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
SQV = dlmread('../restart_out/sq_v.out',' ',3,0);
T = dlmread('../restart_out/dump_t_f.out',' ',3,0);
COM = dlmread('../restart_out/com.out',' ',3,0);
M = dlmread('../restart_out/dump_vel.out',' ',3,0);



%% Add here the obtained center of mass velocity of the first
    run

comx_s
    =[-0.000841977472499998;-0.00247912859300000;-0.00216857371169000;

-0.00345033738600000;-0.00368481507880000;-0.00498350404300000;
-0.00493610153480000;-0.00262584159863000;-0.00245722753360000;
-0.000418514288189999;0.000100975940300000;-3.58626853000019e
    -05;
-0.000429194938239999;-0.000306703727199999;0.00104789230711000;
-0.00163243138100000;-0.00198992369004000;-0.00235051701120000;
0.000194537305400001;-0.00277965946025000;-0.00117342116330000;
-0.00110650076620000;-0.00203835234334000;0.000855322499399999;
```

```
0.000854326022900000;0.00180478429980000;0.00199046271170000;
0.00112102648360990;0.00150552782960000;0.00237341075500000;
0.00249303186520000;0.00272917758580000;0.00113189349860000;
0.000258297046400000;0.00143992420550000;0.00100410939400000;
0.000715356813200000;-0.00121754851184800;-2.64912628999990e-05;
-0.000355739849900002]
comy_s
    =[0.000641526616420000;0.00223623610500000;0.000497581526399999;

0.00131193280230000;0.00184977061390000;0.000190403312725998;
0.00184243141850000;0.000384107771700000;0.000665134029969999;
-1.89347987000001e
    -05;-0.000234730901330001;0.000859769157200001;
-0.00241131342390000;0.000371820925000001;0.00142937210980000;
0.000444135345500001;-0.000389465769600000;0.00114012557240000;
3.78720117899988e-05;0.000537275802000002;-0.000330470278500000;
-0.000335707853600000;-0.00362521917820000;-0.000276149461100000;

-0.00232032840687000;-0.000310429217800000;-0.000926280177000000;

0.00122054021180000;-0.00137006156440000;0.000202092459000000;
0.000232507242099998;-0.00069449819508000;-0.000593944292600000;

-0.00163139502170000;-0.000496034857100001;-0.00137643709260000;
-0.000105488907000000;6.88062306700004e
    -05;-0.000591650348960001;
-0.00106514156684000]
comz_s
    =[0.0347505353620000;0.0370297429336000;0.0360053720963000;
0.0370061855270000;0.0369013298360000;0.0359891963510000;
0.0365040507900999;0.0365754285005000;0.0372035341561000;
0.0353576013370000;0.0724462380490000;0.0704826364580000;
0.0715579377300000;0.0718646893336000;0.0731630622330000;
0.0730348074940000;0.0706202067140000;0.0705295869730000;
0.0747277858490000;0.0709363792105000;0.0702163429420001;
0.0721470777000001;0.0742375056426000;0.0736890942100000;
0.0702790216357001;0.0716341871359999;0.0740855132930001;
0.0760092544420001;0.0744231669670000;0.0714772711075000;
0.0422435126930000;0.0371497465620000;0.0375023322037000;
0.0376907377572000;0.0382451755795000;0.0371909402500000;
0.0382524802090000;0.0379659250984000;0.0399909290710000;
0.0367827729683000]

chunkq = SQV(1,2);
nq = 1;
sq = 1;
lq = size(SQV,1);
s_steadyq = 0;
```

```
partq = lq/(chunkq+1);

sq_x = zeros(chunkq);
sq_y = zeros(chunkq);
sq_z = zeros(chunkq);
Nq_steady = zeros(chunkq);

temp_steady = zeros(chunkq);


N_steady_t = zeros(chunkq);


vx_steady = zeros(chunkq);
vy_steady = zeros(chunkq);
vz_steady = zeros(chunkq);

com_x = zeros(chunkq);
com_y = zeros(chunkq);
com_z = zeros(chunkq);



sq_x_corr = zeros(chunkq);
sq_y_corr = zeros(chunkq);
sq_z_corr = zeros(chunkq);

sq_x = zeros(chunkq);
sq_y = zeros(chunkq);
sq_z = zeros(chunkq);

while sq < partq



    for iq = 1:chunkq


        if sq > 0.5*partq


          com_x(iq) = com_x(iq) + COM(nq+iq,2);
          com_y(iq) = com_y(iq) + COM(nq+iq,3);
          com_z(iq) = com_z(iq) + COM(nq+iq,4);


          vx_steady(iq) = vx_steady(iq) + M(nq+iq,6);
          vy_steady(iq) = vy_steady(iq) + M(nq+iq,7);
```

```
            vz_steady(iq) = vz_steady(iq) + M(nq+iq,8);


            sq_x(iq) = sq_x(iq) + SQV(nq+iq,6)./SQV(nq+iq,5);
            sq_y(iq) = sq_y(iq) + SQV(nq+iq,7)./SQV(nq+iq,5);
            sq_z(iq) = sq_z(iq) + SQV(nq+iq,8)./SQV(nq+iq,5);
            Nq_steady(iq) = Nq_steady(iq) + SQV(nq+iq,5);


            sq_x_corr(iq) = sq_x_corr(iq) + SQV(nq+iq,6)./SQV(nq
               +iq,5) - 2.*comx_s(iq).*M(nq+iq,6)./SQV(nq+iq,5)
               + comx_s(iq).^2;
            sq_y_corr(iq) = sq_y_corr(iq) + SQV(nq+iq,7)./SQV(nq
               +iq,5) - 2.*comy_s(iq).*M(nq+iq,7)./SQV(nq+iq,5)
               + comy_s(iq).^2;
            sq_z_corr(iq) = sq_z_corr(iq) + SQV(nq+iq,8)./SQV(nq
               +iq,5) - 2.*comz_s(iq).*M(nq+iq,8)./SQV(nq+iq,5)
               + comz_s(iq).^2;

            temp_steady(iq) = temp_steady(iq) + T(nq+iq,6);
            N_steady_t(iq) = N_steady_t(iq) + T(nq+iq,5);



            if iq == chunkq
            s_steadyq = s_steadyq + 1;
            end
          end


    end
        nq = nq + chunkq +1;
        iq = 1;
        sq = sq+1;
end

sq_x = sq_x./s_steadyq;
sq_y = sq_y./s_steadyq;
sq_z = sq_z./s_steadyq;
Nq_steady = Nq_steady./s_steadyq;

vx_steady = vx_steady./s_steadyq;
vy_steady = vy_steady./s_steadyq;
vz_steady = vz_steady./s_steadyq;

sq_x_corr = sq_x_corr./s_steadyq;
sq_y_corr = sq_y_corr./s_steadyq;
sq_z_corr = sq_z_corr./s_steadyq;
```

```
com_x = com_x./s_steadyq;
com_y = com_y./s_steadyq;
com_z = com_z./s_steadyq;

temp_steady = temp_steady./s_steadyq;


N_steady_t = N_steady_t./s_steadyq;

T_sq = (sq_x + sq_y + sq_z)./(3);




% Temp correction factors

corr_fac1z = (2/3).*com_z.*vz_steady;
corr_fac2z = (1/3).*Nq_steady.*com_z.^2;

corr_fac1x = (2/3).*com_x.*vx_steady;
corr_fac2x = (1/3).*Nq_steady.*com_x.^2;

corr_fac1y = (2/3).*com_y.*vy_steady;
corr_fac2y = (1/3).*Nq_steady.*com_y.^2;


temp_corr = T_sq - corr_fac1z./Nq_steady +corr_fac2z./Nq_steady
    - corr_fac1x./Nq_steady +corr_fac2x./Nq_steady  - corr_fac1y
   ./Nq_steady +corr_fac2y./Nq_steady;

Tcorr = (sq_x_corr + sq_y_corr + sq_z_corr)/(3)


xt=linspace(0,chunkq,chunkq);



figure(9);
hold on;
plot(xt,temp_steady(:,1),'-','LineWidth',1.5);
plot(xt,T_sq(:,1),'x','LineWidth',1.5);
plot(xt,temp_corr(:,1),'x','LineWidth',1.5);
plot(xt,Tcorr(:,1),'o','LineWidth',1.5);
title('Averaged Temp after 50% of time');
xlabel('Chunk');
ylabel('Temp');
```

```
legend('Lammps with COM', 'Postprocess with COM', 'Postprocess
    without COM - Method 1', 'Postprocess without COM - Method
    2');
ylim([0.9 1.1])
grid on
box
hold off
```

A.8.7 Porous medium, input script for generating restart-file

```
variable mu equal 1.00
variable L equal 5.00
atom_style          atomic
variable            a equal 20.0
#lattice constant

variable            R22 equal 9.0
#Coefficient for LJs-potential

variable            s22 equal ${R22}+1.0
#Coefficient for LJs-potential

variable            s12 equal (${s22}+1.0)/2.0
#Coefficient for LJs-potential

variable            R12 equal ${s12}-1.0
#Coefficient for LJs-potential

variable           W equal $L+2
variable            l equal $L+1
variable            dens equal 4.0/$a^3.0
#Density of fcc lattice

variable            T equal 2
#Temperature

variable            dump equal 100

lattice             fcc ${dens}

region              box block 0 $W 0 1 0 1
region              porous block 1 $l 0 1 0 1

create_box          2 box
lattice             fcc 0.5
create_atoms        1 region box
lattice             fcc ${dens}
create_atoms        2 region porous
```

```
pair_style          lj/spline
pair_coeff          1 1 1 1 1 0
pair_coeff          1 2 1 ${s12} 1 ${R12}
pair_coeff          2 2 1 ${s22} 1 ${R22}

dump                dump all custom ${dump} dump_init.out type
   id x y z

mass                * 1

group               fluid type 1
group               dynamic_fluid dynamic fluid region box
group               solid type 2
#dump                dump_matrix solid custom 500000 dump_matrix
   .out type id x y z radius

neigh_modify        one 10000
delete_atoms        overlap ${s12} fluid solid
delete_atoms        overlap 1 fluid fluid

velocity            fluid create $T 29873
compute             mdtemp dynamic_fluid temp
fix                 nvt dynamic_fluid nvt temp $T $T 0.02
fix_modify          nvt temp mdtemp

thermo_style        custom step time c_mdtemp pe ke etotal atoms
thermo              ${dump}
timestep            0.002
run                 500000

write_restart       restart
```

A.8.8 Porous medium, input script for equilibrium simulation

```
variable mu equal 1.00
variable L equal 5.00
variable p equal 1.00
read_restart        ../restart
variable            a equal 20.0
variable            R22 equal 9
variable            s22 equal ${R22}+1.0
variable            s12 equal (${s22}+1.0)/2.0
variable            R12 equal ${s12}-1.0
variable            W equal ($L+1)*$a+18
variable            l equal ($L+1)*$a+10
variable            volume equal 8.0*18.0*18.0
variable            dens equal 4.0/$a^3.0
variable            T equal 2
variable            dump equal 100
```

```
lattice             fcc ${dens}

region              bulk_liq1 block 2 10 1 19 1 19 units box
region              bulk_liq2 block $l $W 1 19 1 19 units box

pair_style          lj/spline
pair_coeff          1 1 1 1 1 0
pair_coeff          1 2 1 ${s12} 1 ${R12}
pair_coeff          2 2 1 ${s22} 1 ${R22}

dump                dump all custom 100 dump_lammps.out id type
   x y z vx vy vz
dump                dump3 all custom 1000000 dump_lammps_last.
   out id type x y z vx vy vz

mass                * 1

group               fluid type 1
group               solid type 2
group               bulk_liq1 dynamic fluid region bulk_liq1
group               bulk_liq2 dynamic fluid region bulk_liq2

dump                dump2 solid custom 100 dump_lammps_solid.out
    id type x y z vx vy vz

fix                 nvt fluid nvt temp $T $T 0.02


variable            rho1 equal count(bulk_liq1)/v_volume
variable            rho2 equal count(bulk_liq2)/v_volume
compute             velx_peratom fluid property/atom vx
compute             velx fluid reduce sum c_velx_peratom
variable            jmx equal c_velx/vol
compute             temp_liq1 bulk_liq1 temp
compute             temp_liq2 bulk_liq2 temp
compute             press1 bulk_liq1 stress/atom temp_liq1
compute             press2 bulk_liq2 stress/atom temp_liq2
compute             p1 bulk_liq1 reduce sum c_press1[1] c_press1
   [2] c_press1[3]
compute             p2 bulk_liq2 reduce sum c_press2[1] c_press2
   [2] c_press2[3]
variable            P1 equal -(c_p1[1]+c_p1[2]+c_p1[3])/(3.0*
   v_volume)
variable            P2 equal -(c_p2[1]+c_p2[2]+c_p2[3])/(3.0*
   v_volume)

compute             temp_fluid fluid temp
```

```
compute             press fluid stress/atom temp_fluid
compute             chunk fluid chunk/atom bin/1d x lower 1.0
   units box
fix                 dump_p fluid ave/chunk ${dump} 1 ${dump} &
                    chunk temp c_press[*] vx file dump_lammps_p.
                       out norm none


fix                 dump_t_f fluid ave/chunk ${dump} 1 ${dump}
   chunk &
                    temp file dump_t_f.out norm none
fix                 dump_vel fluid ave/chunk ${dump} 1 ${dump}
   chunk &
                    vx vy vz file vel.out norm none

################################################
############## Compute / Dump Temperature ##
################################################

### Temperature with COM ###
compute                 temp_fluid_M fluid temp

fix                     dump_t_f_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        temp file dump_t_f_M.out norm none

################################################
############## Compute / Dump PRESSURE ######
################################################

### Pressure without COM ###

compute                 press_M fluid stress/atom temp_fluid


fix                     dump_p_f_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        c_press[*] file dump_p_fluid_M.out norm
                           none


### Pressure with COM correction ###

# Configurational Part #


compute                 press_fvir_M fluid stress/atom NULL
   virial
```

```
fix                     dump_p_vir_M fluid ave/chunk ${dump} 1 $
   {dump} chunk &
                        c_press_fvir_M[*] file dump_p_vir_M.out
                           norm none


# Square Velocity #

variable                sq_vx atom vx*vx
variable                sq_vy atom vy*vy
variable                sq_vz atom vz*vz

fix                     dump_sq_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        v_sq_vx[*] v_sq_vy[*] v_sq_vz[*] file
                           sq_v_M.out norm none

# Center of Mass Velocity #

compute                 com fluid vcm/chunk chunk
fix                     com_ave_M fluid ave/time ${dump} 1 ${
   dump} c_com[*] file com_M.out mode vector


##############################
########  Velocity ########
##############################

fix                     dump_vel_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        vx vy vz file dump_vel_M.out norm none


########################################
############## 2D Velocity Field ######
########################################

compute                 chunk_2d_zy fluid chunk/atom bin/2d z
   lower 1.0 y lower 1.0
compute                 chunk_2d_zx fluid chunk/atom bin/2d z
   lower 1.0 x lower 1.0


fix                     dump_vel_zx_M fluid ave/chunk ${dump} 1
   ${dump} chunk_2d_zx &
                        vx vy vz file vel_zx_M.out norm none
```

```
fix                     dump_vel_zy_M fluid ave/chunk ${dump} 1
   ${dump} chunk_2d_zy &
                        vx vy vz file vel_zy_M.out norm none

##########################################

#########################
######### MOP #########
#########################
compute                 cmop1 fluid stress/mop/profile x 0.0 0.1
    kin conf total
#(length of chunk)/(number of planes)=(lattice spacing/number of
    planes)=1.74716/10
#Calculate the kinetic, configurational and total=kinetic+
   configurational contributions.
#Will however only use the configurational part from the MOP.

fix                     press_mop1 fluid ave/time ${dump} 1 ${
   dump} c_cmop1[*] file mop1.out mode vector



thermo_style        custom step time pe ke v_P1 v_P2 c_temp_liq1
    c_temp_liq2 v_jmx v_rho1 v_rho2
thermo              ${dump}
timestep            0.002
run                 1000000
write_restart       restart
```

A.8.9 Porous medium, input script for non-equilibrium simulation

```
variable mu equal 1.00
variable L equal 5.00
variable p equal 1.00
read_restart        ../restart
variable            a equal 20.0
variable            R22 equal 9
variable            s22 equal ${R22}+1.0
variable            s12 equal (${s22}+1.0)/2.0
variable            R12 equal ${s12}-1.0
variable            W equal ($L+1)*$a+18
variable            l equal ($L+1)*$a+10
variable            volume equal 8.0*18.0*18.0
variable            dens equal 4.0/$a^3.0
variable            T equal 2
variable            dump equal 100

lattice             fcc ${dens}
```

```
region              bulk_liq1 block 2 10 1 19 1 19 units box
region              bulk_liq2 block $l $W 1 19 1 19 units box

pair_style          lj/spline
pair_coeff          1 1 1 1 1 0
pair_coeff          1 2 1 ${s12} 1 ${R12}
pair_coeff          2 2 1 ${s22} 1 ${R22}

dump                dump all custom 100 dump_lammps.out id type
   x y z vx vy vz
dump                dump3 all custom 1000000 dump_lammps_last.
   out id type x y z vx vy vz

mass                * 1

group               fluid type 1
group               solid type 2
group               bulk_liq1 dynamic fluid region bulk_liq1
group               bulk_liq2 dynamic fluid region bulk_liq2

dump                dump2 solid custom 100 dump_lammps_solid.out
    id type x y z vx vy vz

fix                 nvt fluid nvt temp $T $T 0.02
fix                 rpm fluid wall/rpm xlo EDGE $p

variable            rho1 equal count(bulk_liq1)/v_volume
variable            rho2 equal count(bulk_liq2)/v_volume
compute             velx_peratom fluid property/atom vx
compute             velx fluid reduce sum c_velx_peratom
variable            jmx equal c_velx/vol
compute             temp_liq1 bulk_liq1 temp
compute             temp_liq2 bulk_liq2 temp
compute             press1 bulk_liq1 stress/atom temp_liq1
compute             press2 bulk_liq2 stress/atom temp_liq2
compute             p1 bulk_liq1 reduce sum c_press1[1] c_press1
   [2] c_press1[3]
compute             p2 bulk_liq2 reduce sum c_press2[1] c_press2
   [2] c_press2[3]
variable            P1 equal -(c_p1[1]+c_p1[2]+c_p1[3])/(3.0*
   v_volume)
variable            P2 equal -(c_p2[1]+c_p2[2]+c_p2[3])/(3.0*
   v_volume)

compute             temp_fluid fluid temp
compute             press fluid stress/atom temp_fluid
compute             chunk fluid chunk/atom bin/1d x lower 1.0
   units box
```

```
fix                     dump_p fluid ave/chunk ${dump} 1 ${dump} &
                        chunk temp c_press[*] vx file dump_lammps_p.
                           out norm none


fix                     dump_t_f fluid ave/chunk ${dump} 1 ${dump}
   chunk &
                        temp file dump_t_f.out norm none
fix                     dump_vel fluid ave/chunk ${dump} 1 ${dump}
   chunk &
                        vx vy vz file vel.out norm none

#################################################
############## Compute / Dump Temperature ##
#################################################

### Temperature with COM ###
compute                 temp_fluid_M fluid temp

fix                     dump_t_f_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        temp file dump_t_f_M.out norm none

#################################################
############## Compute / Dump PRESSURE ######
#################################################

### Pressure without COM ###

compute                 press_M fluid stress/atom temp_fluid


fix                     dump_p_f_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        c_press[*] file dump_p_fluid_M.out norm
                           none


### Pressure with COM correction ###

# Configurational Part #


compute                 press_fvir_M fluid stress/atom NULL
   virial

fix                     dump_p_vir_M fluid ave/chunk ${dump} 1 $
   {dump} chunk &
```

```
                        c_press_fvir_M[*] file dump_p_vir_M.out
                           norm none


# Square Velocity #

variable                sq_vx atom vx*vx
variable                sq_vy atom vy*vy
variable                sq_vz atom vz*vz

fix                     dump_sq_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        v_sq_vx[*] v_sq_vy[*] v_sq_vz[*] file
                           sq_v_M.out norm none

# Center of Mass Velocity #

compute                 com fluid vcm/chunk chunk
fix                     com_ave_M fluid ave/time ${dump} 1 ${
   dump} c_com[*] file com_M.out mode vector


################################
#########  Velocity #########
################################

fix                     dump_vel_M fluid ave/chunk ${dump} 1 ${
   dump} chunk &
                        vx vy vz file dump_vel_M.out norm none


#########################################
############### 2D Velocity Field ######
#########################################

compute                 chunk_2d_zy fluid chunk/atom bin/2d z
   lower 1.0 y lower 1.0
compute                 chunk_2d_zx fluid chunk/atom bin/2d z
   lower 1.0 x lower 1.0


fix                     dump_vel_zx_M fluid ave/chunk ${dump} 1
   ${dump} chunk_2d_zx &
                        vx vy vz file vel_zx_M.out norm none

fix                     dump_vel_zy_M fluid ave/chunk ${dump} 1
   ${dump} chunk_2d_zy &
                        vx vy vz file vel_zy_M.out norm none
```

```
##########################################

#########################
######### MOP #########
#########################
compute                 cmop1 fluid stress/mop/profile x 0.0 0.1
    kin conf total

fix                     press_mop1 fluid ave/time ${dump} 1 ${
   dump} c_cmop1[*] file mop1.out mode vector


thermo_style        custom step time pe ke v_P1 v_P2 c_temp_liq1
    c_temp_liq2 v_jmx v_rho1 v_rho2
thermo              ${dump}
timestep            0.002
run                 1000000

write_restart       restart
```

A.8.10 Porous medium, post-processing script for geometry

```
#!/usr/bin/python3

#############################################################################

#
   #
#                       Geometry analytical
   #
#
   #
#   Computes surface area and volume in bins along
   #
#   the x-axis of _non-overlapping_ spheres analytically.
   #
#
   #
#   Usage:
   #
#       python3 geometry_analytical.py <dump-file>
   #
#       <bin size or None> <# of bins or None> <output_file>
   #
#
   #
#   Dependencies:
   #
```

```
#       * utilities.py
   #
#
   #
##############################################################################


import math
import sys
import numpy
import utilities
import util

# Settings
delimiter = " "

deviation = 0.0

def simple_geometry(data, bin_width=None, bins=None):
    if (bins == None and bin_width == None):
        print ("ERROR: Bin width or number of bins needs to be
           specified.")
        exit()
    elif (bins == None):
        bins = math.ceil( (data[1]['x-dim'][1] - data[1]['x-dim
           '][0])/bin_width )
    elif (bin_width == None):
        bin_width = (data[1]['x-dim'][1] - data[1]['x-dim'][0])/
           bins

    print (bins, bin_width)
    max_radius = max(numpy.transpose(data[0])[-1])
    X = [data[1]['x-dim'][1], data[1]['x-dim'][0]]
    Y = [data[1]['y-dim'][1], data[1]['y-dim'][0]]
    Z = [data[1]['z-dim'][1], data[1]['z-dim'][0]]
    print('Y:', Y)
    print('Z:', Z)
    V_rev = bin_width*(Y[1]-Y[0])*(Z[1]-Z[0])
    print('V_rev=', V_rev)

    # Volume and surface area of spheres in each bin
    V = []
    A = []
    Rh = []

    for i in range(0, bins):
        # Start and end of bin
        x_s = i*bin_width
```

```
        x_e = (i+1)*bin_width

        # Volume and surface area of spheres in bin
        V_i = V_rev
        A_i = 0

        # For hydraulic diameter/radius
        Ah_i = (Y[1]-Y[0]) * (Z[1]-Z[0])
        P_i = 0

        # Find spheres that are in bin
        for j in data[0]:
            if (j[1]==2):
                R=5.0       #Radius of solid particles
                x_p = j[2]
                if (x_p > x_s-R and x_p < x_e+R):
                    tmp = geometry_of_bin(x_s, x_e, x_p, R)
                    V_i -= tmp[0]
                    A_i += tmp[1]
                    Ah_i -= tmp[2]
                    P_i += tmp[3]


        V.append(V_i)
        A.append(A_i)
        try:
            Rh.append(Ah_i/P_i)
        except ZeroDivisionError:
            Rh.append(float(0))
    return [V, A, Rh, bin_width, bins]


# Determining width of segment h, radius a at x_e and radius b
    at x_s
def geometry_of_bin(x_s, x_e, x_p, R):
    if (abs(x_p-x_s) < R):
        b = math.sqrt(R**2.0-(x_p-x_s)**2.0)
        if (abs(x_p-x_e) < R):
            a = math.sqrt(R**2.0-(x_p-x_e)**2.0)
            h = x_e-x_s
        else:
            a = 0.0
            h = R-x_s+x_p
        V_i = (1.0/6.0)*math.pi*h*( 3.0*a**2+3.0*b**2+h**2 )
        A_i = 2.0*math.pi*R*h
        # For hydraulic diameter/radius
        Ah =  math.pi*a**2.0
        P = 2.0*math.pi*a
```

```
    elif x_s > x_p-R or x_e < x_p+R:
        a = math.sqrt(R**2.0-(x_p-x_e)**2.0)
        b = 0.0
        h = R-x_p+x_e
        V_i = (1.0/6.0)*math.pi*h*( 3.0*a**2+3.0*b**2+h**2 )
        A_i = 2.0*math.pi*R*h
        # For hydraulic diameter/radius
        Ah =  math.pi*a**2.0
        P = 2.0*math.pi*a
    else:
        V_i = (4.0/3.0)*math.pi*R**3.0
        A_i = 4.0*math.pi*R**2.0

        # For hydraulic diameter/radius
        Ah =  math.pi*R**2.0
        P = 2.0*math.pi*R
    return [V_i, A_i, Ah, P]



if __name__ == "__main__":
    if (len(sys.argv) < 5):
        print ("Too few arguments\nUsage python
           geometry_analytical.py <dump-file> <bin size> <bins>
           <output_name>")
        exit(0)

    dump_file_name = sys.argv[1]
    if (sys.argv[2] == "None"):
        bin_width = None
    else:
        bin_width = float(sys.argv[2])

    if (sys.argv[3] == "None"):
        bins = None
    else:
        bins = int(sys.argv[3])
    output_name = sys.argv[4]
    [raw_data, meta_data] = utilities.read_positions_dump(
       dump_file_name)
    [V, A, Rh, bin_width, bins] = simple_geometry([raw_data,
       meta_data],
            bin_width, bins)
    util.print_geometry(V, A, Rh, meta_data, bin_width,
       output_name)
    print('Total volume of fluid:', sum(V))
```

A.8.11 Porous medium, post-processing script for generating profiles of pressure, compressional energy etc.

```
import numpy as np
import os
import sys
import math
import matplotlib
matplotlib.use('Agg')
matplotlib.rcParams.update({'font.size':45})
matplotlib.rcParams.update({'lines.linewidth':3})
matplotlib.rcParams.update({'lines.markersize':15})
import matplotlib.pyplot
import matplotlib.image
import scipy.optimize
import utilities
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm


# Settings
position_filename   = "dump_lammps.out"
pressure_filename   = "dump_lammps_p.out"
geometry_filename   = "geometry.out"
temp_filename       = "dump_t_f.out"
vel_filename        = "vel.out"
snap_filename       = "snapshot.png"
profile_fig         = "_profile.svg"
fig_directory       = "figures"

# Plotting configuration

def plot_profile(options, folder, pressure_filename="
   dump_lammps_p.out"):
    current_plot = 0

    fig, ax = matplotlib.pyplot.subplots(
            options['n_plots'][0], sharex=True, figsize=(20,
               2*5*options['n_plots'][0])
            )
    # Initialize plot area
    if (options['n_plots'][0] == 1):
        ax = [ax]

    for i in range(0, len(ax)):
        ax[i].spines['right'].set_visible(False)
        ax[i].spines['top'].set_visible(False)
        ax[i].ticklabel_format(axis='y', style = 'sci',
           useOffset=False, scilimits=(-2,2))
```

```
        ax[i].set_xlim([0, 1])

        # Handle special cases
        if (i != len(ax)-1):
            ax[i].spines['bottom'].set_visible(False)
        else:
            ax[i].set_xlabel("x/L")
    vel = None

    if pressure_filename != "dump_lammps_p.out":
        tmp = (geometry_filename[:-4]+pressure_filename[-6:])

        geometry = utilities.read_geometry('geometry.out')


    else:

        geometry = utilities.read_geometry('geometry.out')



    pressure, N, dx=utilities.read_pressure_dump('dump_lammps_p.
       out')
    print('pressure length:', len(pressure[0]))
    T=utilities.read_temp_dump("dump_t_f.out")
    fit = []

    if (pressure.shape[1] != geometry.shape[1]):
        new_shape = geometry.shape[1]
        N = N[:new_shape]
        pressure = np.asarray([
                pressure[0][:new_shape],
                pressure[1][:new_shape],
                pressure[2][:new_shape]
                ])


    shape = N.shape
    n, samples = shape
    print('samples:',samples)
    print('n:',n)
    x = np.linspace(0, 1, n)
    w = 0
    for i in range(0, n):
        if (geometry[1][i] == geometry[3][i]):
#Volume of fluid in bin is equal to bin volume. This means that
   the bin contains only fluid.
```

```
            w += 1
        else:
            break
    L = n-2*w
#Length of the porous region.
    l1 = int(n*(1.0/8.0))-4
    l2 = int(n*(3.0/8.0))+4
    l3 = int(n*(5.0/8.0))-4
    l4 = int(n*(7.0/8.0))+4

    # Include snapshot?
    if options['snapshot'][1]:
        try:
            ax[current_plot].axis('off')
            img = matplotlib.image.imread(snap_filename)
            ax[current_plot].set_ylim([0, 1])
            ax[current_plot].imshow(img, extent=(0, 1, 0,1))
        except FileNotFoundError:
            print("Error: 'snapshot.png' not found in folder")
        current_plot += 1

    # Include porosity?
    if options['porosity'][1]:
        ax[current_plot].set_ylabel("Porosity, $\phi$")
        V = []
        for i in range(0, n):
            V.append(geometry[1][i]/geometry[3][i])
        ax[current_plot].plot(x, V, marker=".")
        current_plot += 1

    # Volume rock
    if options['volume_rock'][1]:
        ax[current_plot].set_ylabel("Volume rock, $V^r$")

        V = []
        for i in range(0, n):
            V.append(geometry[3][i]-geometry[1][i])
        ax[current_plot].plot(x, V, marker=".")
        current_plot += 1

    # Volume fluid
    if options['volume_fluid'][1]:
        ax[current_plot].set_ylabel("Volume fluid, $V^f$")
        V = []
        print('n=', n)
        for i in range(0, n):
            V.append(geometry[1][i])
        print('V:', V)
```

```
    ax[current_plot].set_ylim([0, 1200])
    ax[current_plot].plot(x, V, marker=".")
    current_plot += 1

# Specific surface area
if options['area'][1]:
    ax[current_plot].set_ylabel(r"Surface area, $\Omega^{fr}
       $")
    sA = []
    for i in range(0, n):
        sA.append(geometry[2][i])
    ax[current_plot].plot(x, sA, marker=".")
    current_plot += 1

# Density
if options['density'][1]:
    ax[current_plot].set_ylabel("Density, $\\rho$")
    density = []

    for i in range(0, n):
        density.append(np.mean(N[i])/geometry[1][i])
    ax[current_plot].plot(x, density, marker=".")
    current_plot += 1


# Smoothed density
if options['smooth_density'][1]:
    k = options['smooth_density'][2]
    ax[current_plot].set_ylabel(r"Smoothed density, $\rho, n
       =%d$" % k)
    density_smooth = utilities.smooth_profile(density, k)
    ax[current_plot].plot(x, density_smooth)
    current_plot += 1

# Pressure tensors
if options['pressure_tensor'][1]:
    ax[current_plot].set_ylabel(r"Pressure tensors")
    p_xx = []
    p_yy = []
    p_zz = []

    for i in range(0, n):
        p_xx.append(-np.mean(pressure[0][i])/geometry[1][i])
        p_yy.append(-np.mean(pressure[1][i])/geometry[1][i])
        p_zz.append(-np.mean(pressure[2][i])/geometry[1][i])
    ax[current_plot].plot(x, p_xx, marker=".", label="$P_{xx
       }$")
    ax[current_plot].plot(x, p_yy, marker=".", label="$P_{yy
```

```
        }$")
    ax[current_plot].plot(x, p_zz, marker=".", label="$P_{zz
        }$")
    ax[current_plot].legend(frameon=False)
    current_plot += 1

# Pressure tensors, but different control volume
if options['pressure_tensor2'][1]:
    ax[current_plot].set_title(r"Divided by $V_f$")
    ax[current_plot].set_ylabel(r"Pressure tensors")
    p_xx = []
    p_yy = []
    p_zz = []

    for i in range(0, n):
        p_xx.append(-np.mean(pressure[0][i])/geometry[1][i])
        p_yy.append(-np.mean(pressure[1][i])/geometry[1][i])
        p_zz.append(-np.mean(pressure[2][i])/geometry[1][i])
    ax[current_plot].plot(x, p_xx, marker=".", label="$P_{xx
        }$")
    ax[current_plot].plot(x, p_yy, marker=".", label="$P_{yy
        }$")
    ax[current_plot].plot(x, p_zz, marker=".", label="$P_{zz
        }$")
    ax[current_plot].legend(frameon=False)
    current_plot += 1
p = []
press_diff = 0

# Pressure
if options['pressure'][1]:
    ax[current_plot].set_ylabel(r"Fluid pressure, $p^f$")
    for i in range(0, n):
        p.append(-(np.mean((pressure[0][i]+pressure[1][i]+
            pressure[2][i]))/(3.0*geometry[1][i])))
    ax[current_plot].plot(x, p, marker=".")
    press_diff = [np.mean(p[0:w]), np.mean(p[w+L:n])]
    print('press_diff:', press_diff)

    dp = [(np.mean(p[:l1])+np.mean(p[l4:]))/2.0, np.mean(p[
        l2:l3])]
    print("dp:", dp)
    sdp = np.sqrt((np.std(p[:l1])**2.0 + np.std(p[l4:])
        **2.0)/4.0
            + np.std(p[l2:l3])**2.0)
    current_plot += 1

# Pressure, but different control volume
```

```
if options['pressure2'][1]:
    ax[current_plot].set_ylabel(r"Fluid pressure, $p^f$")
    for i in range(0, n):
        p.append(-(np.mean((pressure[0][i]+pressure[1][i]+
            pressure[2][i]))/(3.0*geometry[3][i])))
    ax[current_plot].plot(x, p, marker=".")
    press_diff = [np.mean(p[0:w]), np.mean(p[w+L:n])]
    current_plot += 1

pf = 0
pr = 0
gamma = 0
# Corrected pressure
if options['corrected_pressure'][1]:
    ax[current_plot].set_ylabel(r"Compressional energy, $p$
        ")
    pfVf = []
    for i in range(0, n):
        pfVf.append(-(np.mean((pressure[0][i]+pressure[1][i
            ]+pressure[2][i])
              /3.0)))
    print('pfVf:', pfVf)

    correction = [pfVf[1]-pfVf[i] for i in range(0, n)]   #
       Hvorfor pfVf[1]?
    GEOM=[[], [], [], []];
    for j in range(7):
        G1=0
        G2=0
        G3=0
        for k in range(20):
            G1+= geometry[1][7*j+k]
            G2+= geometry[2][7*j+k]
            G3+= geometry[3][7*j+k]
        GEOM[1].append(G1)
        GEOM[2].append(G2)
        GEOM[3].append(G3)
    (pr, gamma), pcov = scipy.optimize.curve_fit(pV_func,
            (geometry[1], geometry[2], geometry[3]),
               correction)
    print('gamma:', gamma)
    print('pr:', pr)
    pf = pfVf[1]/geometry[1][1]
    pf_phi = [ pfVf[i]/geometry[3][i] for i in range(0, n)]
    pr_phi = [ pr*(geometry[3][i]-geometry[1][i])/geometry
       [3][i]
            for i in range(0, n)]
    gamma_om = [ gamma*geometry[2][i]/geometry[3][i] for i
```

```
        in range(0, n)]
    print('gamma_om:', gamma_om)


    cp= [pfVf[i]+pr*(geometry[3][i]-geometry[1][i])-gamma*
        geometry[2][i] for i in range(0,n)]
    a = [pf_phi[i]+pr_phi[i] for i in range(0, n)]
    ax[current_plot].plot(x, [pfVf[i] for i in range(0,n)],
        label = r"$p^fV^f$")
    ax[current_plot].plot(x,[pfVf[i]+pr*(geometry[3][i]-
        geometry[1][i]) for i in range(0,n)] , label = r"$p^
        fV^f+\hat{p}^rV^r$")
    cp_smooth = utilities.smooth_profile(cp, 21) #21 is the
        number of bins in a unit cell
    ax[current_plot].plot(x, cp_smooth, marker=".", label=r"
        $p^fV^f+\hat{p}^rV^r-\gamma^{fr}\Omega^{fr}$")


    ax[current_plot].legend(ncol=1,loc=3, fontsize=25)
    fit = [pr, gamma]
    current_plot += 1

# Corrected pressure energy
if options['corrected_pressure_energy'][1]:
    ax[current_plot].set_ylabel(r"Pressure energy, $pV$")
    pfVf = []
    for i in range(0, n):
        pfVf.append(-(np.mean((pressure[0][i]+pressure[1][i
            ]+pressure[2][i])
              /3.0)))
    print("pfVf:", pfVf)
    correction = [pfVf[1]-pfVf[i] for i in range(0, n)]
    print("Correction:", correction)
    (pr, gamma), pcov = scipy.optimize.curve_fit(pV_func,
            (geometry[1], geometry[2], geometry[3]),
               correction)
    print("gamma from curve fit:", gamma)
    print("rock pressure from curve fit:", pr)
    pf = pfVf[1]/geometry[1][1]
    prVr = [ pr*(geometry[3][i]-geometry[1][i])
            for i in range(0, n)]
    gamma_om = [ gamma*geometry[2][i] for i in range(0, n)]

    cp = [pfVf[i]+prVr[i]-gamma_om[i] for i in range(0, n)]
    a = [pfVf[i]+prVr[i] for i in range(0, n)]
    ax[current_plot].plot(x, pfVf, label = r"$p^fV^f$")
    ax[current_plot].plot(x, a, label = r"$p^fV^f+\hat{p}^rV
        ^r$")
```

```
    ax[current_plot].plot(x, cp, marker=".",
            label=r"$p^fV^f+\hat{p}^rV^r-\gamma^{fr}\Omega^{
               fr}$")

    cp_smooth = utilities.smooth_profile(cp, 21)
    ax[current_plot].legend(ncol=1,loc=3, fontsize=25)
    fit = [pr, gamma]
    current_plot += 1

# Corrected pressure gradient
if options['corrected_pressure_gradient'][1]:
    ax[current_plot].set_ylabel(r"Compressional energy, $pV$
       ")
    pf = []
    for i in range(0, n):
        pf.append(-np.mean((pressure[0][i]+pressure[1][i]+
           pressure[2][i])
            /(3.0*geometry[1][i])))

    def gamma(pf):
        return 0.84198*pf +0.005962
    def pr(pf):
        return 1.1188*pf -0.0062652


    pfVf = [pf[i]*geometry[1][i]
            for i in range(0, n)]
    prVr = [pr(pf[i])*(geometry[3][i]-geometry[1][i])
            for i in range(0, n)]
    gamma_om = [gamma(pf[i])*geometry[2][i]
            for i in range(0, n)]
    cp1 = [pfVf[i]+prVr[i] for i in range(0, n)]
    cp = [cp1[i]-gamma_om[i] for i in range(0, n)]
    print("gamma from equilib. simulations: ", gamma(pf[i]))
    print("pr from equilib. simulations:",pr(pf[i]))
    print('length compressional energy:',len(cp))
    ax[current_plot].plot(x, pfVf, marker=".", label = r"$p^
       fV^f$")
    ax[current_plot].plot(x, cp1, marker=".", label = r"$p^
       fV^f+\hat{p}^rV^r$")
    ax[current_plot].plot(x, cp, marker=".",
            label = r"$p^fV^f+\hat{p}^rV^r-\gamma^{fr}\Omega
               ^{fr}$")
    print("length smooth profile:", len(utilities.
       smooth_profile(pfVf, 11)))

    ax[current_plot].legend(loc=3, fontsize=35)
    current_plot += 1
```

```
# Corrected pressure gradient, smooth
if options['corrected_pressure_gradient'][1]:
    ax[current_plot].set_ylabel(r"Compressional energy, $pV$
       ")
    pf = []
    for i in range(0, n):
        pf.append(-np.mean((pressure[0][i]+pressure[1][i]+
           pressure[2][i])
            /(3.0*geometry[1][i])))

    def gamma(pf):
        return 0.84198*pf +0.005962
    def pr(pf):
        return 1.1188*pf -0.0062652


    pfVf = [pf[i]*geometry[1][i]
            for i in range(0, n)]
    prVr = [pr(pf[i])*(geometry[3][i]-geometry[1][i])
            for i in range(0, n)]
    gamma_om = [gamma(pf[i])*geometry[2][i]
            for i in range(0, n)]
    cp1 = [pfVf[i]+prVr[i] for i in range(0, n)]
    cp = [cp1[i]-gamma_om[i] for i in range(0, n)]
    print("gamma from equilib. simulations: ", gamma(pf[i]))
    print("pr from equilib. simulations:",pr(pf[i]))
    print('length compressional energy:',len(cp))
    ax[current_plot].plot(x, utilities.smooth_profile(pfVf,
       21), label = r"$p^fV^f$")
    ax[current_plot].plot(x, utilities.smooth_profile(cp1,
       21), label = r"$p^fV^f+\hat{p}^rV^r$")
    ax[current_plot].plot(x, utilities.smooth_profile(cp,
       21),
            label = r"$p^fV^f+\hat{p}^rV^r-\gamma^{fr}\Omega
               ^{fr}$")
    print("pV:", utilities.smooth_profile(cp, 21))
    print("pf:", utilities.smooth_profile(pf, 21))

    print("length smooth profile:", len(utilities.
       smooth_profile(pfVf, 11)))

    ax[current_plot].legend(loc=3, fontsize=35)
    current_plot += 1

# Smoothed pressure
if options['smooth_pressure'][1]:
    k = options['smooth_pressure'][2]
```

```
    ax[current_plot].set_ylabel(r"Smoothed pressure, $P, n=%
        i$" %k)
    pxx = []
    pyy = []
    pzz = []
    for i in range(0, n):
        pxx.append(-(pressure[0][i]/geometry[3][i]))
        pyy.append(-(pressure[1][i]/geometry[3][i]))
        pzz.append(-(pressure[2][i]/geometry[3][i]))
    q = 3
    pxx_smooth = utilities.smooth_profile(pxx, k)
    pyy_smooth = utilities.smooth_profile(pyy, k)
    pzz_smooth = utilities.smooth_profile(pzz, k)
    ax[current_plot].plot(x, pxx_smooth, marker=".", label="
        $P_{xx}$")
    ax[current_plot].plot(x, pyy_smooth, marker=".", label="
        $P_{yy}$")
    ax[current_plot].plot(x, pzz_smooth, marker=".", label="
        $P_{zz}$")
    ax[current_plot].legend(frameon=False)
    current_plot += 1

t = []
# Temperature
if options['temperature'][1]:
    ax[current_plot].set_ylabel(r"Temperature, $T$")
    for i in range(0, n):
        t.append(np.mean(T[i]))
    ax[current_plot].plot(x, t, marker=".")
    current_plot += 1
    """
    dT = -(np.mean(t[:l1])+np.mean(t[l4:]))/2.0 + np.mean(t[
        l2:l3])
    sdT = np.sqrt((np.std(t[:l1])**2.0 + np.std(t[l4:])
        **2.0)/4.0
            + np.std(t[l2:l3])**2.0)
    print ("dT = %.5f +. %.5f" % (dT, sdT))
    """

mass_flux = np.zeros([2, n])
# Mass flux
if options['mass_flux'][1]:
    ax[current_plot].set_ylabel(r"Mass flux, $J_m$")
    vel = utilities.read_velocity_dump('vel.out')
    density = np.zeros([2, n])
    for i in range(0, n):
        phi = geometry[1][i]/geometry[3][i]
        density[0][i] = np.mean(N[i])/geometry[1][i]
```

```
            density[1][i] = np.std(N[i])/(geometry[1][i]*np.sqrt
                (samples))
            mass_flux[0][i] = np.mean(vel[i])*density[0][i]*phi
            mass_flux[1][i] = np.sqrt(
                    np.power(density[0][i]*phi*np.std(vel[i]),
                        2.0) +
                    np.power(np.mean(vel[i])*phi*density[1][i],
                        2.0)
                    )/np.sqrt(samples)
        ax[current_plot].plot(x[1:-1], mass_flux[0][1:-1])
        current_plot += 1

    # p hat (compressional energy divided by V_REV)
    if options['phat'][1]:
        ax[current_plot].set_ylabel(r"Integral pressure, $\hat{p}
            $")
        pf = []
        for i in range(0, n):
            pf.append(-np.mean((pressure[0][i]+pressure[1][i]+
                pressure[2][i])
                 /(3.0*geometry[1][i])))

        def gamma(pf):
            return 0.84198*pf +0.005962
        def pr(pf):
            return 1.1188*pf -0.0062652


        pfVf = [pf[i]*geometry[1][i] for i in range(0, n)]
        prVr = [pr(pf[i])*(geometry[3][i]-geometry[1][i]) for i
            in range(0, n)]
        gamma_om = [gamma(pf[i])*geometry[2][i] for i in range(0,
            n)]
        cp1 = [pfVf[i]+prVr[i] for i in range(0, n)]
        cp = [cp1[i]-gamma_om[i] for i in range(0, n)]
        phat=[cp[i]/geometry[3][i] for i in range(0,n)]
        ax[current_plot].plot(x, phat, marker=".", label = r"$\
            hat{p}$")
        current_plot += 1

    # p hat smooth (compressional energy divided by V_REV)
    if options['phats'][1]:
        ax[current_plot].set_ylabel(r"Integral presssure, $\hat{p
            }$")
        pf = []
        for i in range(0, n):
            pf.append(-np.mean((pressure[0][i]+pressure[1][i]+
                pressure[2][i])
```

```
        /(3.0*geometry[1][i])))

def gamma(pf):
    return 0.84198*pf +0.005962
def pr(pf):
    return 1.1188*pf -0.0062652


pfVf = [pf[i]*geometry[1][i] for i in range(0, n)]
prVr = [pr(pf[i])*(geometry[3][i]-geometry[1][i]) for i
   in range(0, n)]
gamma_om = [gamma(pf[i])*geometry[2][i] for i in range(0,
    n)]
cp1 = [pfVf[i]+prVr[i] for i in range(0, n)]
cp = [cp1[i]-gamma_om[i] for i in range(0, n)]
phat=[cp[i]/geometry[3][i] for i in range(0,n)]
ax[current_plot].plot(x, utilities.smooth_profile(phat,
   21), label = r"$\hat{p}$")

x1 = np.linspace(15/140, 125/140, 110)
x1 = np.array(x1).reshape((-1, 1))
y=np.array(utilities.smooth_profile(phat, 21)[16:126])

model = LinearRegression()
model.fit(x1, y)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
y_pred = model.predict(x1)
ax[current_plot].plot(x1, y_pred, label = r"$linear fit")
print('y_pred', y_pred)
current_plot += 1
average_p=sum(utilities.smooth_profile(phat, 21)[16:126])
   /len(utilities.smooth_profile(phat, 21)[16:126])
print('Average pressure:', average_p)

print('x1:', x1)
print('y_pred', np.array(y_pred).reshape((-1, 1)))
print('x:', np.array(x).reshape((-1, 1)))
print('int_press:', np.array(utilities.smooth_profile(
   phat, 25)).reshape((-1, 1)))

x1=sm.add_constant(x1)
model=sm.OLS(y, x1)
results=model.fit()
print(results.params)
print("std err:",results.bse)
current_plot += 1
```

```
    fig.tight_layout()
    os.system("mkdir -p figures")
    fig.savefig("figures/"+folder.replace("/", "_")+"_"+
       pressure_filename+"_profile.svg")
    n = len(p)
    return [np.mean(mass_flux[0]),
            np.mean(mass_flux[1]),
            np.mean(p[1:w-1]),
            np.std(p[1:w-1])/np.sqrt(w-2),
            np.mean(p[w+L+1:n-1]),
            np.std(p[w+L+1:n-1])/np.sqrt(w-2),
            np.mean(t),
            np.std(t)/np.sqrt(n),
            w, L, samples, pf, pr, gamma]

def pV_func(tmp, pr, gamma):
    (g1, g2, g3) = tmp
    return pr*(g3-g1)-gamma*g2

def handle_options(args):
    # Initialize options
    options = {
            'n_plots' :              [0, "N/A", "N/A"],
            'pressure' :             ["p", False, "N/A"],
            'pressure2' :            ["p2", False, "N/A"],
            'pressure_tensor' :      ["pt", False, "N/A"],
            'pressure_tensor2' :     ["pt2", False, "N/A"],
            'corrected_pressure' :   ["cp", False, "N/A"],
            'corrected_pressure_energy' :  ["cpv", False, "N/A
               "],
            'corrected_pressure_gradient' :  ["cpg", False, "N/A
               "],
            'smooth_pressure' :      ["sp", False, None],
            'temperature' :          ["t", False, "N/A"],
            'density' :              ["d", False, "N/A"],
            'smooth_density' :       ["sd", False, None],
            'snapshot' :             ["s", False, "N/A"],
            'area' :                 ["a", False, "N/A"],
            'porosity' :             ["phi", False, "N/A"],
            'volume_fluid' :         ["vf", False, "N/A"],
            'volume_rock' :          ["vr", False, "N/A"],
            'mass_flux' :            ["jm", False, "N/A"],
            'phat':                  ["phat", False,"N/A"],
            'phats':           ["phats", False, "N/A"],
```

```
        'cpgs':             ["cpgs", False, "N/A"],
}

for i in range(0, len(args)):
    if args[i].lower() == options['pressure'][0]:
        options['pressure'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['pressure2'][0]:
        options['pressure2'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['pressure_tensor'][0]:
        options['pressure_tensor'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['pressure_tensor2'][0]:
        options['pressure_tensor2'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['corrected_pressure
       '][0]:
        options['corrected_pressure'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['
       corrected_pressure_energy'][0]:
        options['corrected_pressure_energy'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['
       corrected_pressure_gradient'][0]:
        options['corrected_pressure_gradient'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['smooth_pressure'][0]:
        if (len(args) <= i+1):
            print("Error: Need number of smoothing bins")
            exit()
        options['smooth_pressure'][1] = True
        options['smooth_pressure'][2] = int(args[i+1])
        options['n_plots'][0] += 1
    elif args[i].lower() == options['temperature'][0]:
        options['temperature'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['density'][0]:
        options['density'][1] = True
        options['n_plots'][0] += 1
    elif args[i].lower() == options['smooth_density'][0]:
        if (len(args) <= i+1):
            print("Error: Need number of smoothing bins")
            exit()
        options['smooth_density'][1] = True
        options['smooth_density'][2] = int(args[i+1])
        options['n_plots'][0] += 1
```

```
        elif args[i].lower() == options['snapshot'][0]:
            options['snapshot'][1] = True
            options['n_plots'][0] += 1
        elif args[i].lower() == options['area'][0]:
            options['area'][1] = True
            options['n_plots'][0] += 1
        elif args[i].lower() == options['porosity'][0]:
            options['porosity'][1] = True
            options['n_plots'][0] += 1
        elif args[i].lower() == options['volume_fluid'][0]:
            options['volume_fluid'][1] = True
            options['n_plots'][0] += 1
        elif args[i].lower() == options['volume_rock'][0]:
            options['volume_rock'][1] = True
            options['n_plots'][0] += 1
        elif args[i].lower() == options['mass_flux'][0]:
            options['mass_flux'][1] = True
            options['n_plots'][0] += 1
        elif args[i].lower() == "all":
            if (len(args) <= i+1):
                print("Error: Need number of smoothing bins")
                exit()
            for j in options:
                if j != "n_plots":
                    options[j][1] = True
                    options['n_plots'][0] += 1
                    if options[j][2] != "N/A":
                        options[j][2] = int(args[i+1])
        elif args[i].lower()==options['phat'][0]:
            options['phat'][1]=True
            options['n_plots'][0] += 1
        elif args[i].lower()==options['phats'][0]:
            options['phats'][1]=True
            options['n_plots'][0] += 1
        elif args[i].lower()==options['cpgs'][0]:
            options['cpgs'][1]=True
            options['n_plots'][0] += 1
    return options


def print_options(options):
    width = max(len(x) for x in options)+5
    print ("\nUsage: python pressure_profile.py <one or more
       codes>")
    print ("#"*50)
    print ("Plot", " "*(width-5), "Code    Used?   Smoothing")
    print ("#"*50)
    for i in options:
```

```
        if (i != "n_plots"):
            print (i, " "*(width-len(i)), end="")
            for j in options[i]:
                print (j, " "*(7-len(str(j))), end="")
            print ("")
    print ("#"*50)
    print ("Number of plots: ", options['n_plots'][0], "\n")


if __name__ == "__main__":
    options = handle_options(sys.argv[1:])

    print_options(options)
    if options['n_plots'][0] == 0:
        print("Error: Need input")
        exit()

    print(plot_profile(options, ""))
```

A.8.12 Porous medium, utilitity script 1 for post-processing

```
#!/usr/bin/python3

#####################################################
#                                                   #
#   Utilities for LAMMPS Porous Media               #
#                                                   #
#       Functions in this file:                     #
#           * read_positions_dump                   #
#           * read_meta_dump                        #
#           * print_geometry                        #
#           * euclidian_distance                    #
#           * find_matrix_positions                 #
#                                                   #
#####################################################

#   Settings
delimiter = " "

import math

# Reads inputfile of matrix particle positions and radii
def read_positions_dump(dump_file_name):
    dump_file = open(dump_file_name, "r")
    dump_data = [i for i in dump_file]
    dump_file.close()

    x = [float(i) for i in dump_data[5].split()]
    y = [float(i) for i in dump_data[6].split()]
```

```
    z = [float(i) for i in dump_data[7].split()]
    N = int(dump_data[3])


    meta_data = {
            'x-dim' : x,
            'y-dim' : y,
            'z-dim' : z,
            'num_particles' : N,
            }

    raw_data = []

    for i in dump_data[9:]:
        if (i.split()[0] == "ITEM:"):
            break;
        else:
            raw_data.append([float(j) for j in i.split()])


    return [raw_data, meta_data]


# Returns only meta data of dump file
def read_meta_dump(dump_file_name):
    with open(dump_file_name, "r") as dump_file:
        dump_data = [next(dump_file) for i in range(10)]

    x = [float(i) for i in dump_data[5].split()]
    y = [float(i) for i in dump_data[6].split()]
    z = [float(i) for i in dump_data[7].split()]
    N = int(dump_data[3])


    meta_data = {
            'x-dim' : x,
            'y-dim' : y,
            'z-dim' : z,
            'num_particles' : N,
            }

    return meta_data


# Returns the smallest of three values
def dmin(a, b, c):
    if (a < c):
        if (a < b):
```

```
            return a
    if (b < c):
        return b
    return c


# Returns the surface to surface distance of two spheres in
   periodic boundary conditions (Y and Z)
def euclidian_distance(r1, r2, R1, R2, X, Y, Z):
    x_sq = dmin((r1[0]-r2[0])**2.0, (r1[0]+(X[1]-X[0])-r2[0])
       **2.0, (r1[0]-(X[1]-X[0])-r2[0])**2.0)
    y_sq = dmin((r1[1]-r2[1])**2.0, (r1[1]+(Y[1]-Y[0])-r2[1])
       **2.0, (r1[1]-(Y[1]-Y[0])-r2[1])**2.0)
    z_sq = dmin((r1[2]-r2[2])**2.0, (r1[2]+(Z[1]-Z[0])-r2[2])
       **2.0, (r1[2]-(Z[1]-Z[0])-r2[2])**2.0)
    return math.sqrt(x_sq+y_sq+z_sq)-(R1+R2)


# Returns all matrix particles that are inside bin bounded by X
def find_atoms(X, data):
    positions = []

    for i in data[0]:
        x_p = i[2]
        R = i[5]
        if (x_p > X[0]-R and x_p < X[1]+R):
            positions.append([x_p, i[3], i[4], R])

    return positions


# Returns total volume and surface area of matrix particles.

def total(V, A):
    n = len(V)
    V_tot = 0
    A_tot = 0

    for i in range(0, n):
        V_tot += V[i]
        A_tot += A[i]

    print (V_tot, A_tot)


# Shifts positions for periodic boundaries
def periodic_boundary(X, x):
    if (X < x[0]):
```

```
            X += (x[1]-x[0])
        elif (X > x[1]):
            X -= (x[1]-x[0])

        return X


def compare_lists(a, b):
    if (len(a) != len(b)):
        print ("Length of lists must be the same")
        exit(0)
    n = len(a)
    for i in range(0, n):
        if (a[i] != b[i]):
            return False
    return True


# Prints bin number, volume of sphere in bin, surface of spheres
    in bin and total bin volume to output_name
def print_geometry(V, A, Rh, meta_data, bin_width, output_name):
    bins = len(V)
    dy = meta_data['y-dim'][1]-meta_data['y-dim'][0]
    dz = meta_data['z-dim'][1]-meta_data['z-dim'][0]
    dV = bin_width*dy*dz

    output = open(output_name, "w")
    for i in range(0, bins):
        output.write(str(i+1)+delimiter+str(V[i])+
                delimiter+str(A[i])+delimiter+str(dV)+delimiter+
                   str(Rh[i])+"\n")
    output.close()
```

A.8.13 Porous medium, utility script 2 for post-processing

```
import numpy as np

# Read geometry file from porous media suite (geometry_*.py)
def read_geometry(geometry_filename):
    try:
        input = open(geometry_filename, "r")
        data = [x for x in input]
        input.close()

        N = len(data)
        n = len(data[0].split())
        geometry = np.zeros([N, n])
        j = 0
        for i in data:
```

```
            i = i.split()
            geometry[j] = i
            j += 1

        return np.transpose(geometry)
    except FileNotFoundError:
        print ("File not found: ", geometry_filename+". Skipping
            , may give errors.")
    return None


# Reads the head of lammps dumpfile to get system size
def read_system_size(positions_filename):
    with open(positions_filename) as i:
        data = [next(i) for x in range(0, 10)]

    x0 = float(data[5].split()[0])
    x1 = float(data[5].split()[1])
    y0 = float(data[6].split()[0])
    y1 = float(data[6].split()[1])
    z0 = float(data[7].split()[0])
    z1 = float(data[7].split()[1])

    return [x0, x1], [y0, y1], [z0, z1]


# Reads inputfile of matrix particle positions and radii
def read_positions_dump(dump_filename):
    dump_file = open(dump_filename, "r")
    dump_data = [i for i in dump_file]
    dump_file.close()

    x = [float(i) for i in dump_data[5].split()]
    y = [float(i) for i in dump_data[6].split()]
    z = [float(i) for i in dump_data[7].split()]
    N = int(dump_data[3])
    n = 0

    meta_data = {
            'x-dim' : x,
            'y-dim' : y,
            'z-dim' : z,
            'num_particles' : N,
            }

    u = [
            (meta_data['x-dim'][1] - meta_data['x-dim'][0]) /2.0,
            (meta_data['y-dim'][1] - meta_data['y-dim'][0]) /2.0,
```

```
            (meta_data['z-dim'][1]-meta_data['z-dim'][0])/2.0
            ]

    raw_data = []
    particle_pos = []
    for i in dump_data:
        i = i.split()


        if len(i) == 2 and i[1] == "TIMESTEP":
            n += 1
        elif len(i) == 8 and i[0] != "ITEM:":
                if (np.linalg.norm([float(i[2])-u[0], float(i
                   [3])-u[1], float(i[4])-u[2]]) <= 55):
#55 is the length from the center to each of the edges of the
   porous region

                    raw_data.append([float(j) for j in i])
                if (i[1] == '2'):
                    particle_pos.append([float(j) for j in i
                       [2:6]])
        particle_pos=particle_pos[0:22]


    particle_pos = np.transpose(particle_pos)
    meta_data['num_images'] = n
    meta_data['particle'] = [
            np.mean(particle_pos[0]),
            np.mean(particle_pos[1]),
            np.mean(particle_pos[2]),
            np.mean(particle_pos[3]),
          ]

    print('Length of particle_pos:', len(particle_pos[0]))
    print(' particle_pos:', particle_pos)
    del particle_pos

    print('meta data:', meta_data)
    return [raw_data, meta_data]


# Reads temperature dump file
def read_temp_dump(temp_filename):
    try:
        input = open(temp_filename, "r")
        data = [i for i in input]
        input.close()
```

```
        bins    = int(data[-1].split()[0])
        try:
            dump    = int(data[3+bins+1].split()[0]) - int(data
                [3].split()[0])
            N       = (int(data[-(bins+1)].split()[0]) - int(
                data[3].split()[0])) // dump+1
        except IndexError:
            N = 1
        dx      = 2.0*float(data[4].split()[1])

        T = np.zeros([bins, N])
        k = -1
        del data[:3]

        for i in data:
            i = i.split()
            if (int(i[0]) == 1):
                k += 1
            if (len(i) == 4):
                j = int(i[0])-1
                T[j][k] = float(i[3])
        print("Temp:", T)

        return T
    except FileNotFoundError:
        print ("File not found: ", temp_filename+". Skipping,
            may give errors.")
    return None

# Reads pressure dump file
def read_pressure_dump(pressure_filename):
    try:
        input = open(pressure_filename, "r")
        data = [i for i in input]
        input.close()

        bins    = int(data[-1].split()[0])
        print('bins:', bins)
        try:
            dump    = int(data[3+bins+1].split()[0]) - int(data
                [3].split()[0])
            print('dump:', dump)
            N       = (int(data[-(bins+1)].split()[0]) - int(
                data[3].split()[0])) // dump+1
            print('N:',N)
        except IndexError:
            N = 1
```

```
        dx      = 2.0*float(data[4].split()[1])
        print('dx:', dx)
        p_xx = np.zeros([bins, N])
        p_yy = np.zeros([bins, N])
        p_zz = np.zeros([bins, N])
        n = np.zeros([bins, N])
        k = -1
        del data[:3]
        for i in data:
            i = i.split()
            if (int(i[0]) == 1):
                k += 1

            if (int(i[0]) <= bins and len(i) > 3):
                n[int(i[0])-1][k] = float(i[2])
                p_xx[int(i[0])-1][k] = float(i[4])
                p_yy[int(i[0])-1][k] = float(i[5])
                p_zz[int(i[0])-1][k] = float(i[6])
        print('n:', len(n))
        print('p_xx:', p_xx)
        print('p_yy:', p_yy)
        print('p_zz:', p_zz)
        return np.asarray([p_xx, p_yy, p_zz]), np.asarray(n), dx
    except FileNotFoundError:
        print ("File not found: ", pressure_filename+". Skipping
            , may give errors.")
    return None, None, None


def read_velocity_dump(vel_filename):
    try:
        input = open(vel_filename, "r")
        data = [i for i in input]
        input.close()

        bins = int(data[-1].split()[0])
        try:
            dump    = int(data[3+bins+1].split()[0]) - int(data
                [3].split()[0])
            N       = (int(data[-(bins+1)].split()[0]) - int(
                data[3].split()[0])) // dump+1
        except IndexError:
            N = 1
        dx      = 2.0*float(data[4].split()[1])

        vel = np.zeros([bins, N])
        k = -1
        del data[:3]
```

```
        for i in data:
            i = i.split()
            if (int(i[0]) == 1):
                k += 1
            if (len(i) == 6):
                vel[int(i[0])-1, k] = float(i[3])
        print('vel:',vel)
        print('len vel:', len(vel))
        return vel
    except FileNotFoundError:
        print ("File not found: ", vel_filename+". Skipping, may
            give errors.")
    return None, None


# Reduce a profile by averaging n bins together
def smooth_profile(profile, n):
    if (n%2 == 0 or n < 1):
        print ("Error: n must be odd and greater than 1")
        exit()

    N = len(profile)
    start = int(n/2)
    end = N-int(n/2)

    smoothed = []

    for i in range(0, start):
        smoothed.append(profile[i])

    for i in range(start, end):
        smoothed.append(0)
        for j in range(0, n):
            smoothed[-1] += profile[i+j-int(n/2)]
        smoothed[-1] /= n

    for i in range(end, N):
        smoothed.append(profile[i])
    return smoothed
```

A.8.14 Viscosity, script for generating input scripts

```
import os, sys
import numpy as np

T =[ 2.0]
D = [0.5]
V = [0.5, 1.5, 2.5, 3.5, 4.0, 4.5]
```

```
main = [i for i in open("maler/mal_main.run", "r")]
cont = [i for i in open("maler/mal_cont.run", "r")]

for t in T:
    for d in D:
        for v in V:
            f = "T_%.2f/d_%.6f/v_%.6f"%(t, d, v)
            os.system("mkdir -p %s"%f)
            os.chdir(f)
            os.system("pwd")
            f = open("main.run", "w")
            f.write("variable T equal %f\n"%t)
            f.write("variable rho equal %f\n"%d)
            f.write("variable s equal %f\n"%v)

            for i in main:
                f.write(i)
            f.close()
            f = open("cont.run", "w")
            f.write("variable T equal %f\n"%t)
            f.write("variable rho equal %f\n"%d)
            f.write("variable s equal %f\n"%v)
            for i in cont:
                f.write(i)
            f.close()
            os.chdir("../../..")
```

A.8.15 Viscosity, input script for generating restart-file

```
# sample LAMMPS input script for viscosity of 2d LJ liquid
# NEMD via fix deform and fix nvt/sllod

# settings

variable                l equal 20

variable                srate equal 0.5

# problem setup

units                       lj
atom_style                atomic
neigh_modify           delay 0 every 1

# problem setup

lattice             fcc ${rho}
region              box prism 0 $l 0 $l 0 $l 0 0 0
create_box          1 box
```

```
create_atoms        1 box

pair_style          lj/spline
pair_coeff          * * 1 1 1 0

mass                * 1.0
velocity            all create $T 97287

fix                 1 all nve
fix                 2 all langevin $T $T 0.1 498094

dump                dump all custom 100 dump.out id type x y z

# equilibration run

thermo              1000
run                 5000
write_restart       restart
```

A.8.16 Viscosity, input script

```
read_restart        restart
variable            srate equal $s
neigh_modify        delay 0 every 1

lattice             fcc ${rho}
pair_style          lj/spline
pair_coeff          * * 1 1 1 0

mass                * 1.0

velocity                all scale $T
variable                xyrate equal ${srate}/ly

print v_xyrate

fix                     1 all nvt/sllod temp $T $T 0.1
fix                     2 all deform 1 xy erate ${xyrate}
   remap v

compute                 layers all chunk/atom bin/1d y center
   0.05 units reduced
fix                     4 all ave/chunk 20 250 5000 layers vx
   vy vz file profile.out

compute                 usual all temp
compute                 tilt all temp/deform

# data gathering run
```

```
variable                    visc equal -pxy/(v_xyrate)
fix                         vave all ave/time 10 100 1000 v_visc
   ave running

thermo_style          custom step temp press pxy v_visc f_vave
thermo_modify         temp tilt

thermo              1000
run                 4000000
write_restart       restart
```

A.8.17 Viscosity, script for plotting

```
import os
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
matplotlib.rcParams.update({'font.size':25})
matplotlib.rcParams.update({'lines.linewidth':3})
matplotlib.rcParams.update({'lines.markersize':15})

def read_profile(filename):
    f = open(filename, "r")
    d = [i.split() for i in f]
    f.close()

    bins = int(d[3][1])
    x = np.zeros(bins)
    v = np.zeros(bins)
    n = 0
    for i in d:
        if len(i) >= 4 and i[0] != "#":
            if i[0] == "1":
                n += 1
            x[int(i[0])-1] = float(i[1])
            v[int(i[0])-1] += float(i[3])
    return x, np.asarray(v)/n

def read_log(filename):
    f = open(filename, "r")
    d = [i.split() for i in f]
    print("d=", d)
    f.close()

    density = 0
    pxy = []
    ly = 0
```

```
    for i in d:
        if len(i) > 0 and i[0] == "triclinic":
            ly = float(i[8])
        if len(i) == 6:
            try:
                pxy.append(float(i[3]))
            except ValueError:
                continue
    return pxy, density, ly


fig_visc, ax_visc = plt.subplots(1, figsize=(9,4))
high_d_visc = []
high_d_temp = []
result = open("NEMD_results.out", "w")
for T in [2.0]:
    v_inf       = []
    density     = []
    for D in [0.5]:
        fig, ax = plt.subplots(1,2, figsize=(12,6))
        figs, axs = plt.subplots(1, figsize=(9,7))
        v = []
        s = []
        STD=[]
        for V in [0.500000, 1.500000, 2.500000, 3.500000,
           4.000000, 4.500000]:
            f = "T_%.2f/d_%.6f/v_%.6f"%(T,D,V)
            try:
                print("%s/log.lammps"%f)
                pxy, _, ly = read_log("%s/log.lammps"%f)
            except (IndexError, FileNotFoundError):
                break

            if len(pxy) != 0:
                fit = V/ly
#This is the gradient in velocity. V is the shear velocity
   whereas ly is the tilt of the triclinic box (see the input
   file).
                s.append(fit)
                v.append(-np.mean(pxy)/fit)
#See equation (1) in "Non-equilibrium simulation with the SLLOD
   algorithm"
                STD.append(np.std(pxy)/fit)



        if len(s) != 0:
            fit, cov = np.polyfit(s, v, 1, cov=True)
```

```
            x = [0, max(s)]
            y = np.polyval(fit, x)
            result.write("%f %f %f\n"%(T, D, fit[1]))
            ax[1].plot(np.sort(s), np.asarray(v)[np.argsort(s)
                ],'blue', marker="x")
            ax[1].plot(x, y, 'blue', ls="dashed")
            axs.plot(np.sort(s), np.asarray(v)[np.argsort(s)], '
                blue', marker="x")
            plt.errorbar(np.sort(s),np.asarray(v)[np.argsort(s)
                ],STD,marker='x')
            axs.set_xlabel(r"Shear rate")
            axs.set_ylabel(r"Shear viscosity")
            axs.plot(x, y, ls="dashed", color="black")
            axs.spines['top'].set_visible(False)
            axs.spines['right'].set_visible(False)
            for i in ax:
                i.spines['top'].set_visible(False)
                i.spines['right'].set_visible(False)
            ax[0].set_xlabel(r"$y$-coordinate")
            ax[0].set_ylabel(r"Velocity, $u_x$")
            fig.suptitle(r"$T=%.2f,\rho=%.3f$"%(T,D))
            plt.close(fig)
            figs.savefig("shear/%.2f_%.6f.svg"%(T,D))
            plt.close(figs)
            if D < 1.0:
                v_inf.append(fit[1])
                density.append(D)
            else:
                high_d_visc.append(fit[1])
                high_d_temp.append(T)

    if len(v_inf) != 0:
        ax_visc.plot(np.sort(density), np.asarray(v_inf)[np.
            argsort(density)],
                label="NEMD, T=%.1f"%T, marker="x")


ax_visc.spines['top'].set_visible(False)
ax_visc.spines['right'].set_visible(False)
ax_visc.set_xlabel(r"Density, $\rho$")
ax_visc.set_ylabel(r"Shear viscosity, $\eta_s$")
fig_visc.tight_layout()
fig_visc.savefig("visc.svg")
fig, ax = plt.subplots(1, figsize=(6,4))
ax.plot(high_d_temp, high_d_visc, marker="x", ls="solid", label
    ="NEMD")
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
```

```
ax.set_xlabel(r"Temperature, $T$")
ax.set_ylabel(r"Shear viscosity, $\eta_s$")
ax.legend()
fig.savefig("high_d_visc.svg")
result.close()
print(v_inf)
print(STD)
```

A.8.18 Evaporation box, script for generating restart-file

```
##################################################
############### Initialization ###################
##################################################

lattice                  fcc 0.7

# Parameter

variable L equal 100
# Length of the box

variable W  equal 15
# width of the system


variable D equal 0.70
# density of the lattice

variable N  equal 87287

# Lennard-Jones-spline Variables
variable eps11 equal 1.0
variable sig11 equal 1.0
variable alp11 equal 1.0


##################################################
########## Regions/Groups/Potentials##############
##################################################

lattice fcc $D
region box block -${W} ${W} -${W} ${W} 0 ${L} units box
region box2 block -9 9 -9 9 30 90 units box

# Create Atoms

create_box 1 box
create_atoms 1 region box2
```

```
# Group fluid and pore atoms

group fluid type 1

mass 1 1.0


# Define the Lennard-Jones potentials between the atoms

pair_style lj/spline
pair_coeff 1 1 ${eps11} ${sig11} ${alp11}  0 0.0

#################################################
######## Neighbors/Computation Balance ##########
#################################################

neighbor 0.3 bin
neigh_modify every 20 delay 0 check no
# Updating Neighbor List

# Optimizing Computation per processor
fix balance fluid balance 1000 1.15 shift xy 20 1.15


#################################################
################ Set Temperature ################
#################################################


# Set velocity to melt the crystal
velocity fluid create 0.75 $N

dump dump all custom 1000 tmp.dump id type x y z


thermo 1000
thermo_style custom step time pe ke etotal press


# Timestep-Size + run

fix 1 fluid nvt temp 0.75 0.75 0.02

timestep 0.002
run 50000
```

```
# Write restart file

write_restart restart.equil
```

A.8.19 Evaporation box, input script

```
##################################################
############### Initialization ###################
##################################################

read_restart ../restart.equil

lattice                 fcc 0.75

variable L equal 40
variable W equal 10
variable D equal 0.75
variable ch equal 1
variable N  equal 87287

# Lennard-Jones-spline Variables

variable eps11 equal 1.0
variable sig11 equal 1.0
variable alp11 equal 1.0



##################################################
########## Regions/Groups/Potentials##########
##################################################


lattice                 fcc $D
region                  box block -${W} ${W} -${W} ${W} 0 ${L}

region                  liquid block -${W} ${W} -${W} ${W} 20 ${
   L}



# Group fluid and pore atoms
group                   fluid type 1

mass                    1 1.0
```

```
# Define the Lennard-Jones potentials between the atoms

pair_style              lj/spline

pair_coeff              1 1 ${eps11} ${sig11} ${alp11}  0 0.0


#################################################
######## Neighbors/Computation Balance ##########
#################################################

neighbor                0.3 bin
neigh_modify            every 20 delay 0 check no   # Updating
   Neighbor List

# Optimizing Computation per processor

fix                     balance fluid balance 1000 1.15 shift xy
    20 1.15



#################################################
################ Prepare Chunks #################
#################################################


# Dividing box into bins(chunks)
compute                 chunk_f fluid chunk/atom bin/1d z lower
   ${ch}



#################################################
##############  Compute / Dump Temperature ##
#################################################

### Temperature with COM ###
compute                 temp_fluid fluid temp

fix                     dump_t_f fluid ave/chunk 100 10 1000
   chunk_f &
                        temp file dump_t_f.out norm none


#################################################
```

```
############### Compute / Dump PRESSURE ######
################################################

### Pressure without COM ###

compute                 press fluid stress/atom temp_fluid


fix                     dump_p_f fluid ave/chunk 100 10 1000
   chunk_f &
                        c_press[*] file dump_p_fluid.out norm
                           none


####################################
############## COM & Velocity ##
####################################

fix                     dump_vel fluid ave/chunk 100 10 1000
   chunk_f &
                        vx vy vz file dump_vel.out norm none

########################################
########## Positions ###############
########################################

### write output file ###
dump                    dump all custom 200000 tmp.dump id type
   x y z



################################
########## Run ################
################################

fix                     1 fluid nvt temp 0.65 0.65 0.02
timestep                0.002
run                     1000000


# Write restart file
write_restart           restart.lg
```

A.8.20 Evaporation box, post-processing script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% PRESS %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P_f = dlmread(sprintf('../dump_p_fluid.out',1),' ',3,0);


chunk = P_f(1,2);
A_f = zeros(chunk);


uc_t = 1.74716;
ch = 1;

dx = ch*uc_t; %chunk thickness
W = 10*uc_t;

%% Volume of each chunk

for u = 1:chunk
      dx_tot = dx * u;
      A_f(u) = 2*W*2*W;
      V_f(u) =  A_f(u) * dx;

end

%% Block Averaging Procedure
st = 1;
en = 10;


for k = st:en

s_tart = 0.00 + (k-1)*0.1;
e_nd = 0.00 + k*0.1;

np = 1;
sp = 1;
lp = size(P_f,1);
p_steady = 0;
partp = lp/(chunk+1);

press_total_f = zeros(chunk);
press_steady_f = zeros(chunk);
press_steady_fxx = zeros(chunk);
```

```
press_steady_fyy = zeros(chunk);
press_steady_fzz = zeros(chunk);



N_total = zeros(chunk);
N_total_steady = zeros(chunk);



while sp <= partp


    for ip = 1:chunk
        P_sum = -((P_f(np+ip,6)+P_f(np+ip,7)+P_f(np+ip,8))/(3*
            V_f(ip)));
        P_xx = - P_f(np+ip,6) / V_f(ip);
        P_yy = - P_f(np+ip,7) / V_f(ip);
        P_zz = - P_f(np+ip,8) / V_f(ip);

        N_new = P_f(np+ip,5)/V_f(ip);



          if sp >= s_tart*partp && sp < e_nd*partp
           press_steady_f(ip) = press_steady_f(ip) + P_sum;
           press_steady_fxx(ip) = press_steady_fxx(ip) + P_xx;
           press_steady_fyy(ip) = press_steady_fyy(ip) + P_yy;
           press_steady_fzz(ip) = press_steady_fzz(ip) + P_zz;

           N_total_steady(ip) = N_total_steady(ip) + N_new;

            if ip == chunk
            p_steady = p_steady + 1;
            end
          end


    end

        np = np + chunk +1;
        ip = 1;
        sp = sp+1;



end
```

```
press_steady_f_v(1:chunk,k) = press_steady_f(1:chunk,1)/p_steady
    ;
press_steady_fxx_v(1:chunk,k) = press_steady_fxx(1:chunk,1)/
    p_steady;
press_steady_fyy_v(1:chunk,k) = press_steady_fyy(1:chunk,1)/
    p_steady;
press_steady_fzz_v(1:chunk,k) = press_steady_fzz(1:chunk,1)/
    p_steady;

Number_F_v(1:chunk,k) = N_total_steady(1:chunk)/p_steady;



end


for i = 1:chunk
    p_sum(i) = mean(press_steady_f_v(i,5:en));
    p_xx(i) = mean(press_steady_fxx_v(i,5:en));
    p_yy(i) = mean(press_steady_fyy_v(i,5:en));
    p_zz(i) = mean(press_steady_fzz_v(i,5:en));
    Num(i) = mean(Number_F_v(i,5:en));


    p_sum_std(i) = std(press_steady_f_v(i,5:en));
    p_xx_std(i) = std(press_steady_fxx_v(i,5:en));
    p_yy_std(i)  = std(press_steady_fyy_v(i,5:en));
    p_zz_std(i)  = std(press_steady_fzz_v(i,5:en));
    Num_std(i) = std(Number_F_v(i,5:en));

end



%% Plotting

x = linspace(1,chunk,chunk);


figure(1);
hold on;
errorbar(x,p_sum(1,:),p_sum_std(1,:),'vertical','x','color
    ',[0.8500, 0.3250, 0.0980],'LineWidth',1.5);
errorbar(x,p_xx(1,:),p_sum_std(1,:),'vertical','x','color
    ',[0.9290, 0.6940, 0.1250],'LineWidth',1.5);
errorbar(x,p_yy(1,:),p_sum_std(1,:),'vertical','x','color
    ',[0.4940, 0.1840, 0.5560],'LineWidth',1.5);
```

```
errorbar(x,p_zz(1,:),p_sum_std(1,:),'vertical','x','color
    ',[0.6350, 0.0780, 0.1840],'LineWidth',1.5);
title('Fluid Pressure');
xlabel('Chunk');
ylabel('P');
grid on
legend('P_{sum}','P_{xx]','P_{yy}','P_{zz}')
box

figure(2);
hold on;
errorbar(x,Num(1,:),Num_std(1,:),'vertical','x','color',[0.8500,
    0.3250, 0.0980],'LineWidth',1.5);
title('Fluid Density');
xlabel('Chunk');
ylabel('Density');
grid on
box

P_gas=(p_sum(1,2)+p_sum(1,3)+p_sum(1,4)+p_sum(1,5)+p_sum(1,6)+
    p_sum(1,7)+p_sum(1,8)+p_sum(1,9)+p_sum(1,10)+p_sum(1,11)+
    p_sum(1,12)+p_sum(1,13)+p_sum(1,14)+p_sum(1,15)+p_sum(1,16)+
    p_sum(1,17)+p_sum(1,18)+p_sum(1,19))/18

Density_gas=(Num(1,2)+Num(1,3)+Num(1,4)+Num(1,5)+Num(1,6)+Num
    (1,7)+Num(1,8)+Num(1,9)+Num(1,10)+Num(1,11)+Num(1,12)+Num
    (1,13)+Num(1,14)+Num(1,15)+Num(1,16)+Num(1,17)+Num(1,18)+Num
    (1,19))/18
Density_liquid=(Num(1, 23)+Num(1, 24)+Num(1, 25)+ Num(1, 26)+
    Num(1, 27)+ Num(1, 28)+Num(1, 29)+Num(1, 30)+Num(1,31)+Num
    (1,32)+Num(1,33)+Num(1,34)+Num(1,35)+Num(1,36)+Num(1, 37)+Num
    (1, 38))/16
```

Mina Sørensen Bratvold

The permeability of porous media as studied by NEMD

NTNU
Norwegian University of
Science and Technology