Bachelor's Thesis Computing Science

Radboud University Nijmegen

The covariance matrix adaptation evolution strategy (CMA-ES) as a method for fitting biological agent-based models

Author:
Julius Benjamins
S1022533

Daily supervisor:
dr. Inge Wortel

First assessor:
dr. Johannes Textor

Second assessor:
Prof. (dr. ir.) Arjen de Vries

August 3, 2022

Abstract

Biological agent-based models (BABMs) are a great tool in research to encode, test, and compare hypotheses on the biological data-generating process. If the output of a model matches the real world data, this suggests that the domain knowledge encoded in the model is sufficient to explain the observed data.

To see if a model can reproduce a certain target dataset, the BABMs have to be tuned, or fitted, in a specific way. This is essentially an optimization problem, but one that comes with challenges like stochasticity and scaling issues.

In this thesis, the covariance matrix adaptation evolution strategy (CMA-ES) is proposed as a candidate for fitting BABMs. To avoid the great time consumption that comes with complex models, the challenges were simulated using a simple model called the Beauchemin model that describes T-cell movement in the lymph node. CMA-ES was able to fit a single parameter of this model with only small variance between the solutions. This variance increased a small amount for fitting two parameters.

The two parameters of this model were fitted most accurately using Restart-CMA-ES, a strategy for CMA-ES where it gets restarted after termination, and problem rescaling, where the parameters got rescaled to fit in a significantly smaller range.

This problem rescaling also proved to counteract the scaling problems caused by the case where parameters live on vastly different scales. After logarithmically rescaling the problem, the solutions generated by CMA-ES were approximately the same for different scales and the base run where no scaling was applied.

Hence, CMA-ES is an algorithm to be considered for fitting BABMs.

Contents

1 Introduction **3**

2 Problem definition **5**

2.1 CMA-ES 5

2.1.1 CMA-ES inner workings 6

2.1.2 CMA-ES for minimising the diagonal of a rectangle . 8

2.2 The Beauchemin model 11

2.2.1 The parameters of the Beauchemin model 11

2.2.2 Using the difference between mean square displacements as the objective function for fitting the Beauchemin model 12

3 Research **16**

3.1 Stochasticity in the Beauchemin model 16

3.2 Fitting the Beauchemin model 18

3.2.1 CMA-ES correctly fitting a single parameter 18

3.2.2 CMA-ES fitting two parameters shows discrepancies between the target and actual values 21

3.2.3 CMA-ES unable to fit three parameters for the Beauchemin model 26

3.3 Restart strategies for CMA-ES 27

3.3.1 Methods for restarting CMA-ES 27

3.3.2 Regular CMA-ES, Restart-CMA-ES and IPOP-CMA-ES compared for fitting two parameters of the Beauchemin model 27

3.4 Scaling problems in ABMs 28

3.4.1 CMA-ES unable to identify linearly scaled parameter values 29

3.5 Rescaling variables 30

4 Methods **33**

4.1 Python implementation of CMA-ES 33

4.2 Beauchemin simulations 34

4.3 MSD calculation 34
4.4 Choosing parameters to optimise 35
4.5 Fitting single parameters 35
4.6 Scaling parameters using a wrapper function 35
4.7 Rescaling the problem 36

5 Discussion **37**
5.1 Fitting more than two parameters using CMA-ES 37
5.2 Additional scaling issues 37
5.3 Choosing CMA-ES parameters 38
5.3.1 Choosing the initial coordinate 38
5.3.2 Choosing the initial sigma 38
5.4 Using the error distribution to evaluate solutions 38
5.5 Cost penalty of using Restart strategies for CMA-ES 39
5.6 Choosing when to rescale a problem 40
5.7 Analytically solving the Beauchemin model 40

6 Related Work **42**
6.1 Stochastic gradient descent using the Adam algorithm 42

7 Conclusion **44**

A Appendix **48**
A.1 Machine specifications 48
A.2 CMA-ES Output 48
A.3 Beauchemin parameter settings 50
A.4 cmaes.py 50
A.5 run-beauchemin.R 52

Chapter 1

Introduction

In the field of bioinformatics, Agent-Based Models (ABMs) are a great tool to simulate the behaviour of their real-world equivalent. This simulated data can, for example, be used to encode, test, and compare hypotheses on the biological data-generating process.

In this thesis we are interested in ABMs that simulate cell behaviour, from here on called Biological ABMs (BABMs). An example of such a BABM is the Beauchemin model [1]. The Beauchemin model describes stochastic lymphocyte migration in tissue [2]. If there is data on actual cell behaviour, the parameters of your BABM can be tuned to simulate the same data, and to compose or corroborate hypotheses.

However, tuning the parameters for a model such as the Beauchemin model to generate the desired output is quite difficult. This tuning is called fitting, and ideally is not done by hand. Fitting is in fact an optimisation problem, where there is a certain chunk of target data, a model that takes parameters to generate new data and an objective function that compares the target and newly generated data. The goal, minimization of the objective function, or the difference between the target and generated data.

While there are algorithms that can be used to fit a model to simulate desired output, these are not all suitable for fitting BABMs for multiple reasons.

A toy example to help illustrate the problems that come with BABMs could be the following. We want to minimise the diagonal of a rectangle with the constraint of having an area as close to, but not exceeding, 100 metres. Here, the input parameters are a for the vertical sides and b for the horizontal sides. Quickly, it can be seen that the solution would be $a = b = 10$cm. Still, if we were interested in fitting this model, the objective function $f(a, b)$ should be minimised. This function has to constrain the area to not exceed 100, as well as minimise the diagonal.

The first problem is that a lot of BABMs are stochastic. This means that the output generated from a simulation can vary run-to-run even though the input is the same. The toy model being stochastic could be caused by measuring errors for the sides, resulting in a different area and diagonal each run. This makes finding the optimal solution more difficult.

Secondly, there are usually several input parameters for these BABMs that need to be fitted. This adds a layer of difficulty as opposed to fitting only a single parameter. These parameters can e.g. influence each other. For the toy example this is clear, while increasing one side of the rectangle, the area will grow, which has consequences for the other side. Additional to influencing each other, the parameters can live on different scales. For the rectangle it can be that the a sides are measured in kilometres, while the b sides are measured in millimetres. If the starting point of both the sides would be 0, the optimisation method would first have to figure out the correct scales for the parameters.

Finally, these models take an extensive amount of time to execute, adding another inconvenience to the trial-by-trial process. As the model has to be executed for each evaluation of the objective function, the optimisation method has to be very efficient to reduce the time spent.

Ultimately, we need an optimization method that is able to deal with all the mentioned problems. An interesting candidate for this is the Covariance Matrix Adaptation Evolution Strategy, or CMA-ES. It is a method of minimising an objective function that promises great performance [3]. This algorithm is compelling for our purposes, as it is defined to be able to deal with multi-parameter optimization problems. It has not been used to fit BABMs such as the Beauchemin model, making it interesting to see how it will handle their problems.

In this paper, we will examine if the CMA-ES algorithm can be used to fit data for stochastic BABMs, especially the Beauchemin model, that have multi-parameter dependence, scaling problems and costly computations.

Chapter 2

Problem definition

To fit a BABM, there are three requirements. First pick a method for optimisation. Second, the model that is to be fitted. Finally, an objective function that will yield the difference between the target and simulation data. This objective function is what will be minimised.

This chapter will describe how these requirements will be met for this thesis, by giving the method for optimisation, the model to be optimised and finally the objective function.

2.1 CMA-ES

The covariance matrix adaptation evolution strategy, or CMA-ES, is an evolution strategy. These are methods of stochastic, or random, search to minimize a certain objective function. Evolution strategies sample solutions from a multivariate distribution, which gets updated each generation of the algorithm. Naturally, the way this distribution is created and structured has a significant impact on the total performance of the evolution strategy. Covariance matrix adaptation is an approach of creating such a distribution using a matrix that changes each iteration. Hence, CMA-ES was designed to be the state-of-the-art application of evolution strategies to real-world search problems [4].

A covariance matrix is a matrix that describes the covariance, or joint coherence, of parameters. The covariance matrix in CMA-ES describes relation between parameters that are in the distribution of potential solutions, where these solutions would minimise the objective function. With each iteration, the goal of the new matrix is having solutions that have a higher probability of being a good solution. Together with the distribution itself, it keeps track of the corresponding mean, and a sigma value for each parameter. This sigma is the deviation of that mean, resulting in a range where solutions can be sampled from.

CMA is very effective way of sampling new solutions, as variants of CMA-

ES were shown to outperform over 30 other algorithms for problems that have difficult functions and large budgets [3]. This performance makes it a compelling strategy to fit BABMs. In addition, there is no other research about applying CMA-ES to biological ABMs such as the Beauchemin model, so it is still unclear if it is able to deal with the stochasticity and multi-parameter scaling problems, which is researched in this thesis.

2.1.1 CMA-ES inner workings

CMA-ES was proposed as an optimisation algorithm by Nikolaus Hansen et al. [4]. Usage, documentation and mathematical foundation can all be found the supplemental paper "The cma evolution strategy: A tutorial" [5]. The information in this section is a brief summary of key points from that paper.

There are many variants of CMA-ES. For this thesis, the most commonly used version, $(\mu/\mu_w, \lambda)$-CMA-ES was chosen.

Population size and the Mean

In $(\mu/\mu_w, \lambda)$-CMA-ES, λ is the population size, and denotes the amount of solutions that will be sampled in each iteration of the algorithm. μ is the number of best solutions from the λ sampled ones that will be used to dictate solutions subsequent iterations. λ gets calculated according to the following equation:

$$\lambda = 4 + \lfloor 3 \ln n \rfloor \tag{2.1}$$

Here, n is an integer that can be chosen by the user. Increasing the value of λ improves the global search capability at the cost of time consumption and consequently convergence speed. A good choice for n can be the amount of parameters that are being fitted. Using λ, μ gets calculated:

$$\mu = \lfloor \lambda/2 \rfloor \tag{2.2}$$

The λ solutions get sampled from the multivariate normal distribution:

$$\mathcal{N}(m, \sigma^2 C) \tag{2.3}$$

Here, C is the covariance matrix that gets updated using covariance matrix adaptation, as briefly discussed in the previous section. Sigma (σ) is the step-size, or deviation. m is the current mean, solutions are expected to lie within σ of this mean. Each parameter that is fitted has its own mean and σ, which dictates their values.

The μ best search points out of the λ sampled possibilities will determine the mean for the next generation $(g + 1)$ of search points. This calculation is a weighted average of the μ selected points:

$$m^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{g+1} \tag{2.4}$$

The actual value of a solution is denoted by $x_{i:\lambda}$, or solution i out of λ. These weights w are larger for solutions that are better, so that these solutions have a greater impact on calculation of the new mean. These can be calculated as described in Equation 49 in [5].

Step-size control in CMA-ES

Together with the covariance matrix C, the step-size σ controls the overall scale of the distribution where solutions will be sampled from (Equation 2.3).

If this step-size is too large, the distribution will be too broad, making it hard to pinpoint exact solutions. If the step-size is too little, the distribution will be too narrow, making CMA-ES not consider solutions outside of the distribution, which may result in sub-optimal solutions.

This step-size gets updated in a way to shift the scale towards better solutions. This update is done based on the previous solutions, which are kept in an evolution path. This path saves the steps that were made in previous iterations, with their corresponding length and direction.

There are three cases in which the evolution path may be present (Figure 2.1). These three cases are compared to an expected length, which is the result of independently taken, and therefore uncorrelated, steps (Figure 2.1, middle). It means that the same ground cannot be covered by bigger or smaller steps. If step are correlated, they point in the same direction, and can be replaced by one large step (Figure 2.1, right). If steps are uncorrelated, they cancel each other out, and have to be replaced by smaller steps to prevent that (Figure 2.1, left).

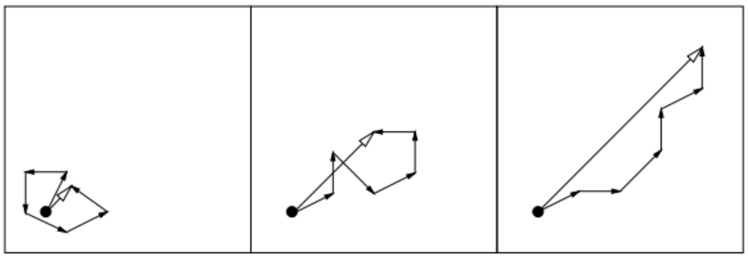


Figure 2.1: Three evolution paths (successive hollow-tipped arrows) with six equally-sized steps (bold-tipped arrow) but drastically different path lengths (image from [5]).

- The first case is where the length of the evolution path is short (Figure 2.1, left). The single steps cancel each other out, meaning that the algorithm is close to a potential solution. It has narrowed down a specific region which has to be explored by small steps. In this case σ has to be decreased.
- The second case is where the length is in the desired situation (Figure 2.1, middle). The steps are uncorrelated. The steps cannot be replaced by larger or smaller steps. In this case σ can stay the same.
- The third case is where the length if long (Figure 2.1, right). The single steps are all somewhat in the same direction. Rather than using small steps to cover the ground, one large step would be more efficient. In this case σ has to be increased.

This scheme is used to update σ. This natural language can be converted in to the formula:

$$\ln \sigma^{(g+1)} = \ln \sigma^{(g)} + \frac{c_\sigma}{d_\sigma} \left(\frac{||p_\sigma^{(g+1)}||}{E||\mathcal{N}(\mathbf{0}, \mathbf{I})||} - 1 \right) \tag{2.5}$$

Here, $p_\sigma^{(g)}$ is the evolution path at generation g, making $||p_\sigma^{(g+1)}||$ is the expected path for the next generation. This gets compared to $E||\mathcal{N}(\mathbf{0}, \mathbf{I})||$, the Euclidean norm of the $\mathcal{N}(\mathbf{0}, \mathbf{I})$, which will result in the expected, desired, path length. Then there is the learning rate for σ, c_σ (< 1), and a damping parameter d_σ (≈ 1). Together, they dictate how drastically σ gets updated each iteration.

It can be seen that if the current path, and expected path are the same, the new sigma will be the same as the old one. Furthermore, if the path length is smaller, $\sigma^{(g+1)}$ will be smaller. If the path length is longer, $\sigma^{(g+1)}$ will be larger.

2.1.2 CMA-ES for minimising the diagonal of a rectangle

Now that the theoretical base for CMA-ES has been established in the previous sections, a simple implementation can be made. The goal of this implementation will be solving the rectangle toy-problem from the introduction. This toy-problem of course has an analytical solution, $a = b = 10$, but still will help illustrate how CMA-ES works, and prepares us to take on a real problem later on.

First, CMA-ES needs an objective function. In this case it is simply 100, which was the area constraint, minus the current area $a \cdot b$. This gets squared to lay emphasis on the difference between the constraint and actual values. Additionally, we want to take into account the diagonal that was to be minimised. A larger diagonal should give a higher error, and vice versa.

The diagonal denoted as D can simply be calculated using the Pythagorean Theorem, $D = \sqrt{a^2 + b^2}$. This results in the following formula.

$$Error = (100 - (a \cdot b))^2 + D \tag{2.6}$$

Given an initial coordinate $(x_0,\ y_0)$ and initial standard deviation σ_0, CMA-ES will find the parameters that minimize this objective function.

After 100 iterations, CMA-ES has solved this minimisation problem, giving $a \approx b \approx 10$, with the minimal error being $14.14 = \sqrt{10^2 + 10^2}$, or the smallest diagonal possible for an area of 100 (Figure 2.2).

In the beginning of the optimisation process, the range of parameter values that are being generated have a considerable wider range than later on in the process. This implies that the σ values for both parameters are still relatively large as compared to the end of the process, and CMA-ES is still highly shifting the mean each iteration. This fact is confirmed by the bottom two graphs of describing the course of the σ values.

Although CMA-ES performs well for this toy-problem, it certainly was not a challenge to begin with. In the next section a BABM will be introduced that will be optimised by CMA-ES. This BABM will be used throughout this thesis to simulate problems of other, more complicated, BABMs.

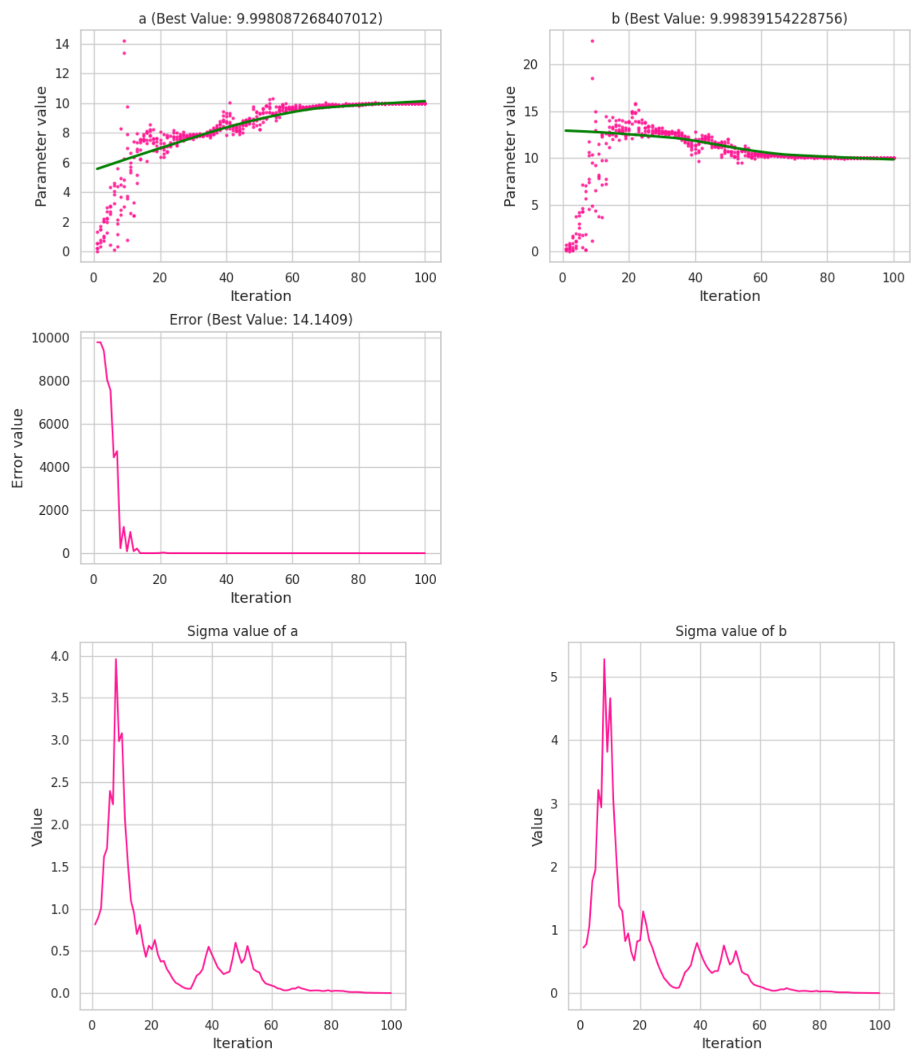


Figure 2.2: CMA-ES correctly fitting the parameters a and b, getting the most optimal solution for the rectangle toy-problem. The top two graphs show how the values of a and b change over-time. Here, the green lines denote the current mean CMA-ES has for the parameters. It shows that the mean for both parameters settled around the value 10. From that mean, using the deviation σ, new solutions will be generated, as shown by the pink dots. The green lines in the top-two plots is generated using locally weighted scatterplot smoothing (LOWESS). Each iteration, $\lambda = 6$ solutions are generated, and $\mu = 3$ solutions are take in to account to calculate the new mean. The middle graph shows the value of the objective function dropping as the iterations go on. The bottom two graphs show the course of the σ value for the parameters. Upon finding the best mean value, it will lower the sigma as described in the previous section. Seemingly, this happens around iteration 60. From there, σ will get as close to 0 as possible to get the best solution for this problem.

2.2 The Beauchemin model

The BABM that will be studied and fitted in this thesis is the Beauchemin model. A model that describes T-Cell movement in the lymph node as proposed by Beauchemin et al. [1].

This model itself is not intrinsically interesting to be fitted. Namely, it has an analytical solution, meaning that given a certain output, the corresponding input can simply be calculated [2]. Still, this model is great for sketching how CMA-ES will tackle certain problems that other, more complex, BABMs face. It can be used as a tool to simulate problems without having to do the costly computations that come with other BABMs.

Thus, while fitting the Beauchemin model is not the greatest challenge by itself, it can be used as a device to predict how CMA-ES will handle more complex and costly BABMs.

2.2.1 The parameters of the Beauchemin model

In the Beauchemin model, cells alternate between moving and pausing. These get dictated by the three parameters of the model.

Firstly, t_{free} dictates the time a cell receives to perform a straight-line walk. Secondly, v_{free} denotes the speed at which the cell performs the random walk. Finally, t_{pause} defines how long a cell may rest between the straight-line walks. Within this pause, the cell randomly picks a new direction, from there on repeating this process until a certain time limit gets reached.

These three parameters will be of main interest in this thesis, as the goal is to fit simulated tracks (Figure 2.3) that closely resemble target data of interest. To judge if a simulation output "matches" the target, a properly defined objective function is necessary.

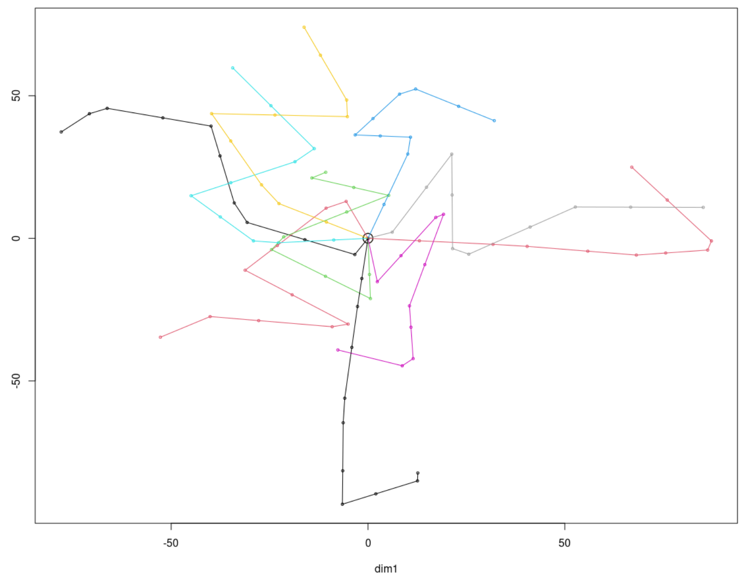


Figure 2.3: To illustrate what the Beauchemin model generates, a simulation of 10 cells/tracks was executed. Every coloured line resembles the path that an individual cell has travelled. These paths were simulated using the parameters $t_{free} = 2.0$, $v_{free} = 18.8$ and $t_{pause} = 0.5$

2.2.2 Using the difference between mean square displacements as the objective function for fitting the Beauchemin model

To optimize and find the target parameters, the optimizer needs an objective function that can be minimized. For the Beauchemin model, the difference between mean squared displacements of the simulated and target data can be used. Namely, the mean squared displacement is a commonly used method for describing cell trajectories.

The mean squared displacement (MSD) denotes the mean deviation of, in this case, a cell from a certain reference point at a certain time t. The parameters discussed in the previous section influence how a cell migrates over time, and thus influencing the MSD. For each of these tracks that a cell creates, subtracks can be created of a certain length t. For every subtrack, the deviation can be calculated. The result is a curve that can be used in the objective function (Figure 2.4).

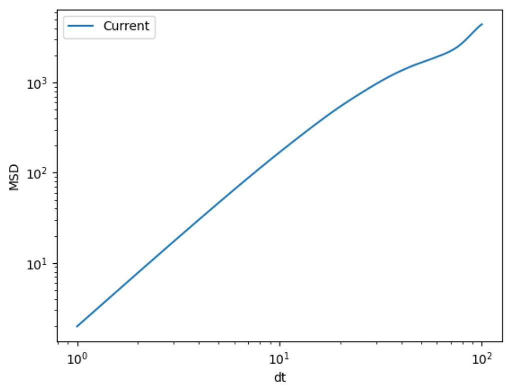


Figure 2.4: An MSD curve for the Beauchemin model generated with parameters $t_{free} = 2.0, v_{free} = 18.8$ and $t_{pause} = 0.5$.

These curves of a Beauchemin output can be compared to the target, by comparing MSD values at several times dt, giving the following equation:

$$\Delta_{MSD}(t) = MSD_{target}(t) - MSD_{current}(t) \tag{2.7}$$

This $\Delta_{MSD}(t)$ gets calculated, and subsequently squared to emphasize the difference, at multiple times $(t_0, t_1, ..., t_n)$. Finally, the mean of these squared values is calculated:

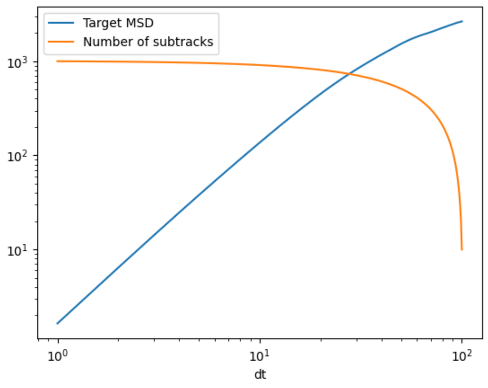
$$Error = \frac{1}{n} \sum_{t=0}^{t_n} (\Delta_{MSD}(t)^2) \tag{2.8}$$

A problem that the MSD of the Beauchemin model experiences is that the MSD values at a low t (e.g. 5) are significantly smaller than at a later t (e.g 6000). This difference in scale results in the later values overshadowing the nuances of the smaller values. To prevent this from happening, the MSD values first will be logged before calculating the difference Δ_{MSD} to bring the values to the same scale. Changing the previous formula to calculate $\Delta_{MSD}(t)$ (Equation 2.7) to:

$$\Delta_{MSD}(t) = \log(MSD_{target}(t) + 1) - \log(MSD_{current}(t) + 1) \tag{2.9}$$

Additional to the log, a '+1' is there to prevent negative values.

Another challenge that arises while using MSDs, is that MSDs calculated at later times are based on less data compared to MSDs at earlier times. This data in our case is the number of subtracks of a track generated by a cell (Figure 2.5).



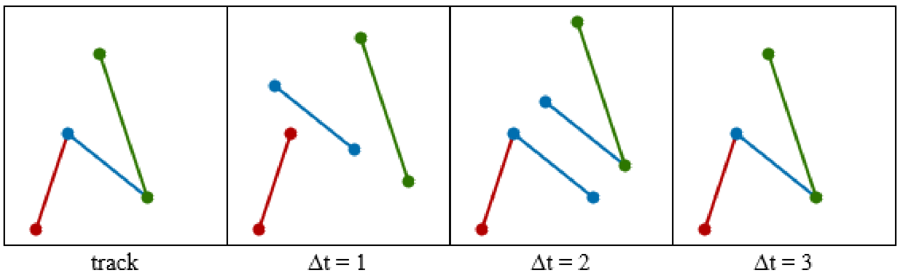


Figure 2.5: The top graph shows the number of subtracks vastly drops for MSD values calculated at higher Δt values, hence those later MSDs can be considered less accurate than MSDs calculated at lower Δt values. The bottom figure shows the number of subtracks that can be made from the 'track' (left) is lower at a higher Δt. Higher Δts yield longer subtracks that consist of shorter subtracks with lower Δts. The lower the Δt, the more combinations of subtracks there are. There is only one subtrack that can be constructed from 'track' with $\Delta t = 3$, while there are three that can be made with $\Delta t = 1$.

When there is a relatively high amount of subtracks for a certain Δt, the MSD will be calculated with more data, and will thus be more accurate compared to MSDs calculated with less subtracks. To value the MSDs that were calculated with more subtracks, weights are introduced.

$$w(t) = \frac{subtracks_{target}(t) + subtracks_{current}(t)}{2} \tag{2.10}$$

The weight can simply be the number of subtracks available at a certain time t. Because two simulations are being compared, the average number of subtracks available between those simulations at a certain time will be used (Equation 2.10).

To calculate the final error, the mean of these squared and weighted $\Delta_{MSD}(t)$ values is calculated:

$$Error = \frac{1}{n} \sum_{t=0}^{t_n} (\Delta_{MSD}(t)^2 \cdot w(t)) \tag{2.11}$$

This formula is used to calculate the error between two MSD curves and judge their similarity, with the goal to fit the Beauchemin model (Figure 2.6).

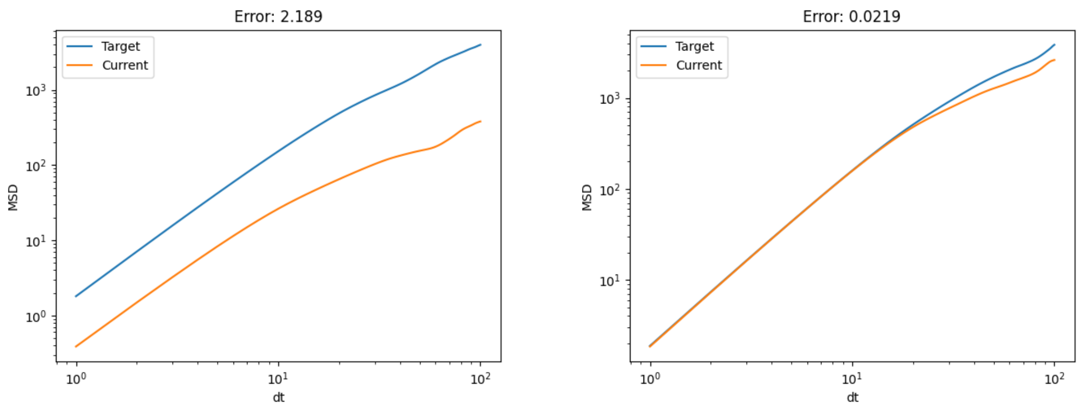


Figure 2.6: log-log plot of two MSDs of Beauchemin simulation. Left, the 'Target' curve was created using the parameters $t_{free} = 2.0$, $v_{free} = 18.8$ and $t_{pause} = 0.5$, and the 'Current' curve with the parameters $t_{free} = 1.0$, $v_{free} = 9.4$ and $t_{pause} = 0.5$ resulting in a relatively high error. Right, it shows two MSD curves created with the exact same parameters, $t_{free} = 2.0$, $v_{free} = 18.8$ and $t_{pause} = 0.5$, consequently having a very low error.

Now, having this objective function, all three requirements as introduced in this section are satisfied. The Beauchemin model is ready to be optimised using CMA-ES, with the MSE of the MSD as objective function. In the next section, this will be put to work.

Chapter 3

Research

3.1 Stochasticity in the Beauchemin model

A major problem when fitting BABMs, is that many of these models are stochastic. Stochasticity entails that even with the same input for a model, the output can still be different. For the Beauchemin model, this means that simulations with the same t_{free}, v_{free} and t_{pause} may still simulate different tracks (Figure 3.1).

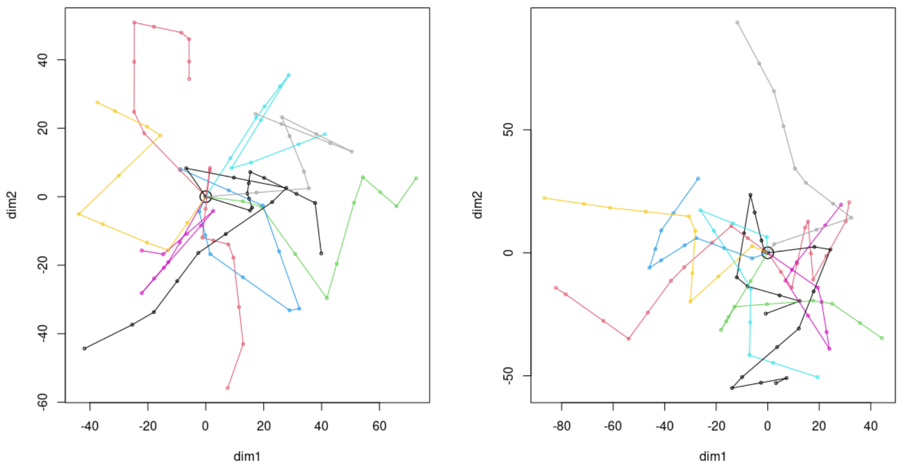


Figure 3.1: Two Beauchemin simulations generated with the same parameters, $t_{free} = 2.0$, $v_{free} = 18.8$ and $t_{pause} = 0.5$.

A model being stochastic makes it difficult to determine the solution to an optimization problem. Even if CMA-ES stumbles on the precise parameters by which the target simulation was generated, there most certainly remains a discrepancy between the outputs.

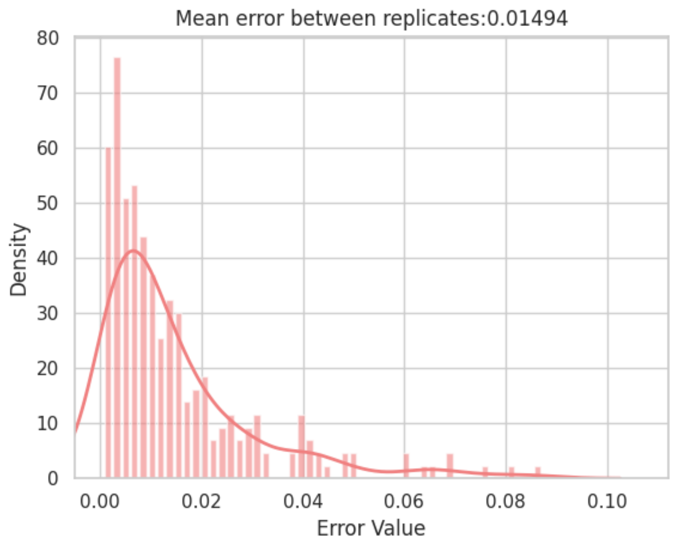


Figure 3.2: A normal distribution of errors of 250 Beauchemin simulation runs, all with the same initial parameters t_{free} = 2.0, v_{free} = 18.8 and t_{pause} = 0.5. This shows the discrepancies between runs with the same input data, and stresses the difficulty when fitting this model.

CMA-ES makes a distribution of where it expects to find good solutions, with a certain deviation σ. With stochastic models, it will result in this distribution ending with a broader range, in other words having a higher σ. Where it is expected that non-stochastic problems with a well-defined error function, like the rectangle problem, end with a σ that is practically 0. It would be an anomaly if the optimisation process for the Beauchemin model would as well.

Besides the noise created by the output of the Beauchemin model, CMA-ES is noisy by itself. Particularly, CMA-ES stochastically samples solutions from its distribution. This inherent stochasticity has a part in the final noisy distribution of error as well, as the algorithm will likely find different solutions each run.

3.2 Fitting the Beauchemin model

As mentioned, the Beauchemin model has three parameters of interest, t_{free}, v_{free} and t_{pause} (Section 2.2). The goal in this section is to retrieve those based on a certain dataset.

Instead of using an actual dataset, for which the parameters are then unknown, we can simulate a new dataset for each optimisation run using target parameters to establish a ground truth. In that way, the target parameters can be directly compared to the parameters obtained by a CMA-ES optimisation run.

In a real-world scenario, this is of course not the case. One would only have dataset where the parameters have to be retrieved for, and no ground truth. This luxury gives us the opportunity to test if CMA-ES is functioning properly, and able to find back these parameters based on the error function as defined in chapter 2.

We will start by fitting a single parameter, as this should be straightforward for CMA-ES (wishful thinking).

3.2.1 CMA-ES correctly fitting a single parameter

To test whether CMA-ES is able to fit the Beauchemin model, we start with the simplest case of fitting a single parameter. The parameter that will be fitted in this experiment is t_{free}, with a goal value of 2.0.

CMA-ES is ran for 100 iterations to generate the final result.

The algorithm is able to get very close to the target value of 2.0 with the final solution being 2.05 after 100 iterations (Figure 3.3). As has been established, the stochasticity makes it difficult to retrieve the exact target value (Section 3.1).

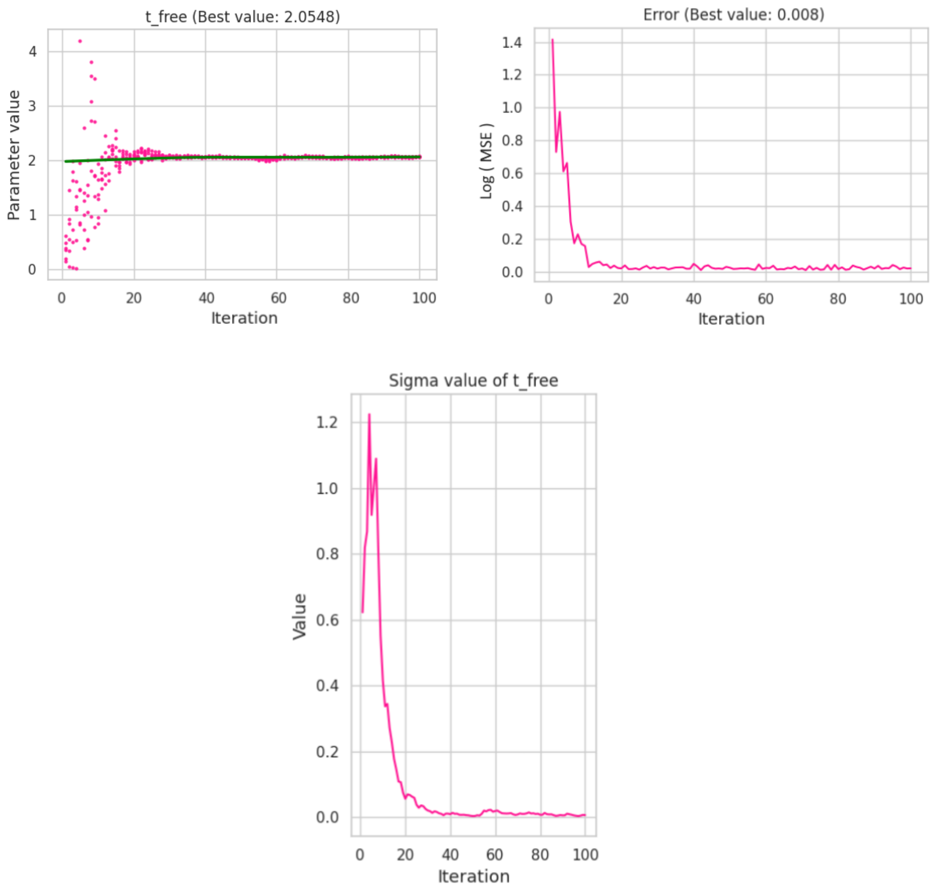


Figure 3.3: Results of CMA-ES fitting a single parameter t_{free} with an initial point $(x_0) = (0.1)$, and an initial $\sigma_0 = 0.5$. The value of λ is set to 6, and the value of μ to 2. Both the error value (top-right) and σ (bottom) quickly drop to around 0, implying it has found the correct mean. This is also reflected in the parameter value stabilising at 2 after around 20 iterations (top-left).

For good measure, another CMA-ES run gets initialized The target value for t_{free} will now be 18.5.

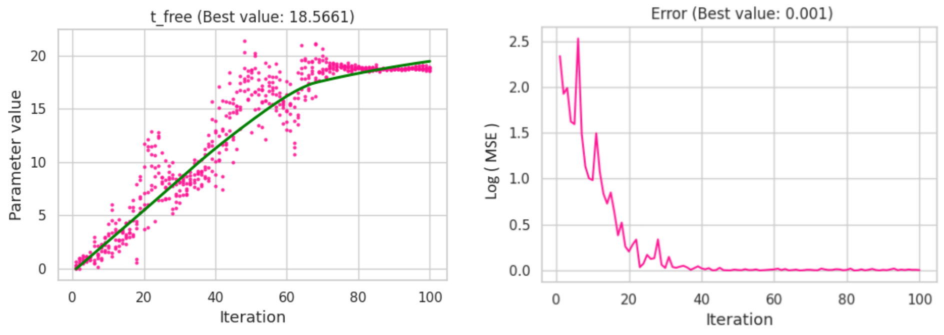


Figure 3.4: Results of CMA-ES fitting a single parameter t_{free} with target $t_{free} = 18.5$. The initial point $(x_0) = (0.1)$, and the initial $\sigma_0 = 0.5$. The value of λ is set to 6, and the value of μ to 2. CMA-ES is able to find the target value with within 100 iterations.

CMA-ES is again able to find the parameter value with great accuracy (Figure 3.4). It is expected that the further a target value is from the initial value, the longer it will take the algorithm to obtain the value. This can be explained by the fact that CMA-ES first has to find the correct mean of the search space. The further the search space is, the longer it will take to reach it.

This begs the question that, if targets are extremely far away from the initial coordinate, can they still be found? This scaling issue is explored in section 3.4.

To outline how CMA-ES performs over several runs, the same optimisation process is ran for 20 times, all with the same target value for t_{free} being 2.0.

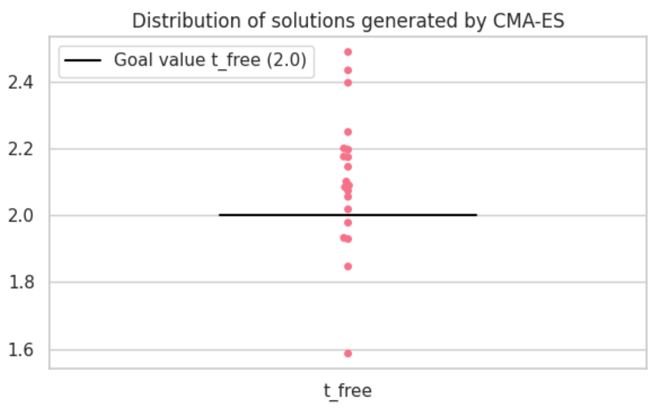


Figure 3.5: Distribution of results of 20 CMA-ES runs fitting the single parameter t_{free} with target $t_{free} = 2.0$. For every run, a new target is simulated. The distribution does show some variation between solutions, though most are within 0.2± of the target value.

The variance shown in the distribution (Figure 3.5) are caused as a result of two reasons, both coming down to stochasticity. First the stochasticity in the Beauchemin model itself. Solutions with other values than $t_{free} = 2.0$ can still generate the same output resulting in a low error. Additionally, CMA-ES is an evolution strategy, which by definition is stochastic. This means that CMA-ES will find different solutions from run-to-run. Sometimes, it will find better solutions than previous runs, and sometimes it will find solutions that are worse.

In conclusion, the results do show variance between solutions, but this is expected due to stochasticity. On average, CMA-ES is able to find the correct value with a fairly low error, indicating it was able to match the MSD curve of the target.

3.2.2 CMA-ES fitting two parameters shows discrepancies between the target and actual values

Now that it has been shown that CMA-ES is able to fit a single parameter of the Beauchemin model with fair accuracy, we move on to fit two parameters.

This is expected to be more difficult, as both parameters influence the final outcome of the simulation. Hence, CMA-ES has to find the balance between the parameters, not favouring fitting one over the other.

The two Beauchemin parameters that will be fitted are t_{free} and v_{free}. The process was ran 100 iterations to obtain the final result.

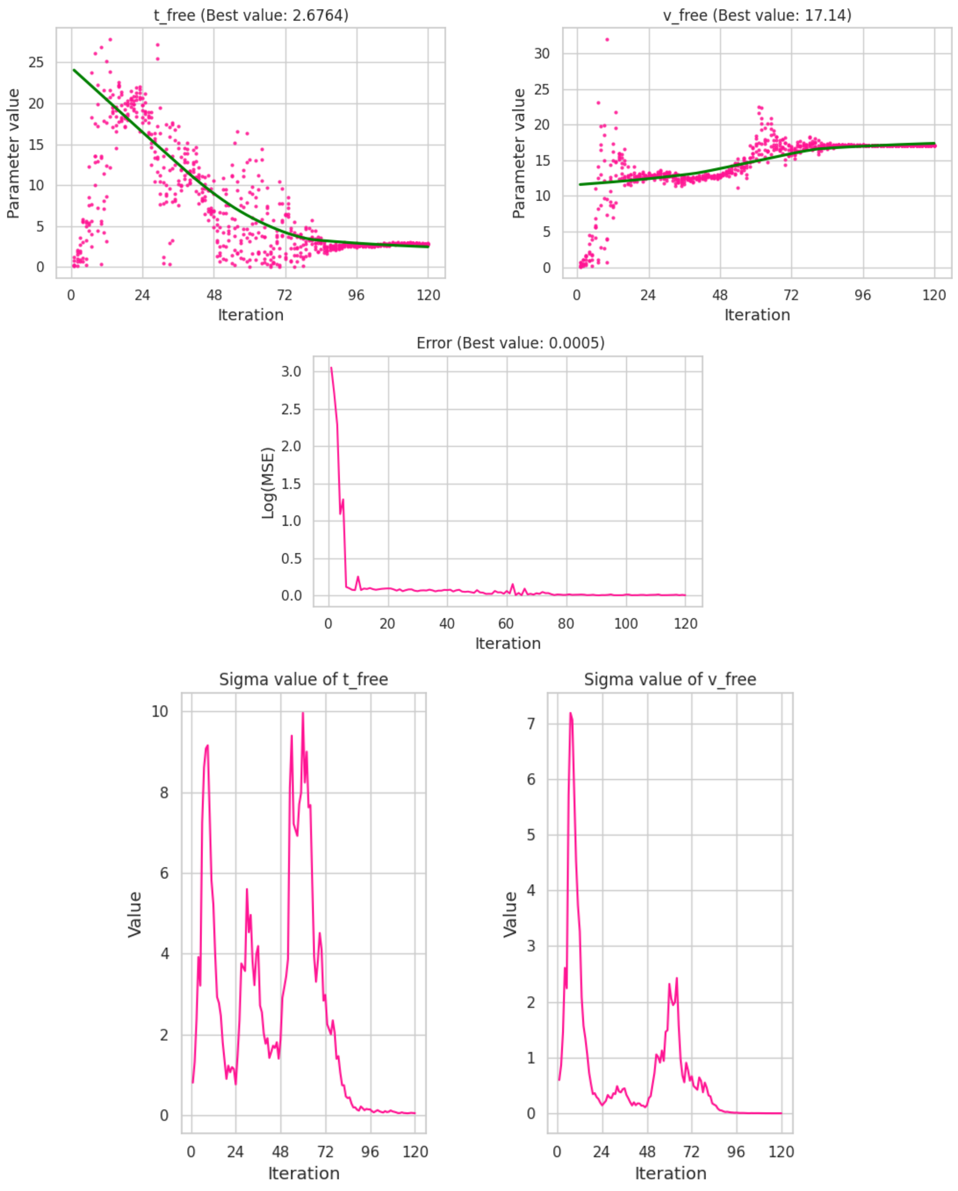


Figure 3.6: CMA-ES simulation with $(x_0, y_0) = (0.1, 0.1)$, $\sigma_0 = 0.5$, $\lambda = 6$, and $\mu = 3$. It finds values for the two fitted parameters, with an error of 0.0005 compared to the target data, that was simulated with parameters $t_{free} = 2.0$ and $v_{free} = 18.8$. The σ values for both parameters show fluctuation, indicating difficulty finding the exact region of solutions. Later in the process they do drop down and stabilise, as also reflected in the parameter values that finally settle to generate the final result.

The final values of this optimisation run are not far off the target values (Figure 3.6), though still some room to the actual target parameter values remains.

The difference in MSDs almost completely disappeared during the optimisation process, in line with the small error (Figure 3.7). Still, some small differences remain, likely due to the stochasticity of both CMA-ES and the model.

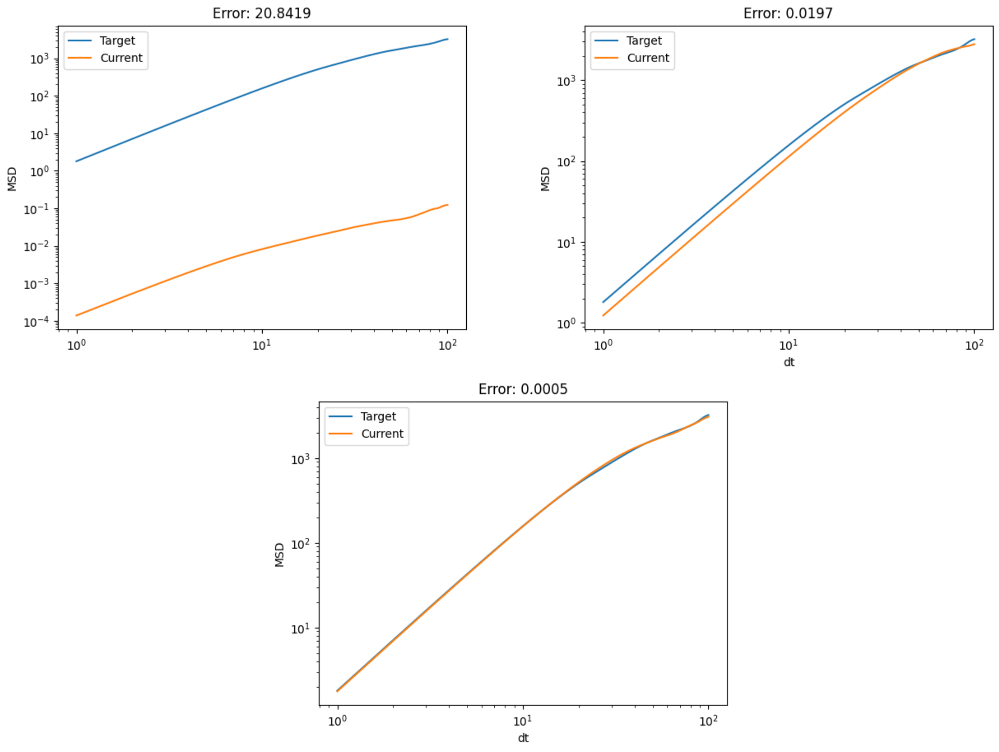


Figure 3.7: The MSD curves of the target simulation and a current simulation. Top-left illustrates the MSD curve of a solution in the first iteration being far off the target. Top-right shows an MSD curve which was created in iteration 60, already being significantly closer to the target. Finally the bottom shows the MSD curve of the best solution found, with the values $t_{free} = 2.6764$ and $v_{free} = 17.14$.

This also shows that even with a solutions that does not precisely match the target parameters, the error can still be extremely low. Again, caused by stochasticity. The solutions generated almost the exact MSD curve as the target, even though the parameter values were not the same.

To test the robustness of the algorithm, a distribution of parameter values of 30 CMA-ES optimisation runs with the same target values, $t_{free} = 2.0$ and $v_{free} = 18.8$ was created (Figure 3.8).

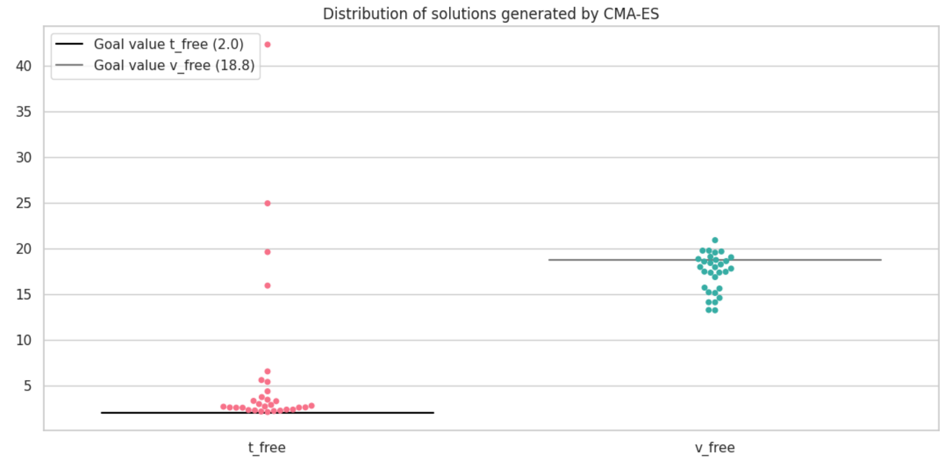


Figure 3.8: Distribution of results of 30 CMA-ES runs, fitting the two parameters t_{free} and v_{free} with target $t_{free} = 2.0$ and $v_{free} = 18.8$. The values go in the right direction, but there are still some outliers. This indicates that the current implementation is not robust enough to reliably get good solutions.

The values for both parameters look to go into the right direction, but several outliers remain. This is especially the case for t_{free}. This indicates that the current method is not robust enough.

A first thought as to where it might be going wrong, could be that the algorithm has not converged yet at the end of the run. Still, it reached the maximum iteration of 100 and prematurely terminated.

Looking at a specific run where it underestimated v_{free}, it shows the the parameters did stabilise to a fixed value towards the end of the run (Figure 3.9, top two graphs), indicating the algorithm had converged.

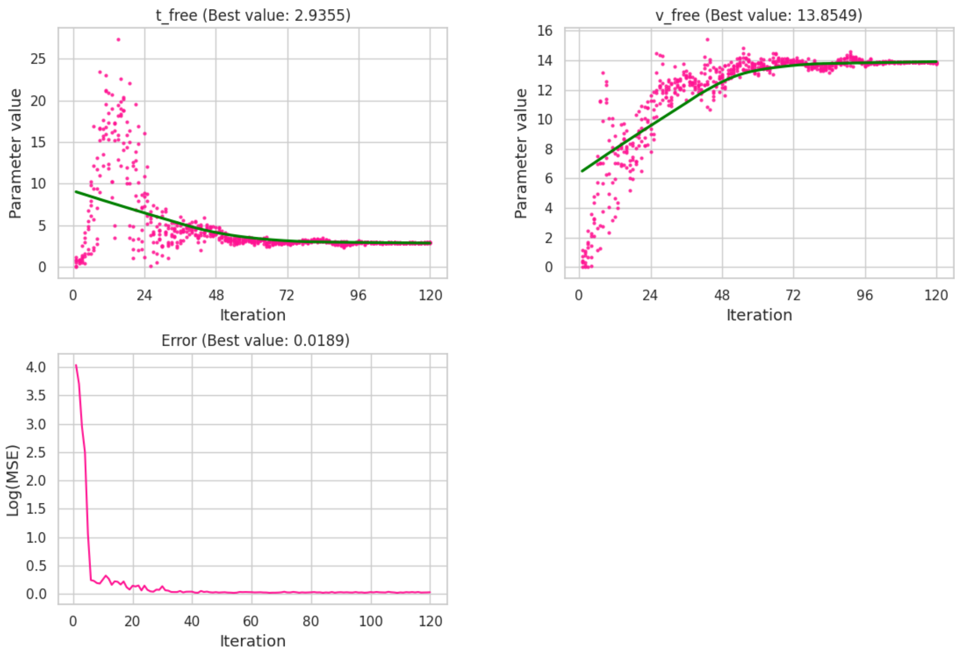


Figure 3.9: Results of CMA-ES fitting the two parameters t_{free} and v_{free} with targets $t_{free} = 2.0$ and $v_{free} = 18.8$. It underestimated v_{free} in this case. The value for both parameters has converged at the end of the run, indicating that the process was finished, and these were the best values CMA-ES could find.

A second thought could be that CMA-ES settled down for a solution too quickly. The error value 0.0189 (Figure 3.9, bottom) seems low, but compared to the previous run where the solution is closer to the target with a corresponding error of 0.0005, it is significantly higher. This indicates that CMA-ES is settling for less. It narrows down the search space too early, resulting in sub-optimal solutions. The idea of opening up this search space again is explored in section 3.3 by the introduction of CMA-ES restart strategies.

To summarize, CMA-ES was able to fit two parameters of the Beauchemin model to go in to the right direction. Still, there remained variance between the solutions and target. Some variance is expected due to stochasticity, but looking at the results for bad runs, the error is significantly higher, possibly indicating a problem in the current strategy.

3.2.3 CMA-ES unable to fit three parameters for the Beauchemin model

The results of fitting two parameters for the Beauchemin model shows promise, as the values go in to the right direction. The main issue that remains is robustness.

To build further upon this, we tried fitting three, or all, Beauchemin parameters. Here, the robustness is evaluated again, to see any shifts as compared to fitting two parameters.

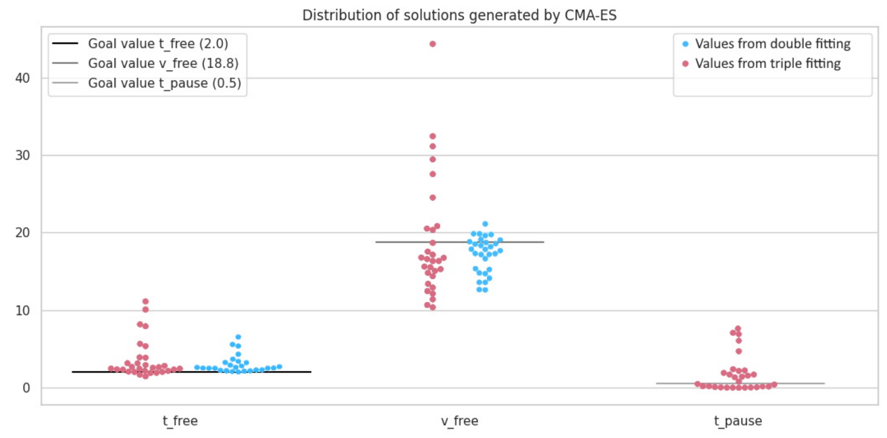


Figure 3.10: Distribution of 30 CMA-ES runs, all with $(x_0, y_0, z_0) = (0.1, 0.1, 0.1)$, $\sigma_0 = 0.5$, $\lambda = 6$, and $\mu = 3$. The three target parameters were $t_{free} = 2.0$, $v_{free} = 18.8$ and $t_{pause} = 0.5$. Side-by-side with the solutions generated by fitting two parameters. While the solutions seem to go in to the right direction for t_{free} and t_{free} they both have a lot more outliers compared to fitting two parameters. Moreover, the range of solutions for both v_{free} and t_{free} has broadened. Then, the range of values for t_{pause} is also very wide. This shows that fitting three parameters of the Beauchemin model is not very accurate.

The solutions generated by CMA-ES for the triple-parameter fitment show more variance as compared to fitting two parameters (Figure 3.10). This indicates that fitting three parameters is less robust than fitting two parameters, and not very accurate.

From these results it can be concluded that CMA-ES is not able to accurately fit all of the three Beauchemin parameters.

This finding is consistent with the finding of Textor et al. that there are infinitely many combinations of t_{pause} and v_{free} that output the same MSD for a fixed t_{free} [2].

Thus, using the mean square displacement for fitting the Beauchemin model, it is infeasible to accurately fit these three parameters. Consequently, two parameters will be fitted for them rest of the experiments.

3.3 Restart strategies for CMA-ES

A problem that arose while trying to fit two Beauchemin parameters, is that the values for σ narrowed down too early. Consequently, CMA-ES settles for a sub-optimal solution. To prevent this from happening, CMA-ES could be restarted where the previous run left of, as proposed by Auger et al [6]. The σ_0 for the restart will be the original σ_0 of the first run, 'opening up' the search space again from a new initial coordinate that is the solutions from the previous run.

3.3.1 Methods for restarting CMA-ES

Two CMA-ES restart strategies will be reviewed in this section, Restart-CMA-ES and IPOP-CMA-ES.

The first strategy, Restart-CMA-ES, entails simply starting CMA-ES again with the initial values being the solutions of the previous run. The σ_0 for each restart, is the σ_0 from the initial run.

Building forth upon Restart-CMA-ES, IPOP-CMA-ES (Increased POPulation) is introduced [6]. Here, CMA-ES starts searching again from the solution obtained from the previous run, while also increasing the population size λ by a factor 2, as in line with the literature. This increases the chance of CMA-ES of finding good solutions, as there are more candidates being generated.

Both strategies will serve the purpose of 'opening up' the search space again. Where the previous ranges ended, the next CMA-ES iteration started from, finding more optimal solutions.

3.3.2 Regular CMA-ES, Restart-CMA-ES and IPOP-CMA-ES compared for fitting two parameters of the Beauchemin model

The two restart strategies as discussed were applied to the optimisation problem of fitting two Beauchemin parameters, and compared to regular CMA-ES.

Restart-CMA-ES shows a shift to a lower overall distance from solutions to the target compared to regular non-restart CMA-ES. IPOP-CMA-ES shows a similar shift compared to Restart-CMA-ES.

IPOP-CMA-ES performed the best, as its solutions are generally closest to the target. At the same time, due to the increased population size the time consumption was the greatest as well. It increases with $\mathcal{O}(n^2)$ for each restart, as every λ in the population needs a function evaluation. Whilst the population size is exponentially increasing, the time spent will be as well.

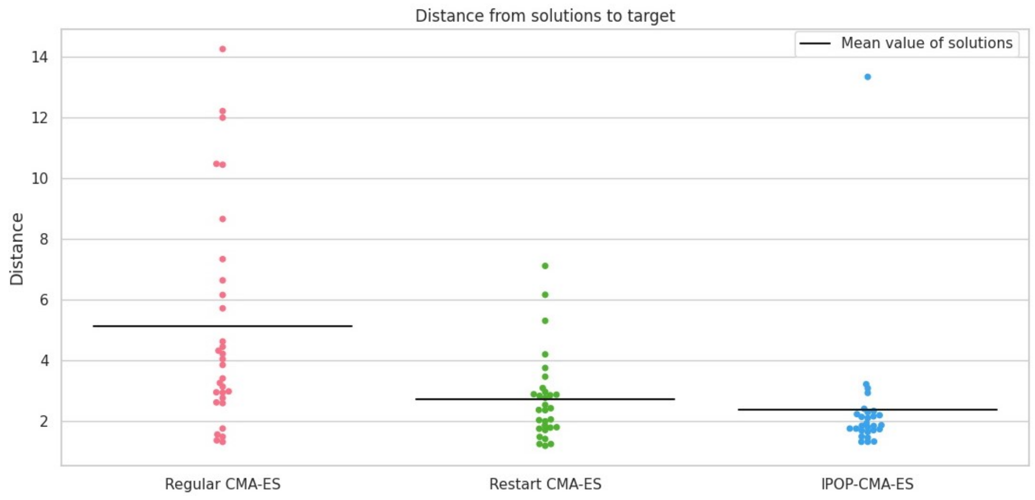


Figure 3.11: Distances from solutions to target coordinate (2.0, 18.8) for three CMA-ES strategies compared. All run with $(x_0, y_0) = (0.1, 0.1)$, $\sigma_0 = 0.5$, $\lambda = 6$, and $\mu = 3$. It shows IPOP-CMA-ES generates the solutions closest to the target, but only marginally better compared to Restart-CMA-ES. Both Restart strategies outperform the regular non-restart CMA-ES in generating solutions close to the target parameter.

For Restart-CMA-ES, the time spent will only increase $\mathcal{O}(n)$ for every restart, as λ does not change.

Summarizing, restart strategies show a great increase in performance compared to regular CMA-ES. Of course this performance comes with a time-consumption penalty, that is the most drastic with IPOP-CMA-ES.

From here on, Restart-CMA-ES will be used for optimisation in this thesis. The reason being better performance as compared to regular non-restart CMA-ES, but not as huge of a performance penalty as compared to IPOP-CMA-ES.

3.4 Scaling problems in ABMs

Biological agent-based models often have multiple parameters that direct their output. These parameters can have varying ranges that are drastically apart from each other. This makes optimisation very hard as opposed to dealing with parameters that live in the same region.

To go back to the rectangle problem, this problem can be simulated by the idea that one side would be measured in kilometres, whereas another side is measured in millimetres. As has been established, the ideal solutions for that problem was to create a rectangle of 10 x 10 metres. Using those other scales, the rectangle would be 0.01 (kilometres) X 10.000 (millimetres), yielding the solution $a = 0.01$ and $b = 10000$. Such great divergence in scales can occur in BABMs. Hence, we tested how CMA-ES responds to such scaling issues in the model to be fitted.

3.4.1 CMA-ES unable to identify linearly scaled parameter values

To examine if CMA-ES is able to handle parameters that exhibit these scaling problems, two parameters of different scales are optimised. The values will not simply be chosen to lie far apart. Instead, the same target values $(2.0, 18.8)$ of previous experiments will be used, where the first parameter gets scaled using a wrapper function. In this way, the experiments can be directly compared, as the Beauchemin parameters, and thus the target, remain the same. The only penalty that the scaled parameters receive is the distance between the starting point and the target, as those are of course further away.

Restart-CMA-ES will be used to fit the parameters. To scale the parameters a division is used. Three scaling values, 10, 100 and 1000, are compared to establish the influence.

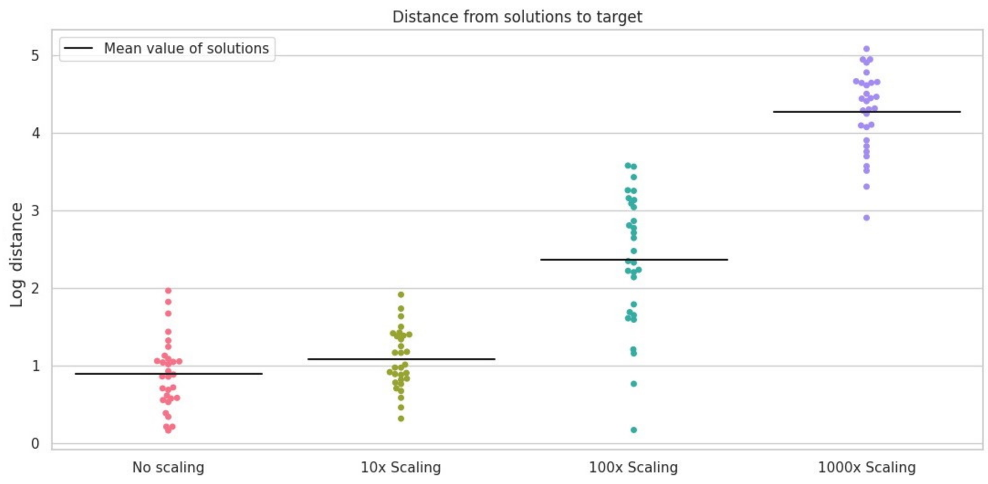


Figure 3.12: Restart-CMA-ES fitting differently scaled parameters. This process is run for 2 restarts with $\sigma_0 = 0.5$ for each run. The initial coordinate (x_0, y_0) is set to $(0.1, 0.1)$, with $\lambda = 6$ and $\mu = 3$. The distances of the solutions to their target coordinate increase as scaling becomes more extreme. This shows that parameters having vastly different scale is a problem for CMA-ES. The more extreme the scaling difference is, the further away the solutions will be from the target.

When the extremity of scaling increases, the worse the solutions generated by CMA-ES will become (Figure 3.12).

This may be caused by the fact that the target coordinate is increasingly further away, resulting in sigma increasingly being 'too small'. The solution is not in the search space, even after several σ-updates. In contrast, the solution for the other parameter that is of lower scale, is being found in the search space after a few updates. This will mean that the error is getting lower, resulting in the sigma lowering, consequently a more narrow search space and CMA-ES ultimately settling for a sub-optimal solution.

Although tempting, σ_0 cannot simply be made larger to solve this. It requires upfront knowledge about the solution to set the σ_0 in a way that the solution can be reached within a few iterations, which is simply not there in a real-world scenario. Trivially, if σ_0 would be infinitely large, the solution will always be in the search space. Doing this though, will make it very hard for CMA-ES to narrow down the final search space, again resulting in not being able to find the precise solution.

The same reasoning goes for the initial coordinate. If the initial coordinate is closer to the target, the probability of CMA-ES finding the solution quicker is simply higher. Again, up front knowledge about the solution is necessary to choose a good initial coordinate, which is not always available.

3.5 Rescaling variables

An idea to counteract these scaling problems, is to re-scale the parameters of this problem in its entirety. This will enforce parameters to be on the same scale, even if not originally the case. Thus, giving the possibility of the initial coordinate and σ_0 being favourable for both parameters.

A simple logarithmic scaling function can be used, as proposed here [7] (Equation 3.1). This will map most values to be in the range [0, 10], where previously the range [0, 10^{10}] was necessary for that.

$$f(x) = 10^x \tag{3.1}$$

This function will be called before calling the objective function. This would mean that if the target parameters of a Beauchemin simulation would be $t_{free} = 20000.0$ and $v_{free} = 18.8$, CMA-ES would have to find $\log 20000.0 = 4.3$ and $\log 18.8 = 1.27$. As opposed to the previous scaling experiments, these target values are feasible to be in the search space after a few iterations.

The same scaling experiments are executed again, but now using the rescaling function.

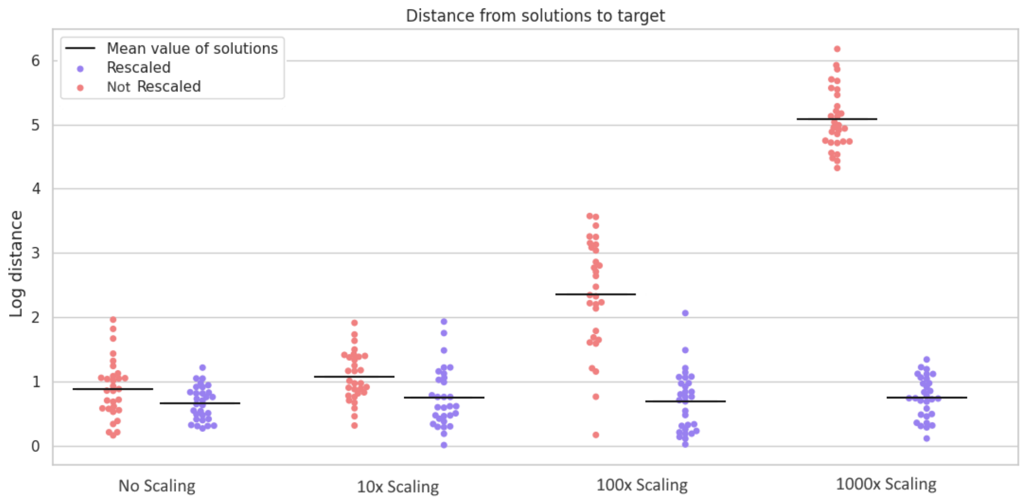


Figure 3.13: Restart-CMA-ES fitting differently scaled parameters compared to scaled parameters after problem rescaling. This process is run for 2 restarts with $\sigma_0 = 0.5$ for each run. The initial coordinate (x_0, y_0) is set to $(0.1, 0.1)$, with $\lambda = 6$ and $\mu = 3$. Rescaling the problem so that the parameters are on the same scale shows the distance between the generated solutions and the target remain the same for each of the scaling runs. Even for the base run, where the target parameters are relatively close to each other, the solutions generated after rescaling improve. Between the rescaled solutions, the mean is almost the same for different scales, indicating no performance penalty if the parameters live on different scales. Hence, it can be concluded that rescaling the problem is a great solution to counter parameter that live on vastly different scales, or even improve performance on problems where these scale differences are not huge.

Problem rescaling proves to have a giant impact on the performance of CMA-ES (Figure 3.18). For BABMs where it is expected that parameters are on different scales, it can be used to improve performance. Moreover, it even improved performance in the 'No Scaling' run where parameters where not far of each other. This can be explained by the fact that the solutions $t_{free} = 2.0$ and $v_{free} = 18.8$ are further apart from each other than the rescaled values $t_{free} = \log 2.0 \approx 0.3$ and $v_{free} = \log 18.8 \approx 1.27$.

When target values are closer to each other as well as to the initial coordinate, the correct values for both parameters can be reached within only a few iterations. From there on, CMA-ES only has to narrow the search space.

Besides, the deviation from the target values gives a greater penalty in the rescaled version. Whereas a positive deviation of 2 from t_{free} would give

4 in the non-rescaled version, it would give $10^{0.3+2} = 200$ in the rescaled version. This will quickly make the σ values drop and get a very narrow search space. If the target value was 2000, it would require an enormous amount of iterations to even get close, if searching started from 0.1.

In conclusion, rescaling the problem shows a great improvement in solutions, even for the base run where the parameters were not far of each other. If rescaling will narrow the range of solutions, it is expected to improve the performance, as the solutions for both parameters are to be reached in only a few iterations.

Chapter 4

Methods

4.1 Python implementation of CMA-ES

To implement CMA-ES, Python was the language of choice, as there already is an existing Python library [8].

For both the implementation of the rectangle problem and the Beauchemin model, a simple skeleton was used to build forth upon (Listing 5.1).

```python
def run_cma(initial_values, target, sigma_0):

    es = cma.CMAEvolutionStrategy(initial_values, sigma_0)

    while not es.stop():
        solutions = es.ask()
        scores = [f(target, solution) for solution in solutions]
        es.tell(solutions, scores)
```

Listing 4.1: Python skeleton for CMA-ES

This CMA-ES implementation makes use of an 'ask-and-tell' interface, where the optimizer gets asked to generate λ coordinates (stored in `solutions`). These will then be evaluated by executing the objective function f to obtain the error value for each of the samples.

Finally, after getting the results from the evaluation, the optimizer will be told the results of that evaluation. The μ coordinates that gave the best score, in other words the lowest error, will be used to compute the distribution of samples for the next iteration.

To initialize the optimizer, it receives an initial coordinate that has to be at least 2-D[1], a target coordinate with the same dimension and a σ_0 value.

[1]This library does not support 1-D fitting out-of-the-box, but can be implemented with a workaround as explained in section 4.5.

4.2 Beauchemin simulations

To execute Beauchemin simulations, 2 predefined functions from `celltrackR` were used [9]. The first, `beaucheminTrack()`, computed a Beauchemin track of a single cell. The second, `simulateTracks()`, was used to simulate multiple of those Beauchemin tracks. Besides the three parameters that were discussed (t_{free}, v_{free} and t_{pause}), there are several more parameters that are not of interest for the optimisation process. Every simulation will be ran with those parameters set to the same values (see Appendix A.3 for exact settings). The three variables that are worth mentioning are n_{tracks}, that dictate how many cells are generated, t_{sim}, that describes how long a cell may move in total, and t_{delta}, which is the interval that states when cell positions are measured.

The larger t_{sim} will be, the longer a Beauchemin simulation will take. Longer Beauchemin simulations naturally result in a slower optimisation process, but do give greater precision as more data is generated.

The same goes for the n_{tracks}, the more tracks, the more data, but more costly computations as there are more cells to be simulated.

Finally, increasing t_{delta} will yield greater precision in final MSD calculation, as when cells are measured more often, more data points will be available, even when the t_{sim} and n_{tracks} do not change. Contrarily, it is not of use making t_{delta} extremely small (lower than 0.1 for example), as a cell will not travel great distance in such small time periods, resulting in only small changes per time point.

Naturally, the sweet spot of these parameters have to be found for the optimisation process. By trial-and-error, these were finally set to $t_{sim} = 20$, $t_{delta} = 0.1$ and $n_{tracks} = 10$, as these proved to give the best results while also not resulting in incredibly lengthy simulations.

4.3 MSD calculation

To calculate the MSD of the Beauchemin simulations that were used to calculate the error, the function `squareDisplacement` from `celltrackR` was used (Listing 4.2) [9]. Here, `tracks` denotes an R dataframe containing a time t, and both an x and y for every cell in a Beauchemin simulation. The R-function `aggregate` will apply `squareDisplacement` to every track.

The `tracks` themselves are the output of a Beauchemin simulation as described in section 4.2.

```
aggregate(tracks, squareDisplacement(), FUN="mean")
```

Listing 4.2: MSD calculation using the squareDisplacement function from celltrackR

4.4 Choosing parameters to optimise

The parameter that were repeatedly chosen to optimise for the Beauchemin model, were $t_{free} = 2.0$, $v_{free} = 18.8$ and $t_{pause} = 0.5$. The reason being that these were the default parameters for the model as set in the `celltrackR` library. Additionally, these parameters found to generate output that matched real-world data the best [2].

4.5 Fitting single parameters

This implementation of CMA-ES needs at least a 2-D vector, or in other words, 2 starting parameters. When only trying to fit one, we can supply the optimizer with a dummy parameter that returns a high penalty upon deviating from its starting value. This penalty is added to the error value.

This method was used to fit the single parameter t_{free} (Section 3.2.1). In reality, there was another parameter in play that was set to 0.05 (v_{free} in Figure 4.1). Deviating from that value added an error of 10 to the error value.

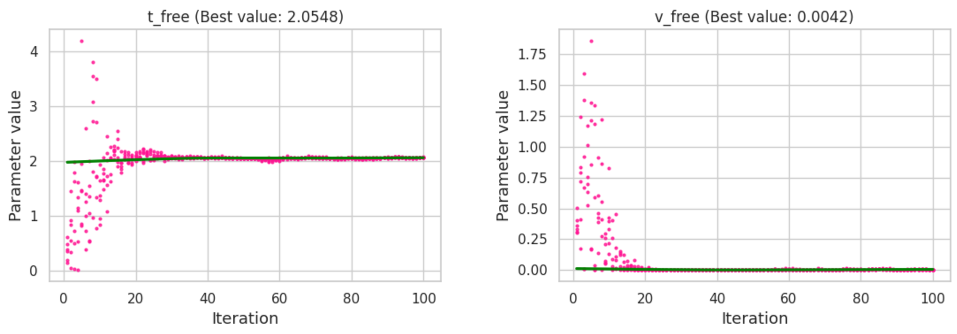


Figure 4.1: Results of CMA-ES fitting a single parameter t_{free}, using a dummy parameter to assert a 2-D starting point. Deviation from $v_{free} = 0.05$ returned a penalty for the error values, forcing CMA-ES to fit it to be around that value.

4.6 Scaling parameters using a wrapper function

To execute the scaling experiments of this thesis, a wrapper function that transforms the parameter function is necessary. This wrapper is called before the values generated by CMA-ES go to the objective function.

In practice, division was used, where the denominator depends on the scale. To scale a parameter by a value of 100, the value generated by CMA-ES gets divided by 100, so that the target is of a scale 100 larger.

Hence, if CMA-ES has a target value of 5, and generates that value of 5, the wrapper makes it $\frac{5}{100} = 0.05$. This value goes to the objective function, resulting in a large error. In other words, CMA-ES actually has to generate the value 500, as in that way $\frac{500}{100} = 5$ will go to the objective function.

This method of scaling was chosen, as opposed to just choose e.g. 500 as target value for the Beauchemin model, so that we can directly compare performance on several scales. If we would pick another target value for each scale, the output of the model would be different, adding another factor to the performance of CMA-ES. This would make it harder to distinguish if a shift in solutions is a result of a different target dataset, or the difference in scales of the parameters.

4.7 Rescaling the problem

The problem rescaling was done in a similar fashion as the scaling of parameters. The rescaling function was called on the CMA-ES value before calling the objective function.

When both rescaling a problem and also scaling a single parameter, the problem rescaling function has to be called before the single parameter scaling function. Otherwise, the final values will not be on the same scale as was the goal of rescaling function.

This means that if CMA-ES were to generate the value ϕ, it would be logarithmically scaled giving 10^{ϕ} (Equation 3.1). Then, if the value was to be scaled to get it to a different scale from another parameter, e.g. 100x, it would be $\frac{10^{\phi}}{100}$.

Chapter 5

Discussion

5.1 Fitting more than two parameters using CMA-ES

The Beauchemin model only has three parameters, whereof only two could be fitted. Contrarily, the majority of interesting BABMs have more than two parameters. Naturally, fitting more than two parameters is possible, CMA-ES was even tested with dimensions up to 640 [10].

The main challenge with BABMs is defining an error function that takes into account all parameters to be fitted. A problem with the Beauchemin model was that the MSD is underdefined to fit three parameters, as several combinations give the same output.

Of course, it is also up to the user what they want CMA-ES to do. The MSD for fitting three Beauchemin parameters could well be used, if the only goal was to find *some* triplet that generates the target MSD. If the goal was to find the exact parameter that target was generated with, it would not be robust.

If the goal is to retrieve the exact parameters a certain dataset was created with, the objective function has to be well-defined in order to distinguish between parameters. This is yet another challenge for fitting BABMs, as this can prove to be difficult.

5.2 Additional scaling issues

In this thesis, linear scaling issues were explored. In practise, these scaling issues can also be non-linear. A realistic problem is *plateaus*, where a change in parameter value does not, or only marginally, change the output. Here, there is only a small region where a parameter exercises its activity.

This poses a considerable challenge for CMA-ES, as if the output does not change, the algorithm will converge, thinking it has found a solution.

The proposed solution for this is rescaling the problem as explored, together with a σ_0 value so that every solution within the range, e.g. [0, 10], is able to be reached within only a few iterations. Additionally, a large population size can be chosen to increase the chance of finding the area where the parameter lives.

This experiment has to be done in the future to determine if CMA-ES is able to handle these plateaus.

5.3 Choosing CMA-ES parameters

5.3.1 Choosing the initial coordinate

Following [5], if the optimum is expected to lie within the search space $[a, b]^n$, the initial coordinate can be chosen uniformly random to be in this space.

For the experiments that were done in this thesis, the initial coordinate was always (0.1, 0.1). If the goal coordinate is (2.0, 18.8), it means that this initial coordinate favours the first target value, as it is simply closer. Ideally, an initial coordinate that is favourable for each parameter is necessary. In this thesis, no upfront knowledge about the target output was presumed, and thus the initial coordinate remained the same.

In real-world experiments though, it could happen that there is some upfront knowledge that points in the right direction of the target coordinate. This information should then be used, as performance might increase.

5.3.2 Choosing the initial sigma

The only constraint given to the value of σ, is that it is in $\mathbb{R}_{>0}$.

It is advised to choose a value for σ_0 so that the solution is expected to lie within about $x_0 \pm 3 \cdot \sigma_0$, with x_0 being the initial coordinate [5]. To enforce this, problem scaling as explored can be a very useful tool.

In this thesis, the influence of different values for σ_0 was not studied, but is expected to influence the performance.

5.4 Using the error distribution to evaluate solutions

In section 3.1, the stochasticity of the Beauchemin model was illustrated using an error distribution, comparing several datasets generated with the same parameters (Figure 3.2).

The variance shown in this distribution indicates that users should interpret results with some caution. By only looking at a single run that almost precisely generates the target solution, users may think the optimisation method is flawless. On the other hand, solutions that are further away from the actual target parameters can also give an output that matches the target dataset, as a result of stochasticity.

This raises the question if every optimisation run should be done multiple times. While this would give more certainty about what the 'right' parameters are (Figure 3.5, 3.8, 3.10), it will be a huge performance penalty, which cannot be afforded with very complex, time-consuming, models. Besides, this distribution might shift depending on the parameters used to create it.

What could be an option, is generating an error distribution with the solution received by CMA-ES. Multiple simulations can be ran with that solution, and be compared to the target dataset. Depending on the distribution, it can be evaluated if the solution was plausible, or still overall generates a too high of an error compared to what would be expected.

This again comes with a performance penalty, but does not require CMA-ES to fit the model multiple times, what will take the most time.

It is up to the users and their purpose to assess if the performance penalty is affordable.

5.5 Cost penalty of using Restart strategies for CMA-ES

Restart strategies for CMA-ES proved to increase the accuracy of the generated solutions for the Beauchemin model considerably. In contrast, there is a big downside with these strategies, the increased time consumption.

For IPOP-CMA-ES, the time consumption increases with $\mathcal{O}(n^2)$ for every restart. A single iterations of a standard CMA-ES run for the fitting the Beauchemin model with $\lambda = 6$, or 6 function evaluations per iteration, takes around 4 seconds (using the specifications as provided in Appendix A.1). For 120 iterations this means that a non-restart CMA-ES run takes around 8 minutes. For 2 restarts this means 8 + 16 + 32 = 56 minutes in total.

Recall that the reason why the Beauchemin model was chosen, is because it is a relatively cheap computation. There are BABMs out there that can take up to hours to simulate. Doing λ of those computations per CMA-ES iterations is of course already extremely costly, exponentially increasing those computations is not realistic (assuming users do not want to wait weeks for an optimization process to finish).

In conclusion, for low-cost simulations such as the Beauchemin model Restart/IPOP-CMA-ES can be considered as an option for optimisation, as the increased time consumption is still manageable as opposed to high-cost BABMs. Namely, the solutions generated by the algorithm do seem to improve using restart strategies. It should be evaluated by the user whether this penalty can be afforded or if it will become too costly.

5.6 Choosing when to rescale a problem

Rescaling the problem was shown to be a powerful method for handling parameters that live on different scales. It also proved to result in similar results for the base run, where the parameters did not live on very different scales. This begs the question when to rescale a problem.

In practise, rescaling a problem the way it was done in this thesis ensures that the values generated by CMA-ES do not have to go outside of the range [0, 10]. If values are in that range regardless, then rescaling would have no, or a negative effect. If values are outside of it though, rescaling would have the benefits as explained. Even if it is uncertain were the target values are going to be, the problem can be rescaled to be sure. While rescaling will maybe will not improve performance, it will only decrease it if the rescaling results in a large range of solutions.

Thus, rescaling of the problem can always be done, and will improve performance if the range of solutions gets narrowed down by rescaling. As seen by the scaling experiments, the larger the difference in ranges before and after rescaling, the larger the performance increase. Of course it depends on the expected parameter values what kind of scaling can be used. No negative values can be generated using the rescaling method as described in this thesis. For the Beauchemin model, negative values for these parameters are not possible. If working with a BABM that does take negative values, another rescaling method has to be composed.

5.7 Analytically solving the Beauchemin model

As mentioned in section 2.2, the Beauchemin model can be solved analytically. The expected MSD can be calculated ([2], proposition 12):

$$E\|D^2(t)\| = 2Mt - 2Mt_{free} \times \begin{cases} \frac{1}{3} & t \geq t_{free} \\ \frac{1}{3}\left(\frac{t}{t_{free}}\right)^3 - \left(\frac{t}{t_{free}}\right)^2 + \left(\frac{t}{t_{free}}\right) & t < t_{free} \end{cases} \tag{5.1}$$

With M being the motility coefficient ([2], Equation 2):

$$M = \frac{(v_{free} \times t_{free})^2}{6(t_{free} + t_{pause})} \tag{5.2}$$

This formula can be used to calculate the squared displacement at a specific time t. Moreover, given a certain squared displacement, the parameters can be retrieved. In this way the solutions generated by the optimisation process can be compared to solutions generated by this formula.

An experiment that was not done in this thesis, is using this formula to determine the difference in stochasticity caused by CMA-ES versus the Beauchemin model.

During this experiment, parameters are used to generate a target dataset. The MSD will be calculated from that dataset, and from there the corresponding parameters using the analytical formula. These parameters will likely be different from the parameters used to generate the dataset, due to stochasticity of the model.

Then, using CMA-ES, the parameters for the dataset can again be retrieved.

Finally, comparing the analytically derived parameters and the parameters generated by CMA-ES. The discrepancies between these parameters are directly caused by the algorithm itself, and not the stochasticity of the model.

Chapter 6

Related Work

6.1 Stochastic gradient descent using the Adam algorithm

Another algorithm that could be used for fitting BABMs is *Adam* [11]. Adam is a method for stochastic optimization.

It is a method for gradient based optimization, meaning it moves through the gradient of objective function values, in search of a local minimum.

Like CMA-ES, Adam keeps track of moving average, which are estimates of the mean and the variance from that mean.

In contrast to CMA-ES, Adam computes individual learning rates for each parameter of the objective function, which changes during the process of optimization. These rates can be compared to the σ values of CMA-ES, in the sense that these are also kept for individual parameters to determine their course. These rates are then directly used to update the parameters:

$$\theta \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) \tag{6.1}$$

Here, θ is the parameter that is getting updated. t is the timestep that increases by one each iteration. α is the stepsize, together with the 1st moment vector m and the second moment vector v it updates the parameter, The first moment vector contains the moving averages of the gradient which themselves are the estimates of the mean and the variance from that mean. The second moment vector v is the squared version of m. Lastly, ϵ is there to prevent division by zero.

The main difference looking at this parameter update, is that it is not stochastic. The value of the parameters, and finally the solutions, are retrieved by a strict calculation, and not stochastic sample from a distribution.

CMA-ES has the arguable downside that it is stochastic, resulting in noise of the solutions that is inherent to its computation. Adam, does not have this downside, and given a certain target dataset, should always get the same solutions.

It would be very interesting to see a comparative study between these algorithms, to see how they perform next to each other, especially for fitting BABMs.

Chapter 7

Conclusion

In this thesis, it was evaluated if CMA-ES is a good option for fitting BABMs.

Out of the box, the algorithm was able to fit single parameter problems with decent accuracy. This accuracy decreased while fitting two parameters for the Beauchemin model. The solutions went in to the right direction, but displayed small discrepancies compared to the target. These deviations from the target values can in some extent be explained by stochasticity of the model used, and CMA-ES by itself. Fitting three Beauchemin model heavily reduced the accuracy of the two parameters that were previously fitted, additionally not being able to fit the third parameter with good precision. This result confirmed the findings by Textor et al, that there are more triplets for this model that give the same output.

To improve the solutions generated by CMA-ES for fitting the Beauchemin model, two restart strategies were introduced. These had the purpose of 'opening up' the search-space again after finding a solution in a previous run, subsequently having the ability to find better solutions. IPOP-CMA-ES proved to give the best results, but also the greatest performance penalty, as time consumption increased by $\mathcal{O}(n^2)$ for each restart. Restart-CMA-ES was chosen for the remaining experiments, as it gave better solutions than regular CMA-ES, but did not have as big of a performance penalty as IPOP-CMA-ES.

The Beauchemin model was then used to simulate the scaling problem that other BABMs might experience. Here it was reviewed if CMA-ES was able to handle target parameters that lived on vastly different scales.

Using Restart-CMA-ES, the solutions increased to be less accurate as this scale-difference increased. To combat this inaccuracy, problem rescaling was introduced.

In this way, the parameters were brought down to the same scale, so that σ_0 and the initial coordinate did not disfavour a parameter, making it hard to find it. The solutions generated by Restart-CMA-ES for the rescaled problems improved enormously, as the solutions generated for the same scaling experiments all had the same accuracy as the run without differently scaled parameters.

It still has to be evaluated how much the stochasticity of CMA-ES influences its final solutions, and how much different values for the CMA-ES hyperparameters influence its performance. Furthermore, the Beauchemin model has an analytical solution. The next step is to apply CMA-ES on a more complex model that experiences the problems that were simulated in this thesis. It will be interesting to see how CMA-ES performs in real-world situations, where there is no ground truth available.

While there still remains research to be done on using CMA-ES for fitting BABMs, the findings in this thesis show great promise for taking on optimisation challenges to come.

Bibliography

[1] C. Beauchemin, N. M. Dixit, and A. S. Perelson, "Characterizing t cell movement within lymph nodes in the absence of antigen," *The Journal of Immunology*, vol. 178, no. 9, pp. 5505–5512, 2007.

[2] J. Textor, M. Sinn, and R. de Boer, "Analytical results on the beauchemin model of lymphocyte migration.," 2013.

[3] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík, "Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009," pp. 1689–1696, 07 2010.

[4] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, pp. 159–195, 06 2001.

[5] N. Hansen, "The cma evolution strategy: A tutorial," 2016.

[6] A. Auger and N. Hansen, "A restart cma evolution strategy with increasing population size," in *2005 IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1769–1776 Vol. 2, 2005.

[7] N. Hansen, "Cma-es source code," Jun 2011.

[8] Y. A. Nikolaus Hansen and P. Baudis, "Cma-es/pycma on github," February 2019. `https://github.com/CMA-ES/pycma`.

[9] I. M. Wortel, A. Y. Liu, K. Dannenberg, J. C. Berry, M. J. Miller, and J. Textor, "Celltrackr: An r package for fast and flexible analysis of immune cell migration data," *ImmunoInformatics*, vol. 1-2, p. 100003, 2021.

[10] K. Varelas, A. Auger, D. Brockhoff, N. Hansen, O. A. Elhara, Y. Semet, R. Kassab, and F. Barbaresco, "A Comparative Study of Large-scale Variants of CMA-ES," in *PPSN XV 2018 - 15th International Conference on Parallel Problem Solving from Nature*, vol. 11101 of *LNCS*, (Coimbra, Portugal), pp. 3–15, Sept. 2018.

[11] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2015. Published as a conference paper at ICLR 2015, `https://arxiv.org/abs/1412.6980`.

Appendix A

Appendix

A.1 Machine specifications

All simulations and experiments were ran on a Pop!_OS virtual machine with the following specifications.

Number of processors	4
Acceleration	VT-x/AMD-V, Nested Paging, KVM Paravirtualization
Video memory	128 MB
Graphics Controller	VMSVGA

Figure A.1: Virtual machine specifications

Respectively run on a machine with the specifications list below.

Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.81 GHz
RAM	16.0 GB
GPU	NVIDIA GeForce GTX 1050
VRAM	2.0 GB

Figure A.2: Machine specifications

A.2 CMA-ES Output

Generally, using the specifications as established, it took CMA-ES around 4 seconds per iteration for $\lambda = 6$, in other words, 6 function evaluations (Section A.1, Listing A.1).

```
(3_w,6)-aCMA-ES (mu_w=2.0,w_1=63%) in dimension 3 (seed=245440,
    Sat Jun 25 22:32:16 2022)
Iterat #Fevals   f-value    axis ratio  sigma  min&max std  t[m:s]
    1      6 4.193917e+01 1.0e+00 5.08e-01  5e-01  6e-01 0:03.9
    2     12 4.470002e+01 1.3e+00 5.12e-01  4e-01  6e-01 0:07.8
    3     18 4.179875e+01 1.3e+00 4.26e-01  4e-01  4e-01 0:11.7
    4     24 3.646529e+01 1.3e+00 4.09e-01  3e-01  4e-01 0:15.4
    6     36 2.403174e+01 1.5e+00 5.10e-01  4e-01  5e-01 0:23.2
    8     48 1.768353e+01 2.2e+00 7.17e-01  6e-01  9e-01 0:30.9
   10     60 7.326296e+00 2.8e+00 1.37e+00  1e+00  2e+00 0:38.7
   12     72 4.175103e+00 3.9e+00 1.65e+00  1e+00  2e+00 0:46.5
   15     90 5.268088e-01 5.9e+00 2.92e+00  4e+00  6e+00 0:58.1
   18    108 7.253512e-02 7.4e+00 2.99e+00  3e+00  4e+00 1:09.7
   21    126 5.808623e-02 6.1e+00 2.03e+00  2e+00  2e+00 1:21.1
   24    144 6.372388e-02 5.2e+00 1.56e+00  1e+00  2e+00 1:33.1
   27    162 6.399808e-02 6.2e+00 1.80e+00  1e+00  2e+00 1:46.8
   30    180 9.289035e-02 7.3e+00 1.90e+00  1e+00  2e+00 2:01.2
   33    198 6.095228e-02 8.2e+00 1.38e+00  8e-01  1e+00 2:15.4
   37    222 5.832487e-02 7.8e+00 1.47e+00  8e-01  2e+00 2:32.5
   41    246 7.568301e-02 7.0e+00 2.15e+00  1e+00  3e+00 2:51.7
   45    270 8.955594e-02 7.0e+00 2.22e+00  1e+00  2e+00 3:11.8
   49    294 2.605863e-01 5.6e+00 2.72e+00  1e+00  3e+00 3:30.2
   54    324 1.178749e-01 6.7e+00 2.76e+00  1e+00  3e+00 3:53.4
   59    354 6.678890e-02 6.7e+00 1.45e+00  6e-01  1e+00 4:16.3
   64    384 6.370684e-02 1.3e+01 1.11e+00  4e-01  8e-01 4:39.1
   69    414 6.847523e-02 1.2e+01 5.71e-01  2e-01  3e-01 5:04.2
   74    444 6.323708e-02 1.4e+01 4.93e-01  1e-01  3e-01 5:28.0
   80    480 6.749566e-02 1.1e+01 7.10e-01  2e-01  4e-01 5:54.7
final/bestever f-value = 6.749566e-02 4.683004e-02
incumbent solution: [20.119893597556953, 15.58871758002261,
    10.796736594807856]
std deviation: [0.7666149133064301, 0.42168548411579937,
    0.5075384795931648]
```

Listing A.1: Example of output generated by CMA-ES

A.3 Beauchemin parameter settings

This section contains the default parameters to run Beauchemin simulations with.

Beauchemin Default Parameters	
Parameter	Value
n_{tracks}	10
t_{sim}	20
t_{delta}	0.1
$p_{persist}$	0
p_{bias}	0.9
dir_{bias}	(0,0,0)
$taxis\ mode$	1
t_{free}	2.0
v_{free}	18.8
t_{pause}	2.0

A.4 cmaes.py

This section contains the Python function used to run CMA-ES, and Restart/IPOP-CMA-ES.

```python
import cma
import numpy as np
from plotcma import plotCMA, plotSigma


def run_cma(max_its, param_names, initial_values, sigma, popsize
    ):
    no_of_params = len(initial_values)
    error_values = []

    # Create a list for each to-be-fitted parameter that
    # holds the evolution of the sigma values, and
    # one for the coordinates
    sigma_values = []
    param_values = []
    for i in range(no_of_params):
        sigma_values.append([])
        param_values.append([])

    # Create new CMA-ES object with target values defined
    # above and bounds between zero and infinite
    es = cma.CMAEvolutionStrategy(initial_values, sigma,
                                  {'bounds': [0, np.inf],
                                  'popsize': popsize})

    i = 1
    while True:
        # Base check, change to preferred stop condition e.g.
        # error value, sigma value, etc.
        if i > max_its:
            print("Maximum iterations reached")
            break

        # CMA ES part
        solutions = es.ask()
        scores = [eval_beauchemin(transform_params(s)) for s in solutions]
        es.tell(solutions, scores)
        es.disp()

        # Adding points for plotting the history of the
        # sigmas and the error.
        sigmas = (es.result[6]).tolist()
        for j in range(no_of_params):
            sigma_values[j].append([i, sigmas[j]])
            for sol in solutions:
                param_values[j].append([i, sol[j]])

        # Calculate best error of this iteration
        err_val = np.amin(scores)

        error_values.append([i, err_val])

        i += 1

    es.result_pretty()
    plotCMA(param_values, param_names, error_values, result)
    plotSigma(sigma_values, param_names)

    return es.result[0], es.result[1]
```

Listing A.2: CMA-ES

```python
def ipop_cmaes(restarts, sigma, initial_values, max_its, popsize, increase_pop):
    values = initial_values
    best_values = []
    best_error = np.inf

    for i in range(restarts):
        res, err = run_cma(max_its=max_its, initial_values=values, sigma=sigma, popsize=popsize)

        if increase_pop:
            popsize = popsize * 2

        if err < best_error:
            best_error = err
```

```
                best_values = res

    return best_values.tolist()
```

Listing A.3: Restart/IPOP-CMA-ES

A.5 run-beauchemin.R

```
library(celltrackR)

no_of_tracks <- 10 # Number of tracks to simulate

# Get arguments from command-line
args <- commandArgs(trailingOnly = TRUE)

# Function that executes a single Beauchemin
# simulation for its given parameters.
single_beauchemin_sim <- function(sim_time, delta_t, p_persist,
    p_bias,
                                    bias_dir_vector, taxis_mode, t
    _free,
                                    v_free, t_pause) {
  beauchemin_sim <- beaucheminTrack(
    sim.time = sim_time, delta.t = delta_t,
    p.persist = p_persist, p.bias = p_bias,
    bias.dir = bias_dir_vector, taxis.mode = taxis_mode,
    t.free = t_free, v.free = v_free,
    t.pause = t_pause
  )

  return(beauchemin_sim)
}

# Function that runs the Beauchemin simulations
# for NO_OF_TRACKS tracks.
execute_beauchemin <- function(input, no_of_args) {
  sim_time <- as.numeric(input[1])
  delta_t <- as.numeric(input[2])
  p_persist <- as.numeric(input[3])
  p_bias <- as.numeric(input[4])
  bias_dir <- input[5]
  taxis_mode <- as.numeric(input[6])
  t_free <- as.numeric(input[7])
  v_free <- as.numeric(input[8])
  t_pause <- as.numeric(input[9])

  # Create the tracks for the Beauchemin simulation.
  beauchemin_tracks <- simulateTracks(
    no_of_tracks,
    single_beauchemin_sim(
      sim_time, delta_t,
      p_persist, p_bias, bias_dir_vector,
```

```r
      taxis_mode, t_free, v_free, t_pause
    )
  )

  # Store the Beauchemin tracks as dataframe.
  df_a <- as.data.frame(beauchemin_tracks)

  # Convert the previous dataframe into a dataframe
  # in the following format:
  # time  cell_id  cellkind  x  y  connectedness
  df_b <- data.frame(
    time = df_a$t, cell_id = df_a$id, cellkind = "N/A",
    centroid_x = df_a$x, centroid_y = df_a$y, connectedness = "N
   /A"
  )

  return(df_b)
}

df <- execute_beauchemin(args, length(args))
print(df, row.names = FALSE)
```
Listing A.4: R script to run Beauchemin simulations