NTNU

Norwegian University of
Science and Technology

Simulation of water self-diffusion in breast tissue

Brynjar Larssen Bakken

Master of Science in Physics and Mathematics
Submission date: June 2018
Supervisor: Pål Erik Goa, IFY
Co-supervisor: Igor Vidic, IFY
Liv Egnell, IFY

Norwegian University of Science and Technology
Department of Physics

Norwegian University of Science and Technology

Simulation of water self-diffusion in breast tissue

by

Brynjar Larssen Bakken

A thesis submitted in fulfilment for the
degree of Master of Science

in the
Faculty of Natural Sciences
Department of Physics

June 2018

Norwegian University of Science and Technology

Abstract

Faculty of Natural Sciences
Department of Physics

Master of Science

by Brynjar Larssen Bakken

Diffusion weighted magnetic resonance imaging (DW-MRI) can be used to distinguish malignant tumour tissue from benign lesions and healthy tissue. Although this is a well established clinical method, the detailed relationship between the tissue microstructure and the measured diffusion properties like the Apparent Diffusion Coefficient (ADC) are still under debate. During winter of 2016/2017 Ane Nordlie Johansen developed a simulation toolbox for water self-diffusion in closed geometries, and Brynjar Larssen Bakken extended this to include diffusion in the presence of solid cylinders as his project thesis during spring 2017. The main aim of this thesis is to simulate the water self diffusion in different geometrical arrangements of collagen fibres as typically found in healthy fibroglandular breast tissue (FGT), and to compare the resulting ADC with experimentally obtained values.

Sammendrag

Diffusjonsvekta magnetisk resonansavbilding kan brukes til å skilne ondarta tumorvev fra godarta lesjoner og friskt vev. Sjøl om dette er en veletablert klinisk metode, blir den detaljerte sammenhengen mellom vevets mikrostruktur og de målte diffusjonsegenskapene som den tilsynelatende diffusjonskoeffisienten fortsatt debattert. I løpet av vinteren 2016/2017 utvikla Ane Nordlie Johansen en verktøykasse for simulering av vanns selvdiffusion i lukka geometrier, og Brynjar Larssen Bakken utvida denne til å inkludere diffusion i nærhet av massive sylindre i hans prosjektoppgave i løpet av våren 2017. Hovedmålet til denne oppgava er å simulere vanns selvdiffusion i forskjellige geometriske oppsetninger av kollagenfibre som typisk finnes i friskt brystvev og sammenligne de resulterende diffusjonskoeffisientene med eksperimentell data.

Acknowledgements

Big thanks to Ane Nordlie Johansen for writing a solid master's thesis and very understandable code, and to Pål Erik Goa for being present, understanding and helpful, and for leading the DW-MRI group with insight and care. The rest of the group have my thanks as well for being helpful, cheerful and knowledgeable. This includes my co-supervisors, Igor and Liv, and Ingrid, my thesis writing partner from the blinds free copy room. You have all made my two DW-MRI focused semesters very much enjoyable.

However, as I soon leave NTNU and Applied Physics and Mathematics behind, there are greater affiliations to be thankful for. The social, cultural, applicable and professional rewards of the extracurricular student life is what makes Trondheim a place like no other. NTNUI Orienteering and the Student society's Internal Theatre has my sincere gratitude, and my warmest regards go to my student organisation Nabla.

Lastly I want to thank my supportive, reasonable and caring Rebekka for always brightening my day.

Contents

Abbreviations

MRI	**M**agnetic **R**esonance **I**maging
ADC	**A**pparent **D**iffusion **C**oefficient
RF	**R**adio **F**requency
DW	**D**iffusion **W**eighted
FGT	**F**ibro**g**landular breast **T**issue
HES	**H**ematoxylin-**E**osin-**S**affron
ICF	**I**ntra**c**ellular **F**luid
ECF	**E**xtra**c**ellular **F**luid
EDC	**E**xpected **D**iffusion **C**oefficient

Chapter 1

Introduction

MRI is a widely used imaging technique in clinical setting. MRI has the advantage of being suitable for differentiating between soft tissues, and unlike many other imaging techniques, it does not require exposure of high energy radiation [1]. MRI has been researched since the 1970s, and diffusion MRI specifically was first proposed as a method of differentiation tissue with varying degrees of diffusivity in 1984. It relies on self-diffusion of water in the sample to map the geometrical structure. This is possible because the geometrical structure influences the mobility of the particles, and therefore the apparent diffusion coefficient, which influences the MRI-signal.

To benefit from diffusion MRI, one needs to know the relationship between the diffusive behaviour and the resulting signal. This relationship is non-trivial, and much research has been done regarding this. A method for doing this is simulating the diffusion and MRI-sequence for a wide range of different environments and analysing the resulting signal. This may be done using Monte Carlo methods, in which statistical phenomenons using random numbers are simulated.

The purpose of this thesis is to further establish an environment appropriate for this kind of simulations and use it to compare simulated results with experimental data. The simulation is run using Monte Carlo methods to implement a diffusion process, and adding hindering or restricting obstacles to disrupt free diffusion. The sequences needed for generating the MRI signal is available in the thesis this work is based on [2].

Chapter 2

Theory

2.1 Diffusion

2.1.1 Free diffusion

Diffusion is the microscopic and seemingly random movement of particles, and is present in all fluids, both when equalising chemical potential and in equilibrium. The movement itself is called Brownian motion, and comes from the many collisions between the particles in the fluid. Brownian motion is often replicated by a random walk. This can be a series of equally long steps with random directions. This motion is described by the diffusion equation, which may be derived using the probability distribution of a particle after a one-dimensional random walk from $x = 0$ given by [3]:

$$P(x,t) = \sqrt{4\pi Dt}\; e^{x^2/4Dt}, \tag{2.1}$$

where D is a constant called the diffusion coefficient. D describes the degree of mobility for the particles. $P(x,t)$ is a symmetric function, so the expectation value is zero. The second moment then reduces to

$$\langle x^2(t)\rangle = \int^{\infty} x^2 P(x,t)\mathrm{d}x = 2Dt. \tag{2.2}$$

The variance is positively proportional with time, meaning the probability of finding a particle far away from its starting position increases over time. We can expand the particles degrees of freedom from one to n, and define a concentration of particles

$$C(\boldsymbol{r},t) = \int_{-\infty}^{\infty} \mathrm{d}\boldsymbol{r}' C_0(\boldsymbol{r}')P(\boldsymbol{r}-\boldsymbol{r}',t), \tag{2.3}$$

given the initial concentration $C_0(\boldsymbol{r})$. Using Fick's law[4] to define the current $\boldsymbol{j}$ as

$$\boldsymbol{j} = -D\nabla C(\boldsymbol{r},t), \tag{2.4}$$

and substituting this into the continuity equation[5]

$$\frac{\partial C}{\partial t} + \nabla \cdot \boldsymbol{j} = 0, \tag{2.5}$$

the diffusion equation is reached:

$$\frac{\partial C}{\partial t} = D\nabla^2 C. \tag{2.6}$$

This partial differential equation can be solved as in [6] to result in the Green function[7]

$$G(\boldsymbol{r},\boldsymbol{r}_0,t) = \frac{\exp(-|\boldsymbol{r}-\boldsymbol{r}_0|^2/4Dt)}{(4\pi Dt)^{n/2}}, \tag{2.7}$$

where $\boldsymbol{r}$ is the n-dimensional position vector, and $\boldsymbol{r}_0$ its initial state. Comparing this to (2.1) tells us that diffusion happens independently in all n dimensions — a result of the separability of the partial differential equation (2.5). $G(\boldsymbol{r},\boldsymbol{r}_0,t)$ is also, as expected, symmetrical about $\boldsymbol{r}_0$, but the second momentum

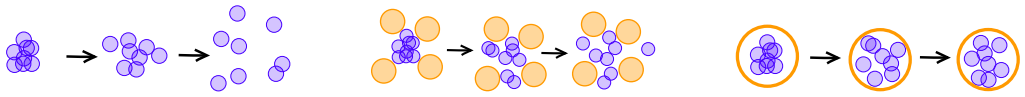
$$\langle \boldsymbol{r}^2(t)\rangle = \boldsymbol{r}_0^2 + 2nDt \tag{2.8}$$

show that (2.2) simply was a specific case for $n = 1, \boldsymbol{r}_0 = 0$. The mean square displacement for a particle after time t is then

$$\langle(\boldsymbol{r}-\boldsymbol{r}_0)^2\rangle = \langle \boldsymbol{r}^2\rangle + \boldsymbol{r}_0^2 - 2\boldsymbol{r}_0\langle \boldsymbol{r}\rangle = 2nDt. \tag{2.9}$$

2.1.2 Non-free diffusion

For non-free diffusion, where the particles are in some way obstructed in its movement, the diffusion coefficient will generally differ from the one discussed in the previous section



(A) free (B) hindered (C) restricted

FIGURE 2.1: Illustration of the three kinds of diffusion evolving over time.

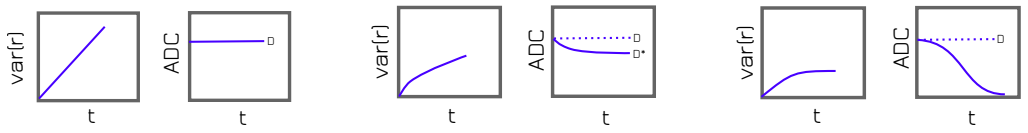
and have a time dependence. The expected length of a particle's single movement over a time t is derived from the true diffusion coefficient:

$$r = \sqrt{6Dt}, \tag{2.10}$$

for diffusion in $n = 3$ dimensions. The length of many particles' total diffusion movement give rise to the apparent diffusion coefficient (ADC). The particles' movement is called non-free when it's *restricted* — the particles are trapped inside an object — and/or when it's *hindered* — the particles are surrounded by obstacles with which to collide. The three types of diffusion is illustrated in figure 2.1. By investigating this new diffusion coefficient $D(t)$ one can find characteristics for the environment the particle is located in, such as shape and characteristic lengths [8].

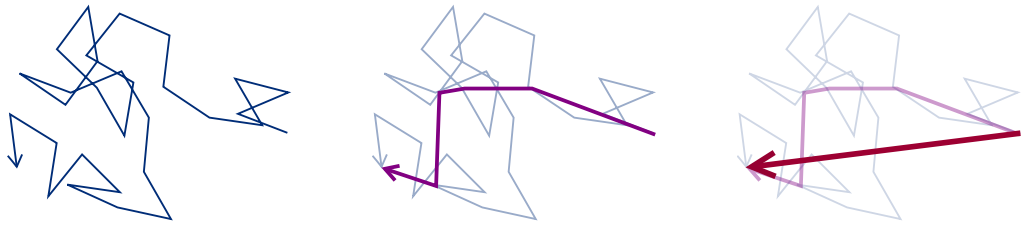
Analytic results for $D(t)$ varies with respect to the diffusion length relative to the dimensions of the surroundings, the size of the obstacles. For particles enclosed in a voxel, meaning the diffusion is restricted, the time dependency varies in the different cases of short or long time behaviour. In the first case particles will not diffuse for long enough time for them to be notably restricted from the surrounding wall, that is $D(t) \sim 1$ as in free diffusion. In the second case most of the particles will have collided many times with the walls, and the particle distribution will no longer develop over time, meaning $D(t) \sim \frac{1}{t}$. There is, of course, an intermediate state where some particles diffuse seemingly freely, and others don't, resulting in a bend in the $D(t)$-curve between the two aforementioned cases. Displacements of particles in free and restricted diffusion is seen illustrated in subfigures 2.2a and 2.2c, where ADC is in this case equivalent to $D(t)$.

If the particles are placed in between hindering objects, the same effect of particles initially moving freely, and later become more suppressed in their movement, is present. Once most of the particle has experienced hindrance and collided with an object, the ADC will expectedly not vary as much over time, but rather flatten out to a lower than true diffusion coefficient. The displacement of hindered particles can be seen in subfigure 2.2b.



(A) free (B) hindered (C) restricted

FIGURE 2.2: Illustrating graphs of displacement for the three kinds of displacement. For the free case, the ADC equals the true diffusion coefficient. For the hindered case, the ADC flattens out to a lower diffusion coefficient after enough time for most particles to have collided with the obstacles. For the restricted case, the ADC flattens out to zero, as no particles move longer than what the enclosing obstacles allow for any amount of time.



(A) Brownian motion (B) Microscopic movement (C) Diffusion result

FIGURE 2.3: Illustration of diffusion simulation for steps of different orders of magnitude. It can be shown that for free diffusion, it bears no difference how long each step is, as long as the number of steps are large.

2.1.3 Combined diffusion

By combined diffusion, a combination of the two types of non-free diffusion is meant. In cases where both hindered and restricted particles diffuse, $D(t)$ will only provide limited information about the diffusion, as it only states an average of the particles' freedom of movement. For short diffusion times are particles, as earlier mentioned, not necessarily obstructed, and the diffusion coefficient starts out as the true one. When time passes both hindered and restricted particles contribute to a decreasing displacement. For long diffusion times, restricted particles are no longer spreading out, and the hindered particles' displacement flattens out. If a fraction f of particles are hindered, then the final diffusion coefficient will turn out as $D_{\text{com}} = fD^*$.

2.1.4 Simulating diffusion

The motion of free can be simulated with computers, either the resulting position is calculated, every single Brownian motion is mapped, or a series of intermediate steps are added together, all of which are illustrated in figure 2.3.

A central size in Brownian motion is the mean free path λ which for liquid water has an order of magnitude of an Ångström. To simulate every Brownian motion, the steps could have a length of the λ and a random angle. The time each step would take is

the t_λ. The mean square displacement for a particle after a total diffusion time T, and therefore $K = \frac{T}{t_\lambda}$ steps is given by

$$\langle (\boldsymbol{r} - \boldsymbol{r}_0)^2 \rangle = \sum_{k=1}^{K} \langle (\boldsymbol{r}_i - \boldsymbol{r}_{i-1})^2 \rangle = K \cdot 2nDt_\lambda, \tag{2.11}$$

which matches the result from (2.9) of $2nDT$. This simple correlation between step by step Brownian motion and diffusion over macroscopic time opens up possibilities to simulate a particle's movement both from calculating its mean free path and from measuring its diffusivity. It also suggests that simulating diffusion with — in contrast to the mean free path — large step sizes is unproblematic.

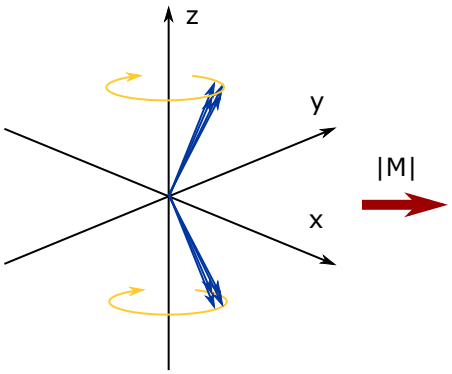
When simulating combined diffusion, there lacks mathematical derivation that step sizes other than those of Brownian motion give unbiased results.

2.2 MRI

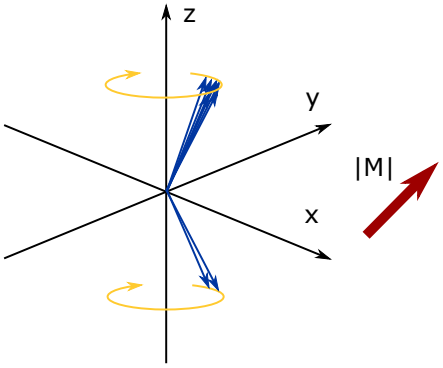
MRI is a tomographic technique used to obtain information about the internal structure of an object. It differs from other tomographic techniques in that it uses the intrinsic properties of the protons, its 1/2-spin, rather than using ionising radiation. MRI measures the magnetisation caused by the protons' magnetic moment $\boldsymbol{\mu}_p$ in the object being scanned. A proton in an external magnetic field $\boldsymbol{B}_0$ has two available energy states $\pm\mu_p B_0$. The spins will favour the lower energy state, making it slightly more occupied, and thus leading to a non-zero magnetisation vector $\boldsymbol{M}$. The spins precess with a frequency called the Larmor frequency given by

$$\omega = -\gamma B_0, \tag{2.12}$$

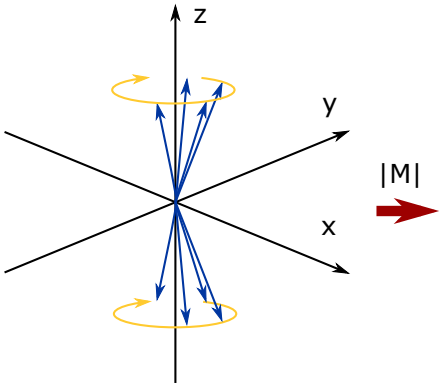
where γ is the gyromagnetic ratio. Spins in a uniform magnetic field will precess with the same frequency, but with arbitrary phase, meaning there will be no oscillating orthogonal to $\boldsymbol{B}$ in $\boldsymbol{M}$. The spins' phase can be synchronised by emitting a radio frequency (RF) signal with the Larmor frequency. With the spins in phase $\boldsymbol{M}$ will now tilt away from $\boldsymbol{B}_0$ and rotate around it — meaning the spins are distributed equally between their up and down states — They will then create a signal we call the MR signal, which naturally will have a frequency ω. The time the spins need for $\boldsymbol{M}$ to become aligned with $\boldsymbol{B}_0$ again defines T_1, an intrinsic property of the object being scanned. The magnetisation vector realigning with the outer field is called spin-lattice relaxation. Another intrinsic property is T_2^*, describing the time it takes for the spins to dephase in the plane, which is



(A) Spins in phase after RF signal.



(B) T_1 relaxation. Spins return to their original state in the direction of the outer magnetic field.



(C) T_2^* relaxation. Spins dephase due to local magnetic field variations and atomic interactions within the tissue.

FIGURE 2.4: Magnetisation relaxation. Spins precess around the axis of the outer magnetic field. Right after the RF signal, spins are in phase and the magnetisation signal is at its strongest. These effects take place at the same time, but are separated here for illustration.

called spin-spin relaxation. T_2 is the time constant for decay of transverse magnetisation arising from natural interactions at the atomic level, while T_2^* also includes dephasing caused by local variations in the magnetic field[1]. All three of these properties is used as contrast in MR images, and is illustrated in figure 2.4.

2.2.1 Diffusion weighted MRI

Diffusion weighted MRI (DW-MRI) relies on the movement of water and water's relaxation times inside tissue. Water molecules are relatively small and light, and will therefore easily diffuse — and with that be affected by the tissue's diffusivity. Mapping of the diffusion process of water in biological tissues helps understand the microscopic details of the tissue. Diffusion in any tissue is not free, but reflects interactions with many obstacles, such as fibres and membranes.

A spin precessing with frequency ω will have a phase at time t given by $\phi = \omega + \phi_0$, where ϕ_0 is the phase at time $t = 0$. The moment the RF signal is turned off, this will be same for all spins. If then a gradient magnetic field G_x increasing in strength perpendicular to the field lines, is applied to the spins, they begin to precess at different frequencies. The total magnetic field is now $\boldsymbol{B} = \boldsymbol{B}_0 + G_x x\hat{\boldsymbol{z}}$. The change in phase over an infinitesimal stretch of time is given by

$$\mathrm{d}\phi(x) = -\omega(x)\mathrm{d}t = -\gamma(B_0 + G_x x)\mathrm{d}t. \tag{2.13}$$

If this gradient is applied for a finite time δ, but still short enough that the spins' movement during the gradients effect still can be neglected, the equation is still valid. Given that a time Δ passes for the spins to diffuse is short enough to assume there are no relaxation of the magnetisation vector, an equally strong, but opposite gradient can be applied to find an exact relationship between phase and displacement:

$$\begin{aligned} \phi_2 - \phi_1 &= \gamma\delta(B_0 + G_x x_1 - B_0 - G_x x_2) \\ &= -\gamma\delta G_x(x_2 - x_1), \end{aligned} \tag{2.14}$$

where $x_1 = x(t)$ and $x_2 = x(t+\Delta)$ for any time t between the gradients. In an actual MR sequence, which is illustrated in figure 2.5, a 180° RF pulse is applied at time t_{180} to invert the magnetisation vectors followed by a gradient of the same sign, but the calculation still stands. Now, if δ is deemed too large to neglect the spin's movement during gradient application, and the two gradients are still applied Δ apart, the x-positions must be integrated over time:

$$\phi_2 - \phi_1 = -\gamma G_x \left(\int_{\Delta}^{\Delta+\delta} x(t)\mathrm{d}t - \int_0^{\delta} x(t)\mathrm{d}t \right), \tag{2.15}$$

or — if we divide the total sequence time into K equal parts t_K in which it is reasonable to say that the spins do not diffuse significantly — expressed in sums:

$$\begin{aligned} \phi_2 - \phi_1 &= -\gamma G_x \left(\sum_{k=1}^{k_\delta} x(t)t_K - \sum_{k=k_\Delta}^{K} x(t)t_K \right) \\ &= -\gamma G_x \left(\sum_{k=1}^{k_\delta} x_2(t) - x_1(t) \right) t_K, \end{aligned} \tag{2.16}$$

with $x(t) = x(kt_K)$, $k_\delta = K\frac{T_E}{\delta}$ the number of steps the gradients last, and $k_\Delta = K\frac{T_E}{\Delta}$ the number of steps between the start of the two gradients. For simplicity we define the

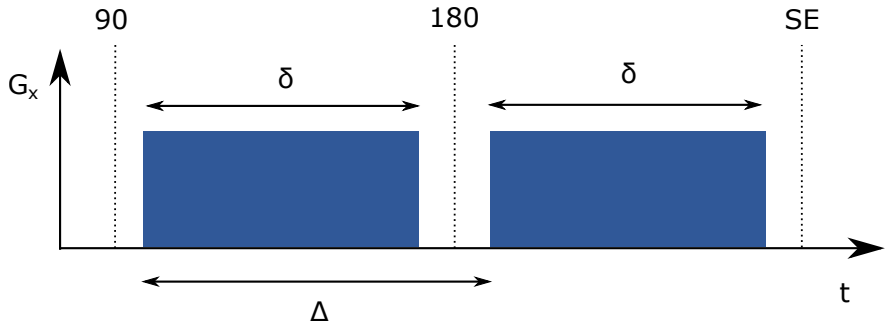


FIGURE 2.5: An illustration of a typical MR spin echo sequence. The time $T_E = 2t_{180}$ is the time from the initial RF pulse (90) to the spin echo is captured (SE).

total diffusion and sequence time $T_E = \delta + \Delta = 2\delta + T_{180}$, where T_{180} is the duration of the 180° signal, even though the figure 2.5 shows the more realistic picture of gaps between pulses, gradients and echo capturing.

In a clinical application, the time between the gradients Δ and the duration of the gradient δ are usually determined by the time it takes for the spin echo to be captured, and forces the other two time variables to be in the same order of magnitude. An MRI machine needs some time t_{sample} before the spin echo occurs, to ensure the sampling of a full echo signal. Between the gradients there also needs to be a small time gap T_{180}, as the 180° signal is applied. The total diffusion time is then

$$T_E = T_{\text{sample}} + \delta + T_{180} + \delta + T_{\text{sample}}. \quad (2.17)$$

This gives a gradient duration of $\delta = T_E/2 - T_{\text{sample}} - T_{180}/2$ and a diffusion time of

$$\Delta = \delta + T_{180}. \quad (2.18)$$

2.2.2 Signals

A central number of MRI is the signal attenuation $S(\delta, \Delta, G_x)/S(\delta, \Delta, 0)$, the fraction of the original signal $S(0)$ — at the spin echo for no gradients — that can be measured in the echo after the application of gradients. For a short δ, freely diffusing particles with initial position distribution $\rho(x_1)$, time between gradients Δ, a propagator $P(x_2, x_1, \Delta)$ from 2.7, signal attenuation for a single particle $\exp i(\phi_2 - \phi_1)$ and phase shift as in 2.14 the total signal attenuation can be calculated:

$$\begin{aligned}\frac{S(\delta,\Delta,G_x)}{S(\delta,\Delta,0)} &= \int\int \rho(x_1)P(x_2,x_1,\Delta)e^{-i\gamma\delta G_x(x_2-x_1)}\mathrm{d}x_1\mathrm{d}x_2 \\ &= \int P(x,0,\Delta)e^{-i\gamma\delta G_x x}\mathrm{d}x \\ &= \exp(-\gamma^2\delta^2 G_x^2 D\Delta),\end{aligned} \tag{2.19}$$

We can here introduce the diffusion sensitising b-value as $b = (\gamma\delta G_x)^2\Delta$, which gives

$$\frac{S(b)}{S(0)} = e^{-bD}. \tag{2.20}$$

The b-value is derived from

$$b = \gamma^2 \int_0^{T_E} \left(\int_0^t G_x(t')\mathrm{d}t' \right)^2 \mathrm{d}t, \tag{2.21}$$

and if we increase δ and simplify the imaging sequence — instead of including the 180°-signal — to two gradients of opposite sign, this resolves to

$$b = \gamma^2 G_x^2 \delta^2 \left(\Delta - \frac{\delta}{3} \right), \tag{2.22}$$

which is in agreement with 2.20 for small δ[9]. The signal attenuation does not have a sine wave of frequency ω as a carrier function, since the oscillating factors in $S(b)$ and $S(0)$ cancel each other out. If no other RF pulse or gradient is applied, a tapering signal could be measured over time. The decay of this signal — caused by T_2^* relaxation — is called the free induction decay, and has the shape of a negative exponential, with the steepness determined by T_2^* in the specific volume of the tissue[10].

T_2^*, like the other relaxation times, do not develop over time. This means that all the spins re-phase an equal amount per time after the 180° signal, given that they do not move. If we ignore diffusion, there will appear a signal peak when the local variations have re-phased every dephased spin, exactly t_{180} after the 180° pulse. This signal is, with gradients applied, called the spin echo and has the highest intensity when no spins are diffusing. Every spin that moves during the diffusion time will not re-phase the exact amount it was dephased, and therefore contribute less to the resulting signal echo.

It should be mentioned that there are several ways of performing a spin echo sequence, all in which the magnetisation vectors are flipped and inverted in a way that the applied gradients ultimately would cancel out any dephasing, either caused by T_1, T_2 or T_2^*

relaxation. The sequence here in focus concerns T_2^* relaxation, a single 180° RF pulse, and two gradients of equal strength and duration.

When varying the diffusion sensitising b-value, different spin echo signals will be captured. The highest intensity of the spin echo will decrease for greater b-values, as the diffused spins are more dephased. A logarithmic plot of the signal attenuation is generally linearly decreasing with b, and the curve's steepness is interpreted as the ADC. This is called the monoexponential representation:

$$S(b) = S(0) \cdot e^{-bD^*}. \tag{2.23}$$

If, however, the tissue contains both hindering and restricting obstacles, a linear logarithmic relationship between $S(b)/S(0)$ and b are no longer as apparent. Especially for large b-values, the signal attenuation has a tendency to curve upwards and flatten to a less, but still, decreasing line[11]. This gives rise to the biexponential representation:

$$S(b) = S(0) \cdot \Big[f_1 \cdot e^{-bD_1} + f_2 \cdot e^{-bD_2} \Big], \tag{2.24}$$

where f_1 and $f_2 = 1 - f_1$ are the fractions of hindered and restricted particles, respectively, and D_1 and D_2 are their respective apparent diffusion coefficients. The fraction of hindered and restricted particles should not affect the steepness of the curves in the parts of where one type of particle dominate, but rather determine where the transition lies.

2.3 The female breast

The female breast is a secretory gland composed of; glandular tissue, which makes and transports milk; connective tissue, which provides support; blood, which nourishes the tissue and provides the nutrients necessary to make milk; lymph, which removes waste; nerves, which allows stimulation for the release of hormones that trigger the milk ejection reflex and the production of milk; and adipose tissue, which offers protection from injury. Water is an essential element in every part mentioned, and it's present both inside and outside the different cells found in the breast.

The intracellular fluid (ICF) is enclosed within the cell walls — and some of that within the nucleus walls — and self-diffuses with restriction, whereas the extracellular fluid (ECF) self-diffuses with hindrance. The cells in question are amongst others fibroblasts, large collagen producing cells with irregular shapes. For water diffusion in breast tissue

the signal attenuation may be found by inserting $f_1 = f_{\text{ECF}} = 1 - f_{\text{ICF}}$, $D_1 = D_{\text{ICF}}$ and $D_2 = D_{\text{ECF}}$ in 2.24.

2.3.1 Collagen

Collagen is the main component in connective tissues, and therefore the most abundant protein in mammalian bodies. The structure of a female breast is not trivial. The collagen fibres themselves can be straightforward to model, but the arrangement of fibres is one thing making the MRI signal of a breast difficult to predict. Except for cases of straightened and aligned tumoral collagen, collagen fibres are both curved and divergent to neighbouring fibres, as stated in [12].

A single collagen molecule is approximately 300 nm long and 1.5 nm wide. Three and three of these molecules create a helix called tropocollagen. Microfibrils are bundles of five tropocollagen helices, but arbitrarily long, with a gap in the joint between molecules. Microfibrils assemble in order of magnitude 10 nm wide fibrils. Fibre diameters range from 1–12 μm. Collagen fibres can be seen as cylindrical in shape, and consist of smaller cylindrical fibrils, although the microstructure may reveal a more complex picture than this.

2.3.2 Microscopy

To distinct the various elements of a breast when studying a sample slice under the microscope, the sample may be stained with Hematoxylin-eosin-saffron, abbreviated HES. HE is commonly used for tissue staining, but HES may be more useful when investigating FGT and other collagen rich tissues, as the saffron marks collagen fibres in yellow. HES also marks erythrocytes in pink, cell nuclei in blue and muscle and elastic fibres in pink, as can be seen in figure 2.6.

2.4 Geometric attributes

There are many ways of describing a structure, and many single numbers that may be useful in some situations and unsatisfactory in others. Fibre density says something about the whole environment, but not about local variations, and in a simulation it may be useful to know how tight the fibre thickets are to make sure to long step sizes are used. It is also worth mentioning the area-to-volume ratio A/V, which can found experimentally as the tangent of the beginning of $D(t)$-curves[13].

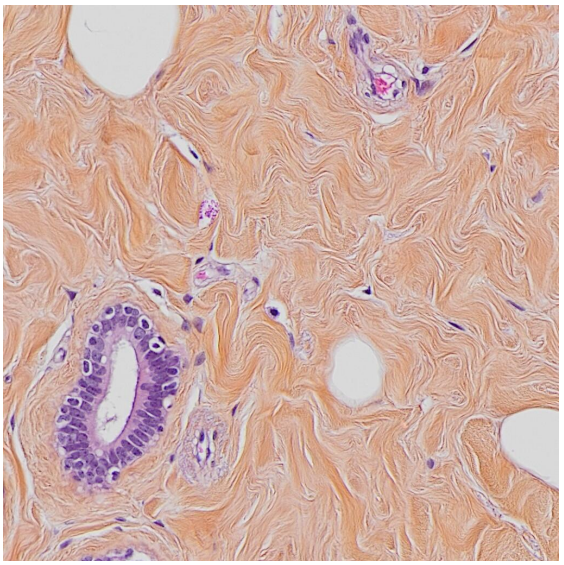


FIGURE 2.6: Example of a HES stained sample of breast tissue. Collagen in yellow, erythrocytes in pink and nuclei in purple. The big white voids are adipose cell, the purple cell cluster is a milk duct, the pink dots may indicate capillaries. The collagen forms a very swirly pattern, which represents healthy tissue.

2.4.1 Tortuosity

The tortuosity α of a structure is number describing the openness or diffusivity of said structure. In some definitions[14][15] it represents the fraction between the shortest possible trajectory S^* and the distance S between two arbitrary points:

$$\alpha_S = \frac{S}{S^*}, \tag{2.25}$$

This indicates that the geometry must be somewhat homogeneous for the tortuosity to bear any value. In this sense, tortuosity can describe how curvy a meandering river is, or how runnable a wooded area is. Others[16][15] have defined the tortuosity as the fraction of the ADC over the true diffusion coefficient, or

$$\alpha_D = \frac{D^*}{D}, \tag{2.26}$$

for long diffusion times. Without further thought it may seems natural that these two numbers are equal for the same structure, but the derivation is not only trivial.

2.4.2 Porosity

Another value that says a lot about a structure or geometry is the porosity, also called the void fraction, simply defined as

$$\beta = \frac{V^*}{V}, \tag{2.27}$$

where V^* is the hollow, populated or "diffusable" volume and V is the total volume.

2.5 Reflection and surface detection

Given a particle with a position $\boldsymbol{r}$ and a constant velocity $\boldsymbol{v}$, its trajectory is described by $\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{v}t$. If a particle is placed in an infinitely long cylinder of radius R with its centre along $\hat{\boldsymbol{z}}$, $\boldsymbol{v}_z$ is irrelevant for finding the time of collision with the cylinder wall. The particle will collide at a time where $\boldsymbol{r}'_{z=0}$ lies R from $\hat{\boldsymbol{z}}$, meaning $|\boldsymbol{r}'|^2 = R^2$. Substituting this into the equation for the particle trajectory and solving for t gives the following equation for collision time:

$$t_c = \frac{-\boldsymbol{r}\cdot\boldsymbol{v} \pm \sqrt{(\boldsymbol{r}\cdot\boldsymbol{v})^2 - \boldsymbol{v}^2(\boldsymbol{r}^2 - R^2)}}{\boldsymbol{v}^2}, \tag{2.28}$$

which will give two possible solutions for t_c. Since the particle's inside the cylinder, one solution will be negative and one solution will be positive — representing the time it takes to move from the wall behind the particle and the time it takes to move to the wall in front of the particle, respectively.

If a particle is placed outside a similar cylinder and moves towards it with a constant velocity $\boldsymbol{v}$, it will still take a t_c fulfilled by equation (2.28), only this time both t_cs will be positive — the latter representing collision from the inside as if ignoring the first wall. If this cylinder's centre is moved to a position $\boldsymbol{r}_{\text{cyl}}$, collision will occur when

$$\begin{aligned} |\boldsymbol{r}' - \boldsymbol{r}_{\text{cyl}}| &= R \\ \boldsymbol{r}'^2 - 2\boldsymbol{r}'\boldsymbol{r}_{\text{cyl}} + \boldsymbol{r}_{\text{cyl}}^2 &= R^2, \end{aligned} \tag{2.29}$$

which can be solved for t_c:

$$t_c = \frac{1}{\boldsymbol{v}^2}\left[\boldsymbol{r}_{\text{cyl}}\cdot\boldsymbol{v} - \boldsymbol{r}\cdot\boldsymbol{v} \pm \sqrt{(\boldsymbol{r} - \boldsymbol{r}_{\text{cyl}})^2 - \boldsymbol{v}^2(\boldsymbol{r}^2 - 2\boldsymbol{r}\cdot\boldsymbol{r}_{\text{cyl}} + \boldsymbol{r}_{\text{cyl}}^2 - R^2)}\right]. \tag{2.30}$$

This, too, gives rise to two positive t_c, of which the negative solution is the correct time of collision for hindered diffusion and the positive for restricted.

When a particle bounces off the surface of the cylinder, the part of the velocity parallel to the wall at the impact, $\boldsymbol{v}_{\parallel}$, will remain the same. The normal component of the velocity, $\boldsymbol{v}_{\perp}$, will change sign in the reflection. This can be expressed as

$$\boldsymbol{v} = \boldsymbol{v}_0 - 2(\boldsymbol{v}_0 \cdot \boldsymbol{n})\boldsymbol{n}. \tag{2.31}$$

2.6 Runtime

Runtime, or program cycle phase, is the period during which a computer program is executing. Simplified, the runtime of a program is given by the number of operations per unit of time the working processors can handle and the number of operations needed to complete the program. The size of the source code does not necessarily correlate to runtime, as both none and many operations can be executed by a line of code.

```
a = a + b;
```

is in some branches of computer science regarded as one single operation all though even here several things are happening in rapid succession. Firstly the script collects the value of variable `a` and `b`. Then it calculates the result of the mathematical operation $a + b$ before it applies that value to `a` and stores it. Now, if `a` and `b` turned out to be arrays or matrices, the number of operations would equal the number of elements in the sets. A dot product of two three-dimensional vectors would therefore demand three operations.

2.6.1 Copying large sets

Array or matrix resizing, such as in the following example, is a source of long runtimes.

```
pot_col_fibres = [];
for i=1:N_fibres
    pot_col_fibres = [pot_col_fibres i];
end
```

This is because when an n-dimensional array is created in a MATLAB environment, only n spaces of potential values is reserved. In the next iteration, there is no way of knowing if the next space, the $n+1$th element, is being used by another variable, and an

entirely new array has to be created to avoid memory loss. This means that there aren't `N_fibres` operations in the for loop above, but rather $1+2+3+\cdots+N_{\text{fibres}} \sim N_{\text{fibres}}^2$, as in the `i`th iteration `i` values must reallocate. This script has order of N_{fibres}^2 time complexity, and is written in terms of order O — a proportionality of runtime often as a function of array size — like $O(N_{\text{fibres}}^2)$. To lower the order to $O(N_{\text{fibres}})$, the size can be set before the loop, like so:

```
pot_col_fibres = zeros(1,N_fibres); %or ones(1,N_fibres) of course
for i=1:N_fibres
    pot_col_fibres(i) = i;
end
```

This is called preallocation. However, if the size of the array is not known on beforehand, a compromise must be found, as shown in the lower example.

```
pot_col_fibres = zeros(1,estimated_N_fibres);
for i=1:estimated_N_fibres
    if fibre_inside_cell(i) == true
        pot_col_fibres(i) = i;
    end
end
```

A too small `estimated_N_fibres` causes the time consuming copying of arrays, and a too large `estimated_N_fibres` increases runtime initially by creating a larger array than necessary. Which case afflicts the largest runtime penalty depends on how much the array is copied later in the script.

```
x = L*rand - l;
y = L*rand - l;
collide = false;
tooTight = false
tooLoose = true;
for n=1:N_fibres
    for i = -L:L:L
        for j = -L:L:L
        if (x-fibre_xpos(n) + i)^2 + ...
            (y-fibre_ypos(n) + j)^2 < ...
            (2*R_fibre+tightness)^2
```

```
            tooLoose = false;
        end
        if (x-fibre_xpos(n) + i)^2 + ...
            (y-fibre_ypos(n) + j)^2 < ...
            (2*R_fibre+looseness)^2
            tooTight = true;
        end
        if (x-fibre_xpos(n) + i)^2 + ...
            (y-fibre_ypos(n) + j)^2 < ...
            (2*R_fibre)^2
            collide = true;
        end
end
```

In this code snippet both `fibre_xpos` and `fibre_ypos` is called three times, and should be replaced `fx` and `fy` with the variables declared in the beginning of the outer for-loop.

```
[...]
for n=1:N_fibres
    fx = fibre_xpos(n);
    fy = fibre_ypos(n);
    [...]
end
```

2.6.2 Logical short-circuiting

In a logical statement with several conditions, it is most efficient to set the conditions more likely to break off the evaluation of remaining conditions early.

```
% often_false is not evaluated when often_true is true
if often_true || often_false
    % often_true is not evaluated when often_false is false
    if often_false && often_true
        use_valuable_time();
    end
end
```

This is called short-circuiting logical operators, and has most effect when the later conditions include time consuming evaluations.

2.6.3 Repetition avoidance

Iteration-independent operations should be placed outside loops. If code does not evaluate differently with each iteration in a loop, it can be moved outside the loop, bringing that part of code from order $O(i)$ to $O(1)$, where i is the number of iterations in the loop.

It may also save time to split the script into several parts, for example if one part of the script generates data that remains constant for the rest of the script and can be used in several runs. One script can generate the data, and another one can be run several times loading the same data and executing the rest of the script.

2.6.4 Replacement of demanding parts

There are surely examples of vastly different codes giving the equivalent output. The same reason simulating restricted diffusion for long diffusion times is processor heavy is the same reason it can be simplified. For a diffusion where almost every particle has collided with the voxel wall at least once, the particles has no 'memory' of its starting point in the sense that an end position has equal chance to have started wherever. In this case the entire stepwise diffusion simulation can be replaced with a random starting and end point. This is called motion averaging, and will be a feasible substitute for stepwise diffusion simulation for a given relationship between the diffusion time and the characteristic size of the voxels.

2.6.5 Example

If an operation is time consuming, it's undesirable to have it run more than it needs. If the operation is finding the shortest distance from a given point p in a given direction and range $\boldsymbol{d}$ to circles in two dimensions, knowing the circles' different x-position, y-position and radius r, finding each individual distance is a second degree problem. The geometric setup for this can be seen in figure 2.7, where the blue circle is over-sized as the point p, the blue arrow is $\boldsymbol{d}$, and the orange arrows are all r long. It would be a waste of time to measure the distance from the given point to all circles, as we are interested in only the circles that crosses the line of direction.

Firstly, if there was a maximum distance $d_{\max}$ of interest, we could discard every circle farther away than this. To simplify, every circle farther away from p than $d_{\max}$ in the x and y direction separately is crossed out, as it is less demanding to check $\min(x_1, x_2)$ and $\min(y_1, y_2)$ than $\min\left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\right)$. This limit is illustrated with the

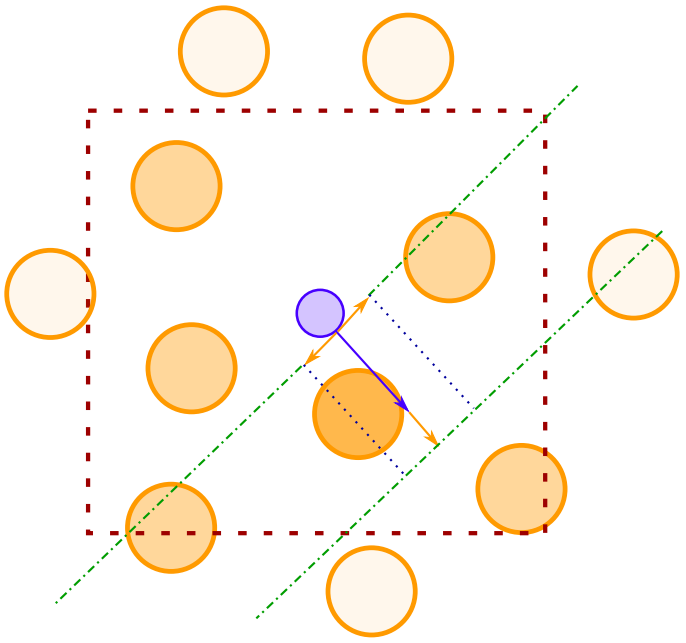


FIGURE 2.7: Collecting circles (orange) from a point (blue) within a certain range along a certain direction (blue arrow). The orange arrows are as long as the circles' radius.The point in question is here shown as a circle for legibility. The circles outside the range both horizontally and vertically are brighter coloured. The only circle in range along the direction is darker coloured.

dashed square, and discarded circles are lighter in colour. After this, every circle "behind" p can be discarded, meaning every circle with a negative $\boldsymbol{d}_{\parallel}$- component in a $\{\boldsymbol{d}_{\parallel}, \boldsymbol{d}_{\perp}\}$ coordinate system. Note that the circle's radius is here irrelevant, since a circle with its centre behind a line cannot intersect with another, perpendicular line in front of the first line (unless the lines intersects within the circle, which we assume is not the case here). In the same way, all circles i farther away than $d_{\max} + r_i$ is discarded. These two constraints is seen in the figure as the dash-dotted lines. Lastly, we can eliminate all circles farther away from p in the $\boldsymbol{d}_{\perp}$-direction than their radii. This is illustrated with the dotted lines.

If the radii are somewhat similar and retrieving the values for the individual radii is demanding, each separate can be exchanged with a $r_{\max} = \max(r_i)$ for all circles i, found on beforehand. By doing this we have possibly shaven off a remarkable fraction of the time consumed. By performing many small operations to narrow down the possibilities, one can make sure that the time consuming operation isn't performed more than necessary. The single circle in reach for p is coloured slightly darker in the figure.

```
parfor n = 1:N_h

    if 100*n/N_h == round(100*n/N_h)
        string = ['ParticleNr: ', ...
         num2str(n),' of ',num2str(N_h)];
        if 10*n/N_h == round(10*n/N_h)
            string = ['ParticleNr: ', ...
            num2str(n), ' of ', ...
            num2str(N_h), ...
            ', additional 10% complete'];
        end
        disp(string)
    end

    x_pos = zeros(1,K); %Initialising
    y_pos = x_pos;      % arrays to save
    z_pos = x_pos;      % all positions.

    x_pos(1) = x0(n);   %Adds the
    y_pos(1) = y0(n);   % initial
    z_pos(1) = z0(n);   % position.

    [...]

end
```

(A) Code

```
for every hindered particle n in parallel

    if n divisible by 100
        string = 'Particle n of N_h'

        if n also divisible by 1000
            add '10% complete' to string



        end
        show string ;)
    end

    position vectors of length K for x,
    y
    and z

    add starting position to x_pos,
    y_pos
    and z_pos

    [rest of loop]

end
```

(B) Pseudocode

FIGURE 2.8: Example of a code snippet and its pseudo code counterpart. The code is part of the script `PulsedGradient.m`

2.7 Writing pseudocode

To efficiently clarify the function of a script – more in-depth than just explaining its essence and more concise than presenting the entire code – a pseudo code, which combines code and words, may be a useful tool. Figure 2.8 shows an example of a code snippet and its pseudo code counterpart. Some parts of the code can be expressed more efficiently with words, while other needs to be expanded to make them easily readable. An overall property of the pseudocode is that it is more self-explanatory.

Chapter 3

Method

A script `DiffusionTest.m` was written to test if the simulated diffusion behaves as expected for a wide range of diffusion times and time step, for whether the step size is constant or normalised, and for the three kinds of diffusion. After the limitation of the simulation are established, the main goal of the thesis — simulating diffusion and calculating the particles' ADC to compare with experimental data — can be executed with confidence that the results are trustworthy.

A fibre environment generating script, like `Fibres.m`, randomly generates and exports x, y, and r values for a number of fibres. A signal calculating script, say `PulsedGradient.m`, imports the fibre values already generated and stores them in the three vectors `fibre_xpos`, `fibre_ypos` and `fibre_radii`, with a certain element number referring to the same fibre in all arrays. These values are used in each time step when finding which fibres the particle may collide into and reflect off of. There are two ways of narrowing the search down to fewer fibres presented later in this section. The list of selected fibres will then be iterated over for a calculation of collision time, always coming in pairs. Hindered and restriction make use of different solutions to the second degree problem. After a collision, the same list of potentially colliding fibre can be used as the particle has no way of moving out of the group of fibres with its total movement adding up to dr for each step. The resulting movement in every dimension of every particle is saved and the resulting signal for different diffusion times and gradient strengths are exported.

Note that the time steps are referred to as dt and step sizes as dr, although they are not infinitesimal.

3.1 Determining parameters

A big portion of this thesis has been running `DiffusionTest.m` in order to find fitting parametres with which to run the later scripts. Firstly a critical number of particles was found to make sure the remaining results are trustworthy.

3.1.1 True diffusion coefficient

There is literature that states that the true diffusion coefficient is linear to the temperature and presents a quite exact relationship as well. However, as the temperature of a breast is far from constant, varying both from breast to breast as well as within each breast, there is no need for an exact coefficient. $D = 3\mu\text{m}^2/\text{ms}$ was chosen for simplicity and represents a temperature of about 36 deg C, probable for an organ outside the ribcage.

3.1.2 Fibre size

The microscopy images such as in figure 2.6 shows a variety of fibre structures. Parts of the fibrous regions in the image seen in figure 2.6 shows a diffuse orange colour, while other parts shows a more distinct zebra pattern. As the optical refraction resolution prevents the further revelation of the microstructure. It may therefore be difficult to choose one fitting fibre size for all experiments onward.

It is simplest for the generation of fibre environments that the fibre have an equal radius. Another alternative would be normally distributed radii. Every fibre could also consist of several smaller cylinders of varying size representing fibrils. For some simulations, whose results are not presented as they didn't seem to bear significance, both varying fibre size as well as fibril bundles were used. Code for generating fibres and fibril bundles for different geometries can be found in the appendix.

A standard radius of 2 μm was used for all simulations were else is specified.

3.1.3 Nucleus size

In the restricted case microscopy images was used to determine a characteristic size for the voxels. As fibroblast cells are large, irregularly shaped and has hard to spot walls, the fibroblast cell nuclei were deemed a better candidate for restricting water particles. Henceforth, intranuclear fluid rather is meant by ICF. The nuclei varied somewhat in size and shape, but to simplify spherical shapes were used to describe the enclosing cell

nuclei when calculating restricted diffusion, all with a radius $R = 5\mu$m — a typical size for the nuclei investigated.

3.1.4 Diffusion times and time steps

The script `DiffusionTest.m` was central in determining how long and how many steps each simulation should have. By varying the total diffusion time T and the duration of the time steps dt — and with that the number of steps $K = T/dt$ — anomalies could be found in the output, and the combinations of T and dt for which the simulation behaves differently were discarded as invalid in later runs. There were several attempts to autogenerate a number describing how dense the geometry generated was, and with with that number deciding what step sizes produce expected results.

3.1.5 Time parameters

In the simulations T_{sample} was ignored both before and after the gradients, as they were thought to not affect the calculation of the signal. The gap between gradients was set to $T_{180} = 2$ ms after investigating typical values. The remaining parameters was the found with 2.17 and 2.18 for the several different T_E in each run.

3.2 Generating fibres

There are many ways to make models of fibre microstructure, but they can be separated into two main methods — random generation and predetermination. The two methods are suited for making the two kinds of microstructure — random generation for environments with locally varying density and predetermination for lattice structures.

3.2.1 Random generation

As in the project thesis, the idea of a unit cell of fibres, and a particle's experience of an endless field of fibres, are continued. A unit cell is created, then duplicated. The cell is a square of width L_x and length L_y. The same rules for overlapping fibrils applies, but in this case overlapping the cell boundary is permitted. To avoid an unnatural gap between the cells, the fibrils can be placed in $-L_x < x < L_x, -L_y < y < L_y$. When checking for overlap with earlier placed fibrils, positions $x \pm L_x, y \pm L_y$ must also be checked, see figure 3.1. The script terminates as before. When initially placing water particles at position (x, y) in the unit cell, one must check for fibres both in (x, y) as well as in $x \pm L_x, y \pm L_y$).

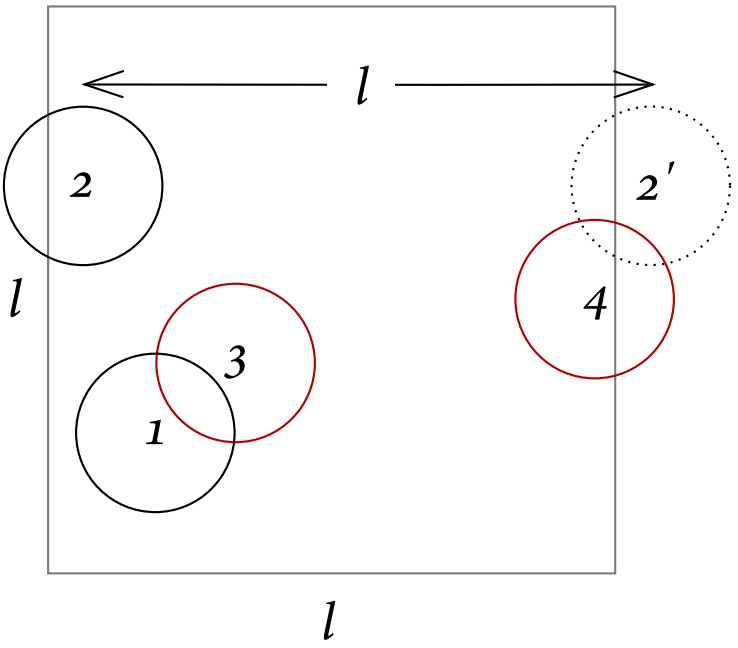


FIGURE 3.1: Attempted placed fibre. 1) No hinder — successful. 2) Legally overlapping the unit cell's outside — successful. 3) Overlapping already placed fibre 1 — unsuccessful. 4) Overlapping fibre 2', the neighbouring unit cell's already placed fibre 2 — unsuccessful. The scale is distorted for legibility.

Geometry 1 is a type of fibre environment that relies only on the random placement of fibres, where geometry 2 and 3 have more restrictions. Geometry 2 has a condition that every fibre should at most a distance $d_{\max}$ away from another fibre, ensuring a more populous environment. Geometry 3 has a restriction in both tightness and looseness, and will only place a fibre if it fulfils geometry 2's condition as well as being at least another distance $d_{\min}$ away from every other fibre.

3.2.2 Fibrils in fibre

For all three geometries mentioned, fibril bundles as fibres is a possibility. When simulating the structure of fibrils in a fibre, cylindrical fibrils are placed tightly, but randomly in a bigger enveloping cylinder representing the fibre. A random number between $-R$ and R for x and y is chosen. If the position $\boldsymbol{r} = [x, y]$ is out of bounds, it is discarded, otherwise stored in the arrays of fibril values. The script terminates when a number threshold of discarded positions for a single fibril placement is reached. The length from the origin r must be less than $R - R_{\text{fibril}}$, and the fibril cannot overlap with any earlier placed fibril. For equal R_{fibril}s, this means $|r - r_i| > 2R_{\text{fibril}}$ for all r_i. The fibrils can also vary in size, with a mean value and a standard deviation for r. In that case $|r - r_i| > R_{\text{fibril}} + R_{\text{fibril},i}$ for all r_i must be satisfied. For a tighter fit than what random number generation and human patience allows, every fibril can be increased in size until they touched one of their neighbours (do not try this yourself).

3.2.3 Predetermination

Two kinds of lattice unit cells were constructed — square and triangular lattices — with fibre radius `R` and distance between fibres `gap` or g as the two parameters. The size of the unit cells follows from the parameters.

3.3 Collecting fibres

Collecting fibres means the process of finding the fibres in the range of a particle for a time step and narrowing down the search until one potential fibre is found to be in the particle's collision course. This includes calculation and comparing of collision times, which are second degree problems. There is lot of processing power demanded when simulating many thousand particles' movement in many thousand time steps, and the by streamlining the collection of fibres, one can save much time.

As was developed and proven the fastest of the tested ways of collecting fibres (then fibrils) in [17] — method 1: The box with optimisation A: tighter box and B: sorting fibres — is used. The method and optimisations are explained in the next subsections.

3.3.1 Method 1 — The box

For every time step, the script loops over all fibres, saving the ones whose x- and y-position lies within reach of the particle. The longest a particle can reach in any movement, also counting for reflection of fibres, is $\mathrm{d}r + R_{\max}$, where $\mathrm{d}r$ is the particles' step size and $R_{\max}$ is the largest radius of all fibres. For small differences in the fibres' radii, it is suggested that the use of maximum radius rather rather than the specific one found in the vector improves the runtime. The search for potentially colliding fibres, however, goes through the xy plane, and fibres never reflect in the $\hat{\boldsymbol{z}}$-direction, meaning that one can confine all potentially colliding fibres within a box of size $\mathrm{d}r \sin \mathrm{d}_\theta + r_{\max}$, where $\mathrm{d}\theta$ is this step's angle from the xy plane.

3.3.2 Optimisation A — Tighter box

The optimisation is performed for every direction change, either at a new time step, after collision or transmission. One may save time by only calculating the collision time of fibres actually in collision course with the particle. Four limits are set — 1: fibre must be in front of particle, 2: fibre cannot be further left than its radius of the particles trajectory, 3: fibre cannot be further away than its radius plus movement in remainder

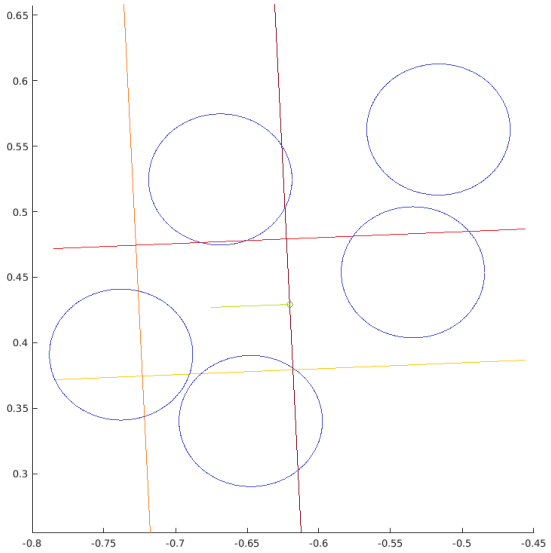


FIGURE 3.2: Example of the four boundaries that constitute the tighter box in Optimisation A. Particle's position and trajectory (green) among potentially colliding fibres (blue) seen in the xy plane. Boundaries 1 (yellow), 2 (orange), 3 (red) and 4 (dark red) shown forming a box of length dl $v + r_{\max}$ and width 2 $r_{\max}$. Only fibres with centers within the boundaries can be collided into, and demands calculation of t_c.

of the time step, and 4: fibre cannot be further right than its radius of the particles trajectory. These four boundaries for a particle and its immediate trajectory is shown in figure 3.2. This will usually shave off every or all but one fibre, saving the script from solving dot products and second degree equations. There is still a small possibility for a fibre to remain as the only fibre in the group of potentially colliding fibres without actually colliding. This could be solved with a parabola instead of a line for the third boundary, but is not expected to save time.

3.3.3 Optimisation B — Sorting fibres

Before simulating the particles' movements, the fibres' x positions vector is sorted from lowest to highest. The other vectors are also sorted so the same element number represents the same fibre in all vectors. An example of this can be seen in 3.1 This allows the search through assorted values to be confined to the y-range only. This will not speed up Method 2's runtime, as that does not make use of the three original fibre vectors in the simulation.

3.3.4 Newer efficiency improvements

The collecting and evaluating of fibres for potential water particle collision is a time consuming chore that must be done for every particle at least once for every time step.

TABLE 3.1: Example of optimisation B's effect on three five-dimensional arrays containing x, y and r values for five fibres.

(A) This is the order the fibres were generated — assorted and arbitrary.

x	-4	5	-2	-1	4
y	0	5	-4	-3	-1
r	0.4	0.5	0.5	0.7	0.3

(B) The same fibres sorted after x positions from lowest to highest. y and r in arbitrary order.

x	-4	-2	-1	4	5
y	0	-4	-3	-1	5
r	0.4	0.5	0.7	0.3	0.5

When in `DiffusionTest.m` this was to be done for restricted diffusion, not just hindered, measures were made to cut unnecessary calculation from the script. The array of potentially colliding fibres `pot_fib_col` was moved up to the scope of the `parfor n = 1:N`-loop and set to contain just the fibre restricting the water particle and its possible equivalent fibres on the other side of the unit cell.

A collision counter was implemented in the scripts to help the debugging process when unexpected behaviour was encountered.

3.4 Finding EDCs

Before the gradient, phase and signal is implemented in `PulsedGradient.m` or `ConstantGradient.m`, it was still useful to have a number to describe the particles' displacement, comparable to the ADC. I introduced the Expected Diffusion Coefficient (EDC), which just is the variance of the displacement over the diffusion time. Therefore it should be the same number as the ADC is everything goes as we want, but it is a shortcut worthy a mention. This was what was done in the project as well, when the script `MakeHistogram_xyz` generated a plot and calculated based off of the displacement values and overlayed in the histogram. It was found by regression, fitting the histogram values to the form $f(r) = \exp(r^2/4D\Delta)$.

3.5 Finding ADCs

After a simulation run of `PulsedGradient.m` or `ConstantGradient.m` with data from the matrix can be plotted to show signal attenuation as a a function of b-value. Either by reading directly off the plot, or by calculating the numbers behind it, the steepness of the curve in a logarithmic plot will give the value of the ADC. If there are two distinct flat parts of the curve, then two ADCs can be extracted from the data, hopefully one D_{IDF} and one D_{EDF}.

Chapter 4

Results

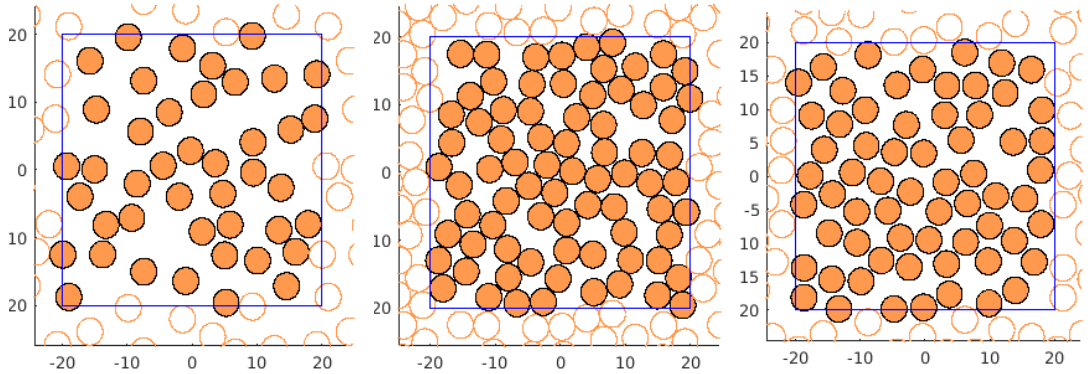
In all runs true diffusion coefficient of $D = 3\mu\text{m}^2/\text{ms}$ were used. Unless other is noted have every EDC and ADC the same unit, and all times are in ms. Signal attenuations are unitless.

4.1 Geometries

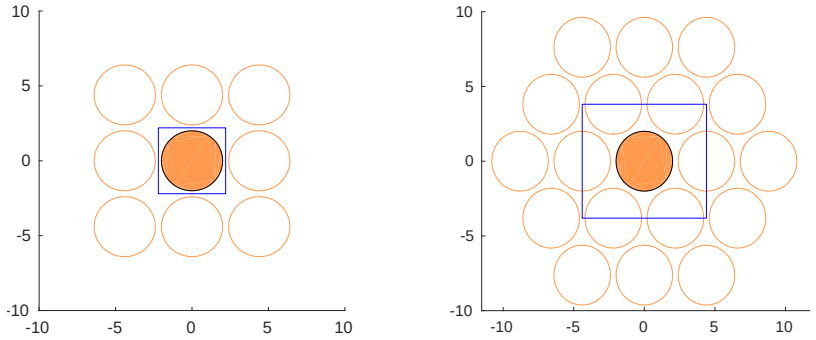
Figure 4.1 shows how the different geometries look like in the MATLAB client. Geometry 1 with fibril bundles for fibres can be seen in figure 4.2.

4.2 Runtime

The different parts of the time consuming script DiffusionTest.m turns out to demand a very skewed distribution of processing power. The script calculates free, hindered and restricted movement for a number of time steps for both constant and normally distributed step sizes, all in a row. The runtimes for all combinations of these parametres is seen in table 4.1 for N = 1000 particles, a total diffusion time T = 10 ms. It is clear that simulating restricted diffusion – where reflections are nearly unavoidable – is more intensive than hindered, which again is more intensive than the free case. As for the step lengths the processor heavier case is of normally distributions, where more calculations are needed. The biggest discrepancy is the runtimes between using regular `for`-loops and parallel `parfor`-loops for already time consuming scripts. Using parallel loops may cut the runtime to a fifth. Scripts only taking seconds to complete do not show this effect.

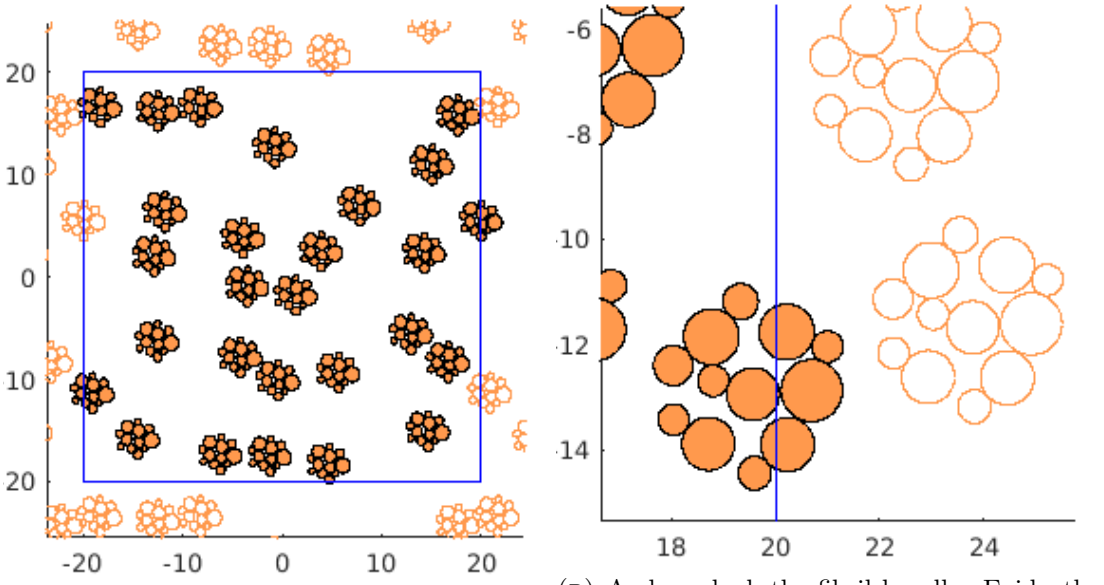


(A) Geometry 1 (B) Geometry 2 (C) Geometry 3



(D) Geometry 4 (E) Geometry 5

FIGURE 4.1: Examples of fibre environments. For G2 $d_{\min} = 0.1R$ was used. G3: $d_{\min} = 0.2R, d_{\max} = 0.3R$. G4 and G5: $d = 0.2R$.



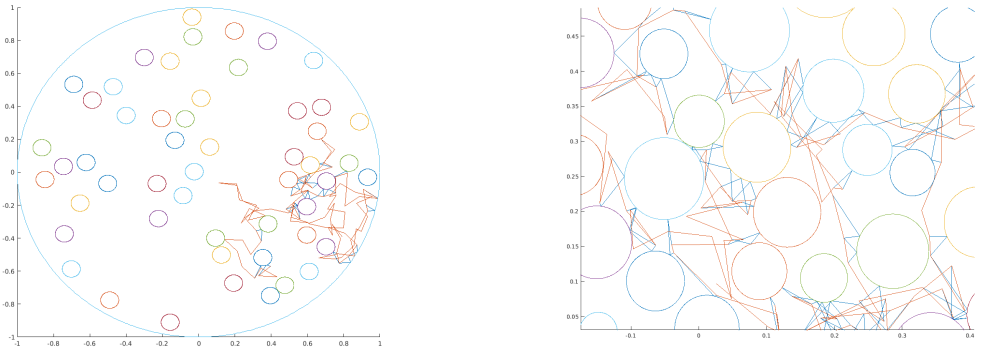
(A) The entire unit cell of fibril bundles.

(B) A closer look the fibril bundle. Evidently, entire fibrils may be placed outside its unit cell.

FIGURE 4.2: Geometry 1 with fibril bundles as replacements for the solid fibres. All bundles are identical. Solid fibrils are original, hollow are copies.

TABLE 4.1: Runtimes in seconds for N = 1000 and T = 10 ms; different time steps dt; constant (c) and normally distributed (nd) step sizes; free, hindered and restricted diffusion.

		Free		**Hindered**		**Restricted**	
	dt (ms)	c	nd	c	nd	c	nd
for	0.1	0.19	1.97	2.34	4.10	9.89	12.23
	0.01	1.3	18.6	10.0	36.3	74.7	99.6
	0.001	13	186	74	275	675	913
parfor	0.1	0.79	1.11	2.41	2.57	3.37	4.09
	0.01	0.9	3.4	3.4	8.9	15.3	21.5
	0.001	2	27	11	63	123	189



(A) Trajectory among equally sized fibrils. (B) Trajectory among fibrils with varying radii.

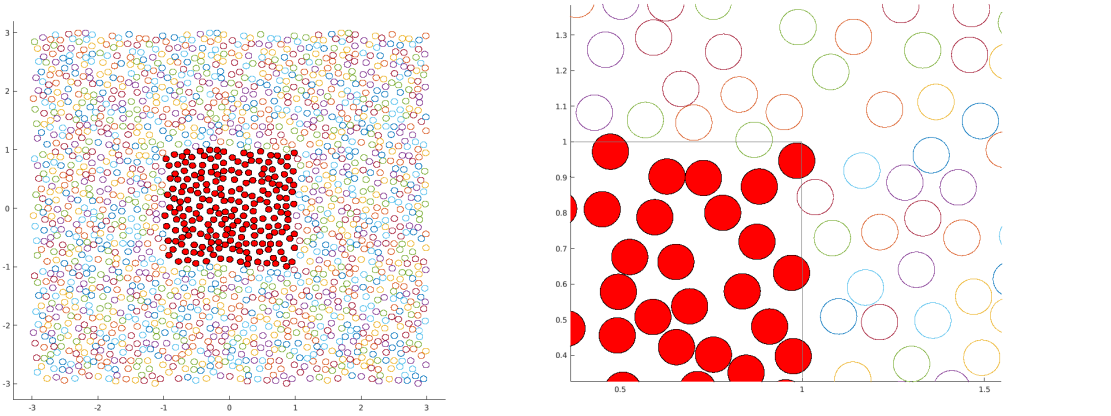
FIGURE 4.3: Trajectory of water particle moving between fibrils and bouncing into them. Orange trace plots particle's position time step by time step. Blue trace plots same particle's every direction change.

4.3 Random walk in unit cell

The trajectory of a single water particle in a scarcely populated fibril environment from the project thesis can be seen in figure 4.3. Figure 4.4 shows a 2 μm sized square unit cell filled with fibrils of size $R_{\text{fibril}} = 0.05$ μm. The cell is duplicated to every side and corner.

A particle that reaches the unit cell wall is transmitted to the other side of the cell. The particles 'sees' an endless grid of unit cells. A particle trajectory is shown twice in figure 4.5, without and with showing transmission at boundary. To the left the particle still interacts with the fibrils to the lower right of a unit cell when it here seems out of bounds. To the right the number of lines count the number of transmissions during the walk.

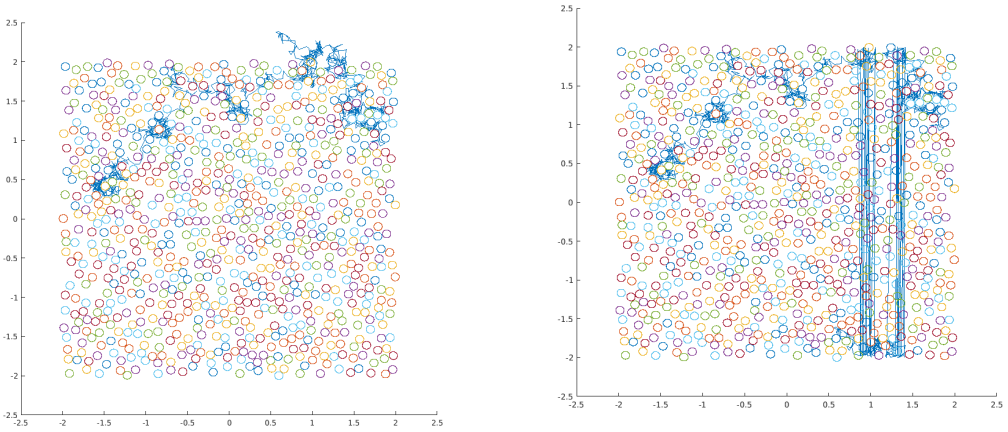
The z value of any run is of the form as seen in figure 4.6.



(A) Nine unit cells in total, original in centre.

(B) Zoomed in on one of the original cell's corner. Cell boundary in grey.

FIGURE 4.4: Generated fibrils in unit cell (filled) and duplicated (hollow) on all sides. Generated with legal boundary overlap.



(A) True trajectory.

(B) Trajectory with shifted positions at wall transmission.

FIGURE 4.5: One particles movement with time step $dt = 0.001$ ms among fibrils of radius $r = 0.05$ μm in a grid of square unit cells of side length $l = 4$ μm. Only one cell is displayed. The two trajectories shown represent the same movement.

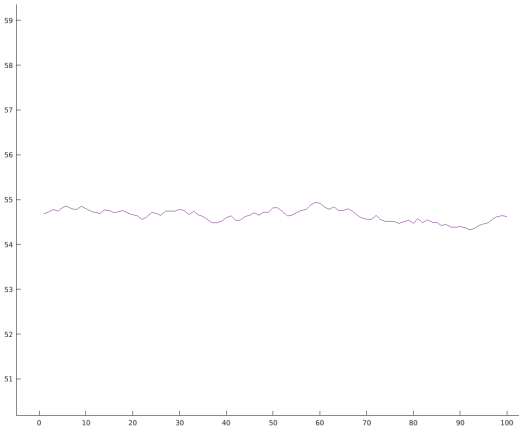


FIGURE 4.6: The z value over 100 time steps.

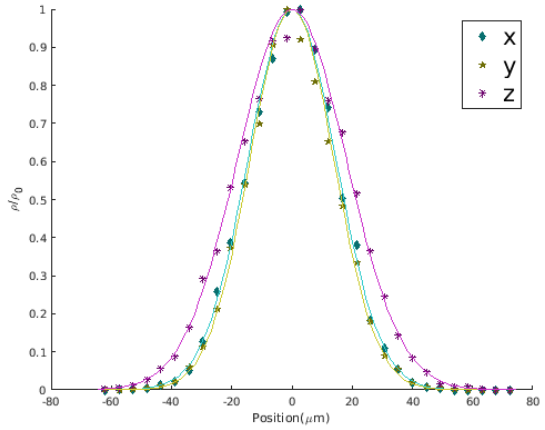


FIGURE 4.7: Histogram visualising x, y, and z values as well as regression curves to fit from a run with $N = 10000$ particles moving in $\Delta = 100$ ms with time steps of $\mathrm{d}t = 0.0001$ ms.

TABLE 4.2: EDCs found by regression of $N = 1000$ water particles in endless fibrils and runtimes of script for different time steps $\mathrm{d}t$. The true diffusion coefficient is $D = 1.65$ m^2/s, and is expected for D_z.

N=1000				
$\mathrm{d}t$ (ms)	D_x	D_y	D_z	τ (s)
1	0.13	0.14	1.31	219
0.3	0.22	0.20	1.67	289
0.1	0.30	0.31	1.68	288
0.03	0.31	0.37	1.65	338
0.01	0.55	0.45	1.50	425
0.003	0.78	0.72	1.67	631
0.001	0.70	0.81	1.58	1 024
0.0003	0.91	0.90	1.39	2 016
0.0001	0.92	0.93	1.31	4 032
0.00003	0.91	1.10	1.58	8 796
0.00001	0.88	0.97	1.44	19 347

4.4 Expected Diffusion Coefficient

4.4.1 Project thesis

Several runs of water particles in endless fibrils were run in the project, and EDCs were found. An example of a histogram showing the experimental data as well as the regression curves can be seen in figure 4.7. The EDCs as well as runtimes for the specific runs is shown in tables 4.2. Note that these numbers are results from the project thesis, with another D, than earlier stated. Without the runtimes the same data is plotted in figure 4.9. Additionally, the true diffusion coefficient is shown.

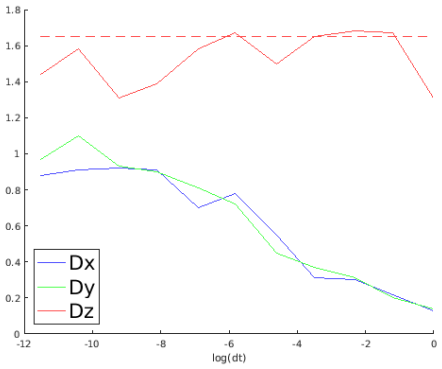


FIGURE 4.8: N=1000

FIGURE 4.9: D-values for all dimensions found by regression. Expected D_z shown in a dashed line.

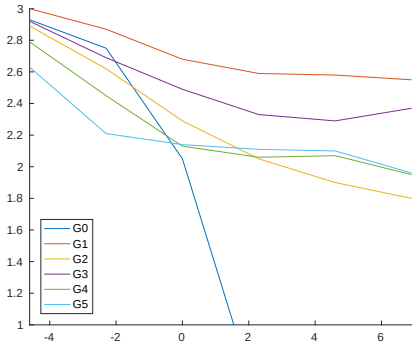


FIGURE 4.10: EDCs for all geometries as a function of $\log(T)$.

4.4.2 Geometries

Calculated EDCs for every geometry can be compared in figure 4.10. Note that these diffusion times greatly exceed what bear clinical relevance.

4.4.3 Normalised step sizes

In `DiffusionTest.m` one of the properties whose results were investigated was the distribution of steps sizes. The EDCs of constant step sizes and those of normally distributed step sizes, can be seen in figure 4.11 for all three kinds of diffusion and a given $T = 100$ ms and dts in the range of $[10^{-9}, 10^{2}]$ ms.

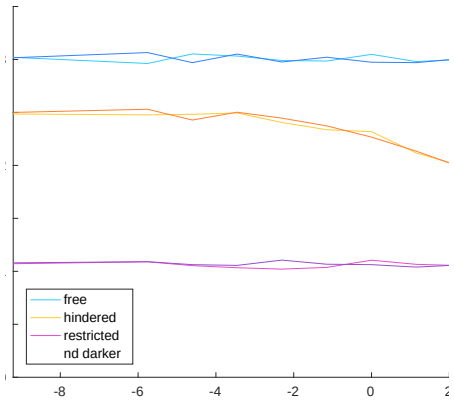


FIGURE 4.11: EDCs of diffusion in geometry 1 for a range of dts, a given diffusion time T, all three types of freedom, and both kinds of steps size distribution.

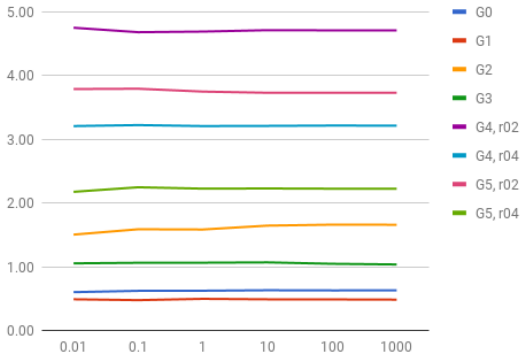


FIGURE 4.12: Average number of collisions per $\sqrt{TK}$ for a range of diffusion times.

4.4.4 Geometric attributes

From the collision counter — originally for debugging purposes — a recurring number can be extracted. The average number of collisions for a particle divided by the square root of the diffusion time and the number of time step, $N^*_{\text{col}} = N_{\text{col}}/\sqrt{TK}$, behaves almost like constant for each geometry. In figure 4.12, this number is shown for all geometries for a range of diffusion times. The unit for N^*_{col} is $1/\sqrt{\text{ms}}$.

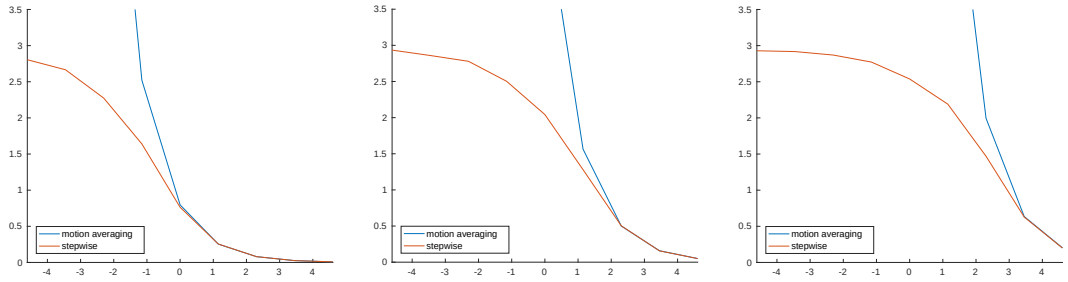
Table 4.3 shows an overview over miscellaneous geometric attributes for every geometry.

4.4.5 Motion averaging

Before motional averaging was implemented in the simulations, it was tested for three different sized spheres. The EDC of stepwise restricted diffusion superimposed on the equivalents of motional averaging can be seen in the three cases in figure 4.13. As we

	G1	G2	G3	G4g0.2	G4g0.4	G5g0.2	G5g0.4
EDC	2.59	2.05	2.33	2.06	2.23	2.11	2.19
α_D	0.86	0.68	0.78	0.69	0.74	0.70	0.73
A/V	0.38	1.48	0.98	1.85	1.20	2.99	1.70
β	0.73	0.40	0.51	0.35	0.45	0.25	0.37
N^*_{col}	0.49	1.61	1.06	4.71	3.22	3.75	2.23

TABLE 4.3: Miscellaneous geometric attributes for all geometries. G stand for geometry, and g stands for gap between neighbouring fibres in numbers of R. The N^*_{col} and EDC (and by that the tortuosity) is collected from a run with $T = 10$ and $K = 10000$. The other values are intrinsic of the particular geometry.



(A) $R = 2\mu$m (B) $R = 5\mu$m. Plausible nucleus size. (C) $R = 10\mu$m

FIGURE 4.13: Every plot shows EDCs of stepwise diffusion and motion averaging for a range of diffusion times. The restricting voxel for the three plots is a sphere of 2, 5 and 10 μm respectively and affects the critical diffusion time for when motional averaging is representable.

TABLE 4.4: Stepwise versus wholewise diffusion

T (ms)	10	3.33	1	0.33	0.1	0.03	0.01
stepwise EDC	0.10	0.30	0.94	1.76	2.33	2.62	2.80
wholewise EDC	0.10	0.30	1.00	2.99	10.99	30.11	100.14
fraction	0.99	1.00	0.93	0.59	0.23	0.09	0.03

see, the diffusion time from which the two ways of calculating EDCs coincide vary with the size of the spheres, becoming larger for larger voxels.

4.4.6 Ballistic regime

When single time steps grow too large for a certain geometry, an interesting effect take place, where the seemingly convergent D^* starts to taper. After an even larger increase in time it may seem to converge again, this time on a lower D^*. This effect is visible in the figure 4.14, and illustrate where the particles' movement goes from being diffusive to ballistic, meaning that for a single time step a particle is will collide with several fibres.

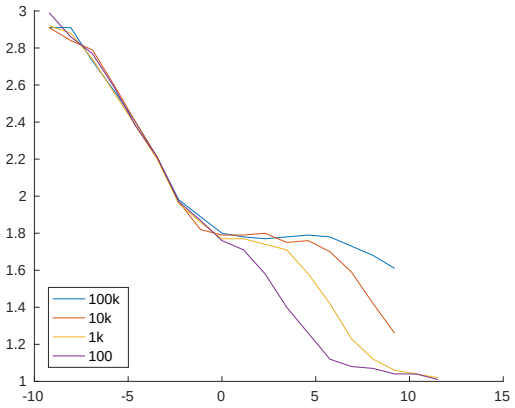
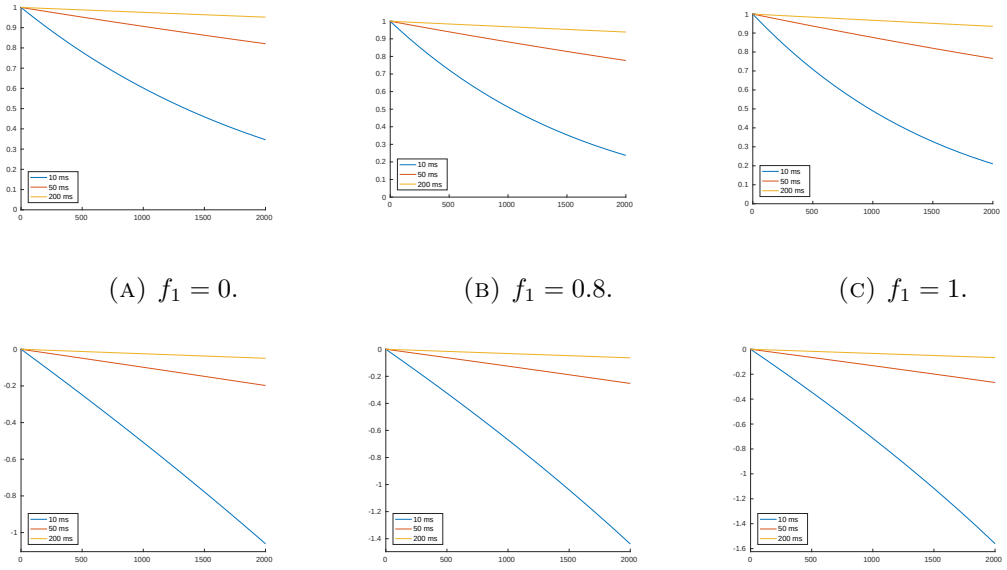


FIGURE 4.14: Geometry 5 with gap $g = 0.05R$ between neighbouring fibres. EDCs as a function of total diffusion time $\log(T)$ for different number of time steps K.



(A) $f_1 = 0$. (B) $f_1 = 0.8$. (C) $f_1 = 1$.

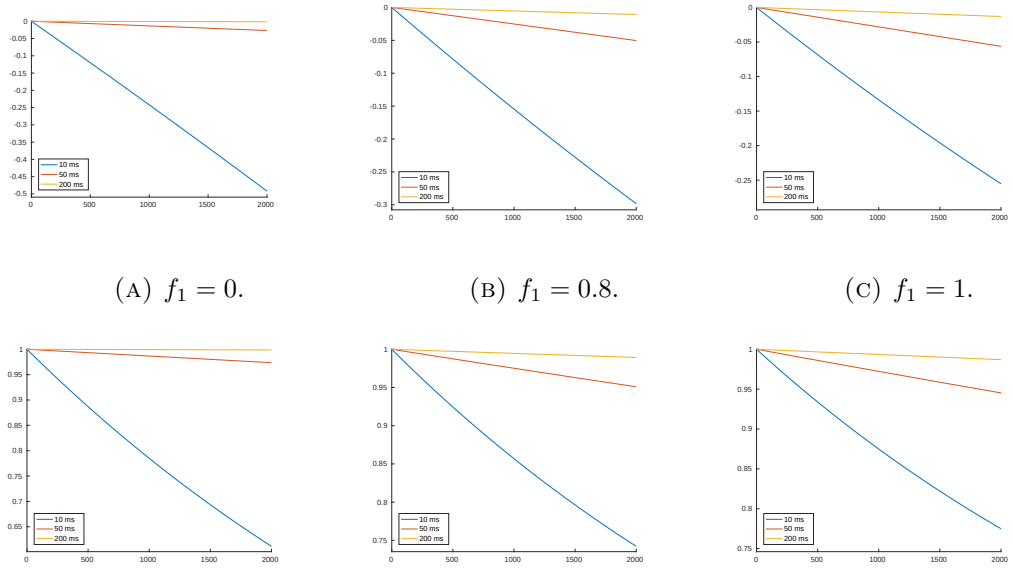
(D) $f_1 = 0$. (E) $f_1 = 0.8$. (F) $f_1 = 1$.

FIGURE 4.15: Signal attenuations after a pulsed gradient spin echo sequence. All restricted left, $f_1 = 0.8$ middle, and all hindered right. Linear upper and logarithmic lower.

4.5 Signals

4.5.1 Pulsed gradient

In the simulations with pulsed gradient, gradient duration $\delta = 0.001$ms was used. In figure 4.15 signal attenuations from a simulated pulsed gradient spin echo sequence with different fractions f_1 of hindered particles can be seen.



(A) $f_1 = 0$. (B) $f_1 = 0.8$. (C) $f_1 = 1$.

(D) $f_1 = 0$. (E) $f_1 = 0.8$. (F) $f_1 = 1$.

FIGURE 4.16: Signal attenuations after a constant gradient spin echo sequence. All restricted left, $f_1 = 0.8$ middle, and all hindered right. Linear upper and logarithmic lower.

4.5.2 Constant gradient

In figure 4.16 signal attenuations from a simulated constant gradient spin echo sequence can be seen. The same values for f_1 as for the pulsed gradient case is used.

Chapter 5

Discussion

In general during simulations, the particles seem well behaved when being initially placed, finding directions for every time step, and reflecting off of both fibril and fibre walls. Both results and debugging during the project and master's thesis indicate no errors or bias.

5.1 Runtime

Note that experience suggests runtimes can vary as much as 10 % off the named results based on composition of fibre and positioning of water particles. The runtime greatly depends on number of reflections, and each run presents great variability in to which degree particles find pockets of free space or get stuck in a tight cluster of fibrils. This is the case both several runs on same set of fibrils as well as runs on different sets of fibrils.

Different measures were initiated for lowering the runtime which we may assume was successful, interpreting the table 4.1.

5.2 Normalised step sizes

As figure 4.11 clearly shows, there are no greater difference between the EDCs of diffusion with constant step sizes and those of normally distributed step sizes than the variations in the EDCs for each individual case. Table **??** shows that including the extra step of changing the step length `dr` for every step consumes a lot of time. When the outcome is indistinguishable from those of constant step sizes, it is only reasonable to discard the normally distributed step sizes in further simulations.

5.3 Ballistic regime

As seen in figure refballistic there needs to be at least to be $K = 1000$ time steps when simulating diffusion in geometry 5 with a gap of $g = 0.05R$. When the diffusion time exceeds 10 ms, and therefore exceeds the order of magnitude for $T2*$ relaxation time and spin echo sequences, $= 100$ can already be deemed too small for accurate calculation of EDCs. For a higher K, the EDC drops off at a higher T. Surprisingly all EDCs seem to flatten out for an even higher T than the drop off diffusion time. The plot is not prioritised completed for all Ts and Ks, because of the time it consumed. Interestingly, all diffusion coefficients seem to converge again at the highest diffusion times. Reading out what should be the correct D^* is not hard, as all curves show sign of convergence at a horizontal line near 1.8 $\mu m^2/ms$.

Unknowingly, this is the effect seen in figure **??** and table 4.2 from the project thesis. This is when diffusion was simulated in a fibril environment, not a fibre environment. Therefore, much lower diffusion times and step sizes are needed for reaching the ballistic regime.

A result that does not drop for large values of T is the recorded number of collisions divided by the square root of TK. The number of collision recorded does not include direction changes due to a new time step. As the number of steps increases, we have more diffusive and less ballistic behaviour. It's not clear whether this plays into the fact that the number of collisions are a constant for every geometry, even for very long diffusion times.

It should be noted that these diffusion times in no way are applicable for clinical uses for at least two reasons. A normal magnetic field gradient strength does not allow $T1$ and $T2$ relaxation over an order of magnitude of a second. Also, water in the two compartments are interchangeable — meaning hindered particles can become restricted and vice versa — which would influence the interpretation of the biexponential representation.

5.4 Geometries

For all geometries the fibre radius is fixed at $r = 2\mu m$. In the figure 4.10, we see EDCs for a range of diffusion times on a logarithmic scale. All of them start at about $D = 3$ and quickly fall for longer diffusion times. With the exception of G0 and G2, the EDCs flattens out, giving a way of defining D^* for those geometries. When listing the EDCs in rising or falling order, G0 and G2 are also the only geometries that keeps that order from being the same for all diffusion times.

5.4.1 Geometry 0 – voxel

This geometry does not contain fibres, but rather a single hollow sphere of radius $r_{vox} = 5\mu$m, in which water diffuses. The EDCs follow a restricted diffusion pattern for the different diffusion times, as one would expect. This curve has deemed harder to replicate with a function of the form $D(T) = D/[1 + (T/T')^a]$ than expected.

5.4.2 Geometry 1 – scattered

This geometry reveals its way of random generation, as it varies a lot in local fibre density. Some clusters are spreads over the unit cell, but most of the fibres are scattered in a way that opens up of almost unhindered movement for the water particles. Firstly we see the EDCs lower than D, but still generally high. True random generation is an inefficient way of packing fibres, and probably far away from how we could expect collagen mass in both healthy and unhealthy FGT.

It is much to consider when determining how long the smallest time step should be, seeing as at the tightest cluster fibres are placed more than a order of magnitude closer than for the most part of the environment.

5.4.3 Geometry 2 – tight

Seeing as this geometry only demands a certain tightness — every new fibre has to be placed with a gap to the closest fibre smaller than a given d_{G2}. This opens up for the possibilities of creating almost closed compartments in the xy-plane, revealing ADCs similar to those of restricting environments. This geometry alone was tested for $K = 10000$ as the EDCs for a lower number of steps never really flattened out.

Although it may be solved with a higher number of time steps, the EDC curve never flattens out for these diffusion times, indicating that restricted diffusion behaviour may be achieved without actual enclosement in an environment of fibres of this size.

5.4.4 Geometry 3 – semi tight

This geometry does not have the restricting properties G2 has, due to the requirement of not generating fibres within a radius of another fibre. It is certainly tighter than G1, which is reflected in its EDC. The geometry naturally has fewer variations and clusters, and its minimum step size is therefore easier to determine.

The EDC of this geometry flattens out to a lower D^* than the one of G1, which is expected, as G3 is more populated than G1.

5.4.5 Geometry 4 – square lattice

The square lattice is quite efficiently packed, with a relatively low porosity. However, in comparison to the triangular lattice, the same porosity does not mean the same A/V ratio or the same gap between fibres. This comes of every fibre having only four equally distant neighbours, and the four next are $\sqrt{2}$ further away. The pocket between four fibres are relatively big compared to the gap between two, at least for an effectively packed structure. This could be thought to give the EDCs for G4 restricting properties, but as we see from figure 4.10 that is not the case.

5.4.6 Geometry 5 – triangular lattice

The triangular lattice is the most volume efficient way of packing equal sized circular particles in two dimensions, and as such may be most similar to how collagen are stacked in breast tissue.

5.5 Geometrical attributes

Given a series of $N_T = 3$ diffusion times $T = [10, 20, 30]$ and a fixed number of steps $K = 1000$, the simulation would have to calculate a total of $K_{TOT} = K = N_T K = 3000$ steps. If rather the duration $\mathrm{d}t$ of the time step was fixed at a value, the shortest diffusion time $K = 1000$ steps would give $\mathrm{d}t = 0.01$. This opens up for continuing the run for $T = [20, 30]$ after $T = 10$ is done and still calculate a total of $K_{TOT} = K = \max(T)/\mathrm{d}t = 3000$ steps. If the distributions of diffusion times weren't linear, but increasingly geometrical, a fixed number of steps would the be the most runtime efficient.

What else needs considering is how long you can make every time step before the simulation enters the ballistic regime, and the results start behaving differently. It would be of use to assign a number $\mathrm{d}r_{\max}$ to every geometry to ensure diffusive behaviour, but it's not easy to describe a geometry or structure with a set of numbers, especially not a single number. None of the attempts to automatically calculate a number to indicate how long the shortest step should be has been successful. There are also few intuitive links between the different values of the geometrical attributes shown in table 4.3 and the geometry's equivalent EDCs.

5.6 Signals

5.6.1 `PulsedGradient.m`

In the figure 4.15 we see a signal attenuation as a function of b-values curving downward both in the linear and logarithmic plot. This may be an artefact of the simulation, and is not expected to appear in an experimental case. As stated and shortly explained in [2], poles may appear for high b-values when calculating the signal from phases. However, b-values far higher than $b = 2000$ shows anomalous behaviour, which is not the case here. Also the poles appear in very different b regimes for the different values of T, while these results show a rather uniform curvature for every T, meaning other reasons could be behind the dip in the signal. The phenomenon probably is a result of the simulation's unrealistic simplicity.

The ADCs from this script do not vary much for the differing f_1s. While the EDCs in general seem to fit experimental data for specifically healthy breast tissue[18], the signal equivalent may have some flaws in its calculation.

5.6.2 `ConstantGradient.m`

The signal attenuation curves downward in the linear plot in figure 4.16, but, in contrast to `PulsedGradient.m`, it curves upward in the logarithmic plot. This is in agreement with the shape of the curves for the equivalent simulations in [2]. It is, however, difficult from the plot to determine the two Ds if we were to fit the signal to a biexponential representation, as the graphs curve very slightly all over and has no flat parts.

5.7 Assumptions and possible sources of error

There are always assumptions made when translating a real phenomenon into something programmable and a into a model used for simulations.

5.7.1 Interpretation of microscopy images

As earlier mentioned, it is a fairly big simplification to give the model of the cell nucleus in the FGT an equal size and spherical shape. It is shown that adjusting the size, will affect the runtime — as the feasibility for motion averaging is given by the diffusion time and the square root of the sphere size. Moreover, adjusting the size will give differing contributions to the MR signal. If we use the bi-exponential representation we would

see that the restricted contribution, determined by ADC_fast, skew the signal decay. An increase in global nucleus size would flatten the curve for low b-values, and a decrease would steepen it. If, however, the nuclei varied in size, as is what we see in the microscopy images, the signal for low b-values would be a sum of steep and flat contributions, maybe nullifying each other, making use of an average a good assumption.

We cannot determine the roughness of the fibroblast nucleus surface, nor the roughness' affect on the signal. If, instead of spheres, the nuclei were ellipsoids, it could be argued that the signal would only be affected in the the directions in which the nuclei are shortened, and that the final affect on the signal would be as if the nuclei were spheres of radii equal to the mean of the ellipsoids' principal semi-axes.

5.7.2 Pseudodiffusion and flow between compartments

We have mainly focused on stagnant water as a source of diffusing hydrogenous matter. While it is with a large margin the biggest contribution to the MR signal after an RF pulse, there is a significant portion of water not hindered by collagen and not restricted by nucleus walls, but driven through a network of capillaries by the pressure of a beating heart. The water in blood flows in circulation to and fro the heart, within a network of small vessels so rich and intricate, that the blood as a whole at no point can be said to flow in any specific direction. Instead, if a collection of arbitrary blood water particles is chosen to observe, it will seemingly diffuse, but with much higher speeds. This is called pseudodiffusion and can only be shown on MR images with low b-values.

Chapter 6

Conclusion

MRI is a widely used tomographic technique that can diagnose many kinds of tissues. There are also a lot of ongoing research in the field of medical physics regarding tomography. Despite this, There are still a lot to explore regarding the microstructure of both healthy and tumour-stricken tissue. Although there are new and precise representations that fit certain experimental data, the step forward in a clinical sense still awaits a model that can explain, justify and predict the same kind of results.

This thesis does not make a breakthrough in understanding the relationship between signals and microstructure, but it may be a step forward. It explores the effects of the ballistic regime, maps the feasibility of motion averaging, discusses the limits between restriction and hindrance, and finds signals in agreement with experimental data. Diffusion coefficients both derived from displacement and phase shift, and for three types of diffusion, have proven themselves attainable.

6.1 Further work

Although much of the tool box that is the MATLAB environment created by Ane and myself has been used to calculate several types of signal, there are still potential for others to pick up the code and continue the work on these scripts. There are plenty of more one can add to these simulations if one sees the necessity in it. Simulating other spin echo sequences, or maybe not spin echo at all, could be done with the same core code of diffusion of water particles in fibre environments. Generating other kinds of tissue representation could also be done without making the diffusion part of the code obsolete. If not writing more code, using already existing code with a purposeful focus to explore other cases than what is touched by this thesis could reveal much with a relatively small effort.

Appendix A

Code and pseudocode

Following are entire scripts written and run during project and master semesters. There are also some pseudocode for scripts whose entire code not deemed necessary for publication. More scripts were written, however the ones presented are carrying the most importance.

A.1 `RestrictedReplacement.m`

This script was written to investigate the possibilities of replacing the stepwise restricted diffusion with motion averaging.

```
%************************************************************************
%*** Measures the EDC of a single random step inside a fibre or cell    ***
%*** as well as the EDC of step wise diffusion for the same time.       ***
%************************************************************************

tic

N = 10000;                  %Number of particles []
R = 5;                      %Voxel radius [μm]
T = 0.01;                   %Total diffusion time [ms]
D = 3;                      %True diffusion coefficient [μm^2/ms]

K = 1000;                   %Number of steps []
dt = T/K;                   %Time step [ms]
dr = sqrt(6*D*dt);          %Stepsize [μm]
```

```
16
17 x0 = zeros(1,N);
18 y0 = zeros(1,N);
19 z0 = zeros(1,N);
20
21 x_end = zeros(1,N);
22 y_end = zeros(1,N);
23 z_end = zeros(1,N);
24
25 cols = zeros(1,N);
26
27 n = 1;
28 while n <= N
29     x0(n) = 2*R*(1-2*rand());
30     y0(n) = 2*R*(1-2*rand());
31     z0(n) = 2*R*(1-2*rand());
32     if sqrt(x0(n)*^2 + y0(n)^2 + z0(n)^2) < R
33         n = n + 1;
34     end
35 end
36
37 n = 1;
38 while n <= N
39     x_end(n) = 2*R*(1-2*rand());
40     y_end(n) = 2*R*(1-2*rand());
41     z_end(n) = 2*R*(1-2*rand());
42     if sqrt(x_end(n)^2+y_end(n)^2+z_end(n)^2) < R
43         n = n + 1;
44     end
45 end
46
47 x = x_end - x0;
48 y = y_end - y0;
49 z = z_end - z0;
50
51 avg_x = sum(x)/N;
52 avg_y = sum(y)/N;
53 avg_z = sum(z)/N;
54 avg_ma = (avg_x+avg_y+avg_z)/3;
```

```
55
56 EDC_x = sum(x.*x)/(2*N*T);
57 EDC_y = sum(y.*y)/(2*N*T);
58 EDC_z = sum(z.*z)/(2*N*T);
59 EDC_ma = (EDC_x+EDC_y+EDC_z)/3;
60
61 %*****************************Loop over all particles*******************
62
63 parfor n = 1:N
64
65     if 100*n/N == round(100*n/N)
66         string = ['ParticleNr : ', num2str(n), ' of ',num2str(N)];
67         if 10*n/N == round(10*n/N)
68             string = ['ParticleNr : ', num2str(n), ...
69             ' of ',num2str(N), ', estimated additional 10% complete'];
70         end
71         disp(string)
72     end
73
74     x_pos = zeros(1,K);    %Initializing arrays to
75     y_pos = x_pos;         %save all positions.
76     z_pos = x_pos;         %
77
78
79     x_pos(1) = x0(n);      %Adds the initial position.
80     y_pos(1) = y0(n);      %
81     z_pos(1) = z0(n);      %
82
83     k = 2;
84     while k <= K+1
85
86         pos = [x_pos(k-1) y_pos(k-1) z_pos(k-1)];   %Current position.
87
88         d_cos_theta = 2*rand - 1;                   %Finds direction.
89         d_theta = acos(d_cos_theta);                %
90         d_phi = 2*pi*rand;                          %
91
92         dx = dr*cos(d_phi)*sin(d_theta);            %Computes the step.
93         dy = dr*sin(d_phi)*sin(d_theta);            % [μm]
```

```
94          dz = dr*d_cos_theta;                          %
95
96          dxy = dr*sin(d_theta);              %Step's reach in xy-plane
97
98          v = [dx dy dz]/dt;                  %Velocity [μm/ms]
99
100         ddt = dt;                           %Remaining time of time step
101
102         %(III)***** Loop in case of multiple collision **********
103         %********** in same time step.                  **********
104
105         collide = 1; %Assuming collision
106
107         while collide == 1
108
109             %***** Computes time until collision from current *****
110             %***** position given the velocity v.             *****
111
112             px = pos(1);                    %Current x position
113             py = pos(2);                    %Current y position
114             dl = norm(v(1:2))*ddt;          %Remaining step movement
115             phi = atan2(v(2),v(1));         %Current direction
116
117             %ABC formula used to find time of collision t_c
118             A = dot(v,v);
119             B = 2*dot(pos,v);
120             C = dot(pos,pos)- R^2;
121             SQ = sqrt(B^2 - 4*A*C);
122             t_c = (-B + SQ)/(2*A);
123             %*****************************************************
124
125             if t_c >= ddt                   %The step does not lead to
126                 collide = 0;                % collision
127             end
128
129             if collide == 1                         %Collision
130
131                 cols(n) = cols(n) + 1;
132
```

```
133                 %Surface normal in collision point
134                 normal = pos + v*t_c;   %Collision point
135                 normal = normal/norm(normal);
136                 v_new = v - 2*dot(v,normal)*normal;
137                 pos = pos + v*t_c;  %Updates position.
138                 v = v_new;          %Updates velocity.
139                 ddt = ddt-t_c;      %Updates ddt in case of multiple
140                 % collisons in one time step.
141
142             else                    %Not collision; assumption wrong.
143
144                 x_pos(k) = pos(1) + v(1)*ddt; %Finds new positions.
145                 y_pos(k) = pos(2) + v(2)*ddt; %
146                 z_pos(k) = pos(3) + v(3)*ddt; %
147
148             end
149         end
150
151         %(III)*************************************************
152         k = k + 1;
153     end
154
155     x_end(n) = x_pos(k-1);
156     y_end(n) = y_pos(k-1);
157     z_end(n) = z_pos(k-1);
158
159 end
160
161 x = x_end - x0;
162 y = y_end - y0;
163 z = z_end - z0;
164
165 avg_x = sum(x)/N;
166 avg_y = sum(y)/N;
167 avg_z = sum(z)/N;
168 avg_r = (avg_x+avg_y+avg_z)/3;
169
170 EDC_x = sum(x.*x)/(2*N*T);
171 EDC_y = sum(y.*y)/(2*N*T);
```

```
172 EDC_z = sum(z.*z)/(2*N*T);
173 EDC_r = (EDC_x+EDC_y+EDC_z)/3;
174
175 cols = sum(cols)/N;
176
177 toc
```

A.2 Fibres.m

The first fibre generating script written. This also includes the possibility of fibres as fibril bundles.

```
%************************************************************************
%*** GEOMETRY 1                                                       ***
%*** Randomly generates a square unit cell filled with small massive  ***
%*** cylinders (fibres) of equal radii. Returns a matrix with all     ***
%*** fibres' x-positions, y-positions, radii as well as information   ***
%*** about the cell, which is width, length and whether or not if this ***
%*** geometry is a lattice structure.                                 ***
%************************************************************************

l = 20;         % Half length of unit cell side [μm]
R_fibre = 2;    % Radius of massive cylinders or bundle boundaries [μm]
R_fibril = 0.5; % Radius of massive cylinders in bundles [μm]
N_fibril_fails = 100000;% # of max consequtive unsuccessful placements []
N_fibre_fails = 10;     % # of max consequtive unsuccessul placements []

L = 2*l;

fibre_xpos = L*rand - l;
fibre_ypos = L*rand - l;

N_fibres = 1;

n_tries = 0;

while n_tries < N_fibre_fails
    newxpos = L*rand - l;
    newypos = L*rand - l;
    collide = false;

    for n = 1:N_fibres
        if ~collide
            for i = -L:L:L
                if ~collide
```

```
34                      for j = -L:L:L
35                          if (newxpos-fibre_xpos(n) + i)^2 + ...
36                                  (newypos-fibre_ypos(n) + j)^2 < ...
37                                  (2*R_fibre)^2
38                              collide = true;
39                              n_tries = n_tries + 1;
40                              break
41                          end
42                      end
43                  end
44              end
45          end
46      end
47
48      if collide == false
49          N_fibres = N_fibres + 1;
50          fibre_xpos(N_fibres) = newxpos;
51          fibre_ypos(N_fibres) = newypos;
52          n_tries = 0;
53      end
54  end
55
56  disp(N_fibres)
57
58  cell_sizes = zeros(1, N_fibres);
59  cell_sizes(1) = l;
60  cell_sizes(2) = l;
61  cell_sizes(3) = 0;
62
63  fibril_xpos = 2*R_fibre*rand - R_fibre;
64  fibril_ypos = 2*R_fibre*rand - R_fibre;
65  while fibril_xpos^2 + fibril_ypos^2 > (R_fibre - R_fibril)^2
66      fibril_xpos = 2*R_fibre*rand - R_fibre;
67      fibril_ypos = 2*R_fibre*rand - R_fibre;
68  end
69  fibril_radii = R_fibril;
70
71  n_fibrils = 1;
72
```

```
n_tries = 0;
temp_R_fibril = R_fibril;

while n_tries < N_fibril_fails
    newxpos = 2*R_fibre*rand - R_fibre;
    newypos = 2*R_fibre*rand - R_fibre;
    collide = false;
    closeTo = 0;

    for n = 1:n_fibrils
        if ((newxpos-fibril_xpos(n))^2 + ...
            (newypos-fibril_ypos(n))^2 < (R_fibril+temp_R_fibril)^2 || ...
            (newxpos^2 + newypos^2 > (R_fibre - temp_R_fibril)^2))
            collide = true;
            n_tries = n_tries + 1;
            if (temp_R_fibril > 0.3)
                temp_R_fibril = temp_R_fibril - 0.01;
            else
                temp_R_fibril = R_fibril;
            end
            break
        end
    end

    if collide == false
        n_fibrils = n_fibrils + 1;
        fibril_xpos(n_fibrils) = newxpos;
        fibril_ypos(n_fibrils) = newypos;
        fibril_radii(n_fibrils) = temp_R_fibril;
        n_tries = 0;
        temp_R_fibril = R_fibril;
    end
end

disp(n_fibrils)

for i = 1:n_fibrils
    minDist = Inf;
    for j = 1:n_fibrils
```

```
112         tempDist = sqrt((fibril_xpos(i)-fibril_xpos(j))^2 +  ...
113                         (fibril_ypos(i)-fibril_ypos(j))^2) - ...
114                         (fibril_radii(i) + fibril_radii(j));
115         if i ~= j && tempDist < minDist
116             minDist = tempDist;
117         end
118     end
119     fibril_radii(i) = fibril_radii(i)+minDist;
120 end
121
122 hold on
123 for i=-L:L:L
124     for j=-L:L:L
125         for n=1:N_fibres
126             PlotCircle(fibre_xpos(n) + i, ...
127                         fibre_ypos(n) + j, R_fibre, false)
128         end
129     end
130 end
131
132 for n=1:n_fibres
133     for o=1:n_fibrils
134         PlotCircle(fibre_xpos(n) + fibril_xpos(o), ...
135                     fibre_ypos(n) + fibril_ypos(o), fibril_radii(o), true)
136     end
137 end
138
139 s = 0.1;            % Recommended smallest step size [μm]
140 cell_sizes(4) = s;
141
142 fibre_radii = R_fibre*ones(1,N_fibres);
143 fibre_vals = [fibre_xpos; fibre_ypos; fibre_radii; cell_sizes];
144 fibril_bundle_vals = [fibril_xpos; fibril_ypos; fibril_radii];
145 save 'fibres1.mat' fibre_vals
146 save 'fibrils.mat' fibril_bundle_vals
```

A.3 Fibres2.m

`Fibres2.m` functions very similarly as `Fibres.m`, the only difference being the restriction of tightness. A parameter `tightness` (in units of micrometers) is chosen in the beginning of the script, and as each fibre is placed, it needs to be at most `tightness` away from another fibre. This means that one close neighbour is sufficient.

```
fibre radius r
tightness t

[...]

for a number of tries
    random legal xpos and ypos
    for all other fibres
        for fibre copies L away in both x and y
            if |xpos, ypos| - fibre's |x, y| < 2r + t
                if |xpos, ypos| - fibre's |x, y| < 2r
                    collision: increase number of unsuccessful tries
                    break to outermost loop
                else (for all cases)
                    save xpos and ypos
                end
            else
                not tight enough: increase number of unsuccessful tries
                break to outermost loop
            end
        end
    end
    save xpos and ypos
end

[...]
```

A.4 Fibres3.m

```
%************************************************************************
%*** Randomly generates a square unit cell filled with small massive    ***
%*** cylinders (fibres). The microstructure is denser than for Fibres.m ***
%*** Generates positions of the fibres with equal radii.                ***
%*** Returns a matrix with all       ***
%*** fibres' x-positions, y-positions, radii as well as information     ***
%*** about the cell, which is width, length and whether or not if this  ***
%*** geometry is a lattice structure.                                    ***
%************************************************************************

l = 20;          %Half length of unit cell side [μm]
R_fibre = 2;     %Radius of massive cylinders or bundle boundaries [μm]
R_fibril = 0.5; %Radius of massive cylinders in bundles [μm]
N_fibril_fails = 100000;%# of max consecutive unsuccessful placements []
N_fibre_fails = 100000; %# of max consecutive unsuccessful placements []
tightness = 0.3*R_fibre;%Maximum distance to all neighbouring fibres [μm]
looseness = 0.2*R_fibre;%Minimum distance to all neighbouring fibres [μm]

L = 2*l;

fibre_xpos = L*rand - l;
fibre_ypos = L*rand - l;

N_fibres = 1;

n_tries = 0;

while n_tries < N_fibre_fails
    newxpos = L*rand - l;
    newypos = L*rand - l;
    collide = false;
    tightfit = false;

    for n = 1:N_fibres
        if ~collide
            for i = -L:L:L
```

```
37                  if ~collide
38                      for j = -L:L:L
39                          if (newxpos-fibre_xpos(n) + i)^2 + ...
40                             (newypos-fibre_ypos(n) + j)^2 < ...
41                             (2*R_fibre+tightness)^2
42                              tightfit = true;
43                              if (newxpos-fibre_xpos(n) + i)^2 + ...
44                                 (newypos-fibre_ypos(n) + j)^2 < ...
45                                 (2*R_fibre+looseness)^2
46                                  n_tries = n_tries + 1;
47                                  collide = true;
48                                  break
49                              end
50                          end
51                      end
52                  end
53              end
54          end
55      end
56
57      if tightfit && ~collide
58          N_fibres = N_fibres + 1;
59          fibre_xpos(N_fibres) = newxpos;
60          fibre_ypos(N_fibres) = newypos;
61          n_tries = 0;
62      end
63  end
64
65  disp(N_fibres)
66
67  cell_sizes = zeros(1, N_fibres);
68  cell_sizes(1) = l;
69  cell_sizes(2) = l;
70  cell_sizes(3) = 0;
71
72  hold on
73  for i=-L:L:L
74      for j=-L:L:L
75          for n=1:N_fibres
```

```
76              PlotCircle(fibre_xpos(n), ...
77                         fibre_ypos(n), R_fibre, false)
78          end
79      end
80  end
81
82  plot([-1,1,1,-1,-1],[1,1,-1,-1,1])
83
84  s = 0.01;
85  cell_sizes(4) = s;
86
87  fibre_radii = R_fibre*ones(1,N_fibres);
88  fibre_vals = [fibre_xpos; fibre_ypos; fibre_radii; cell_sizes];
89  save 'fibres3.mat' fibre_vals
```

A.5 Fibres4.m

Unlike the previous geometries, this one is determined by two parameters and no randomness. The fourth geometry is a square lattice structure, naturally with a square unit cell and fibres along the cell wall. Additionally, the next closest neighbours are added in the information matrix `fibre_vals` and exported to be read by a diffusion script. This is done to ensure that all potentially colliding fibres are collected even for step sizes large enough to be comparable to the unit cell size.

```
%************************************************************************
%*** Generates a square unit cell filled with a single small massive    ***
%*** cylinder (fibre). This represents a square lattice.                ***
%***                                                                    ***
%***    o o o   Square unit cell                                        ***
%***    o|ø|o   l_y = sqrt(3)/2 * l_x                                   ***
%***    o o o                                                           ***
%***                                                                    ***
%************************************************************************

R = 2;                  % Radius of massive cylinders [μm]
gap = 0.2*R;            % Distance between fibres [μm]

l = R + gap/2;          % Implied half length of unit cell side [μm]
L = 2*l;                % Whole length of unit cell [μm]

disp('Number of fibres in cell')
disp(1)
disp('Number of fibres exported')
disp(9)

hold on
for i=-L:L:L
    for j=-L:L:L
        PlotCircle(0+i, 0+j, R, false)
    end
end

plot([-l,l,l,-l,-l],[l,l,-l,-l,l])
```

```
30
31 s = gap/10;
32
33 fibre_vals = [[-L,-L,-L, 0, 0, 0, L, L, L]; ...
34               [-L, 0, L,-L, 0, L,-L, 0, L]; ...
35               [ R, R, R, R, R, R, R, R, R]; ...
36               [ l, l, 1, s, 0, 0, 0, 0, 0]];
37
38 save 'fibres4.mat' fibre_vals
```

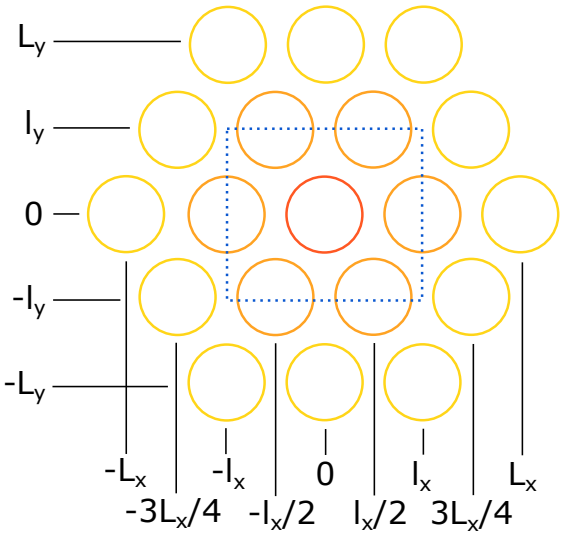


FIGURE A.1: A fibre (red) has six neighbouring fibres (orange) equally far away that all half-intersects the rectangular unit cell (blue) of width L_x and length L_y. Exported are also the next closest fibres (yellow), making the total number of fibres 19.

A.6 `Fibres5.m`

In the fifth geometry, a triangular lattice geometry, every fibre has six neighbours of equal distance. This means that the unit cell cannot be square. Once again, `gap` determines the distance between fibres, making l_x the same as in the square lattice. When every neighbouring fibres is placed on the unit cell wall, the length is $\sqrt{3}/2$ greater than the width. There are seven fibres in the unit cell, and 19 if we include one more layer. As in geometry 4, the extra layer is exported to `'fibres5.mat'`. The positions then becomes:

A.7 PulsedGradient.m

```
%************************************************************************
%*** Computes the final signal of a spin echo sequence with an          ***
%*** infinitesimal gradient duration for N particles diffusing in a     ***
%*** certain fibre environment.                                         ***
%************************************************************************

tic

%*** Parametres ***
N = 10000;              %Number of particles
Gs = 0:50:1700;          %Magnetic field gradient strength [T/m]
delta = 0.001;           %Duration of gradient application [ms]
f = 0.8;                 %Fraction of hindered particles
Ts = [10,50,200];       %Diffusion times [ms]
%******************

%*** Constants ***
D = 3;                          %True diffusion coefficient [μm^2/ms]
gamma = 2.675*10^8;             %Gyromagnetic ratio rad/(T*s)
K_min = 1000;                   %Number of hindered minimum time steps
R_r = 5;                        %Radius of spherical nuclei [μm]
K_r = 100;                      %Good assumption for R_r = 5 μm
%*****************

%*** Initialisations ***
N_h = f*N;                      %Number of hindered particles
N_r = N-N_h;                    %Number of restricted particles
N_T = length(Ts);               %Number of diffusion times
N_G = length(Gs);               %Number fof gradient strengths
Ks = K_min*ones(1,N_T);         %Number of time steps for each diff. time
dts_r = Ts/K_r;                 %Time steps for each diffusion time

avg_xyz_h = zeros(N_T,3);       %Average displacements
avg_h = zeros(N_T,1);           %
avg_xyz_r = zeros(N_T,3);       %
avg_r = zeros(N_T,1);           %
```

```
37 avg = zeros(N_T,1);              %
38
39 EDC_xyz_h = zeros(N_T,3);        %Expected diffusion coefficient (var/T)
40 EDC_h = zeros(N_T,1);            %
41 EDC_xyz_r = zeros(N_T,3);        %
42 EDC_r = zeros(N_T,1);            %
43 EDC = zeros(N_T,1);
44
45 ADC_xyz_h = zeros(N_T,3);        %Apparent diffusion coefficient
46 ADC_h = zeros(N_T,1);            %
47 ADC_xyz_r = zeros(N_T,3);        %
48 ADC_r = zeros(N_T,1);            %
49 ADC = zeros(N_T,1);              %
50
51 cols_h = zeros(N_T,1);           %Number of hindered collisions
52
53 d_phase_h = zeros(N_h,N_G);
54 d_phase_r = zeros(N_r,N_G);
55
56 signal_matrix = zeros(N_T*N_G,5); %Matrix for results
57 %Adding the diffusion times in ms and the b-values in s/mm^2 for every G
58 for n_T = 1:N_T
59     signal_matrix((n_T-1)*N_G+(1:N_G), 1) = Ts(n_T);
60     bs =  (Gs*delta*gamma).^2*Ts(n_T)*10^-15;
61     signal_matrix((n_T-1)*N_G+(1:N_G),2) = bs;
62 end
63 %***********************
64
65 %*** Begin: Fibre composition ***
66 geometry = load('fibres5.mat'); %Fibre structure (1-5)
67 fibre_xpos = geometry.fibre_vals(1,:);
68 fibre_ypos = geometry.fibre_vals(2,:);
69 fibre_radii = geometry.fibre_vals(3,:);
70 N_fibres = length(fibre_radii); %Number of fibres
71
72 l_x = geometry.fibre_vals(4,1); %Half width unit cell size [μm]
73 l_y = geometry.fibre_vals(4,2); %Half length unit cell size [μm]
74 L_x = 2*l_x;                    %Whole width of unit cell [μm]
75 L_y = 2*l_y;                    %Whole length of unit cell [μm]
```

```
76 lattice = geometry.fibre_vals(4,3); %
77
78 equal_radii = true;
79 R = fibre_radii(1);              %Radius of cylindrical fibres
80 for i=2:N_fibres                     %Open up for varying fibre radii
81     if fibre_radii(i) > R             %
82         R = fibre_radii(i);      %In this case maximum radius of cyl. fibres
83         equal_radii = false;          %
84     end                               %
85 end                                  %
86
87 dts = geometry.fibre_vals(4,4)*ones(1,N_T);
88 for n_T = 1:N_T
89     if Ts(n_T)/dts(n_T) < K_min      %If T is too short for K_min steps,
90         dts(n_T) = Ts(n_T)/K_min;    % make sure the time step is shortened.
91     end
92     Ks(n_T) = floor(Ts(n_T)/dts(n_T));      %Adjusting number of time steps
93 end
94 %**** End: Fibre composition ****
95
96 if lattice == 0
97     %*** Begin: Add the fibres near cell edges on the opposite side ***
98     for i = 1:n_fibres
99         if fibre_xpos(i) < -l + fibre_radii(i)
100             n_fibres = n_fibres + 1;
101             fibre_xpos(n_fibres) = fibre_xpos(i) + L;
102             fibre_ypos(n_fibres) = fibre_ypos(i);
103             fibre_radii(n_fibres) = fibre_radii(i);
104             if fibre_ypos(i) < -l + fibre_radii(i)
105                 n_fibres = n_fibres + 1;
106                 fibre_xpos(n_fibres) = fibre_xpos(i) + L;
107                 fibre_ypos(n_fibres) = fibre_ypos(i) + L;
108                 fibre_radii(n_fibres) = fibre_radii(i);
109             elseif fibre_ypos(i) > l - fibre_radii(i)
110                 n_fibres = n_fibres + 1;
111                 fibre_xpos(n_fibres) = fibre_xpos(i) + L;
112                 fibre_ypos(n_fibres) = fibre_ypos(i) - L;
113                 fibre_radii(n_fibres) = fibre_radii(i);
114             end
```

```
115         elseif fibre_xpos(i) > l - fibre_radii(i)
116             n_fibres = n_fibres + 1;
117             fibre_xpos(n_fibres) = fibre_xpos(i) - L;
118             fibre_ypos(n_fibres) = fibre_ypos(i);
119             fibre_radii(n_fibres) = fibre_radii(i);
120             if fibre_ypos(i) < -l + fibre_radii(i)
121                 n_fibres = n_fibres + 1;
122                 fibre_xpos(n_fibres) = fibre_xpos(i) - L;
123                 fibre_ypos(n_fibres) = fibre_ypos(i) + L;
124                 fibre_radii(n_fibres) = fibre_radii(i);
125             elseif fibre_ypos(i) > l - fibre_radii(i)
126                 n_fibres = n_fibres + 1;
127                 fibre_xpos(n_fibres) = fibre_xpos(i) - L;
128                 fibre_ypos(n_fibres) = fibre_ypos(i) - L;
129                 fibre_radii(n_fibres) = fibre_radii(i);
130             end
131         end
132         if fibre_ypos(i) < -l + fibre_radii(i)
133             n_fibres = n_fibres + 1;
134             fibre_ypos(n_fibres) = fibre_ypos(i) + L;
135             fibre_xpos(n_fibres) = fibre_xpos(i);
136             fibre_radii(n_fibres) = fibre_radii(i);
137         end
138         if fibre_ypos(i) > l - fibre_radii(i)
139             n_fibres = n_fibres + 1;
140             fibre_ypos(n_fibres) = fibre_ypos(i) - L;
141             fibre_xpos(n_fibres) = fibre_xpos(i);
142             fibre_radii(n_fibres) = fibre_radii(i);
143         end
144     end
145     %**** End: Add the fibres near cell edges on the opposite side ****
146
147     %*** Begin: Sort every array to fibre_xpos' rising order ***
148     tempfib_xpos = fibre_xpos(1);
149     tempfib_ypos = fibre_ypos(1);
150     tempfib_radii = fibre_radii(1);
151
152     for fibit=2:N_fibres
153         for tempit = 1:fibit
```

```
154             if tempit == fibit
155                 tempfib_xpos = [tempfib_xpos, fibre_xpos(fibit)];
156                 tempfib_ypos = [tempfib_ypos, fibre_ypos(fibit)];
157                 tempfib_radii = [tempfib_radii, fibre_radii(fibit)];
158                 break
159             end
160             if fibre_xpos(fibit) < tempfib_xpos(tempit)
161                 tempfib_xpos = [tempfib_xpos(1:tempit - 1), ...
162                     fibre_xpos(fibit), ...
163                     tempfib_xpos(tempit:fibit - 1)];
164                 tempfib_ypos = [tempfib_ypos(1:tempit - 1), ...
165                     fibre_ypos(fibit), ...
166                     tempfib_ypos(tempit:fibit - 1)];
167                 tempfib_radii = [tempfib_radii(1:tempit - 1), ...
168                     fibre_radii(fibit), ...
169                     tempfib_radii(tempit:fibit - 1)];
170                 break
171             end
172         end
173     end
174 
175     fibre_xpos = tempfib_xpos;
176     fibre_ypos = tempfib_ypos;
177     fibre_radii = tempfib_radii;
178     %**** End: Sort every array to fibre_xpos' rising order ****
179 end
180 
181 disp('Fibrils in order')
182 
183 %****************** BEGIN: LOOP OVER ALL DIFFUSION TIMES ******************
184 
185 for n_T = 1:N_T
186 
187     T = Ts(n_T);
188     dt = dts(n_T);
189     K = Ks(n_T);
190 
191     disp(' ')
192     string = ['Diffusion time: ', num2str(T), ' ms, time ', ...
```

```
193         num2str(n_T), ' of ',num2str(N_T)];
194     disp(string)
195
196     dr = sqrt(6*D*dt);          %Stepsize [μm]
197
198
199     cols = zeros(1,N_h);        %Number of collisions
200
201     %*** Finding initial positions for all N hindered particles ***
202     x0 = zeros(1,N_h);
203     y0 = zeros(1,N_h);
204     z0 = zeros(1,N_h);
205     n = 0;
206
207     while n < N_h
208         x = L_x*rand - l_x;
209         y = L_y*rand - l_y;
210         inside_fibre = false;
211         for i=1:N_fibres
212             if equal_radii && norm([x, y] - [fibre_xpos(i), fibre_ypos(i)]) < R || ...
213                     norm([x, y] - [fibre_xpos(i), fibre_ypos(i)]) < fibre_radii(i)
214                 inside_fibre = true;
215                 break
216             end
217         end
218         if inside_fibre == false
219             n = n + 1;
220             x0(n) = x;
221             y0(n) = y;
222             if 10*n/N_h == round(10*n/N_h)
223                 string = [num2str(n), ' of ',num2str(N_h), ' hindered particles placed'
224                 disp(string)
225             end
226         end
227     end
228     %************************************************************
229
230     x_end = zeros(1,N_h);   %*****Initialising final positions and
231     y_end = zeros(1,N_h);   %***** out-of-bounds-counter for all N particles
```

```
232     z_end = zeros(1,N_h);   %
233                             %
234     x_oob = zeros(1,N_h);   %
235     y_oob = zeros(1,N_h);   %
236
237     %*** Begin: Loop over all particles ***
238
239     parfor n = 1:N_h
240
241         if 100*n/N_h == round(100*n/N_h)
242             string = ['ParticleNr: ', num2str(n), ' of ',num2str(N_h)];
243             if 10*n/N_h == round(10*n/N_h)
244                 string = ['ParticleNr: ', num2str(n), ' of ', ...
245                     num2str(N_h), ', estimated additional 10% complete'];
246             end
247             disp(string)
248         end
249
250         x_pos = zeros(1,K);    %Initializing arrays to
251         y_pos = x_pos;         %save all positions.
252         z_pos = x_pos;         %
253
254         x_pos(1) = x0(n);      %Adds the initial position.
255         y_pos(1) = y0(n);      %
256         z_pos(1) = z0(n);      %
257
258         %*** Begin: Loop over all steps ***
259
260         k = 2;
261         while k <= K+1
262
263             pos = [x_pos(k-1) y_pos(k-1) z_pos(k-1)];   %Current position.
264
265             d_cos_theta = 2*rand - 1;                   %Finds direction.
266             d_theta = acos(d_cos_theta);                %
267             d_phi = 2*pi*rand;                          %
268
269             dx = dr*cos(d_phi)*sin(d_theta);            %Computes the step.
270             dy = dr*sin(d_phi)*sin(d_theta);            % [μm]
```

```
271             dz = dr*d_cos_theta;                          %
272
273             dxy = dr*sin(d_theta);              %Step's reach in xy-plane
274
275             v = [dx dy dz]/dt;                  %Velocity [μm/ms]
276
277             ddt = dt;                           %Remaining time of time step
278
279             %Minimum distance to every outer border
280             dist = min(abs(abs(x_pos(k-1))-l_x), abs(abs(y_pos(k-1))-l_y));
281             pot_outer_trans = (dist < dxy);      %Possible outer transmission
282
283
284             %*** Begin: Loop in case of several collisions in one step ***
285
286             collide = 1; %Assuming collision
287
288             while collide == 1
289
290                 %***** Computes time until collision from current *****
291                 %***** position given the velocity v.             *****
292
293                 t_c = inf;                  %Time until collision.
294                 pos_fc = inf;               %Position of colliding fibre.
295
296                 %First calculate a t_c for the outer limit, if needed.
297                 % Actually not a collision, but a transmission.
298                 % t_c used on both for simplicity.
299                 if pot_outer_trans
300                     if v(1) > 0
301                         t_c = (l_x - pos(1))/v(1);
302                         wall_trans = 1;         %Right limit
303                     end
304                     if (l_y - pos(2))/v(2) < t_c && v(2) > 0
305                         t_c = (l_y - pos(2))/v(2);
306                         wall_trans = 2;         %Upper limit
307                     end
308                     if (l_x + pos(1))/-v(1) < t_c && -v(1) > 0
309                         t_c = (l_x + pos(1))/-v(1);
```

```
310                     wall_trans = 3;         %Left limit
311                 end
312                 if (l_y + pos(2))/-v(2) < t_c && -v(2) > 0
313                     t_c = (l_y + pos(2))/-v(2);
314                     wall_trans = 4;         %Lower limit
315                end
316             end
317
318             px = pos(1);                    %Current x position
319             py = pos(2);                    %Current y position
320             ds = norm(v(1:2))*ddt;          %Remaining step movement
321             phi = atan2(v(2),v(1));         %Current direction
322
323             %Then evaluate the potential fibre collisions
324             for i=1:N_fibres
325
326                 %Position of fibre to collide into
327                 pos_f = [fibre_xpos(i), fibre_ypos(i)];
328
329                 %Eliminates every fibre outside a smaller box,
330                 % 2*R wide and dl*ddt+R long.
331                 fx = pos_f(1);
332                 fy = pos_f(2);
333
334                 bound1 = py - cot(phi)*(fx-px);
335                 bound2 = py + sec(phi)*R + tan(phi)*(fx-px);
336                 bound3 = py + csc(phi)*(ds+R) - cot(phi)*(fx-px);
337                 bound4 = py - sec(phi)*R + tan(phi)*(fx-px);
338
339                 if v(1) < 0
340                     tempbound = bound4;
341                     bound4 = bound2;
342                     bound2 = tempbound;
343                 end
344                 if v(2) < 0
345                     tempbound = bound1;
346                     bound1 = bound3;
347                     bound3 = tempbound;
348                 end
```

```
349
350                     if fy > bound1 && fy < bound2 && fy < bound3 && fy > bound4
351
352                         %ABC formula used to find time of collision t_c
353                         A = dot(v(1:2),v(1:2));
354                         B = 2*dot(pos(1:2),v(1:2)) - 2*dot(v(1:2),pos_f);
355                         C = dot(pos(1:2),pos(1:2)) + dot(pos_f,pos_f) - ...
356                             2*dot(pos(1:2),pos_f) - R^2;
357                         SQ = sqrt(B^2 - 4*A*C);
358                         %Negative solution used in (2) since
359                         %particle is reflecting on fibre's outside
360                         new_t_c = (-B - SQ)/(2*A);
361                         %Non-real new_t_c: fibre outside trajectory
362                         %Negative new_t_c: fibre behind particle
363                         %Larger new_t_c than t_c: fibre behind another fibre
364                         %None of the above: Colliding fibre
365                         if isreal(new_t_c) && new_t_c >= 0 && new_t_c < t_c
366                             t_c = new_t_c;
367                             pos_fc = pos_f;     %Current fibre is reflecting.
368                         end
369                     end
370                 end
371                 %*****************************************************
372
373                 if t_c >= ddt                   %The step does not lead to
374                     collide = 0;                % collision or transmission
375                 end
376
377                 if collide == 1                         %Collision
378
379                     cols(n) = cols(n) + 1;
380
381                     if pos_fc ~= inf
382                         %Surface normal in collision point
383                         normal = pos + v*t_c;   %Collision point
384                         normal(3) = 0;          %No reflection in z-direction
385                         normal = normal - [pos_fc(1), pos_fc(2), 0];
386                         normal = normal/norm(normal);
387                         v_new = v - 2*dot(v,normal)*normal;
```

```
388                     pos = pos + v*t_c;  %Updates position.
389                     v = v_new;          %Updates velocity.
390                 else
391                     %Updates position including unit cell movement
392                     % and saves it in the out-of-bounds-vector
393                     if wall_trans == 1
394                         pos = pos + v*t_c - [L_x,0,0];
395                         x_oob(n) = x_oob(n) + 1;
396                     elseif wall_trans == 2
397                         pos = pos + v*t_c - [0,L_y,0];
398                         y_oob(n) = y_oob(n) + 1;
399                     elseif wall_trans == 3
400                         pos = pos + v*t_c + [L_x,0,0];
401                         x_oob(n) = x_oob(n) - 1;
402                     elseif wall_trans == 4
403                         pos = pos + v*t_c + [0,L_y,0];
404                         y_oob(n) = y_oob(n) - 1;
405                     end
406                 end
407                 ddt = ddt-t_c;      %Updates ddt in case of multiple
408                 % collisons in one time step.
409
410             else                    %Not collision; assumption wrong.
411
412                 x_pos(k) = pos(1) + v(1)*ddt; %Finds new positions.
413                 y_pos(k) = pos(2) + v(2)*ddt; %
414                 z_pos(k) = pos(3) + v(3)*ddt; %
415
416             end
417         end
418
419         %**** End: Loop in case of several collisions in one step ****
420         k = k + 1;
421     end
422
423     %**** End: Loop over all steps ****
424
425     x_end(n) = x_pos(k-1);
426     y_end(n) = y_pos(k-1);
```

```
427         z_end(n) = z_pos(k-1);
428
429         %Phase displacement, 10^-9 is to make up for prefixes
430         for n_G = 1:N_G
431             d_phase_h(n,n_G) = ...
432                 Gs(n_G)*delta*gamma*(x_pos(1) - x_pos(k-1))*10^-9;
433         end
434
435     end
436
437     %**** End: Loop over all particles ****
438
439     %Adding the signal for every G
440     for n_G = 1:N_G
441         signal_matrix((n_T-1)*N_G + n_G,3) = sum(exp(1i*d_phase_h(:,n_G)))/N_h;
442     end
443
444     %Reads the subtracted movement out of bounds.
445     x_end = x_end - x0 + L_x*x_oob;
446     y_end = y_end - y0 + L_y*y_oob;
447     z_end = z_end - z0;
448
449     r_end = sqrt(y_end.^2 + y_end.^2 + z_end.^2);
450
451     avg_xyz_h(n_T,:) = [sum(x_end),sum(y_end),sum(z_end)]/N_h;
452     avg_h(n_T) = sum(avg_xyz_h(n_T,:))/3;
453
454     EDC_xyz_h(n_T,:) = [sum(x_end.*x_end), ...
455                         sum(y_end.*y_end),sum(z_end.*z_end)]/(2*N_h*T);
456     EDC_h(n_T,:) = sum(EDC_xyz_h(n_T,:))/3;
457
458     cols_h(n_T) = sum(cols)/N_h;
459
460     %************* END: HINDERED DIFFUSION *************
461     %************ BEGIN: RESTRICTED DIFFUSION **********
462
463     if sqrt(2*T) < R_r
464
465         string = ['T = ', num2str(T), ...
```

```
466             ' ms too small for representable motion averaging'];
467         disp(string)
468         disp('Calculating stepwise restricted diffusion')
469 
470         x0_r = zeros(1,N_r);
471         y0_r = zeros(1,N_r);
472         z0_r = zeros(1,N_r);
473 
474         x_end_r = zeros(1,N_r);
475         y_end_r = zeros(1,N_r);
476         z_end_r = zeros(1,N_r);
477 
478         n_r = 1;
479         while n_r <= N_r
480             x0_r(n_r) = 2*R_r*(1-2*rand());
481             y0_r(n_r) = 2*R_r*(1-2*rand());
482             z0_r(n_r) = 2*R_r*(1-2*rand());
483             if sqrt(x0_r(n_r)^2 + y0_r(n_r)^2 + z0_r(n_r)^2) < R_r
484                 n_r = n_r + 1;
485             end
486         end
487 
488         parfor n_r = 1:N_r
489 
490             if 100*n_r/N_r == round(100*n_r/N_r)
491                 string = ['ParticleNr : ',num2str(n_r),' of ',num2str(N_r)];
492                 if 10*n_r/N_r == round(10*n_r/N_r)
493                     string = ['ParticleNr : ',num2str(n_r),' of ',num2str(...
494                         N_r),', estimated additional 10% complete'];
495                 end
496                 disp(string)
497             end
498 
499             x_pos_r = zeros(1,K_r);     %Initializing arrays to
500             y_pos_r = x_pos_r;        %save all positions.
501             z_pos_r = x_pos_r;         %
502 
503 
504             x_pos_r(1) = x0_r(n_r);      %Adds the initial position.
```

```
505             y_pos_r(1) = y0_r(n_r);      %
506             z_pos_r(1) = z0_r(n_r);      %
507
508             k = 2;
509             while k <= K_r+1
510
511                 pos = [x_pos_r(k-1) y_pos_r(k-1) z_pos_r(k-1)]; %Current position.
512
513                 d_cos_theta = 2*rand - 1;               %Finds direction.
514                 d_theta = acos(d_cos_theta);           %
515                 d_phi = 2*pi*rand;                    %
516
517                 dt = T/K_r;             %Restricted time step [ms]
518                 dr = sqrt(6*D*dt);      %Restricted step size [μm]
519
520                 dx = dr*cos(d_phi)*sin(d_theta);     %Computes the step.
521                 dy = dr*sin(d_phi)*sin(d_theta);       % [μm]
522                 dz = dr*d_cos_theta;                  %
523
524                 dxy = dr*sin(d_theta);    %Step's reach in xy-plane
525
526                 v = [dx dy dz]/dt;       %Velocity [μm/ms]
527
528                 ddt = dt;                 %Remaining time of time step
529
530                 %*** Begin: Loop in case of multiple collisions in time step. ***
531
532                 collide = 1; %Assuming collision
533
534                 while collide == 1
535
536                     %***** Computes time until collision from current *****
537                     %***** position given the velocity v.             *****
538                     px = pos(1);              %Current x position
539                     py = pos(2);              %Current y position
540                     dl = norm(v(1:2))*ddt;    %Remaining step movement
541                     phi = atan2(v(2),v(1));     %Current direction
542
543                     %ABC formula used to find time of collision t_c
```

```
544                     A = dot(v,v);
545                     B = 2*dot(pos,v);
546                     C = dot(pos,pos)- R_r^2;
547                     SQ = sqrt(B^2 - 4*A*C);
548                     t_c = (-B + SQ)/(2*A);
549                     %*****************************************************
550
551                     if t_c >= ddt                   %The step does not lead to
552                         collide = 0;                % collision
553                     end
554
555                     if collide == 1                 %Collision
556
557                         %Surface normal in collision point
558                         normal = pos + v*t_c;       %Collision point
559                         normal = normal/norm(normal);
560                         v_new = v - 2*dot(v,normal)*normal;
561                         pos = pos + v*t_c;          %Updates position.
562                         v = v_new;                  %Updates velocity.
563                         ddt = ddt-t_c;      %Updates remaining time step.
564
565                     else                %Not collision; assumption wrong.
566
567                         x_pos_r(k) = pos(1) + v(1)*ddt; %Finds new positions.
568                         y_pos_r(k) = pos(2) + v(2)*ddt; %
569                         z_pos_r(k) = pos(3) + v(3)*ddt; %
570
571                     end
572                 end
573
574                 %**** End: Loop in case of multiple collisions in time step. ****
575                 k = k + 1;
576             end
577
578             x_end_r(n_r) = x_pos_r(k-1);
579             y_end_r(n_r) = y_pos_r(k-1);
580             z_end_r(n_r) = z_pos_r(k-1);
581
582             %Phase displacement, 10^-9 is to make up for prefixes
```

```
583             for n_G = 1:N_G
584                 d_phase_r(n_r,n_G) = Gs(n_G)*delta*gamma* ...
585                                     (x_pos_r(1) - x_pos_r(k-1))*10^-9;
586             end
587 
588         end
589 
590         x_r = x_end_r - x0_r;
591         y_r = y_end_r - y0_r;
592         z_r = z_end_r - z0_r;
593 
594         avg_xyz_r(n_T,:) = [sum(x_r),sum(y_r),sum(z_r)]/N_r;
595         avg_r(n_T) = sum(avg_xyz_r(n_T,:))/3;
596 
597         EDC_xyz_r(n_T,:) = [sum(x_r.*x_r), ...
598                             sum(y_r.*y_r),sum(z_r.*z_r)]/(2*N_r*T);
599         EDC_r(n_T) = sum(EDC_xyz_r(n_T,:))/3;
600 
601         %Adding the signal for every G
602         for n_G = 1:N_G
603             signal_matrix((n_T-1)*N_G + n_G, 4) = ...
604                 sum(exp(1i*d_phase_r(:,n_G)))/N_r;
605         end
606 
607     else
608 
609         disp('Starting motion averaging')
610 
611         x0_r = zeros(1,N_r);
612         y0_r = zeros(1,N_r);
613         z0_r = zeros(1,N_r);
614 
615         x_end_r = zeros(1,N_r);
616         y_end_r = zeros(1,N_r);
617         z_end_r = zeros(1,N_r);
618 
619         n_ma = 1;
620         while n_ma <= N_r
621             x0_r(n_ma) = 2*R_r*(1-2*rand());
```

```
622             y0_r(n_ma) = 2*R_r*(1-2*rand());
623             z0_r(n_ma) = 2*R_r*(1-2*rand());
624             if sqrt(x0_r(n_ma)^2+y0_r(n_ma)^2+z0_r(n_ma)^2) < R_r
625                 n_ma = n_ma + 1;
626             end
627         end
628
629         n_ma = 1;
630         while n_ma <= N_r
631             x_end_r(n_ma) = 2*R_r*(1-2*rand());
632             y_end_r(n_ma) = 2*R_r*(1-2*rand());
633             z_end_r(n_ma) = 2*R_r*(1-2*rand());
634             if sqrt(x_end_r(n_ma)^2+y_end_r(n_ma)^2+z_end_r(n_ma)^2) < R_r
635                 n_ma = n_ma + 1;
636             end
637         end
638
639         x_r = x_end_r - x0_r;
640         y_r = y_end_r - y0_r;
641         z_r = z_end_r - y0_r;
642
643         avg_xyz_r(n_T,:) = [sum(x_r),sum(y_r),sum(z_r)]/N_r;
644         avg_r(n_T) = sum(avg_xyz_r(n_T,:))/3;
645
646         EDC_xyz_r(n_T,:) = ...
647             [sum(x_r.*x_r),sum(y_r.*y_r),sum(z_r.*z_r)]/(2*N_r*T);
648         EDC_r(n_T) = sum(EDC_xyz_r(n_T,:))/3;
649
650         disp('Motion averaging complete')
651
652         %Adding the signal for every G, 10^-9 is to make up for prefixes
653         for n_G = 1:N_G
654             d_phase_r(:,n_G) = Gs(n_G)*delta*gamma*x_r*10^-9;
655             signal_matrix((n_T-1)*N_G+n_G,4) = ...
656                 sum(exp(1i*d_phase_r(:,n_G)))/N_r;
657         end
658
659     end
660
```

```
661     %**** End: Motion averaging ****
662
663 end
664 %****************** END: LOOP OVER ALL DIFFUSION TIMES ******************
665
666 signal_matrix(:,5) = signal_matrix(:,3)*f+signal_matrix(:,4)*(1-f);
667 signal_matrix = abs(signal_matrix);
668
669 save 'pulsed_f08.mat' signal_matrix
670
671 toc
```

A.8 ConstantGradient.m

Like in PulsedGradient.m water particles self-diffuses with both hindrance and restriction for a number of particles, diffusion times and magnetic gradient strengths. The difference lies in the calculation of the signal. Where the last script only took into account the first and last position of every particle, ConstantGradient.m calculates the signal from every time step a gradient is applied. Therefore motion averaging is not performable.

Only the last lines of the code is included, showing the calculation of the signal for restricted diffusion. This is also done for the hindered case, but not shown here.

```
[...]

T = Ts(n_T);            % Total diffusion time
T_180 = 2 ms;           % Time between gradients
delta = T - T_180/2;    % Gradient duration

[...]

%Adding the signal for every G, 10^-9 is to make up for prefixes
        for n_G = 1:N_G
            d_phase_r(:,n_G) = Gs(n_G)*delta*gamma*mean( ...
            x_pos(1:(T-T_180)*K/(2*T)) - ...
            x-pos((T+T_180)*K/(2*T):K))*10^-9;
            signal_matrix((n_T-1)*N_G+n_G,4) = ...
                sum(exp(1i*d_phase_r(:,n_G)))/N_r;
        end
    end
end
%****************** END: LOOP OVER ALL DIFFUSION TIMES ******************

signal_matrix(:,5) = signal_matrix(:,3)*f+signal_matrix(:,4)*(1-f);
signal_matrix = abs(signal_matrix);

save 'pulsed_f08.mat' signal_matrix

toc
```

A.9 DiffusionTest.m

This script was written before the signal generating scripts to ensure that assumptions made, such as constant step sizes longer than the Brownian motion, didn't affect the results for any of the three types of diffusion.

```
D = 3
Specifying parameters: Ts, dt, N
N_T = length(Ts)
types of diffusion: 3
types of step length distribution: 2
number of dimensions: 3
Initialising output arrays of length N_T*3*2*3: avg, EDC

Specifying geometry
Copying cell wall intersecting fibres

for n_T=1:N_T
    T = Ts(n_T);
    K = T/dt;
    dr = sqrt(6*D*dt);

    for freedom_type=1:3
        for step_type=1:2 1: c, 2: nd
            placing particles in environment
                (freedom_type
                 1: wherever
                 2: outside fibres
                 3: inside fibres)
            parfor n=1:N
                if freedom_type == 3
                    collect one fibre and its copies
                end
                for k=1:K
                    if step_type == 2
                        dr = nrmrnd(dr,dr*0.3);
                    end
                    perform diffusion as otherwise presented
                        (freedom_type
```

```
                        1: no fibres collected
                        2: fibres collected, outside reflection
                        3: already collected)
                end
            end
            save the final position of all particles in x, y and z
            avg((n_T-1)*18+(freedom_type-1)*6+(step_type-1)*3+(1:3)) ...
                = [sum(x), sum(y), sum(z)]/N;
            EDC((n_T-1)*18+(freedom_type-1)*6+(step_type-1)*3+(1:3)) ...
                = [sum(x.^2), sum(y.^2), sum(z.^2)]/(N^2*2*T);
        end
    end
end

save avg and EDC
```

A.10 WalkInFibre.m

The following script `WalkInFibre.m` was the most executed one during the work on the project thesis on beforehand, as well as the one into which the most time had then gone. The name lingers from when it originally only handled fibrils in a fibre. This script was the starting point from which many of the other scripts were developed. Consistently ADC terminology were used as EDC is a newer expression.

```
%*************************************************************************
%*** Simulates particles movement in fibre with random walks.           ***
%*** Returns the final position of all particles.                       ***
%*************************************************************************
%*** It takes about a second a particle with diffusion time 100 ms,     ***
%*** a little over 10000 particles in three hours.                      ***
%*************************************************************************
%*** NOTE: 'Colliding fibril' terminology used even though the          ***
%*** particles are colliding into fibrils, not the other way around.    ***
%*************************************************************************

tic

%********** Parameters **********
N = 1000;                           %Total number of particles []
H = 100;                            %Height cylinder [μm]
L = 2;                              %Half length unit cell size [μm]
D = 1.65;                           %Diffusion coefficient [(μm)^2 /ms]
diff_times = 100;                   %Diffusion times [ms]
dt = 0.001;                         %Time step [ms]
%*******************************

dr = sqrt(6*D*dt);                  %Step size [μm]
s = dr/dt;                          %Movement speed [μm/s]

%********** Fiber composition ***
imp_vals = load ('fibrils.mat');
fibril_xpos = imp_vals.fibril_vals(1,:);
fibril_ypos = imp_vals.fibril_vals(2,:);
fibril_radii = imp_vals.fibril_vals(3,:);
```

```
31 n_fibrils = length(fibril_radii);
32 equal_radii = true;
33 radmax = fibril_radii(1);              %May save time to use maximum
34 for i=2:n_fibrils                      % fibril radius when anticipating
35     if fibril_radii(i) > radmax        % collision for relatively small
36         radmax = fibril_radii(i);      % differences in fibril radii.
37     equal_radii = false;
38     end
39 end
40
41 %***** Sorting every array to fibril_xpos' rising order ****
42
43 tempfib_xpos = fibril_xpos(1);
44 tempfib_ypos = fibril_ypos(1);
45 tempfib_radii = fibril_radii(1);
46
47 for fibit=2:n_fibrils
48     for tempit = 1:fibit
49         if tempit == fibit
50             tempfib_xpos = [tempfib_xpos, fibril_xpos(fibit)];
51             tempfib_ypos = [tempfib_ypos, fibril_ypos(fibit)];
52             tempfib_radii = [tempfib_radii, fibril_radii(fibit)];
53             break
54         end
55         if fibril_xpos(fibit) < tempfib_xpos(tempit)
56             tempfib_xpos = [tempfib_xpos(1:tempit - 1), ...
57                             fibril_xpos(fibit), ...
58                             tempfib_xpos(tempit:fibit - 1)];
59             tempfib_ypos = [tempfib_ypos(1:tempit - 1), ...
60                             fibril_ypos(fibit), ...
61                             tempfib_ypos(tempit:fibit - 1)];
62             tempfib_radii = [tempfib_radii(1:tempit - 1), ...
63                             fibril_radii(fibit), ...
64                             tempfib_radii(tempit:fibit - 1)];
65             break
66         end
67     end
68 end
69
```

```
70  fibril_xpos = tempfib_xpos;
71  fibril_ypos = tempfib_ypos;
72  fibril_radii = tempfib_radii;
73
74  %**************************
75
76  %*******************************
77
78  N_time = length(diff_times);        %Number of different diffusion times.
79
80  limit = round(diff_times/dt);            %Finds the step numbers associated
81  for i = 1:N_time                         %with the different diffusion
82      if limit(i)/2 ~= round(limit(i)/2)   %times, and forces is to be even.
83          limit(i) = limit(i) + 1;          %
84      end
85  end
86
87  K = max(limit);                       %Total number of steps.
88
89  %***** Finding initial positions for all N particles *****
90  x0 = zeros(1,N);
91  y0 = zeros(1,N);
92  n = 0;
93  while n < N
94      x = 2*L*rand - L;
95      y = 2*L*rand - L;
96      inside_fibril = false;
97      for i=1:n_fibrils
98          if (equal_radii && norm([x, y] - [fibril_xpos(i), fibril_ypos(i)])) < radmax ||
99              norm([x, y] - [fibril_xpos(i), fibril_ypos(i)]) < fibril_radii(i)
100             inside_fibril = true;
101             break
102         end
103     end
104     if inside_fibril == false
105         n = n + 1;
106         x0(n) = x;
107         y0(n) = y;
108     end
```

```
109 end
110 z0 = H*rand(1,N);
111 %*********************************************************
112
113 %*****Initializing final positions and                 ***
114 %***** out-of-bounds-counter for all N particles       ***
115 x_end = zeros(1,N);
116 y_end = zeros(1,N);
117 z_end = zeros(1,N);
118
119 x_oob = zeros(1,N);
120 y_oob = zeros(1,N);
121 %*********************************************************
122
123 %***** Writes out total number of steps *****
124 string = ['Number of steps : ', num2str(K)];
125 disp(string)
126 %*****************************************
127
128 %(I)****************** Loop over all the particles **********************
129 parfor n = 1:N
130
131     if n/100 == round(n/100)
132         string = ['ParticleNr : ', num2str(n), ' of ',num2str(N)];
133         disp(string)
134     end
135
136     x_pos = zeros(1,K);    %Initilizing arrays to
137     y_pos = x_pos;         %save all positions.
138     z_pos = x_pos;         %
139
140
141     x_pos(1) = x0(n);      %Adds the initial position.
142     y_pos(1) = y0(n);      %
143     z_pos(1) = z0(n);      %
144
145     %(II)*********** Finds position 2 to K ************
146     k = 2;
147     while k <= K
```

```
148
149         if k/10000 == round(k/10000)
150             string = ['MovementNr : ', num2str(k), ' of ',num2str(K)];
151             disp(string)
152         end
153
154         %Minimum distance to every outer border
155         dist = min(abs(abs(x_pos(k-1))-L), abs(abs(y_pos(k-1))-L));
156         pot_outer_trans = (dist < dr);      %Possible outer transmission
157
158         pos = [x_pos(k-1) y_pos(k-1) z_pos(k-1)];   %Current position.
159
160         d_cos_theta = 2*rand - 1;                   %Finds direction.
161         d_theta = acos(d_cos_theta);                %
162         d_phi = 2*pi*rand;                          %
163
164         dx = dr*cos(d_phi)*sin(d_theta);            %Computes the step.
165         dy = dr*sin(d_phi)*sin(d_theta);            %
166         dz = dr*d_cos_theta;                        %
167
168         dxy = dr*sin(d_theta);              %Step's reach in xy-plane
169
170         v = [dx dy dz]/dt;                  %Velocity
171
172         ddt = dt;                           %Remaining time of time step
173
174         %********** Finds all fibrils in a      **********
175         %********** dxy+radmax-sized box        **********
176         %********** around the particle.        **********
177
178         pot_fib_col = zeros(1,10);  %Array of potentially colliding fibrils
179         n_pot_fib_col = 0;          %Number of fibrils found in box
180
181         %Finds the first fibril whose x position fulfills box constraint
182         startit = ceil((pos(1)+L)/(2*L)*n_fibrils); %Start iterator for x
183         while true
184             if fibril_xpos(startit) > pos(1)-radmax-dxy && startit ~= 1
185                 startit = startit - 1;
186             elseif startit ~= n_fibrils && ...
```

```
187                 fibril_xpos(startit + 1) < pos(1)-radmax-dxy
188                 startit = startit + 1;
189             else
190                 break
191             end
192         end
193
194         for i=startit:n_fibrils
195             %If an x position is outside, the remaining fibrils are outside
196             if fibril_xpos(i) > pos(1)+radmax+dxy
197                 break
198             end
199             %Checking y positions
200             if fibril_ypos(i) > pos(2)-radmax-dxy && ...
201                fibril_ypos(i) < pos(2)+radmax+dxy
202                 n_pot_fib_col = n_pot_fib_col + 1;
203                 pot_fib_col(n_pot_fib_col) = i;
204             end
205         end
206
207         %************************************************
208
209         %(III)***** Loop in case of multiple collision **********
210         %********** in same time step.                  **********
211         collide = 1;                              %Assuming collision
212         while collide == 1
213
214             %***** Computes time until collision from current *****
215             %***** position given the velocity v.             *****
216
217             t_c = inf;                  %Time until collision.
218             pos_fc = 0;                 %Position of colliding fibril.
219
220             %First calculate a t_c for the outer limit, if needed.
221             % Actually not a collision, but a transmission.
222             % t_c used on both for simplicity.
223             if pot_outer_trans
224                 if v(1) > 0
225                     t_c = (L - pos(1))/v(1);
```

```
226                     wall_trans = 1;         %Right limit
227                 end
228                 if (L - pos(2))/v(2) < t_c && v(2) > 0
229                     t_c = (L - pos(2))/v(2);
230                     wall_trans = 2;         %Upper limit
231                 end
232                 if (L + pos(1))/-v(1) < t_c && -v(1) > 0
233                     t_c = (L + pos(1))/-v(1);
234                     wall_trans = 3;         %Left limit
235                 end
236                 if (L + pos(2))/-v(2) < t_c && -v(2) > 0
237                     t_c = (L + pos(2))/-v(2);
238                     wall_trans = 4;         %Lower limit
239                 end
240             end
241
242             px = pos(1);                    %Current x position
243             py = pos(2);                    %Current y position
244             dl = norm(v(1:2))*ddt;          %Remaining step movement
245             phi = atan2(v(2),v(1));         %Current direction
246
247             %Then evaluate the potential fibril collisions
248             for i=1:n_pot_fib_col
249
250                 %Radius of fibril to collide into
251                 if equal_radii
252                     col_rad = radmax;
253                 else
254                     col_rad = fibril_radii(pot_fib_col(i));
255                 end
256
257                 %Position of fibril to collide into
258                 pos_f = [fibril_xpos(pot_fib_col(i)), ...
259                         fibril_ypos(pot_fib_col(i))];
260
261                 %Eliminates every fibril outside a smaller box,
262                 % 2*radmax wide and dl*ddt+radmax long.
263                 fx = pos_f(1);
264                 fy = pos_f(2);
```

```
265
266             bound1 = py - cot(phi)*(fx-px);
267             bound2 = py + sec(phi)*radmax + tan(phi)*(fx-px);
268             bound3 = py + csc(phi)*(dl+radmax) - cot(phi)*(fx-px);
269             bound4 = py - sec(phi)*radmax + tan(phi)*(fx-px);
270
271             if v(1) < 0
272                 tempbound = bound4;
273                 bound4 = bound2;
274                 bound2 = tempbound;
275             end
276             if v(2) < 0
277                 tempbound = bound1;
278                 bound1 = bound3;
279                 bound3 = tempbound;
280             end
281
282             if fy > bound1 && fy < bound2 && fy < bound3 && fy > bound4
283
284                 %ABC formula used to find time of collision t_c
285                 A = dot(v(1:2),v(1:2));
286                 B = 2*dot(pos(1:2),v(1:2)) - 2*dot(v(1:2),pos_f);
287                 C = dot(pos(1:2),pos(1:2)) + dot(pos_f,pos_f) - ...
288                     2*dot(pos(1:2),pos_f) - col_rad^2;
289                 SQ = sqrt(B^2 - 4*A*C);
290                 %Negative solution used since particle is reflecting
291                 % on fibril's outside
292                 new_t_c = (-B - SQ)/(2*A);
293                 %Non-real new_t_c: fibril outside trajectory
294                 %Negative new_t_c: fibril behind particle
295                 %Larger new_t_c than t_c: fibril behind another fibril
296                 %None of the above: Colliding fibril
297                 if isreal(new_t_c) && new_t_c >= 0 && new_t_c < t_c
298                     t_c = new_t_c;
299                     pos_fc = pos_f;     %Current fibril is reflecting.
300                 end
301             end
302         end
303         %*****************************************************
```

```
304
305             if t_c >= ddt                   %The step does not lead to
306                 collide = 0;                % collision or transmission
307             end
308
309             if collide == 1                         %Collision
310
311                 if pos_fc ~= 0
312                     %Surface normal in collision point
313                     normal = pos + v*t_c;   %Collision point
314                     normal(3) = 0;          %No reflection in z-direction
315                     normal = normal - [pos_fc(1), pos_fc(2), 0];
316                     normal = normal/norm(normal);
317                     v_new = v - 2*dot(v,normal)*normal;
318                     pos = pos + v*t_c;  %Updates position.
319                     v = v_new;          %Updates velocity.
320                 else
321                     %Updates position including unit cell movement
322                     % and saves it in the out-of-bounds-vector
323                     if wall_trans == 1
324                         pos = pos + v*t_c - [2*L,0,0];
325                         x_oob(n) = x_oob(n) + 1;
326                     elseif wall_trans == 2
327                         pos = pos + v*t_c - [0,2*L,0];
328                         y_oob(n) = y_oob(n) + 1;
329                     elseif wall_trans == 3
330                         pos = pos + v*t_c + [2*L,0,0];
331                         x_oob(n) = x_oob(n) - 1;
332                     elseif wall_trans == 4
333                         pos = pos + v*t_c + [0,2*L,0];
334                         y_oob(n) = y_oob(n) - 1;
335                     end
336                 end
337                 ddt = ddt-t_c;      %Updates ddt in case of multiple
338                                     % collisons in one time step.
339
340             else                    %Not collision; assumption wrong.
341
342                 x_pos(k) = pos(1) + v(1)*ddt; %Finds new positions.
```

```
343                 y_pos(k) = pos(2) + v(2)*ddt; %
344                 z_pos(k) = pos(3) + v(3)*ddt; %
345 
346             end
347         end
348         %(III)*************************************************
349         k = k + 1;
350     end
351     %(II)*********************************************
352 
353     x_end(n) = pos(1);
354     y_end(n) = pos(2);
355     z_end(n) = pos(3);
356 
357 end
358 %(I)*************************************************************************
359 
360 %Reads the subtracted movement out of bounds.
361 x_end = x_end - x0 + 2*L*x_oob;
362 y_end = y_end - y0 + 2*L*y_oob;
363 z_end = z_end - z0;
364 
365 r_end = sqrt(y_end.^2 + y_end.^2 + z_end.^2);
366 
367 save 'end_pos1000dt0001.mat' x_end y_end z_end r_end
368 
369 toc
```

Bibliography

[1] J. Lilley. Nuclear physics, principles and applications. 2001.

[2] A. N. Johansen. Monte Carlo simulations of diffusion MRI in restricted geometries. March 2017.

[3] J. O. Andersen. Introduction to statistical mechanics. pages 6–12, August 2011.

[4] K. M. Bundell S. J. Bundell. Concepts in thermal physics. second edition. *Oxford University Press*, 2010.

[5] A. Sparr G. Sparr. Kontinuerliga system. *Studentlitteratur AB*, 1999, 2000.

[6] T. S. Ursell. The diffusion equation a multi-dimensional tutorial. October 2007.

[7] T. S. Urseel. The diusion equation a multi-dimensional tutorial. *California Institute of Technology*, 2007.

[8] P. N. Sen. Time-dependent diffusion coefficient as a probe of geometry. Concepts in magnetic resonance Part A. 2004.

[9] T. E. J. Behrens P. J. Basser, E. Özarslan; red. H: Johansen-Berg. Introduction to diusion mr, chapter 1. 2009.

[10] J. P. Hornak. The basics of mri. *Rochester Institute of Technology.*

[11] N. P. Jerome T. E. Sjøbakk A. Østlie H. E. Fjøsne R. Karunamuni N. S. White R. Rakow-Penner A. M. Dale T. F. Bathen P. E. Goa I. Vidic, L. Egnell. Non-gaussian dwi of breast lesions at high b-value. 2017.

[12] M. J. Engstrøm O. A. Haugen L. A. Dyrnes B. O. Åsvold M. B. Lilledahl A. M. Bofin A. Brabrand, I. I. Kariuki. Alterations in collagen fibre patterns in breast cancer. A premise for tumour invasiveness? *APMIS*, 2015.

[13] D. M. Hoang Y. Z. Wadghiri D. S. Novikov S. G. Kim O. Reynaud, K. V. Winters. Surface-to-volume ratio mapping of tumor microstructure using oscillating gradient diffusion weighted imaging. *Magn Reson Med*, 2016.

[14] Z. Koza M. Matyka, A. Khalili. Tortuosity-porosity relation in porous media ow. *Physical Review*, 2008.

[15] P. N. Sen. Time-dependent diffusion coefcient as a probe of geometry. *Concepts Magn Reson*, 2004.

[16] P. N. Sen T. N. de Swiet. Time dependent diffusion coefficient in a disordered medium. *The Journal of Chemical Physics*, 1996.

[17] B. L. Bakken. Monte carlo simulations of diffusion weighted mri in restricted geometries. 2017.

[18] S. Kan H. Hata M. Ozaki K. Wabuchi M. Kuranami M. Watanabe K. Hayakawa R. Woodhams, K. Matsunaga. Adc mapping of benign and malignant breast tumors. *Magnetic Resonance in Medical Sciences*, 2005.