Deep-learning-accelerated Bayesian inference for FRAP experiments

Master's thesis in Engineering mathematics and computational science

Harald Westling

DEPARTMENT OF MATHEMATICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Deep-learning-accelerated Bayesian inference for FRAP experiments

Harald Westling

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2021

Deep-learning-accelerated Bayesian inference for FRAP experiments Harald Westling

Supervisor: Magnus Röding, Research Institute of Sweden, RISE
Examiner: Aila Särkkä, Department of Mathematics

Master's Thesis 2021
Department of Mathematics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Deep-learning-accelerated Bayesian inference for FRAP experiments
Harald Westling
Department of Mathematics
Chalmers University of Technology

Abstract

Fluorescence recovery after photobleaching (FRAP) is an experimental method for determining properties such as the diffusion coefficient and binding rate of molecules in solutions, and is used extensively in areas such as food science and biology. By utilizing a high intensity laser the fluorescent molecules in a region of interest are bleached. The mean fluorescence intensity in the region, and its development over time, can be modeled as a function of the mobility parameters. Bayesian inference with regards to the model parameters, using the likelihood function, can then be performed. This likelihood function is very computationally heavy to evaluate. In this work a neural network has been implemented to approximate the likelihood function with regards to the four most central parameters in the model. The method is promising since three of the four marginal posterior distributions of the parameters were well approximated, with the result being comparable to traditional methods for posterior sampling. We demonstrated that the method is much more computationally efficient than traditional methods.

Keywords: Fluorescence recovery after photobleaching, Diffusion, Neural network, Bayesian inference, MCMC, Metropolis-Hastings.

Acknowledgements

First of all I would like to thank my supervisor Magnus Röding at RISE Agrifood Bioscience who gave me a great deal of support and advice during the course of the project.
I would also like to thank my examiner Aila Särkkä from Chalmers University of Technology, for her support and guidance during the thesis work.

Harald Westling, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

Correctly understanding the microscopic properties of materials and products is something of high importance in both industry and science, with new discoveries potentially leading to improvements in areas such as medicine and soft matter research. By studying the manner of particle movement, when dissolved in a solvent, new insights could be obtained in these areas. One experimental method for observing the properties related to particle movement in a solvent is fluorescence recovery after photobleaching (FRAP). When utilizing the FRAP method fluorescent particles in a region of interest are bleached using a laser. The change of fluorescence concentration over time, in the region, can then be used to estimate the different properties. One of these properties is the diffusion coefficient which is the rate of random particle movement in a solvent.

When performing experiments, Bayesian posterior inference is often used in analyzing the results, and assessing uncertainties in the model. By first expressing ones prior belief of the distribution of an unknown parameter, $\boldsymbol{\theta}$, one can use the likelihood function $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})$, which encodes the probability of the parameters $\boldsymbol{\theta}$ generating the data $\boldsymbol{y}$, in order to update the prior knowledge. By the use of Bayes' theorem the updated prior then becomes the posterior distribution of the unknown parameters, conditioned on the observed data and the likelihood of observing the data. The posterior distribution is often the distribution of interest, but it can be difficult to obtain. It could require a normalizing constant, that is unknown, or computationally hard to approximate. One solution is to use Markov chain Monte Carlo sampling methods to approximate the distribution by iteratively constructing a Markov chain, which converges towards the target posterior distribution. One such method is the Metropolis-Hastings algorithm. This algorithm needs to compute the likelihood function at each step, and if the likelihood is complex then the computation might be challenging. One proposed solution to this is to approximate the likelihood function using a neural network, which can approximate the likelihood and requires much less computational power.

In this report the aim is to approximate the likelihood function of the data originated from the FRAP method. The data resulting from a FRAP experiment is a sequence of images which is typically reduced to a curve, known as the recovery curve, expressing the mean intensity of the bleached region. From the recovery curve a likelihood function can be calculated. The implementation of this model is computationally heavy. In order to alleviate this problem a new algorithm for doing Bayesian inference for this model is proposed, based on utilizing deep learning and

training an artificial neural network to approximate the likelihood function. In this report a fully connected feedforward neural network was used.

This thesis aims to explore if deep learning can be used to accelerate the Metropolis-Hastings algorithm without sacrificing the performance, with regards to accuracy and efficiency. The project is limited in scope with regards to the parameters of the model and network architecture, as well as time constraints.

The outline of the report is as follows. Chapter 2 contains the background covering the basics of the FRAP method, the tool for numerical solutions for the diffusion equation, and simulating the FRAP data, the basics of Bayesian statistics and inference and a brief introduction to neural networks. Chapter 3 contains the training setup, the pre-processing done on the data, a description of the training of the neural network as well as a description of the implemented accelerated Metropolis-Hastings algorithm. This chapter also presents and discusses the results from the accelerated Metropolis-Hastings algorithm. In Chapter 4, some conclusions are drawn from the results and some further work on the topic is suggested.

2
Background

2.1 FRAP

In this section a basic theoretical framework of diffusion and the fluorescence recovery after photobleaching method is introduced.

2.1.1 Diffusion

Diffusion is a term for the random movement of atoms, or molecules driven by fluctuations in the thermal energy. This process can be described, on a macroscopic level, with Ficks first law

$$J(\boldsymbol{x}, t) = -D\nabla c(\boldsymbol{x}, t) \tag{2.1}$$

where $D > 0$ is the *diffusion coefficient* and $c(\boldsymbol{x}, t)$ is the concentration at position $\boldsymbol{x}$ and time t. The flux, J, is proportional to the negative concentration gradient. Ficks second law, also known as the diffusion equation, is expressed as

$$\frac{\partial c}{\partial t} = D\nabla^2 c(\boldsymbol{x}, t), \tag{2.2}$$

which describes how the concentration of particles is affected by diffusion [6].

2.1.2 Fluorescence recovery after photobleaching

Fluorescence recovery after photobleaching (FRAP) is a method used for estimating the diffusion coefficient of molecules dissolved in a liquid. By using an intense laser light on a *region of interest* (ROI) the fluorescent molecules, called *fluorophores*, inside the area are bleached, meaning that they are unable to fluoresce.

The photobleaching results in a bleached region which the unbleached fluorophores from the surrounding areas will diffuse into, while the bleached fluorophores will move out of, the area. The result of this process is that the bleached area both broadens and fades at the same time, in a manner that depends on the diffusion coefficient.

When performing FRAP it is typically assumed that the fluorophore concentration is linearly proportional to the fluorescence intensity. This intensity profile can then be used in order to estimate D [9].

The FRAP method utilizes a fluorescence microscope together with lasers as light source, along with photo-multipliers as detectors. One usually talks about the *prebleach-time* as the period of time before any bleaching of the molecules has occurred and the *postbleach-time* as the period of time after the molecules have been bleached. Figure 2.1 illustrates an example of the bleaching process of a typical sample measured using the FRAP method.

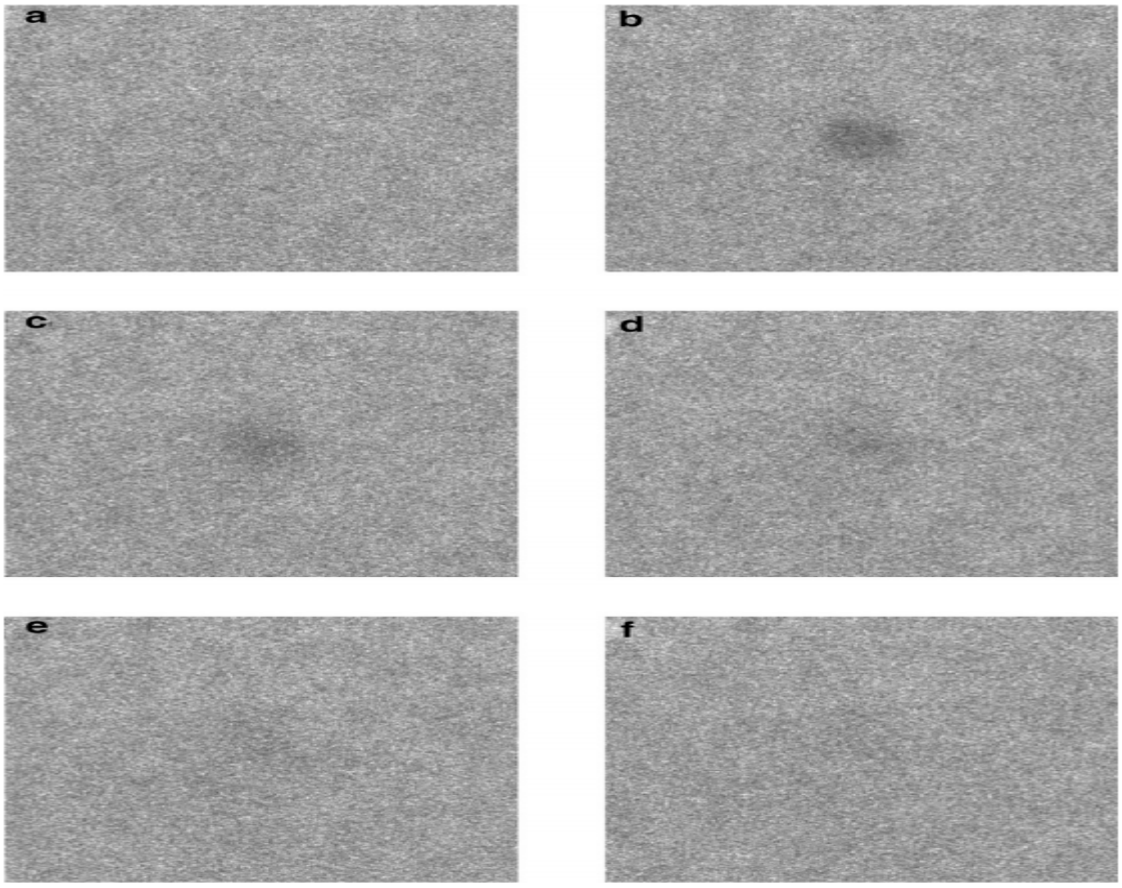


Figure 2.1: Images showcasing a typical FRAP measurement. Image (a) depicts the prebleach phase; the second image (b) is taken directly after the bleaching; the following images are recorded (c) 5 s, (d) 20 s, (e) 50 s, and (f) 100 s after bleaching [12].

2.1.3 The FRAP Model

In the FRAP model, see [9], the fluorescence decay from photobleaching is often assumed to be an irreversible first order reaction

$$\text{flourophore} + \text{photon} \rightarrow \text{bleach product} \tag{2.3}$$

It is assumed that the bleaching occurs instantaneously, and the proportion of

bleached flourophores are used as the initial condition for Ficks second law in Equation 2.2. It is usually assumed that the size of the liquid suspension is infinite in the 2D plane, meaning that the diffusing molecules should not reach any stopping boundaries. Another common assumption made is that that there is no net diffusion in the z direction, so only the 2D plane needs to be considered.

Let $c(x, y, z, t)$ be the concentration of flourophores, and $I_b(x, y, z)$ be the photobleaching illumination intensity in the sample. This intensity is often modeled as a Gaussian distribution. Then the decay of fluorescence is described by

$$\frac{dc(x, y, z, t)}{d(t)} = c_0 I_b(x, y, z) c(x, y, z) \tag{2.4}$$

which leads to the solution

$$c(x, y, z, t) = c_0 e^{-\alpha I_b(x,y,z)t} \tag{2.5}$$

with c_0 being the initial proportion of flourphore molecules before bleaching and α being a flourophore-depended parameter impacting the bleaching [9].

2.1.4 Parameter estimation

In order to estimate the parameters from the frames resulting from the FRAP experiment one can either estimate them by fitting a model to the pixels of the data or to a *recovery curve*.

The general noise model that is assumed for most methods is that each pixel is viewed as independent from another. The experimental noise is assumed to follow a normal distribution with zero mean and the variance $\sigma^2(c(x, y, t))$ defined as

$$\sigma^2(c(x, y, t)) = a + bc(x, y, t) \tag{2.6}$$

for the concentration $c(x, y, t)$ and where a represents a constant noise factor and b a noise factor proportional to the concentration.

The recovery curve method calculates a curve related to the fluorescence intensity of the sample. An example of a recovery curve, and how it relates to the bleaching process, is shown in Figure 2.2. From experimental data a simple model for describing the recovery curve has been found. It is defined as

$$F(t) = F_0(1 - \exp(\frac{-w^2}{4Dt}) + b), \tag{2.7}$$

where F_0 is the initial fluorescence intensity, $F(t)$ the fluorescence in the area of interest at time t, after the photobleaching has occurred, w the radius of the area of interest, b the fraction of the fluorophores that are bleached in the beginning and D being the diffusion coefficient [7]. This, however, is not the model used in this project. Instead the recovery curve will be extracted using a numerical solution for the diffusion equation.

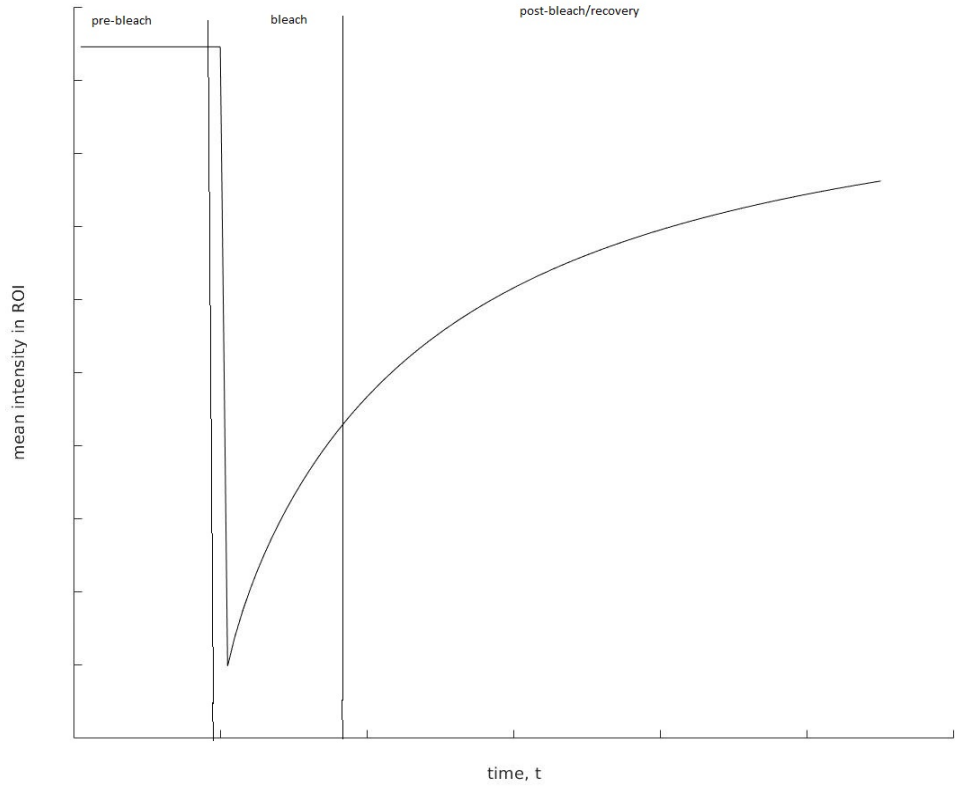


Figure 2.2: A typical recovery curve form a FRAP experiment. The curve is constant in the pre-bleach period. After the bleaching, where the flourophores are photobleached, the particles diffuse in and out of the area, leading to a recovery in the intensity in the ROI.

2.2 Numerical solution to the diffusion equation

The software for simulating FRAP experiments was implemented in MATLAB [11] by solving Ficks second equation of diffusion, Equation 2.2, numerically.
The initial concentration is assumed to be $c(x, y) = c_o$. After the first bleach frame the concentration becomes

$$c(x,y) = \begin{cases} c_0\alpha, & (x,y) \in Q \\ c_0, & (x,y) \notin Q \end{cases} \tag{2.8}$$

where Q is the bleached region of interest, typically of circular shape and with radius r, and α is a flourophore-dependent photobleaching parameter.

The numerical solution for the diffusion equation is solved by using periodic boundary conditions on a $(N+2M)(N+2M)$ grid, where $N = 256$ represents the resolution of the simulated frames, and $M = 128$ represents padding added in order to avoid periodicity artifacts from the diffusion.

The concentration $c(x, y, t)$, is represented as a $(N + 2M)(N + 2M)(t)$ matrix with the bleach concentration for each grid in each frame corresponding to the frame associated with each value t. The bleaching is represented by a multiplication with a mask consisting of the value 1 outside Q and α inside Q, with some intermediate value on the edges. Smooth edges of the ROI are ensured by supersampling, which is a technique used for reducing the jaggedness of the pixels in an image. An illustration of the simulated FRAP experiments is shown in Figure 2.3 with leftmost image being taken before the bleaching occurred, the middle just after bleaching, and the rightmost being the final simulated image in the series. The matrix $c(x, y, t)$ is then

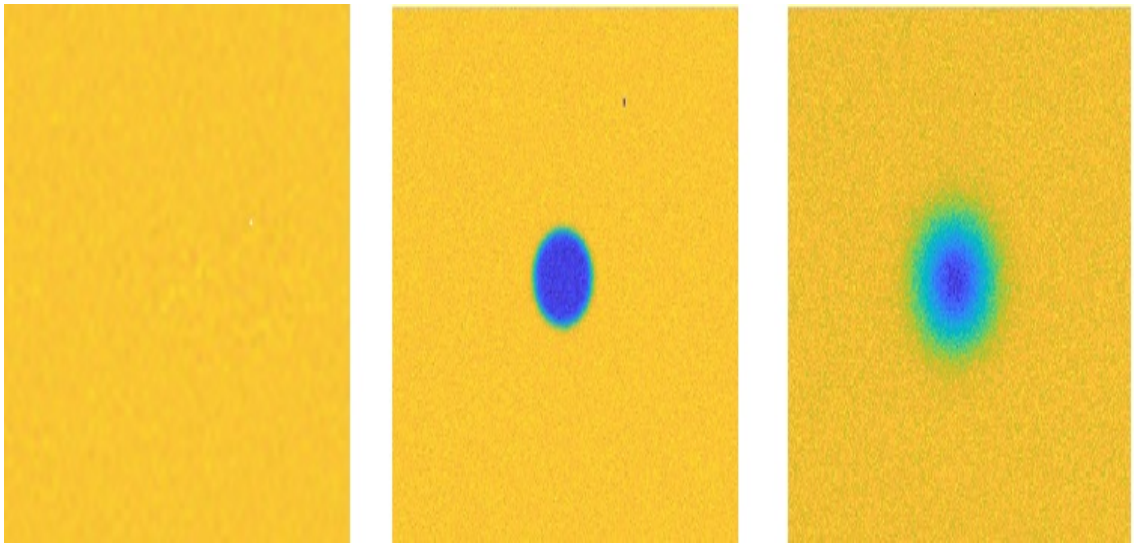


Figure 2.3: Simulated data from a FRAP experiment. The leftmost image being in before any bleaching has occurred, in first frame of the pre-bleach time. The middle image illustrating the simulated sample directly after the bleach has occurred, the first frame of the post-bleach time. The rightmost image illustrating the final frame of the post-bleach time.

transformed into a spectral representation $\hat{c}(\xi, \eta, t)$ using the fast Fourier Transform. This transform yields a set of independent ordinary differential equation on the form

$$\frac{\partial \hat{c}(\xi, \eta, t)}{\partial t} = -(\xi^2 + \eta^2) D \hat{c}(\xi, \eta, t) \tag{2.9}$$

for each grid point (ξ, η). Their solutions are available explicitly on the form

$$\hat{c}(\xi, \eta, t + \Delta t) = e^{-(\xi^2+\eta^2)D\Delta t} \hat{c}(\xi, \eta), \tag{2.10}$$

where Δt is the time between two frames. The solution in the spatial domain is obtained by performing an inverse Fourier transform for each t.

In order to extract the recovery curve the mean intensity from the simulated frames the equation

$$F(t) = \sum_{x,y} w(x, y, t) c(x, y, t) \tag{2.11}$$

was used. Where $w(x, y, t)$ is a normalization matrix and c(x,y,t) the concentration of the sample. The noise variance of the recovery curves is also i.i.d and follows a normal distribution with zero mean and variance

$$\sigma^2(F(\theta, t)) = \sum_{x,y} w(x, y)^2(a + bc(x, y, t)). \tag{2.12}$$

The log likelihood function of the residuals between the measured experimental recover curve versus the recovery curve for a given set of parameters then becomes

$$l(\theta) = -\frac{1}{2}\sum_t \log(2\pi\sigma^2(F(\theta, t))) - \frac{1}{2}\sum_t \frac{(F_{\text{exp}}(t) - F(\theta, t))^2}{\sigma^2 F(\theta, t)}, \tag{2.13}$$

where $F(\theta, t))$ is the recovery curve given the parameter vector $\theta = [D, \alpha, C_0, a]$ and the time t. F_{exp} is the experimental recovery curve calculated from the measured data. When simulating the data the noise parameter proportional to the concentration was set to zero, $b = 0$, we are making the assumption that the noise variance is constant. The log likelihood function then simplifies to the negative sum of the residuals squared and the likelihood parameters, $[D, C_0, \alpha]$, can be estimated using least squares.

The recovery curve $F_{exp}(t)$ was simulated to be as close to experimental data as possible, given the true parameters θ, while the recovery curve $F(\theta, t)$ was simulated as noise free data.

2.3 Bayesian inference

In this section the theory for Bayesian inference will be presented. Firstly, a brief introduction of the Bayesian framework will be presented followed by the Metropolis-Hastings algorithm for posterior sampling.

2.3.1 Bayesian framework

The core principle of the Bayesian framework is *Bayes' theorem*, stated as

$$\text{P(A|B)} = \frac{\text{P(B|A)P(A)}}{P(B)}, \tag{2.14}$$

where A and B are events and P(B) $\neq$ 0. The theorem relates the conditional probability P(A|B) of observing the event A after observing the event B, to the conditional probability P(B|A) and the probabilities P(A) and P(B).
Equation 2.14 has an analogous form for probability densities,

$$\text{f}_{X|Y}(x|y) = \frac{\text{f}_{X,Y}(x, y)\text{f}_X(x)}{\text{f}_Y(y)}, \tag{2.15}$$

where X and Y are random variables and $f_X(x)$,$f_{X,Y}(x, y)$ and $f_{X|Y}(x)$ are the marginal, joint and conditional distributions.

The way the Bayesian framework differs, for inference, from a frequentist framework is in the way a parameter vector for a model, $\boldsymbol{\theta} \in \mathbb{R}^d$, is viewed. In the Bayesian framework the parameters in $\boldsymbol{\theta}$ are considered to be random variables, while in the frequentist framework, they are considered to be constant.
In general, in the Bayesian framework, we are interested in the distribution of the of the parameter $\boldsymbol{\theta}$ conditionally on the observed data, $\mathbf{y}$. This can be expressed, using Bayes' theorem as

$$p(\boldsymbol{\theta}|\mathbf{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y})}. \tag{2.16}$$

.
Here, $p(\boldsymbol{\theta})$ is called the *prior distribution* which represents the prior belief of the parameter values and should be decided before any measurements take place. $p(\boldsymbol{\theta}|\boldsymbol{y})$ is known as the *posterior distribution* and represents the change in prior distribution after observing the data, $\boldsymbol{y}$, in accordance with Bayes' theorem. The distribution $p(\boldsymbol{y}|\boldsymbol{\theta})$ is known as the *likelihood function* and is commonly denoted as $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})$, which expresses the likelihood of observing the data given a fixed set of parameter values. More commonly, it is used as a probability of how likely the parameter values are, given the fixed observed data, which explains the switch in the conditional variables in the $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})$ notation. The density $p(\boldsymbol{y})$ is in the Bayesian framework sometimes called *evidence* and can be expressed as the integral $\int \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})p(\boldsymbol{\theta})d\boldsymbol{\theta}$ which is often analytically intractable. In figure 2.4 the relationship between the prior distribution, the likelihood function and the posterior distribution is illustrated, with 30 measured samples.

Since Equation 2.16 is a probability density function which by definition needs to integrate to 1, the evidence can then be regarded as a normalising constant. We can then omit the evidence and express Equation 2.16 as

$$p(\boldsymbol{\theta}|\boldsymbol{y}) \propto \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})p(\boldsymbol{\theta}), \tag{2.17}$$

meaning that we in practice only need the prior and likelihood distributions. In most cases the posterior distribution will not be analytically tractable. This might happen if $p(y)$ is impossible or computationally heavy to calculate or if it is not possible to generate samples from the resulting posterior, $p(\boldsymbol{\theta}|\boldsymbol{y})$, from Equation 2.16.

One way of solving these problems is by approximating the distributions using sampling methods. One possibility is to use the *Markov Chain Monte Carlo* (MCMC) methods, and one example of this is the Metropolis-Hastings algorithm.

The Bayesian framework does not only provide methods for estimating the posterior distribution but, similarly to the frequentist framework, there are also methods for point estimation. Multiple choices exist, but the most common onew are *maximum a posteriori* (MAP), *posterior mean* and *posterior median*. All of these three minimize the posterior expected value with regards to different loss functions.

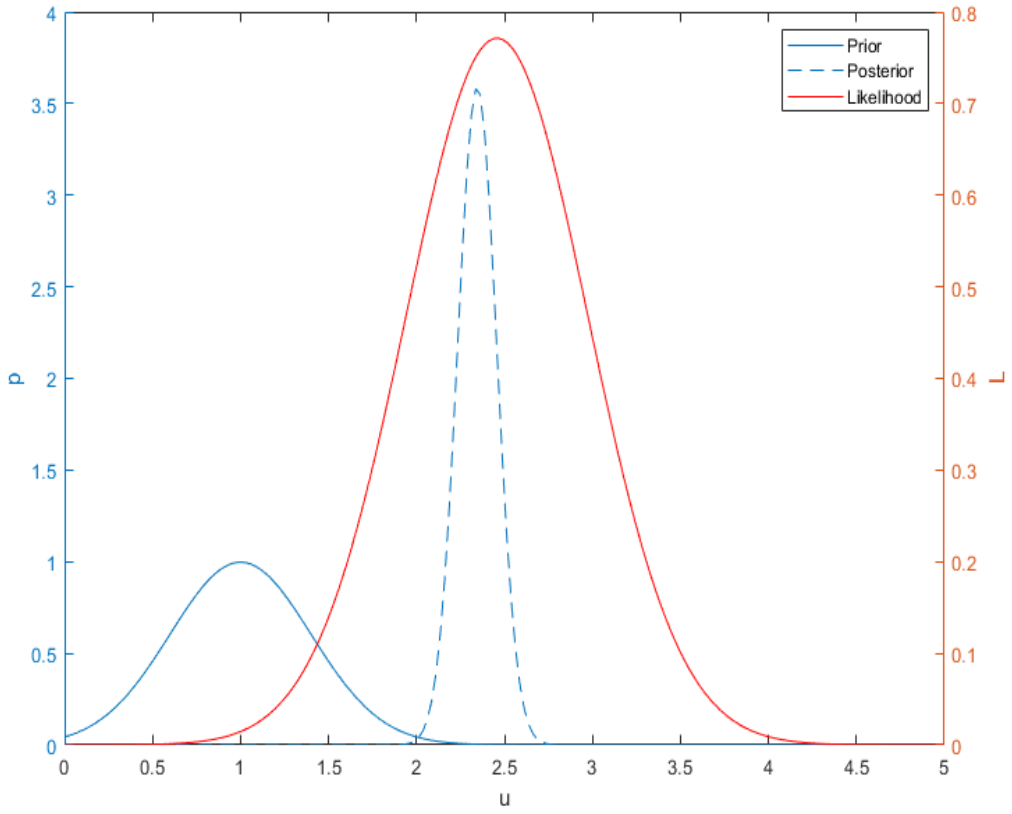


Figure 2.4: Illustration of how the posterior distribution is related to the prior and likelihood. Here, the prior u $\sim N(1, 0.4^2)$ and with 20 samples simulated from $N(2.5, 0.5^{)}$. The posterior then becomes Normal distributed with mean 2.34 and variance 0.1^2.

2.3.2 Metropolis-Hastings

As mentioned above the posterior distribution might often lack a simple analytical expression making it hard to sample from it. An MCMC algorithm is an algorithm used for sampling from probability distributions. This works as long as we know of some function, $f(\boldsymbol{x})$, that is proportional to the probability density that we want to estimate, $p(\boldsymbol{x})$,

$$f(\boldsymbol{x}) \propto p(\boldsymbol{x}). \tag{2.18}$$

Since $p(\boldsymbol{x})$ is a probability density it must integrate to 1, with the proportionality constant being $\int f(\boldsymbol{x})d\boldsymbol{x}$ over the entire domain of $f(\boldsymbol{x})$. The Markov chain will, under certain assumptions, asymptotically converge towards the target distribution $p(\boldsymbol{x})$.

The Metropolis-Hastings algorithm utilizes a *candidate distribution* $g(\boldsymbol{x}'|\boldsymbol{x})$ which proposes a new candidate for the next sample value, $\boldsymbol{x}'$, conditioned on the previous sample value $\boldsymbol{x}$. A common choice of a candidate distribution is a Gaussian centered around the previous proposed value, $\boldsymbol{x}$, which increases the likelihood of values close

to the previous one of being selected. It should be noted that if $g(\boldsymbol{x}'|\boldsymbol{x})$ is chosen as a symmetric function then the part of the ratio involving it will always equal 1. Then an *acceptance ratio* A is computed where A is defined as

$$A := \min\left(1, \frac{f(\boldsymbol{x}')g(\boldsymbol{x_t}|\boldsymbol{x}')}{f(\boldsymbol{x_t})g(\boldsymbol{x}'|\boldsymbol{x_t})}\right), \tag{2.19}$$

with $\boldsymbol{x_t}$ being the candidate value at time step t. The acceptance ratio is used in order to decide if the proposed candidate value is to be accepted or not. A value $u \sim U(0,1)$ is simulated and if $u \leq Aa$ the proposed value is accepted and $\boldsymbol{x}_{t+1} = \boldsymbol{x}'$. If not then the proposed value will be rejected and the previous value is kept, i.e. $\boldsymbol{x}_{t+1} = \boldsymbol{x_t}$. If a proposed move is more probable than the previous one it will always be accepted while a proposed move that is less probable might get rejected, with a larger drop in probability increasing the likelihood of rejection of the proposed move. This will result in the algorithm exploring more values in high density areas and fewer in low density areas. Doing this will allow the distribution of the state $\boldsymbol{x}_t$ to converge towards the target distribution, $p(\boldsymbol{x})$, as $t \to \infty$.

The convergence rate of the Metropolis-Hastings algorithm depends on both the initial guess of $\boldsymbol{x}_0$, and how similar the $f(\boldsymbol{x})$ is to the target distribution. It can vary greatly and is not always guaranteed. The time that it takes for the chain to converge is called the *burn-in period*. When all the states from the burn-in period are removed the remaining states in the chain are then considered as samples from the target distribution. It should be noted that this does require independent samples, which is something that the chain does not generate. One solution to this issue is to sample every jth state for some j. This will result in an approximately independent set of states. The optimal acceptance rate of the Metropolis-Hastings algorithm when applied to certain target distributions is 0.234, which indicates that an acceptance ratio of between $20 - 25\%$ is reasonable to aim for [3].

2.3.3 Metropolis-Hastings in Bayesian inference

Using the Metropolis-Hastings algorithm, outline in algorithm 1, for Bayesian inference then the target distribution is the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$. Since, as seen in Equation 2.17, the posterior is proportional to the likelihood function multiplied with the prior, the Metropolis-Hastings algorithm can then be used to generate samples from the posterior distribution, without need to compute the normalization constant.

Often computations are done on the log likelihood $l(\boldsymbol{\theta}|\boldsymbol{y}) = \text{Log } \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})$. This transformation is done since the likelihood function is often represented as a product of multiple probability distributions. This function is called the *Log Likelihood*. This transform is also beneficial since the range of the value in the likelihood function is often large, which could lead to numerical issues. By reducing this range a more stable algorithm can be achieved. After a log transform the acceptance ratio, step 4.2 in algorithm 1, becomes

$$\log A := \min(0, \log f(\boldsymbol{x}') + \log g(\boldsymbol{x_t}|\boldsymbol{x}') - \log f(\boldsymbol{x_t}) - \log g(\boldsymbol{x}'|\boldsymbol{x_t})) \tag{2.20}$$

Algorithm 1 Metropolis-Hastings

1: **Input**: $\boldsymbol{x}_0, f(\boldsymbol{x}), g(\boldsymbol{x}'|\boldsymbol{x}), T_{end}$
2: **Output**: $(\boldsymbol{x}_t)_{t=1}^{T_{end}}$
3: **Initialize**
   1. Set an initial starting point x_0.
   2. Set $t := 0$.
4: **Proposal:**
   1. Sample a value $\boldsymbol{x}'$ from $g(\boldsymbol{x}'|\boldsymbol{x_t})$.
   2. Compute acceptance ratio $A := \min(1, \frac{f(\boldsymbol{x}')g(\boldsymbol{x_t}|\boldsymbol{x}')}{f(\boldsymbol{x_t})g(\boldsymbol{x}'|\boldsymbol{x_t})})$.
5: **Accept or Reject**
   1. Sample value $u \sim U(0, 1)$.
   2. If $u \leq A$, then $\boldsymbol{x}_{t+1} = \boldsymbol{x}'$.
   3. If $u > A$, then $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t$.
6: **End Condition**
   1. Set $t := t + 1$
   2. If $t < T_{end}$ then continue to step 4.
   3. If $t = T_{end}$ then break.

and the check performed to see if the proposed state is accepted becomes $\log u \leq \log A$. With the corresponding change for the rejection case.

2.4 Artificial Neural Networks

An artificial neural network (ANN) is a model used for both supervised classification and regression problems. By constructing a network of *nodes*, inspired by biological neurons, with one or more layers, each one containing nodes that are interconnected with the all the other nodes in the adjacent layers. One of the simpler forms of an artificial neural network is a *feedforward neural network*.

2.4.1 Feedforward Neural Network

The simplest feedforward neural network consists of a single layer of output nodes with weights connecting the input nodes to the output nodes. Each layer in the network is only able to learn linearly separable patterns, but by increasing the number of layers more complex mappings can be represented.

For a network with layers $L \geq 1$ with the number of nodes in each layer, l_i, being $N_1, ..., N_L$. The layers are connected with a weight matrix $W_{j,k} \in \mathbb{R}^{N_i} \times \mathbb{R}^{N_{i+1}}$, with j being the index for the nodes in layer l_i and k for the nodes in layer l_{i+1}. The layers also have a corresponding bias vector $\boldsymbol{b_i} \in \mathbb{R}^{n_i}$. Each layer also has an *activation function*, g, associated with it. These are often nonlinear since it allows for the network to approximate nonlinear functions [5]. This results in the equation

$$\boldsymbol{V_j^{i+1}} = g(\sum_k W_{j,k}^i \boldsymbol{x_k^i} - \boldsymbol{b_j^i}) \tag{2.21}$$

with $\boldsymbol{V_j^{i+1}}$ being node j in layer l_{i+1} and with $\boldsymbol{x_k^i}$. The superscript represents the layer, l, while the subscript represents the nodes of the layer. The network, $\mathcal{F}$, represents a mapping which is the combination of function composition and matrix multiplication of all the earlier layers, such as $\mathcal{F} := g(W^L) \circ ... \circ g(W^1\boldsymbol{x})$.

If the activation function is chosen to be a linear, identity, function then the network will perform a simple linear regression. For regression problems the most common activation functions are the ReLu, tanh and Elu.

$$\text{ReLu}(x) = \max(0, x) \in [0, \infty] \tag{2.22}$$

$$\tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}} \in [-1, 1] \tag{2.23}$$

$$\text{Elu}(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases} \in [-\alpha, \infty] \tag{2.24}$$

Most of these activation functions are differentiable everywhere, or at least almost everywhere. This function enables the differentiation of the whole network with respect to the weights and bias using the *backpropagation algorithm* which utilises gradient descent in order to minimize the *loss function*.

2.4.2 Optimization

In an optimization problem a *loss function*, denoted here as C, is a function that is utilized in optimization problems. A loss function, $C : R^n \to R$, has as input a vector of size n, and returns a scalar that is proportionate to the algorithms prediction. For regression problems the most common loss functions is the mean square error (MSE).

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathcal{F}(y_i))^2, \tag{2.25}$$

with y_i being the target and $\hat{y_i}$ the predicted value.

The goal of optimization problems is to minimize the loss function. In neural networks one common algorithm used for this is the *gradient descent* algorithm. The loss function is minimized with respect to the weights and biases in the network. This process is done by propagating backwards through the feedforward network and updating the values such that the loss function is minimized. Since the parameters of each layer l depends on the value of weights and biases of previous layers. This results in a recursive process where the partial derivative of each weight and bias is determined through the chain rule, propagating backwards from the output layer until the weight/bias has been reached.

Consider the set of all weights and bias

$$w := (W^L, ..., W^1, b^L, ..., b^1). \tag{2.26}$$

Using gradient descent the weights/biases are updated according to

$$w_i = w_i - \eta\frac{\partial C}{\partial w_i}, \tag{2.27}$$

where $\eta > 0$ is referred to as the learning rate and w_i ith element of w.

In the context of neural networks it is a common choice to use the *Stochastic Gradient Descent* (SGD) algorithm instead of the ordinary gradient descent. By avoiding to calculate the whole, true gradient, as in Equation 2.27, the SGD algorithm estimates the whole gradient based on multiple, random, smaller subset of the data called *mini-batch* at each step. This speeds up the computation, especially if the number of samples in the data set is large. It has been suggested that the solution obtained by the SGD algorithm generalizes better than other adaptive gradient descent methods [13].

Sometimes a momentum, ν, is added to the SGD algorithm in order to accelerate it. This is done by adding a fraction of the previous estimated gradient and thus reducing the oscillation of the algorithm in the non descent directions. By adding momentum the update rule in Equation 2.27 becomes

$$\begin{aligned} v_t &= \nu v_{t-1} + \eta \frac{\partial C}{\partial w_i} \\ w_i &= w_i - v_t \end{aligned} \quad . \tag{2.28}$$

A typical value for the momentum is $\nu = 0.9$ [10].

3
Results and Discussion

3.1 Training environment

The neural network was implemented using Tensorflow [1] and was trained using NVIDIA T4 GPUs. In each simulated recovery curve from the FRAP sample the first 10 points correspond to the pre-bleach frames and the final 100 to the post-bleach frames. The simulation was performed using, $\Delta t = 0.265$ s between frames, and with a pixel size of 7.6×10^{-7} m/pixels.

The four parameters are sampled as $D_{data} \sim \text{Log U}(D_{low}, D_{high})$, the diffusion constant, $C_{0data} \sim U(C_{0low}, C_{0high})$, the original flourescence intensity, $\alpha_{data} \sim U(\alpha_{low}, \alpha_{high})$, the bleach factor, and $a_{data} \sim \text{Log } U(a_{low}, a_{high})$, the additive noise defined in Equation 2.6. Where U is the uniform distribution, and Log U is a log-uniform distribution, with the D_{low} and D_{high} being the lower and upper bounds for the distributions. And likewise for the other parameters. The lower and upper boundaries for Ds distribution are [10^{-12},10^{-9}] in the units m^2/s. While the boundary values for C_0 are [0.50,1], for α, [0.45,0.95], and for a, $[1 \times 10^{-4}, 1 \times 10^{-2}]$, with C_0 and a being defined in arbitrary units, a.u, and α being a dimensionless quantity. By converting the boundaries of D into the more appropriate unit pixels^2/s one changes the boundary values to $[1.7, 1.7 \times 10^3]$. All values and distributions for the parameters can be found in Table 3.1. The model outlined in section 2.2 allows for a broader choice of parameters, but it was decided to focus on the four most central ones.

The network was trained to minimize the mean square error loss function, defined as

$$\text{MSE}(\boldsymbol{y}, \mathcal{F}(\boldsymbol{y})) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - \mathcal{F}(\boldsymbol{y}_i))^2, \tag{3.1}$$

with $\hat{\boldsymbol{y}}$ being the predicted value and $\boldsymbol{y}$ the corresponding target value.

3.2 Pre-processing and generation of the data

The data, Ξ, was generated using the method outlined in section 2.2 along with the experimental parameters found in Table 3.2. The data generation was implemented using Matlab [11]. From Table 3.1 a region of allowed parameter values can be extracted, call this region of accepted values in the parameter space Ω.

Table 3.1: The distribution that the parameters used for generating the FRAP data are simulated from. The distributions are chosen to accurately represent the parameter values encountered in FRAP experiments.

Parameter	Distribution
D	Log $U(1.7$ pixels^2/s$, 1.7 \times 10^3$ pixels^2/s$)$
C_0	$U(0.5, 1)$
α	$U(0.45, 0.95)$
a	Log $U(10^{-4}, 10^{-2}$

Table 3.2: The fixed experimental parameters used in the simulation of the FRAP data.

Parameter	Value
Δt	0.265 s
Pre-bleach frames	10
Post-bleach frames	100
Pixel size	7.6×10^{-7} m/pixels
Number of pixels	256
Padding	128

Along with fixed experimental parameters found in Table 3.2, the generated data parameter vector, $\theta = [D_{data}, C_{0data}, \alpha_{data}, a_{data}]$, was used to simulate 110 frames showcasing the diffusion. These frames were then the basis of the generated, experimental, recovery curve $F_{exp}(t)$. Then, four new parameters were generated, $\hat{\theta} = [\hat{D}, \hat{C}_0, \hat{\alpha}, \hat{a}]$, and the process was repeated resulting in the recovery curve $F(\hat{\theta}, t)$. Using this, the log likelihood function was then calculated in accordance with Equation 2.13. Each data sample consists of 110 data points of $F_{exp}(t)$, the four parameters in the vector $\hat{\theta}$, as well as the target $l(\hat{\theta})$.

A total of $N_{data} = 1,200,000$ data points were generated, since the data was simulated there were never any shortage of data which is something that neural networks benefits immensely from. The simulated data was split into a training, validation and a test set, where $N_{train} = 0.6N_{data}$, $N_{val} = 0.3N_{data}$ and $N_{test} = 0.1N_{data}$.

Since the log likelihood values differ substantially in scale, between $[-10^8, 10^2]$, and is skewed towards very low values since the probability of generated parameters $\hat{\theta}$ being the true parameters θ in all dimensions is low, thus leading to a majority of the simulated data points having a large negative log likelihood value associated with it. To combat this a transform of the values was performed in order to both speed up the convergence and reduce the risk of *exploding gradients*.

First, an affine transform was applied to ensure that all simulated log likelihood values had the same sign. Afterwards a logarithmic transform was applied on the

data set, Ξ, resulting in the total transform

$$\Xi' = \log(-(\Xi - 1000)), \tag{3.2}$$

with Ξ' being the new transformed data set. This transform resulted in approximately $\Xi' \in [10, 20]$.

The feature vectors of the parameters D and a were standardized to values in between [0,1] using a minmax-scaler

$$x_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{3.3}$$

with x being a feature vector. A transform of the input feature is associated with a better performance of the neural network [8] since the features are of the same order of magnitude. The transform was not applied to the features C_0 and α since they are bounded between $[0, 1]$, nor was the generated recovery curve rescaled since the values were also already bounded between $[0, 1]$.

The rescaling was calculated based on the training data. The transform was then applied on the entire data set Ξ'.

3.3 Neural Network

3.3.1 Training the neural network

When it comes to the design of neural networks it often comes down to intuition and guidelines, since no foolproof method for finding the best hyperparameters exists. There exists a trade off between the design of a network and time. A deep and wide network might be able to learn the data better, but will take longer time before being sufficiently trained.

Since we are performing regression with a fully connected feedforward network the initial idea was that a shallow, and wide, network might be ideal [2]. Often times when using neural networks for regression a shallow network is preferable, compared to networks for classification were a deeper network might be more advantageous.

3.3.2 Hyperparameter Optimization

The hyperparameters in the fully connected feedforward network was optimized with respect to the MAE metric. The hyperparameters were batch size N_B, learning rate η, momentum ν, amount of layers L and amount of neurons per layer N, and this was done using a grid search. It was found that the best values were $N_B = 128$, $\eta = 1 \times 10^{-4}$, $\nu = 0.95$, $L = 5$ and $N = 256$. In order to decrease the complexity of the network it was decided that each layer should have the same activation function and consist of the same amount of neurons per layer.

Table 3.3: The hyperparameters used for the training of the neural network.

Parameter	Value
N_B	128
η	10^{-4}
ν	0.95
L	5
N	256

It was found that the network generalized better the starting learning rate was chosen a bit larger and then allowed to decay after a fixed amount of epochs [14]. These hyperparameters were optimized through a grid search and it was found that a learning rate decay of 50% every 5000 epochs performed the best. All hyperparameters were optimized with respect to the loss metric *mean absolute error* (MAE), defined as

$$\text{MAE} = \frac{1}{n}\sum_{i}^{n} |\boldsymbol{y}_i - \mathcal{F}(\boldsymbol{y}_i)|, \tag{3.4}$$

on the validation set.

The activation function was chosen as the Elu function, previously defined in Equation 2.24, since it performs equally well compared, with optimal hyperparameters selection, to the ReLu activation function, with respect to the MAE value, and it is also associated with a faster learning speed, [4] with the added bonus of also being differentiable everywhere.

The final network then becomes a fully connected feedforward network with 4 layers, each having 256 neurons in each layer, with the ELU activation function and with the final layer as a linear function with one neuron.

3.4 Accelerated Metropolis-Hastings

In this section the accelerated Metropolis-Hastings (AMH) algorithm will be presented. The algorithm is similar to the one presented in Algorithm 1. The AMH utilizes the neural network, $\mathcal{F}$, to approximate the shifted log log likelihood function.

The proposal function was specified as a multivariate Gaussian with covariance Σ centered around the latest accepted proposal, x_t,

$$x' \sim g(\boldsymbol{x}'|\boldsymbol{x_t}) = N(\boldsymbol{x}_t, \Sigma). \tag{3.5}$$

Since this function is symmetric it can be removed from the computation of the acceptance ratio defined in Equation 2.19, and the acceptance ratio becomes $\log A := \min(0, \hat{\mathcal{F}}(\boldsymbol{x}') - \hat{\mathcal{F}}(\boldsymbol{x_t}))$. Where $\hat{\mathcal{F}}$ is the inverse transformed output from the neural network.

The accelerated likelihood function is only defined on a bounded parametric space,

Ω, that covers the data set and since the new proposed values from the Metropolis-Hastings algorithm might cross this region there needs to be method of dealing with this issue. One solution is to only accept proposed values that are inside the region Ω, and reject it otherwise. In implementation of this algorithm, described in algorithm 2. If a state outside the region Ω is proposed by the proposal function it will automatically get rejected and the algorithm continues to the next step.

The likelihood value for each proposed state was transformed from Ξ' back into Ξ after being estimated, meaning that the normal log likelihood was sampled in the algorithm. The covariance matrix Σ was chosen with the aim of allowing for approximately 20 to 25% of all proposed states to be accepted.

Algorithm 2 Accelerated Metropolis-Hastings Algorithm (AMH)

1: **Input**: $\boldsymbol{x}_0, \hat{\mathcal{F}}(\boldsymbol{x}), g(\boldsymbol{x}'|\boldsymbol{x}, \Sigma), T_{end,}, \Sigma$
2: **Output**: $(\boldsymbol{x}_t)_{t=1}^{T_{end}}$
3: **Initialize**
   1. Set an initial starting point x_0.
   2. Set $t := 0$.
4: **Proposal:**
   1. Sample a value $\boldsymbol{x}'$ from $g(\boldsymbol{x}'|\boldsymbol{x_t}, \Sigma)$.
   2. Compute acceptance ratio $\log A := \min(0, \hat{\mathcal{F}}(\boldsymbol{x}') - \hat{\mathcal{F}}(\boldsymbol{x_t}))$.
5: **Accept or Reject**
   1. Sample value $u \sim U(0, 1)$.
   2. If $\log u \leq \log \alpha$, then $\boldsymbol{x}_{t+1} = \boldsymbol{x}'$.
   3. If $\log u > \log \alpha$, then then $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t$.
6: **End Condition**
   1. Set $t := t + 1$
   2. If $t < T_{end}$ then continue to step 4.
   3. If $t = T_{end}$ then break.

The burn-in period for algorithm 2 was visually deduced by finding the point were the algorithm had seemingly converged to a somewhat narrow stripe. Call that point t_{burn}. All states with $t \leq t_{burn}$, $x^1, .., x^{t_{burn}}$, are discarded and the marginal distributions are then obtained using the remaining states.

3.5 Performance metrics

To ensure that the AMH algorithm performs well compared to the classical implementation of the MH algorithm three evaluation metrics will be used. Firstly, since the AMH is an accelerated algorithm it should be faster at proposing and accepting/rejecting a state than the corresponding classical MH algorithm.

Secondly, the posterior mean estimation and the posterior variance of the two densities should agree with each other. The posterior mean should ideally also agree with the original parameters used for generating the underlying data. And lastly

the estimated marginal posterior distributions using the two algorithms should be as similar as possible. To quantify this distribution similarity the *Jensen-Shannon distance* was used.

The Jensen-Shannon divergence is defined as

$$\mathrm{JSD}(P||Q) = \frac{1}{2}(D_{KL}(P||M) + D_{KL}(Q||M), \tag{3.6}$$

with the probability measure $M = \frac{1}{2}(P + Q)$ and D_{KL} being the Kullback-Leibler divergence defined as

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x)\log\,\frac{p(x)}{q(x)}, \tag{3.7}$$

where P and Q are continuous random variable with p and q denoting the corresponding probability density functions. When using the base 2 logarithm then the Jensen-Shannon divergence is bounded between 0 and 1. By taking the square root of the Jensen-Shannon divergence one gets the Jensen-Shannon distance (JSD). A JSD value of 0 indicates that the two probability density are equal while a value of 1 indicates that the densities have very few, or zero, similarities.

3.6 Results

In this section the results of the Accelerated Metropolis-Hastings algorithm as well as the results related to the neural network is presented. The section starts by introducing the convergence of the neural network and continue by presenting the results of the AMH algorithm.

3.6.1 Results from training

The neural network was trained with the hyperparameters defined in Table 3.3. The network was trained for a total of 14000 epochs. Then the best performing model, with respect to the validation metric MAE, was chosen.

In Figure 3.1 the logarithmic loss of the MSE for both the training and validation set of the neural network are shown. It is clear where the learning rate decay occurred, and that it also had a positive impact on the final achieved loss. One of the reasons why the loss curve appears to be so noisy can be that we used the SGD algorithm for the minimization of the loss function.

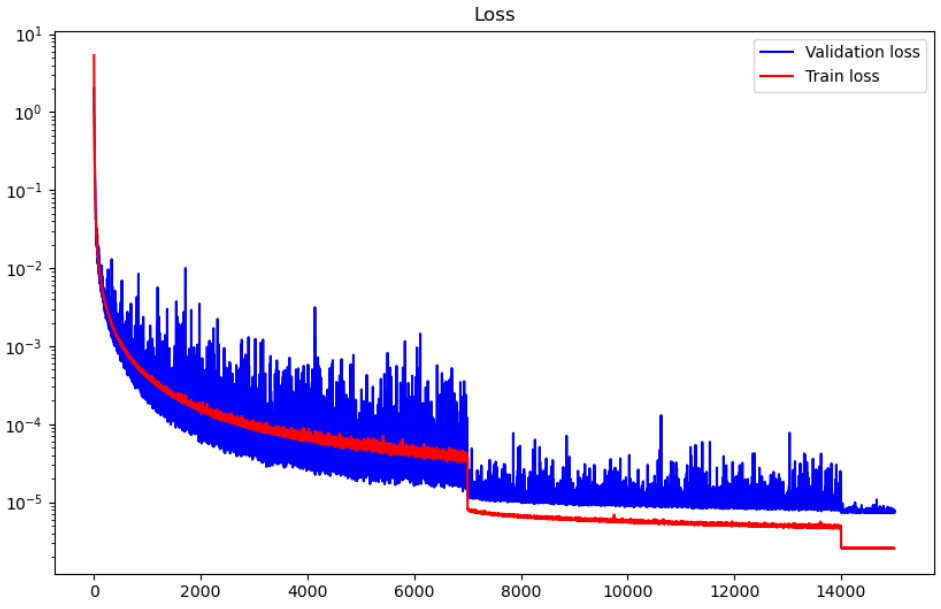


Figure 3.1: The validation and training loss of the network with the hyperparameters defined in Table 3.3.

3.6.2 Results from the algorithms

In this section the resulting marginal distributions for the parameters D, C_0, α and a from two different sets of true parameter values will be presented. The resulting marginal distributions from both the AMH algorithm and the classic MH algorithm will be compared, both through a histogram where the marginal densities are showcased, the posterior mean estimate, the posterior variance, a comparison through the JSD metric and in regards to the time per sample. Where the time per sample is defined as the time that it takes to propose a state, calculate the associated acceptance ratio, and update the state for all four parameters, measured in seconds.

The first set of true parameters are defined in Table 3.7, the parameters where generated from their respective distribution, defined in Table 3.1, and then rounded to two significant digits. The second set of parameter values were generated in the same fashion, and then rounded, and can be found in Table 3.8. The priors for the parameters $\theta = [D, C_0, \alpha, a]$ were chosen to be uniform, with the boundaries defined in section 3.1, this choice was made in order to simplify the implementation. Since the aim was to compare the resulting marginal posterior distributions estimated from the AMH algorithm compared with the classical MH, the choice of prior should not have an impact, as long as it is the same in both implementations.

There are quite a few metrics mentioned in this report that are used for evaluating the results of the algorithms, but they are not equally important. The metrics of main importance in the context of this report are the posterior mean estimates and the time per sample metric. The posterior mean since we are interested in estimating the true parameter values and the time per sample metric since the algorithm is supposed to be accelerated, compared to the classical MH.

In Figure 3.2 the convergence for all parameters for the AMH algorithm, outlined in algorithm 2 for the first set of generated parameters, is shown. The initial value $x_0 \in \Omega$ was chosen at random. In order to guarantee convergence the algorithm ran for a total of 2×10^5 iterations. The corresponding convergence plots are shown in Figure A.4, where a generous burn-in cutoff was chosen at 1.6×10^5.

While in the Figure 3.3 the convergence for the second set of parameters, found in Table 3.8, are shown. Similarly to the first case the initial value was chosen at random, and the algorithm ran for a total of 1.8×10^5 iterations, with the burn-in cutoff chosen at 1.5×10^5 iterations.

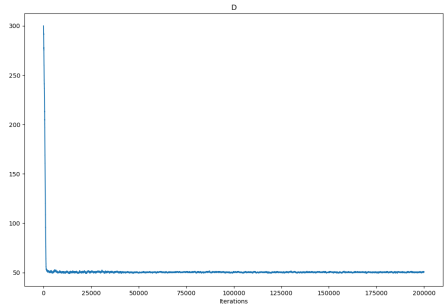
In Figure A.2 the marginal posterior distribution, generated from the AMH algorithm, for the parameters D, C_0, α and a are shown, with the true parameter values found in Table 3.7. In Figure A.3 the empirical marginal posterior distributions, that are generated using the classical Metropolis-Hastings algorithm defined in Algorithm 1, are shown, and in Figure 3.4 the marginal posterior distributions from both the AMH algorithm and the classical MH algorithm are overlayed on top of each other, and in the Table 3.9 the posterior mean estimates for both algorithms are displayed, along with acceptance rate and time needed per sample.

Since there exists fewer samples from the marginal distributions, generated from

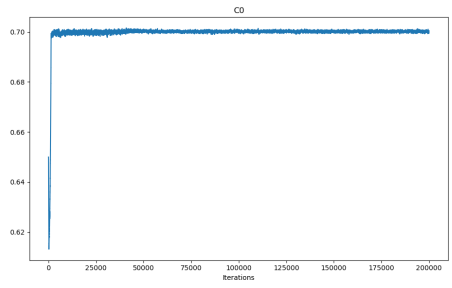
Table 3.4: The resulting JSD values for each parameter by comparing the marginal posterior distributions of the AMH and the classical MH algorithms. From the first run of the algorithm with parameters defined in Table 3.7.

Parameters	D	C_0	α	a
JDS metric	0.27	0.28	0.53	0.96

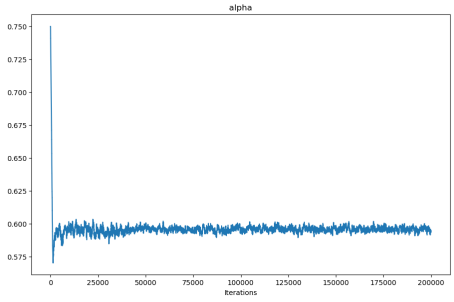
the classical algorithm, the distributions generated from the AMH algorithm were randomly subsampled so that the amount of samples in the corresponding distribution would be equal. In order to ease the convergence for the classical MH algorithm the initial starting point was chosen as the true parameter values. By comparing



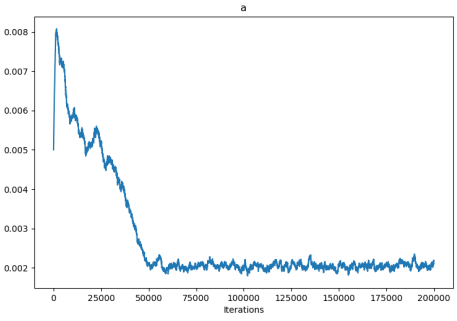
(a) D



(b) C_0



(c) α



(d) a

Figure 3.2: The convergence plots of the parameters D (a), C_0 (b), α (c) and a (d). These are the results from the first run of the AMH algorithm with the parameters defined in Table 3.7.

posterior mean values from Table 3.9 with the parameter values used for generating the experimental recovery curve, $F(\theta, t)_{exp}$, found in Table 3.7, it is clear that both algorithms works well with regards to the parameters D, C_0 and α but that the AMH algorithm underperforms in the a dimension while the classical algorithm, as expected, performs well in that dimension. It is also clear from comparing the two posterior marginal distributions in Figure 3.4d that they do not match. Estimating

Table 3.5: The resulting JSD values for each parameter by comparing the marginal posterior distributions of the AMH and the classical MH algorithms. From the second run of the algorithm with parameters defined in Table 3.8.

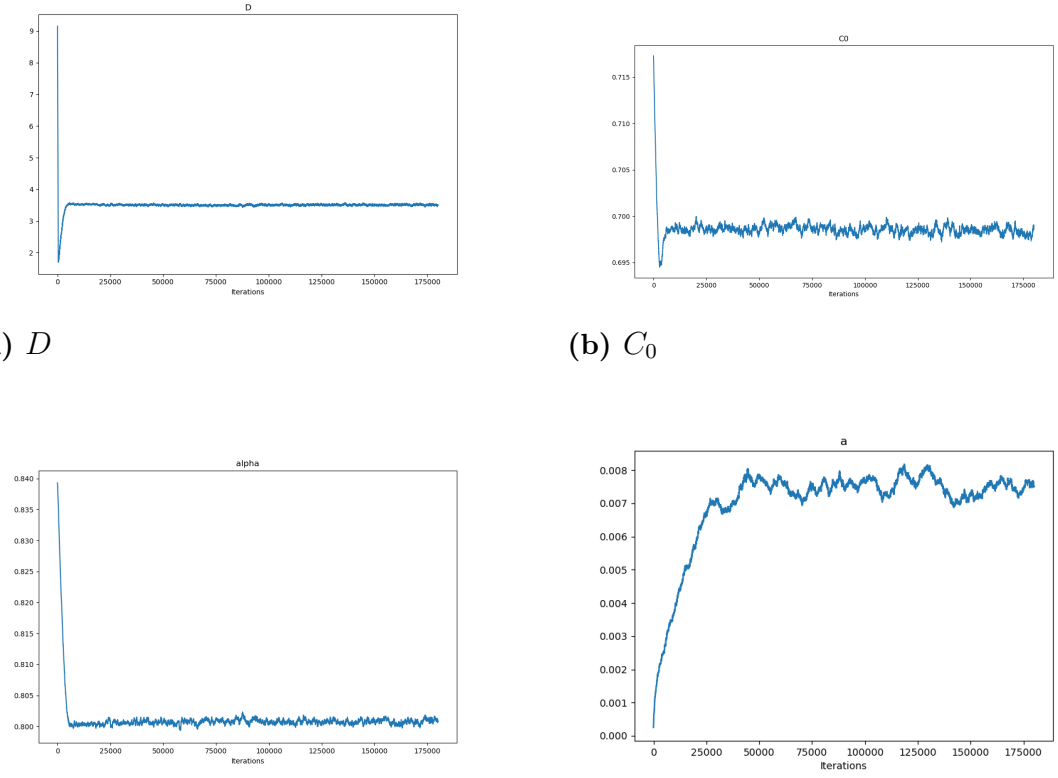
Parameters	D	C_0	α	a
JDS metric	0.58	0.97	0.81	1

the noise parameter, a, is seemingly not as simple as the other parameters D, C_0 and α, when using a fully connected neural network. There seems to be a bias for the neural network in predicting the likelihood with respect to the noise, a. This is apparent from both the posterior mean estimation, from Table 3.9, and the posterior distributions themselves, from Figure 3.4d.

It is unsurprising that the AMH algorithm managed to estimate the marginal posterior distribution for the parameter C_0 well since a lot of information about the initial concentration parameter, C_0, is contained in the pre-bleached frames. This argument is also strengthened by studying the speed of convergence of the C_0 parameter in Figure 3.2b, where it converges quickly to the correct value, indicating that it is a relatively easy parameter to estimate. The marginal posterior distribution for the parameter α is quite similar to the one obtained from the classic MH. Similarly to the parameter C_0, information about the bleach parameter α is also contained in the pre-bleach frames, meaning that it is probably one of the two easier marginal posterior distributions to estimate, along with C_0. This is clear from studying the estimated marginal posterior distribution in Figure 3.4b for C_0 and Figure 3.4c for α. By comparing the estimated posterior mean values for α and C_0 from Table 3.9 with the parameters used to generate the data, from Table 3.7 it is clear that the posterior mean estimates are consistent with the values used for the simulation of the data.

The situation for the second case, with the true parameters found in Table 3.8, is similar to the one mentioned above. The true data was generated using the parameters found in Table 3.8, and the corresponding convergence plots for all parameters are shown in Figure 3.3. The resulting metric values can be found in Table 3.6. Again, the marginal posterior distribution for the parameters D, C_0 and α seems to be relativity easy to estimate with the posterior mean being very similar between both the two algorithms as well the parameters used to generate the data. By studying the convergence plots the AMH algorithms seems to converge relatively quickly for the true values of the parameters D, C_0 and α. The comparison of the marginal posterior distributions for the second set of generative parameters are shown in Figure 3.5 The marginal variance for these parameters differ more, with the variance of the marginal distributions estimated by the AMH algorithm being, in general, an order of magnitude larger than the distributions estimated by the classic MH algorithm.

Similarly to the situation in the first case the AMH algorithm fails to converge to the true value of the noise parameter a, as seen in Figure 3.5d. The difference in the posterior mean estimate, between the two algorithms, is larger for the second



(a) D **(b)** C_0

(c) α **(d)** a

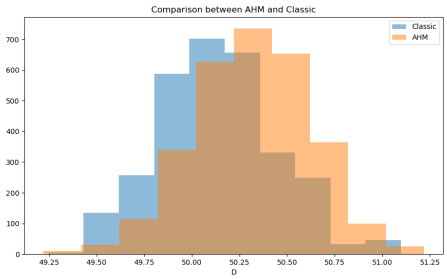
Figure 3.3: The convergence plots of the parameters D (a), C_0 (b), α (c) and a (d). These are the results from the second run of the AMH algorithm with the parameters defined in Table 3.8.

set of parameters than the first. One partial explanation for this is the fact that the true noise parameter is much lower in the second case, and that the neural network seems to find a minimum where the a parameter value is relatively large, no matter what the true value of the other parameters are.

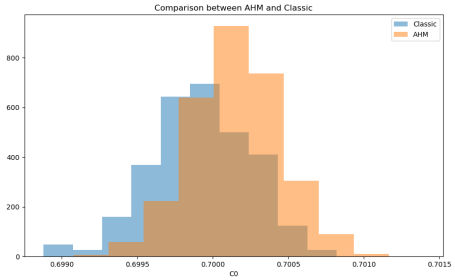
3.6.3 Comparing the algorithms

At first glance it is clear that the results from the two algorithms differ quite a bit, and that the neural network used in this project was not able to perfectly approximate the posterior density for any of the parameters. This is clear from the JSD metric, found in Table 3.4 for the first set of true parameter values and in Table 3.5 for the second set. It is clear from this metric that it, in general, agrees with the previous assessment that the marginal posterior distribution for the three parameters D, C_0 and α are somewhat well approximated, but the one for a parameter is not.

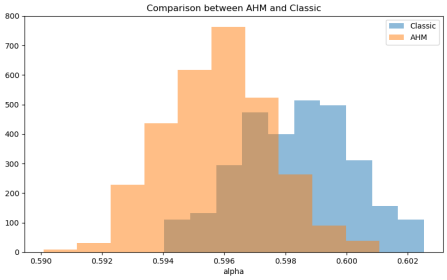
But the difference between the two algorithms is the time per sample metric. With regards to that metric it is clear that the AMH algorithm performs well, being able to propose samples 38 times faster than the classical implementation is a clear



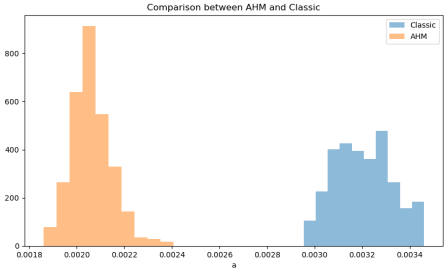
(a) Comparison of the marginal posterior distribution between the algorithms of the parameter D



(b) Comparison of the marginal posterior distribution between the algorithms of the parameter C_0

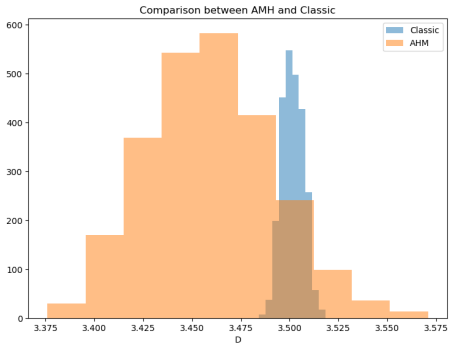


(c) Comparison of the marginal posterior distribution between the algorithms of the parameter α

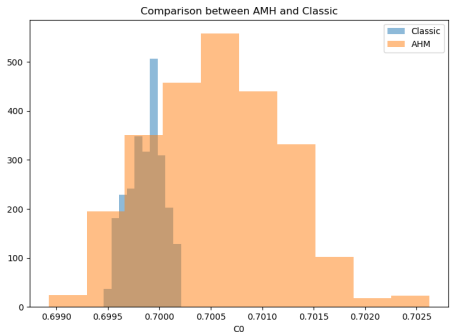


(d) Comparison of the marginal posterior distribution between the algorithms of the parameter a

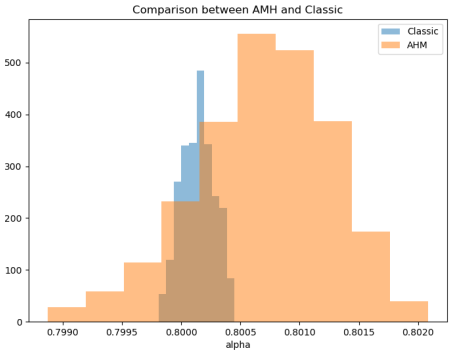
Figure 3.4: Histograms of the marginal posterior distributions resulting from the two algorithms AMH and classical MH for the parameters D (a), C_0 (b), α (c) and a (d). These are the results from the first run of the AMH and the classical MH algorithm with the parameters defined in Table 3.7.



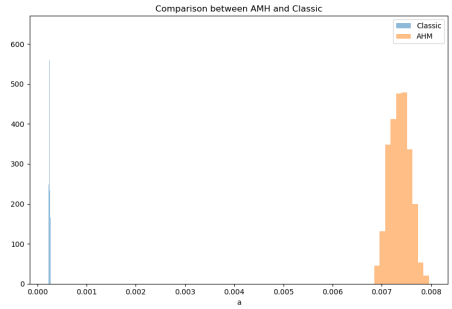
(a) Comparison of the marginal posterior distribution between the algorithms of the parameter D



(b) Comparison of the marginal posterior distribution between the algorithms of the parameter C_0



(c) Comparison of the marginal posterior distribution between the algorithms of the parameter α



(d) Comparison of the marginal posterior distribution between the algorithms of the parameter a

Figure 3.5: Histograms of the marginal posterior distributions resulting from the two algorithms AMH and classical MH for the parameters D (a), C_0 (b), α (c) and a (d). These are the results from the second run of the AMH and the classical MH algorithm with the parameters defined in Table 3.8.

Table 3.6: Resulting metrics for the parameters from the second run of the AMH and classical MH algorithm. The true parameters for the data can be found in Table 3.8.

Metric	Classic MH	AMH
Posterior Mean D	3.5	3.46
Posterior Variance D	3×10^{-5}	1×10^{-3}
Posterior Mean C_0	0.69	0.69
Posterior Variance C_0	2.8×10^{-8}	2×10^{-7}
Posterior Mean α	0.8	0.8
Posterior Variance α	1.8×10^{-8}	1.2×10^{-7}
Posterior Mean a	2.4×10^{-4}	2.8×10^{-3}
Posterior Variance a	1×10^{-10}	3.4×10^{-9}
Time per sample	7.5s	0.2s
Acceptance rate	26 %	21 %

Table 3.7: The true parameters used for simulating the underlying recovery curve for the first run of the AMH algorithm.

Parameters	D	C_0	α	a
Value	50	0.7	0.6	3.3×10^{-3}

improvement.

3.6.4 Estimating the parameter a

The network performs worse at estimating the marginal posterior density for the parameter a than the other parameters is clear. However the exact reason for this is hard to pinpoint. One possible reason for could be a bias in the networks approximation of the log likelihood function, with this bias mainly being visible in the estimation of the marginal posterior density of a. When the approximated log likelihood function is studied near the set of true parameter values the function appears to be very 'flat' in the a-dimension while the approximated log likelihood had a very optima with regards to the other three parameters. This 'flatness' of the approximated log likelihood function might results in the fact that even if the AMH algorithm manages to converge to the correct true a value, it might still drift in the a dimension and get 'stuck' in an optima that might be a residue from the approximation done by the network. This flatness is also something found in the true log likelihood function.

Table 3.8: The true parameters used for simulating the underlying recovery curve for the second run of the AMH algorithm.

Parameters	D	C_0	α	a
Value	3.5	0.7	0.8	2.5×10^{-4}

Table 3.9: Resulting metrics for the parameters from the first run of the AMH and classical MH algorithm. The true parameters for the data can be found in Table 3.7.

Metric	Classic MH	AMH
Posterior Mean D	50.13	50.32
Posterior Variance D	9.5×10^{-2}	9.5×10^{-2}
Posterior Mean C_0	0.7	0.7
Posterior Variance C_0	1.1×10^{-7}	9×10^{-8}
Posterior Mean α	0.6	0.59
Posterior Variance α	3.4×10^{-6}	3×10^{-6}
Posterior Mean a	3.2×10^{-3}	7×10^{-3}
Posterior Variance a	1.4×10^{-8}	3×10^{-8}
Time per sample	7.4s	0.2s
Acceptance rate	23 %	27 %

4
Conclusion

The goal with this project was to develop a deep learning-accelerated Metropolis-Hastings algorithm which works for Bayesian inference on the simulated FRAP data. The purpose was to decrease the computational time needed for Bayesian inference on the recovery curve model. A fully connected feedforward neural network was trained with regards to the four most central parameters of the model. Numerically simulated FRAP data has served as the training, validation and test data for the network.

The results indicate that it is possible to preserve the shape of the marginal posterior densities, of the chosen parameters, compared to the traditional Metropolis-Hastings. The results show that the posterior mean estimates for three out of four parameters are consistent with the estimates from the traditional algorithm, as well as the true values used for generating the data. Importantly, the implemented algorithm is much more computationally efficient than the traditional method.

Future work on the topic of this thesis should aim to improve the estimation of the noise parameter, a. Either by utilizing different types of network architecture that might be better suited for this type of problem, or by not approximating the log likelihood and instead focusing on the negative sum of the residuals squared to estimate the parameters D, C_0 and α and estimating a either through an additional neural network, or using some other method. Another idea is to perform a different transform in the preprocessing step and thus perhaps simplifying the estimation of the parameters. It is also possible that the noise parameter a is hard to estimate in general, and that no method performs well. This is also something can can be examined. Lastly, future work could also be done by extending the framework by including more FRAP parameters.

Bibliography

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep?, 2014.

[3] Mylène Bédard. Optimal acceptance rates for metropolis algorithms: Moving beyond 0.234. *Stochastic Processes and their Applications*, 118(12):2198–2222, 2008.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[6] Adolf Fick. Ueber diffusion. *Annalen der Physik*, 170(1):59–86, 1855.

[7] McGregor G. Jacobson K. A. Kapitza, H. G. Direct measurement of lateral transport in membranes by using time-resolved spatial photometry. *Proceedings of the National Academy of Sciences of the United States of America*, 82(12):4122–4126, 1985.

[8] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, page 9–50, Berlin, Heidelberg, 1998. Springer-Verlag.

[9] Niklas Lorén, Joel Hagman, Jenny K. Jonasson, Hendrik Deschout, Diana Bernin, Francesca Cella-Zanacchi, Alberto Diaspro, James G. McNally, Marcel Ameloot, Nick Smisdom, and et al. Fluorescence recovery after photobleaching in material and life sciences: putting theory into practice. *Quarterly Reviews of Biophysics*, 48(3):323–387, 2015.

[10] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[11] Magnus Röding, Leander Lacroix, Annika Krona, Tobias Gebäck, and Niklas Lorén. A highly accurate pixel-based frap model based on spectral-domain numerical methods. *Biophysical Journal*, 116(7):1348–1361, 2019.

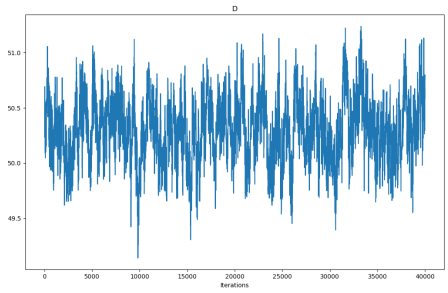
[12] Hermansson A. M. Ohgren C. Rudemo M. Lorén N. Schuster, E. Interactions and diffusion in fine-stranded -lactoglobulin gels determined via frap and binding. *Biophysical journal*, 106(1):253–262, 2014.

[13] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning, 2018.
[14] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I. Jordan. How does learning rate decay help modern neural networks?, 2019.

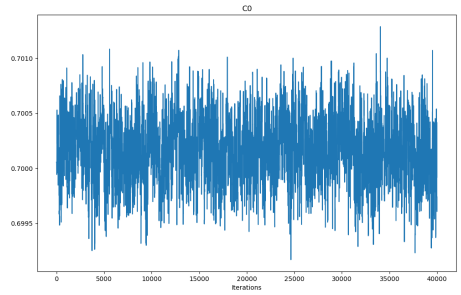
A

Appendix 1

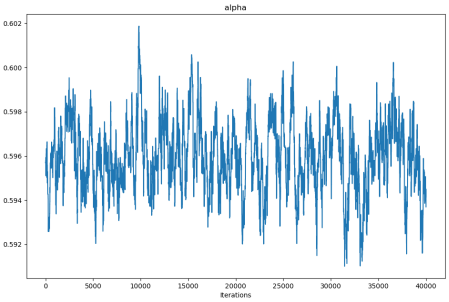
A.1 Convergence plots and histograms from AHM and classical MH algorithms



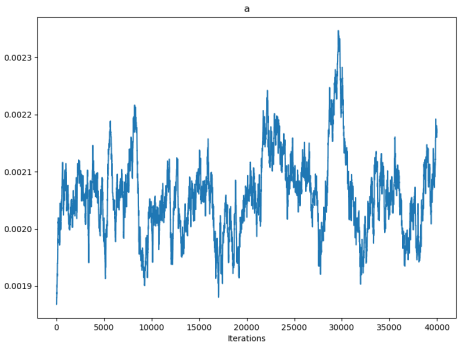
(a) D



(b) C_0

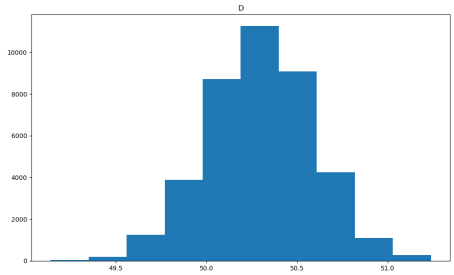


(c) α

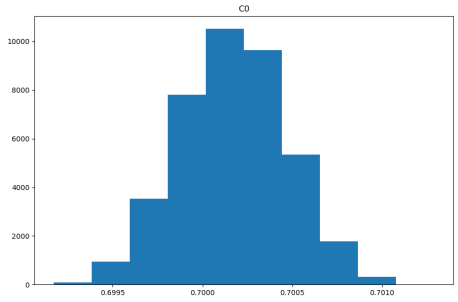


(d) a

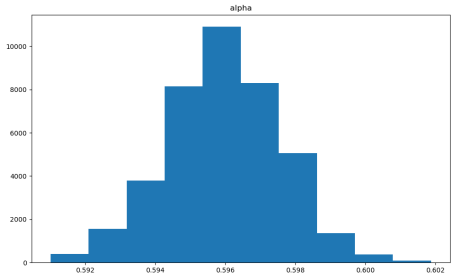
Figure A.1: The samples after the burn-in time of the parameters D (a), C_0 (b), α (c) and a (d). These are the result from the first run of the AMH algorithm with the parameters defined in table 3.7.



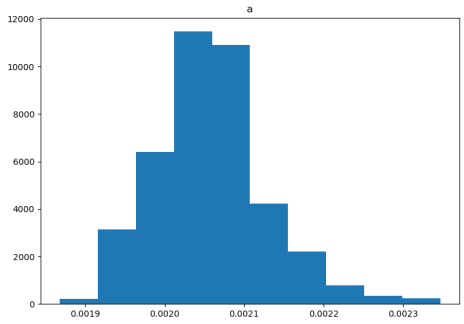
(a) D



(b) C_0

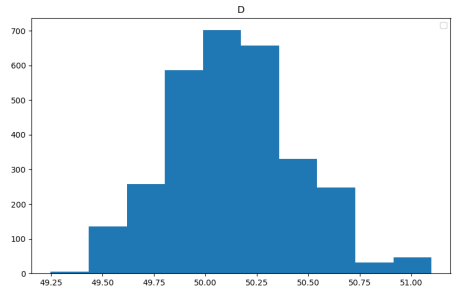


(c) α

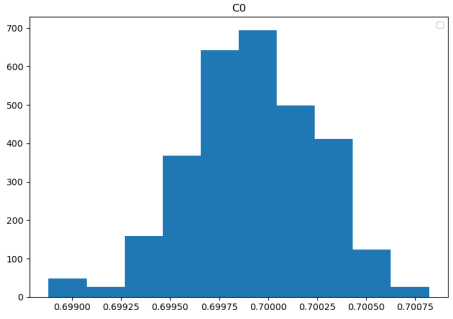


(d) a

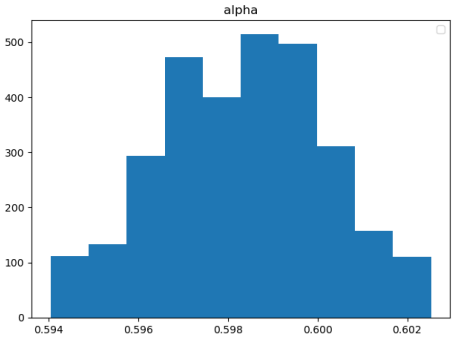
Figure A.2: The histograms of the parameters D (a), C_0 (b), α (c) and a (d). These are the result from the first run of the AMH algorithm with the parameters defined in table 3.7.



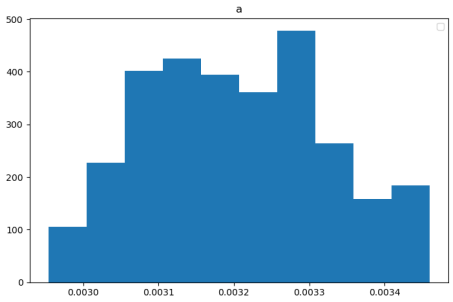
(a) D



(b) C_0

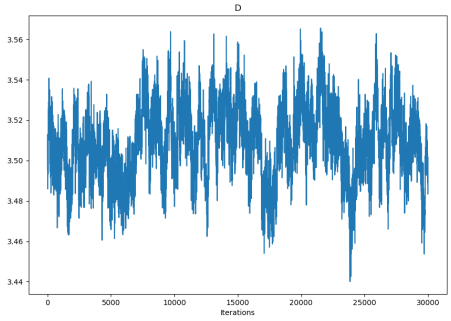


(c) α

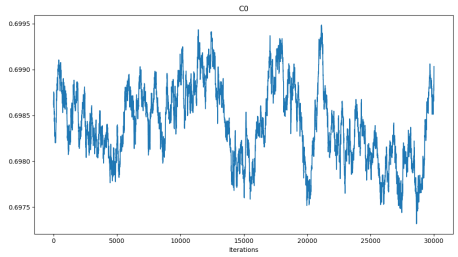


(d) a

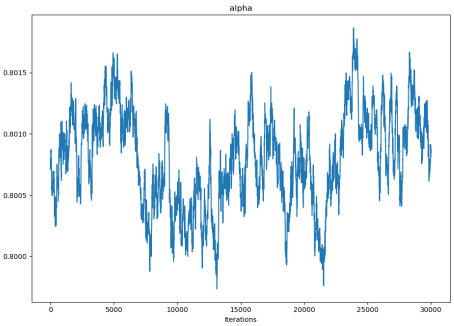
Figure A.3: The histograms of the parameters D (a), C_0 (b), α (c) and a (d). These are the result from the first run of the classical MH algorithm with the parameters defined in table 3.7.



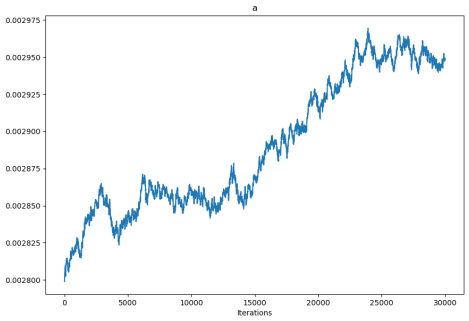
(a) D



(b) C_0



(c) α



(d) a

Figure A.4: The samples after the burn-in time of the parameters D (a), C_0 (b), α (c) and a (d). These are the result from the second run of the AMH algorithm with the parameters defined in table 3.8.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY