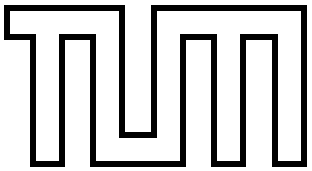


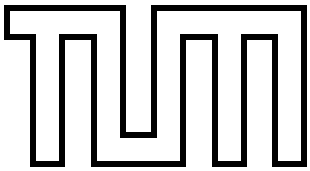
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Efficient Bayesian Inference of Hydrological Model Parameters: Implementation of a Parallel Markov Chain Monte Carlo Approach

Chengjie Zhou



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Efficient Bayesian Inference of Hydrological Model Parameters: Implementation of a Parallel Markov Chain Monte Carlo Approach

Effiziente Bayesianische Inferenz der Parameter von der hydrologischen Modelle: Implementierung eines Markov-Chain-Monte-Carlo-Verfahrens im parallelen Modus

Author:	Chengjie Zhou
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Ivana Jovanovic Buha, M.Sc.
Date:	09.08.2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 09.08.2024 Chengjie Zhou

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Zusammenfassung

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

Part I.

Introduction and Background

1. Introduction

2. Introduction to Markov Chain Monte Carlo

2.1. The Idea of Markov Chain Monte Carlo

Markov chain Monte Carlo (abbr. MCMC) is an algorithm that performs sampling. General usage of the Markov chain Monte Carlo algorithm started in the fields of chemistry, biochemistry, and physics up until after 1990 when it was also adopted by the field of statistics and scientific computing.[BGJM11] The general idea of the Markov chain Monte Carlo includes, as its name suggests, a combination of the Monte Carlo methods and the usage of Markov chains. The Monte Carlo methods solve numerical problems by repeatedly generating random numbers,[KTB13] whereas the Markov chains provide this algorithm a property so that each sample that is generated depends on the sample that is generated before.[Wan09]

The Markov chain Monte Carlo tries to sample an unknown target distribution using a proposal distribution that completely lies above the target distribution. The selection proposal distribution is crucial to the success of the algorithm, since it may lead to different behaviors of convergence and acceptance rate.[BGJM11] The core of the Markov chain Monte Carlo is the application of a Markov chain, where the Monte Carlo integration is used. Given a distribution $\pi(\cdot)$ and its probability function $f(\cdot)$, a typical Monte Carlo simulation would perform the following mathematical approximation:

$$\mathbb{E}[f(X)] \approx \frac{1}{n} \sum_{i=1}^{n} f(X_i) \tag{2.1}$$

where $\{X_i | i \in [1, n]\}$ is the sample space that is drawn from the distribution $\pi(\cdot)$.[GRS95] Since this approximation is used on a Markov chain, each sample from the sample space is dependent on the sample before. In Markov chain Monte Carlo, this dependence is given by a transition kernel that is essentially a conditional distribution $\Pr[X_{i+1}|X_i]$.[GRS95] In other words: After the last sample was generated, a distribution that takes this generated sample as a parameter is created. The next sample is then generated based on this newly created sample, which creates a dependence between both samples. Markov chains Monte Carlo functions thanks to a property called ergodicity that is shared by all Markov chains. The ergodic theorem states that the distribution of the states on the chain converges to a certain stationary distribution regardless of the starting state, as time approaches infinity.[Cas07] This property could be applied in the case of Markov chain Monte Carlo as well. The starting states that do not sample from the stationary distribution before the ergodic states can be regarded as "burn-in states" and should be discarded. Thus, the approximation of the Monte Carlo simulation can be altered to

$$\mathbb{E}[f(X)] \approx \frac{1}{n-b} \sum_{i=b+1}^{n} f(X_i) \tag{2.2}$$

where b denotes the position of the last state in the burn-in phase that should be discarded[GRS95]. The determination of the burn in phase plays an important part in terms of the sample space accuracy and will be discussed later in this thesis in a more detailed manner.

A crucial prerequisite of the Markov chain Monte Carlo algorithm is the detailed balance condition. The ergodicity property after the burn-in period for every Markov chain can be mathematically defined as:

$$\forall X, Y \in S.\ \pi(X)\Pr[Y|X] = \pi(Y)\Pr[X|Y] \tag{2.3}$$

[GRS95] where S is the set of the state of a Markov chain. From this equation, we can derive the following property:

$$\int \pi(X)\Pr[Y|X]dX = \int \pi(Y)\Pr[X|Y]dX = \pi(X_j) \tag{2.4}$$

What this equation essentially points out is that if X is sampled from the stationary distribution $\pi(\cdot)$, Y will also be from this stationary distribution.[GRS95] This corresponds to the idea of ergodicity and proves that using the detailed balance equation, all of the subsequent samples will eventually come from the stationary distribution.

2.2. The Metropolis-Hastings Algorithm

Metropolis-Hastings is a widely used algorithm that performs Markov chain Monte Carlo sampling. It is extremely versatile and often used to sample multivariate distribution. It was extensively used in the physics field, but later on also in the statistics field.[CG95]

As mentioned before, the Markov chain Monte Carlo algorithms use a transition kernel to create dependence between two states.

The main idea is that for each iteration, a sample is drawn from the target distribution. For the generation of the next state, a distribution that takes the last sampled data point as a parameter will be created so that the new sample can be drawn from the newly created distribution.[GRS95] An acceptance probability is then calculated using the following formula:

$$\alpha(X, Y) = \min(\frac{\pi(Y)q[Y|X]}{\pi(X)q[X|Y]}, 1) \tag{2.5}$$

where $q[X|Y]$ denotes the proposal density from X to Y. With the proposal density multiplied by the acceptance probability, the transition probability is derived:

$$\Pr[X|Y] = q(X|Y)A(Y, X) \tag{2.6}$$

Due to the detailed balance equation:[GRS95]

$$\pi(X)q(Y|X)A(X, Y) = \pi(Y)q(X|Y)A(Y, X) \tag{2.7}$$

Bringing the ratio of the acceptance probability to the left

$$\frac{A(Y, X)}{A(X, Y)} = \frac{\pi(Y)q[Y|X]}{\pi(X)q[X|Y]} := \alpha(X, Y) \tag{2.8}$$

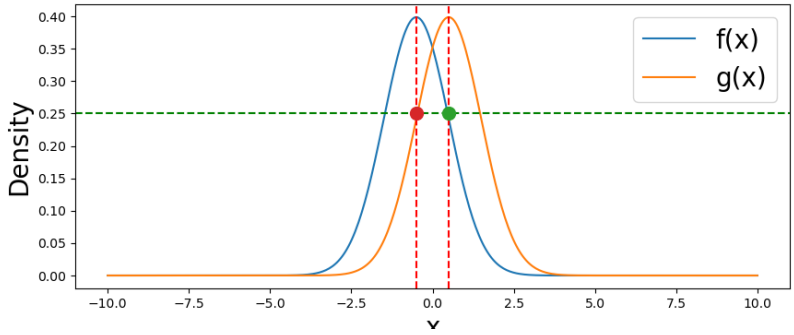


Figure 2.1.: Caption: TODO

As we can see, the acceptance probability of the Metropolis-Hastings algorithm is calculated based on the ratio of the acceptance density from the old to the newly generated sample point to the acceptance density from the newly to the old generated sample point. However, since the probability space adds up to one, the maximum acceptance probability can not exceed one. Therefore, a minimum condition must be added.

A special case of the Metropolis-Hastings algorithm is the Metropolis algorithm. The Metropolis algorithm applies a symmetric distribution as the transition kernel, such as a normal distribution.[Bha88] Due to the symmetry, the proposal density $q[Y|X]$ equals to $q[X|Y]$. The idea is that the position of the density of the newly generated sample in the normal distribution centering the last generated sample is at the same altitude as the density of the last generated sample in the normal distribution centering the newly generated sample, as the visualization in Figure 2.1 suggests.

In this graph, we can see that the green point is the newly sampled point conditioned on the distribution drawn by the last generated point. The red point denotes, on the contrary, the last generated point conditioned on the distribution drawn by the newly generated sample. These two points lie on the same level and share the equivalent value. Thus, the term $\frac{q[Y|X]}{q[X|Y]}$ cancels out to 1. In this case, the acceptance probability becomes

$$\alpha(X, Y) = \min(\frac{\pi(Y)}{\pi(X)}, 1) \tag{2.9}$$

which is virtually the ratio of the probability density of the newly generated sample to the probability density of the last generated sample. An illustration of this equation would be as follows: if $\pi(Y) > \pi(X)$, which means that the probability density of the newly generated sample is greater than its of the last generated sample, $\frac{\pi(Y)}{\pi(X)}$ is greater than 1 and the acceptance of the newly generated point is guaranteed. Over time, the samples that are generated will have greater and greater probability density, and eventually, the peak will be reached. If $\pi(X) > \pi(Y)$, which means that the probability density of the newly generated sample is less than it's of the last generated sample, $\frac{\pi(Y)}{\pi(X)}$ is greater than 1 and it is still probable to accept the newly generated point, however a less probability that is calculated equation.[CG95]

2.3. Advantages of Markov Chain Monte Carlo Methods over Other Commonly-Used Sampling Methods

Markov chain Monte Carlo methods differ from other popular sampling methods and have specific advantages over them. In this section, the rejection sampling and the importance sampling are discussed and are compared to Markov chain Monte Carlo sampling.

Rejection sampling, also known as acceptance-rejection sampling, is a sampling method that generates samples that are independent from one another. Instead of generating the next sample from a newly created distribution that takes the last generated sample as input, the samples are generated from a sampling distribution that lies above the target distribution. The acceptance probability is than the ratio of the density of the target distribution over the sampling distribution.[GRS95] However, if the target distribution is complicated, especially in multivariate cases, the ratio might be very low, causing the acceptance probability to be low as well, which leads to inefficiency. By creating a dependency between samples, Markov chain Monte Carlo methods avoid this inefficiency.[AFDJ03]

Importance sampling is a sampling method that is based on the calculation of weights. A typical implementation includes the generation of samples, calculating the weights of all of these samples, and calculating the expected value by summing them up together by the weights.[Atz10] The process of importance sampling is rather easy to implement. However, a disadvantage is that the samples that have higher weights dominate the calculation of the expected value, which essentially reduces the sample space since the samples that have lower weights play almost no role in the calculation. Markov chain Monte Carlo methods, on the other hand, eventually samples from the stationary distribution after the burn-in period, resulting in consistency of the sampling.[AFDJ03]

2.4. The Idea of Parallel Implementation of Markov Chain Monte Carlo

The objective of this thesis is to implement parallel versions of Markov chain Monte Carlo algorithms for the Bayesian inverse problem. Therefore, paralleling Markov chain Monte Carlo is a significant part of the thesis. The base idea is that instead of one single chain, several chains are run simultaneously.[VS13] Since after the burn-in period, every single state from each chain samples from the stationary distribution due to the ergodic property of the Markov chains,[Cas07] all of the samples from different chains could be merged to present the target stationary distribution. However, other variations involve generating multiple points at the same time conditioned on the last generated sample and then evaluating the forward model.[WSB23] Since this chapter only provides an overview of the Markov chain Monte Carlo algorithm, a following chapter that is specifically dedicated to the parallel of the algorithm will be given.

2.5. An example of the Metropolis Hastings Algorithm

In this section, an example of the Metropolis-Hastings algorithm will be given and visualized to provide an illustrative comprehension of the different steps of the algorithm. In this

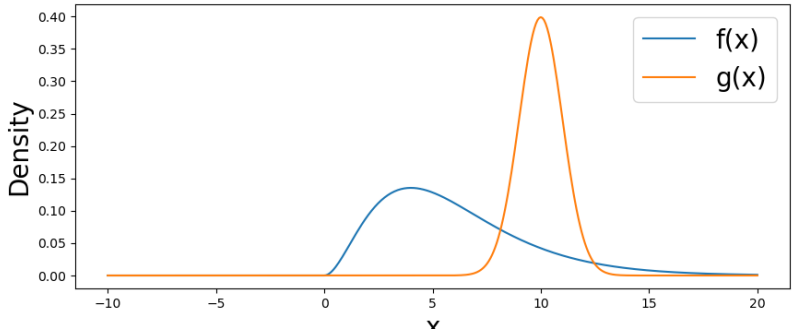


Figure 2.2.: Caption: TODO

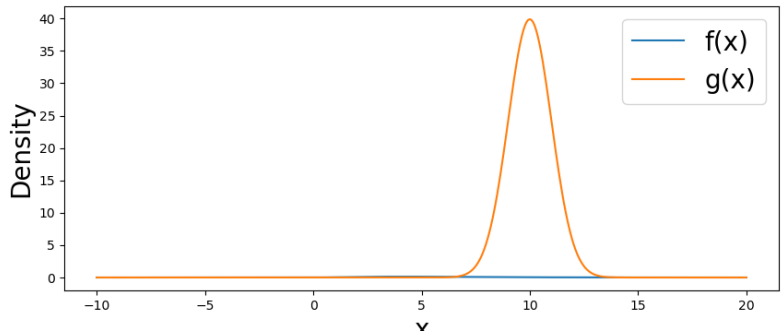


Figure 2.3.: Caption: TODO

example, we want to sample an Erlang distribution from a normal distribution. We are set to generate 100,000 samples to compare how the generated samples fit the Erlang distribution. The Erlang distribution $f(x)$ has a shape of 3 and a scale of 2, whereas the normal distribution $g(x)$ has a mean of 10 and a standard deviation of 1.

As we can see from Figure 2.2, the normal distribution that is sampled does not lie above the Erlang distribution. Therefore, we need to scale up the normal distribution. The scaled up graph is then shown in the Figure 2.3.

The transition kernel is set to be a normal distribution that sets the mean as the last generated sample and the standard deviation of 4. Since the normal distribution is symmetric, the acceptance probability could be set to $\alpha(X, Y) = \min(\frac{\pi(Y)}{\pi(X)}, 1)$, as mentioned above. The probability density function of the sampling distribution, $\pi(\cdot)$, is defined as follows:

$$\pi(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{2.10}$$

where μ denotes the mean and σ denotes the standard deviation of the normal distribution. [DS13]

We select 0 as our starting point and start the iterations from there on. In the first iteration, we acquire the random sample with the value of 1.5437347713886516. The acceptance probability is then $\alpha(X, Y) = \min(\frac{\pi(Y)}{\pi(X)}, 1) = 1$. In this case, since the acceptance

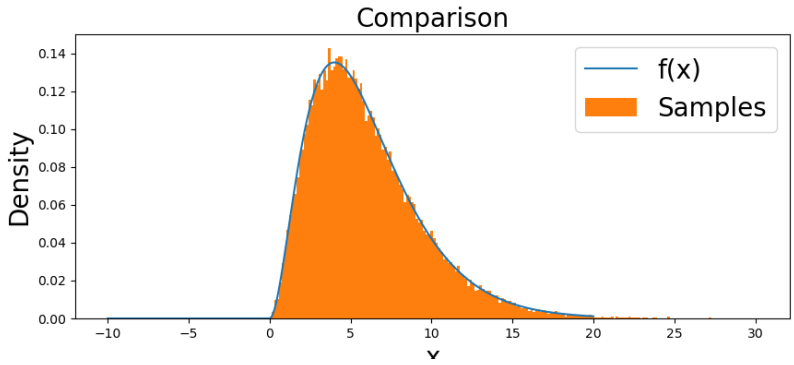


Figure 2.4.: Caption: TODO

probability is 1, it is guaranteed that this sample is accepted. We append this sample to the sample array and move on to the next iteration.

In the next iteration, we sample the value -1.802047900190033. The acceptance probability is then $\alpha(X, Y) = \min(\frac{\pi(Y)}{\pi(X)}, 1) = 0$. This means that this generated sample should by no means be sampled and we should carry on with the last sample. In this case, we repeat our old sample append it to the sample array, and continue.

We continue the rest of the iterations and draw a set of samples. If we plot the samples out and compare them with the target Erlang distribution, it would look something that the graph shown in Figure 2.4. As we can see, the samples that we generate resemble the targeted Erlang distribution. We can conclude that Markov chain Monte Carlo works perfectly in this specific example.

3. Bayesian Inference and its Use Case in Hydrological Model

In this chapter, the idea of Bayesian inference will be introduced, including the instantiation of this problem using the Markov chain Monte Carlo algorithm. Afterward, the hydrological model that is used in this paper will be discussed.

3.1. Bayesian Inference

The Bayesian inference problem is a method of statistical inference that is used to calculate the probability estimate based on evidence and the likelihood of the set of parameters.[BT11] Given a prior distribution that provides information on the preexisting data, the Bayesian inference problem uses the Bayes theorem to update the prior distribution using a likelihood function and derives the actual possibility.

Given the Bayes theorem:[Har03]

$$\Pr[B|A] = \frac{\Pr[B] \cdot \Pr[A|B]}{\Pr[A]} \tag{3.1}$$

where A and B are two different incidents. This equation can than be formed into

$$\Pr[B|A] = \frac{\Pr[B] \cdot \Pr[A|B]}{\int \Pr[B] \cdot \Pr[A|B] dB} \tag{3.2}$$

$\Pr[A|B]$ is called the likelihood function, which is generated by a set of data to interpret how likely a particular set of observations is.[Bar67a] The $\Pr[B]$ is called prior, since this is the preexisting knowledge that is given.[Gel02] The denominator of the equation is called evidence, which is a constant that depicts the probability of observing the data across all values of the model parameters.[GRS95] The result of the above equation is the posterior, which is the object of the Bayesian inference problem.[GRS95]

The different implementations of the Bayesian inference problem are versatile and vary from one another. In this paper, we implement the Bayesian inference problem using the Metropolis-Hastings algorithm with a normal distribution transition kernel. The idea is that we calculate the acceptance rate based on the posterior calculation. For revision, the acceptance rate of the Metropolis-Hastings algorithm with a normal distribution transition kernel is given by:[GRS95]

$$\alpha(X, Y) = \min(\frac{\pi(Y)}{\pi(X)}, 1) \tag{3.3}$$

Replacing $\pi(\cdot)$ with the posterior distribution, we derive:

$$\alpha(X, Y) = \min(\frac{\Pr[Y] \Pr[X|Y]}{\Pr[X] \Pr[Y|X]}, 1) \tag{3.4}$$

In plain language: The acceptance probability is calculated by the ratio of the prior and likelihood of the newly proposed point over the prior and likelihood of the last generated point. Since the evidence is a constant, they cancel each other out and will therefore not be taken into account.

Different variants of implementations are used throughout this paper to perform Bayesian inference of the hydrological model. They will be discussed in later chapters. For now, we will take a look at the hydrological model.

3.2. Overview of the Hydrological Model

The HBV-SASK conceptual model is a renowned mathematical model that is commonly used in the field of hydrology. HBV is a model that describes the subroutines for snow accumulation and melts, for soil moisture accounting and river routing.[B+95] SASK stands for the province of Saskatchewan, the province in Canada in which the model is developed. The creation of the HBV-SASK model is therefore based on the HBV model but involves local data calibration and integration with local water management needs.[SRH19]

The HBV-SASK model has twelve different hyperparameters, of which seven are relevant to this thesis. All of these These include:[GR18]

- TT: ranges from -4 to 4, stands for the air temperature threshold in °C for melting/freezing and separating rain and snow
- C0: ranges from 0 to 10, describes the base melt factor in mm/°C per day
- β (beta): ranges from 0 to 3, depicts the shape parameter (exponent) for the soil release equation
- ETF: ranges from 0 to 1, describes the temperature anomaly correction in 1/°C of potential evapotranspiration
- FC: ranges from 50 to 500, depicts the field capacity of soil in mm.
- FRAC: ranges from 0.1 to 0.9, stands for the fraction of soil release entering the fast reservoir
- K2: the slow reservoir coefficient ranges from 0 to 0.05, which determines what proportion of the storage is released per day

To run this model, these hyperparameters need to be determined. Since the only prior information given is the lowest and the highest bound of each hyperparameter, uncertainty quantification of these hyperparameters is, therefore, necessary to gain posterior information. Apart from these hyperparameters, the starting and the end date of the period that is used for uncertainty quantification are also required to be specified.[GR18] However, the very first phase at the start is used for the spin-up phase, in which the model runs for some time using historical data. This phase stabilizes internal model states such as soil moisture and groundwater levels, which are important for accurate simulation.[GR18] TODO: states

3.3. Overview of the Data Set

There are two existing data sets to the hydrological model, which are respectively called Oldman Basin and Banff Basin, since they are each measured at the Oldman River and in

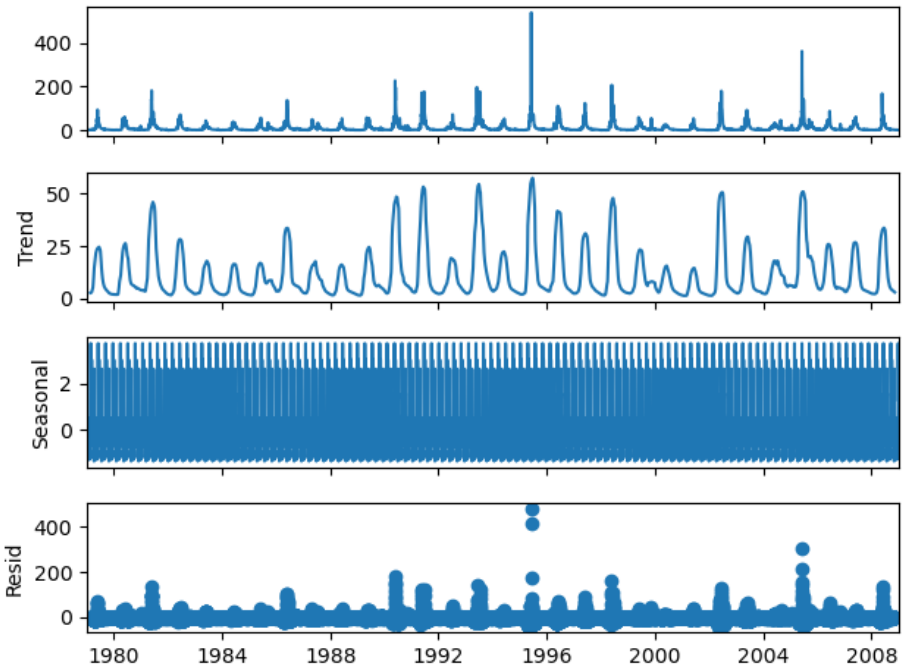


Figure 3.1.: Caption: TODO

the town of Banff in Alberta, Canada.[SR20] The value that is measured is called streamflow. It describes the movement of water within a river or stream channel and is the combined result of all climatological and geographical factors that operate in a drainage basin.[Her08] Both of these data sets are presented in the format of a time series, in which the value of each measurement is collected against the dates over a long period. The Oldman basin data set is available from 1979 to 2008, whereas the Banff basin data set is available from 1950 to 2011.[GR18]

Since the data is presented in a format of time series, a time series decomposition is required to present more information on trends, and seasonal and regression effects.[Wes97] After the decomposition, the trend, seasonal, and residue of the dataset over the whole period can be observed. For the time series decomposition in this section, the function TSA.seasonal.seasonal_decompose from the python framework stats models is used.[MPS11]

First, we take a look at the result of the decomposition of the Oldman basin data set. It is shown in Figure 3.1. The seasonal component of this time series is regular over the years. However, there were significant peaks around the early 1980s, mid-1990s, and around 2005. The streamflow in these periods is higher than anytime else, which indicates that there might be possible heavy rainfalls, floods[GR18], or even ecologic disasters. Anomalies in this period can be also found in the residue component, which confirms that there might be odd behaviors happening in these periods. Therefore, quantifying the uncertainty of these three periods would be a challenging task.

We then take a look at the result of the decomposition of the Banff basin data set. It is shown in Figure 3.2. Similar to the case of Oldman Basin, it shows a regular seasonal component. The trend, on the other hand, fluctuates across the entire measured period

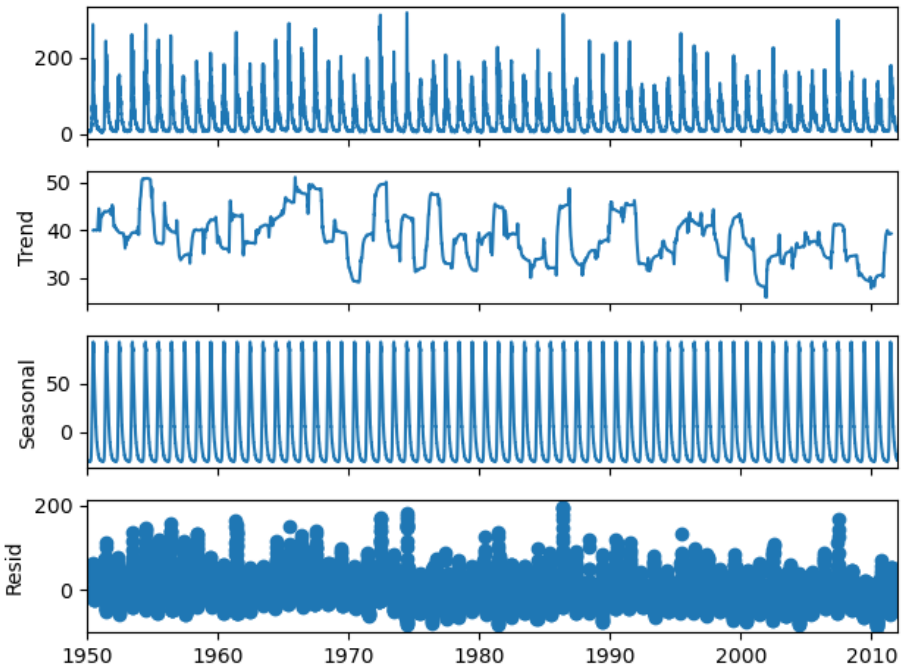


Figure 3.2.: Caption: TODO

but does not show a general upward or downward direction. This means, that the whole measurement is indeed relatively stable. However, the fluctuation might still impose an influence on the accuracy and stability of uncertainty quantification. The residuals of the Banff basin data set are much more varied than the Oldman basin data set. There are obvious clusters of high activity and anomalies, which confirms that the uncertainty quantification of this data set might be a hard task to deal with.

3.4. Task

Since the task of this paper is to caliber the input parameters of the hydrology model, the ultimate goal is to acquire the posterior of the parameters. The main idea in this use case is that we infer the posterior probability of each parameter using Markov chain Monte Carlo. The exact process goes as follows: First, a model is created with the configuration of the period and target distributions being given. Then, we select an appropriate Markov chain Monte Carlo algorithm and perform sampling to solve the Bayesian inference. In this process, the output of each generated set of parameter arrays will be calculated by the model. The output will be later compared to the measured data points, which will provide the algorithm with an acceptance rate. In each iteration, the algorithm decides whether to accept or reject this sample. After all these iterations, the posterior probability is obtained and is prepared to be used on a testing data. Last but not least, we compare the projected results that are calculated by the model to the measured result. Since the posterior probability that is obtained is denoted in the form of a list of sampled data, we can use the Monte Carlo simulation to generate a certain amount of samples, pass them into the model and acquire

the result. Then, we can either calculate the mean of the results or sample the maximum of the result to compare them with the actual data.

In order to operate The training and the testing data periods are needed to be determined and extracted. For the Oldman Basin, the trend of the data from 1990 until 2000 remains constantly fluctuated within a specific upper bound and a specific lower bound, which indicates the stability of the record. Therefore, the posterior is inferred by the time period of the entire 90s. An additional ten years beforehand, which is equivalent to the time period on which the posterior is trained, is used for the spin up phase. The time period after 2000 is then used as the testing phase. Since there are some anomalies around 2005, any time period that includes the year 2005 would be an excellent time period for testing. For the Banff Basin data set, the recorded data is kept constant after the year 1970 and the year 1990. However, in order to use the data in the 2000s for testing purpose, the data before 1990s does not play such an important role. Therefore, we use the data in the time period of the 1990s for training purpose, whereas the 10 years beforehand are used for the spin up phase. The entire time period after the 2000s would be excellent for testing purpose, since anomalies and extreme values can be found throughout the entire sequence.

To test the generalized application of the algorithm, we select three challenging yet representative periods for testing, with one being long (5 years), another being medium long (around 2 years) and the other being short (1 year). Due to the constant fluctuation of the Banff basin data set, we could select a long period within it to perform uncertainty quantification. Since the residue in the early 1970s seems to show extreme anomalies, we select 1970 to 1974. The other two periods are both picked from the Oldman Basin data set. The period from 1994 to 1996 is chosen, since there are apparent anomalies of the residue in this period. Another period is the year 2005, since there are also anomalies being showcased in this year.

4. Fundamental Implementation

In this chapter, a basic version of Metropolis-Hastings is implemented. Complications including the parallel aspect, more complicated algorithms, and further enhancement are not considered in this chapter. A general idea of the implementation is explained, with specific sections contributing to possible issues with the general algorithm and the choice of the likelihood function.

4.1. Hardware Specification and Required Frameworks

All of the code that is run and tested in this paper is run on a single computation machine, namely the MacBook Pro 2021 by Apple Inc. It has an Apple M1 Pro chip, which has an ARM architecture. It has 10 CPU cores, 8 of which are for performance and the rest are for efficiency. It has a RAM of 32 GB and also 16 GPUs available[1]. The entire code is run on macOS Sonama 14.4.1.

The software implementation of the fundamental Metropolis-Hastings algorithm is rather basic. Since the sample space is multivariate, the calculation involves operation between vectors. To ease the process of these calculations, the popular software package of Numpy is used.[HMVDW+20] Additionally, the Tensorflow Probability package is used. It does not only offer implemented probability distribution functions, but also randomness and sampling functions.[DLT+17] To use the Tensorflow Probability framework, the software package Tensorflow is required to be installed[2]. For the visualization part, standard data visualization libraries including Matplotlib and Seaborn are used for the creation of histograms, kernel density estimation, and boxplots.[Was21]

4.2. Basic Metropolis Hastings and Evaluation Metrics

First of all, we take a look at the basic implementation of the Metropolis-Hastings algorithm. This implementation applies the idea of the algorithm mentioned in the chapter above. It takes the following required input parameters: the proposal distribution that data points are sampled from, the sampling kernel for transition, the likelihood kernel for calculation of the likelihood and acceptance rate, the initial state where the algorithm starts, and the number of iterations. Since the prior distributions of the parameters that are going to be calibrated are normal, the boundaries need to be given as input parameters, so that the algorithm can examine whether the generated samples are out of bounds, which should be ignored. The sampling kernel that is used throughout this paper is the Gaussian normal distribution since it provides symmetry and simplicity.[Do08] The concrete algorithm goes as follows.

[1] https://support.apple.com/en-us/111902
[2] https://www.tensorflow.org/probability

Algorithm 1: Basic Metropolis Hastings

Input: proposal distribution, sampling kernel, likelihood kernel, initial state, iterations

Output: sampled data points

```
1  Function MH(arr):
       // Extract parameters
2      [proposal_dist, sampl_kernel, likel_kernel, init_state, iterations] = arr
3      samples ← [init_state]
4      for i ← 0 to iterations - 1 do
           // Calculate probability
5          old ← samples[−1]
6          new ← sampl_kernel(old)
7          p ← proposal_dist(new) ∗ likel_kernel(new)/(proposal_dist(old) ∗
            likel_kernel(old))
           // Decide to accept or reject the sample
8          if random > min(1, p) then
9              samples.add(samples[-1])
10         else
11             samples.add(new)
12     return samples
```

There are two required kernels for the input parameters, namely the sampling kernel and the likelihood kernel. The sampling kernel, as mentioned above, acts like the transition of the Markov chain between two consecutive samples. It plays a critical role in the performance and results of a Markov chain Monte Carlo simulation.[GRS95] Due to the ease and efficiency of calculation that is discussed above, symmetric distribution, notably normal distribution, is used in this paper.[Do08] However, the shape of the kernel is left unknown and is to be determined by the standard deviation.[Wei02] The choice of the standard deviation is crucial. If the standard deviation is too wide, the generated samples will keep getting out of bounds. If the standard deviation is too narrow, it will take too long time to explore the whole distribution range, because it is more likely that the new sample generated is near the mean.

The likelihood kernel also plays a crucial role in the Markov chain Monte Carlo. It measures the probability of observing the data given the set of the generated parameters under the model.[Bar67b] In this paper, the likelihood function for the kernel is implemented as a normal distribution probability function due to the Central Limit Theorem, which suggests that the mean of a large amount of independent random variables will be approximately normally distributed regardless of the underlying distribution.[KK17] In other words, the normal distribution describes the noise of the data points around the mean, which is the sampled value. The likelihood function of the Gaussian normal distribution is given by the product of all of the probability densities of each point:[Gop98]

$$L(\mu, \sigma^2; x_1, ..., x_n) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}} \tag{4.1}$$

Given that the calculation process is numerically hard to solve, the log-likelihood function is commonly used.[PB95] The log-likelihood function of the Gaussian normal distribution is given by:[Ste20]

$$\log[L(\mu, \sigma^2; x_1, ..., x_n)] = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=1}^{n}(x_j - \mu)^2 \tag{4.2}$$

Both of these above equations are implemented as help functions in this thesis for easier further applications. However, an easier way of implementation is to use a probability modeling framework to create a multivariate normal distribution centered around the given value, calculate the likelihood for each pair of points, and sum them up together. This version is also implemented using the TensorFlow probability library and is mainly used throughout this thesis. The only unknown thing is the standard deviation of the distribution. Similar to the sampling distribution, the standard deviation is a left-to-be-determined variable that describes the shape of the distribution. In the context of the likelihood function, the standard deviation implies how concentrated the data are expected to be around the mean value.[Bla06] A smaller standard deviation suggests that the data points are expected to cluster tightly around the mean and have less tolerance, whereas a larger standard deviation implies a broader spread of data points and more tolerance.

Using the above-listed algorithm, we can perform Bayesian inference on the HBV-SASK model. Because the calibrated parameters in the HBV-SASK model are seven-dimensional instead of one-dimensional, the acceptance rate is, therefore, a seven-dimensional array. In this case, to represent the generality of the acceptance rates given by each parameter, the mean of the entire array is taken as the final acceptance rate. There is also an option of taking the maximum value from the array, but the selection will be discussed later in the section of "Input Parameter Exploration".

To quantify the accuracy mathematically, we need to introduce metrics that can calculate the accuracy of the Bayesian inference results. First, we calculate the mean of the absolute difference between the calculated time series run by the model and the measured time series. By calculating the absolute difference, the similarity of the times series could be well quantified. Afterward, two metrics that are used to test the goodness of fit are calculated, namely root mean square error (RMSE) and mean absolute error (MAE). The RMSE is a metric that is often used to evaluate the model performance in climate research studies and ecology. The calculation is as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.3}$$

The square root in RMSE plays an important role. On the one hand, RMSE penalizes larger discrepancies more severely by squaring the errors.[CD14] Moreover, it helps to stabilize the variance of the error terms, making it particularly useful in the use case of this paper, since the usage of standard deviation and variances are present in the implementation.[Hod22]

The MAE is another widely used metric for model performance evaluation.[Hod22]

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.4}$$

Calculating the average of absolute errors means that it is easier to understand and interpret the calculation directly. Besides, due to the less impact of anomalies on the metric,[Hod22] MAE can offer a more robust estimate in contexts when extreme values are expected to be anomalies.

The whole process of evaluation of the results is going to look like this. To collect this result, we first randomly generate 1000 samples from the given posterior using the Monte Carlo simulation [Moo97] and take the mean. The results of the calculation are saved as "posterior mean". Calculating the mean value, which generalizes the entire results, would allow us to observe the actual accuracy of the result since every single individual time series contributes to the calculation process. Next, the maximum value of each timestamp is also found, so that we can observe how extreme the posterior samples could be. The result is then saved as "posterior max", which would indicate how stable the sampling process is. If the posterior max does not differ much from the posterior mean, then the entire sampling space is stable. Otherwise, the individual samples vary too much from each other, causing destabilization. These two time series are then compared to the "prior mean" time series, which is the mean of all the calculated time series of the model that takes samples from prior as input, to observe how much the prior distribution influences the result. After calculating all of the above, a visualization is going to be done so that these results can be compared to each other.

For simplicity, we only use the Oldman Basin dataset for training purposes in this paper. For the exploration phase, the model will be run and evaluated on the same training dataset, so that we can observe how well the trained posterior fits the data on which it is trained. For the actual evaluation, the model will be run on the training dataset, whereas the accuracy score will be evaluated from a testing data set, which is a year 2005 of the Oldman Basin dataset.

Apart from the accuracy test, efficiency also plays an important role in the MCMC algorithms.[QKVT18] To test the efficiency, the run times of different implementations are going to be recorded, so that a comparison can be done later and we can infer which factors have impacts on the computation time.

Another thing that needs to be done is to select meaningful values or instances to be input parameters. In the following sections, we are going to discuss the choices of input parameters based on two methods. First, the input parameters are going to be selected using existing knowledge. Afterward, the input parameters are going to be explored, in which models with different input values are going to be run multiple times. Afterward, the accuracy tests that are mentioned above are going to be carried, out so that an overview of the relation between different input parameters and the accuracy and efficiency result can be visualized. An overview of which set of input parameters delivers the best overall accuracy and efficiency is also going to be shown later on. To test how well the posterior fits the data on which it is trained, the evaluation part is going to be carried on the training data as well. In the end, the sets of input parameters from both methods are going to be used for another run of the algorithm, so that the results of the Bayesian inference problems

can be compared with each other. The evaluation, however, is going to be carried on on the testing data, so that we can observe how well the posterior would perform in actual cases.

4.3. Visualization

After deriving the results, visualization helps us to understand the data well, regardless of posterior distributions or relations between input parameter configurations and accuracy.

For the use case of the Bayesian inference problem, two types of visualizations are necessary. First, we need to know how the individual parameter distributes in the posterior probability. The result of the Metropolis-Hastings is a multidimensional list of values, representing the calculated posterior. For each parameter, a separate histogram is generated. Alongside the histogram is the Kernel Density Estimation (KDE) graph. It is used to estimate the probability density function of a random variable based on a data sample by averaging contributions from specific kernel functions that are centered at each data point.[We18] By combining the histogram and the KDE graph, an overview of the general distribution of the posterior can be understood.

However, if the histogram and the KDE plot resemble the prior distribution, little or no useful information can be retrieved. Therefore, we can visualize the data using boxplots to get specific information on the distributions of the posterior of each parameter. The data we can read from the boxplot are the locations of different samples so that we can know how many samples are located under the first quantile, above the third quantile, or the median. It provides us with an easier understanding by explaining the locations of all of the samples in the posterior distribution.

After considering these parameters individually, we regard all of the parameters as a whole. In this case, we investigate whether specific parameters create dependency on each other. To deal with this, a heatmap is plotted, which shows the correlation of each pair of parameters. If the two parameters are too highly correlated, it would be possible that the samples of one parameter are dependent on another parameter. In this case, decoupling is needed to prevent entanglement, so that each parameter can sample on its own without the influence of other parameters.

At the very end of the parameters visualization, the calculated results are going to be compared. The four times series including posterior mean, posterior max, prior mean and measured data, which are mentioned before, are going to be plotted on the same graph so that the results can be visually compared.

The other type of visualization is presented during the exploration phase of the input parameters. The charts present the relations between the input parameter configurations and the metrics results. If the configuration is the form of numeric values, the line plot is drawn, so that a possible existing trend could be detected. If the configuration contains categorical values, then a bar chart is drawn instead. By plotting the metrics against the configuration, a clear comparison of the results calculated by different sets of input parameters is provided.

4.4. Knowledge-Based Input Parameter Selection

Before exploring the model's input parameters, we can determine the input parameters by logic previous knowledge, and logical thinking. In this section, the Bayesian inference

problem will be directly executed, as well as the visualization part. First, all different input parameters are going to be individually discussed.

The proposal distribution is, for the use case of the hydrological data set, relatively straightforward. Since there is no further information regarding the shape and look of the distribution other than the upper and the lower bound, a multivariate uniform distribution that ranges from the lower and the upper bound could be modeled and used as the proposal distribution.

The determination of the standard deviation of the sampling kernel is crucial to the result of the algorithm. Since the standard deviation should generally not be a bigger value than $\frac{1}{4}$ of the range,[And14] we start to find a maximum value of the factor going down from there so that the transition kernel of the algorithm does not go out of bounds too often, which causes numerical errors. After several test runs of the model, we found out that the first appropriate sampling kernel is a normal distribution with the standard deviation set as $\frac{1}{6}$ of the range, since it is neither too wide nor too narrow, which allows the algorithm to run smoothly and effectively reduce the amount of sampling out of bounds. This value is thus set as the default standard deviation.

The default standard deviation value is set to 1. The intention of setting the standard deviation to a relatively low value is to expect better precision. Since normal distributions with narrow standard deviations give out lower probabilities if the value is away from the mean, the samples that are far away from the mean will receive more penalties, if the standard deviation is set low. This results in a lower likelihood value, which might lead to a lower acceptance probability.

For the rest of the attributes: The initial state does not affect the accuracy of the result[GRS95]. Therefore, it is set as a random set of values within the uniform distribution for now. To increase the random effect, we generate 1000 random samples and take the mean as the random result. Optimization for efficiency is going to be performed later in this paper. For now, we simulate the Bayesian inference problem using Markov chain Monte Carlo for 10.000 iterations. As suggested, 20 percent of the data are discarded due to the burn-in phase.[BSSW$^+$23]

We execute the algorithm with the above-suggested input parameters. The execution was successful and produced decent results. Several graphs are produced to provide a great visualization of the result.

First, we take a look into the posterior distribution of individual parameters. The graph is shown in Figure 4.1. As we can see from the histogram and the KDE plot, the calculated posterior does not resemble specific distributions. They share a similarity, that is the probability of samples near the boundaries are relatively lower than the samples near the center. All of these parameters also have several peak values, which are sampled in the posterior distribution more than other values. All of the values are relatively widespread and evenly distributed.

Since little information is gathered from the above plot, we can visualize the data using boxplots to get specific information on the distributions of the posterior of each parameter. As we can see from the boxplots that are shown in Figure 4.2, the boundaries of the posterior distributions are retained. They share the same boundaries as the prior distribution. However, the first and third quantile as well as the median of the posterior distribution. This gives us a general idea of how the posterior would look like. For further application of Markov chain Monte Carlo algorithms, the starting values could potentially be altered based on this

information to improve efficiency. This aspect will be discussed later in this chapter.

After considering these parameters individually, we regard all of the parameters as a whole. In this case, we investigate whether specific parameters create dependency together using the heatmap. From the heatmap shown in Figure 4.3, we can see that none of the parameters are strictly dependent on each other. Specific pairs of parameters such as TT and FC do share more or less level of dependency, nevertheless, these are extremely low so this factor can be neglected. In this case, no specific changes to the likelihood functions need to be made.

After analyzing the parameters individually and as a group, the result that is computed by the posterior of the Bayesian inference is revealed in Figure 4.4 and compared to the observed data. The calculated result resembles the actual measured data, particularly the posterior mean. It reaches its peak in the same period as the measured data, whereas it shows stable behavior for the rest of the time, just like the the measured data. The posterior max shows slightly more extreme behaviors at certain points. Both of the posterior time series show significant improvement based on the prior mean.

After visualizing the parameters, we take a look at the accuracy of the data. For this part, the RMSEs and MAEs are calculated. The RMSE of the posterior mean is 22.14475485613421 and the MAE of the posterior mean is 11.399487080387862. From the graph, we can see that a significant difference in the results is shown around the peak, which might contribute to the relatively high MAE and a decent value of RMSE. The RMSE of the posterior max is 26.72454579307846 and the MAE of the posterior max is 13.196081237340492. They do not show too many differences from the posterior means, which means that the posterior distribution is relatively stable.

However, these values currently mean nothing, as they will be compared later to the results after the input parameters exploration. In the following sections, we are going to explore specific parameters of the Metropolis-Hastings algorithm to achieve maximum accuracy. Besides, the efficiency will also be taken into account, where the run time of the algorithm will be observed and compared.

4.5. Input Parameters Exploration

After finding the appropriate values by knowledge-based selection, all of the input parameters will be explored in this section. By trying out different reasonable values as input and interpreting the accuracy and efficiency results, we might be able to figure out a specific relation between the different input values against the accuracy or the efficiency score. Later on, the algorithm will be run using the set of input parameters which delivers the best performance by accuracy and efficiency metrics and be compared with the input parameters by knowledge-based selection on the testing data.

4.5.1. Sampling Out of Bounds

While executing this algorithm for the hydrological model, however, there is a certain issue. Since no specific information regarding the distribution is given, we are required to use the uniform distribution to describe the parameters that need to be calibrated. Since the uniform distribution ranges from a certain lower bound to a certain upper bound, it does not have an unlimited range. In this case, there is a possibility that the newly generated samples

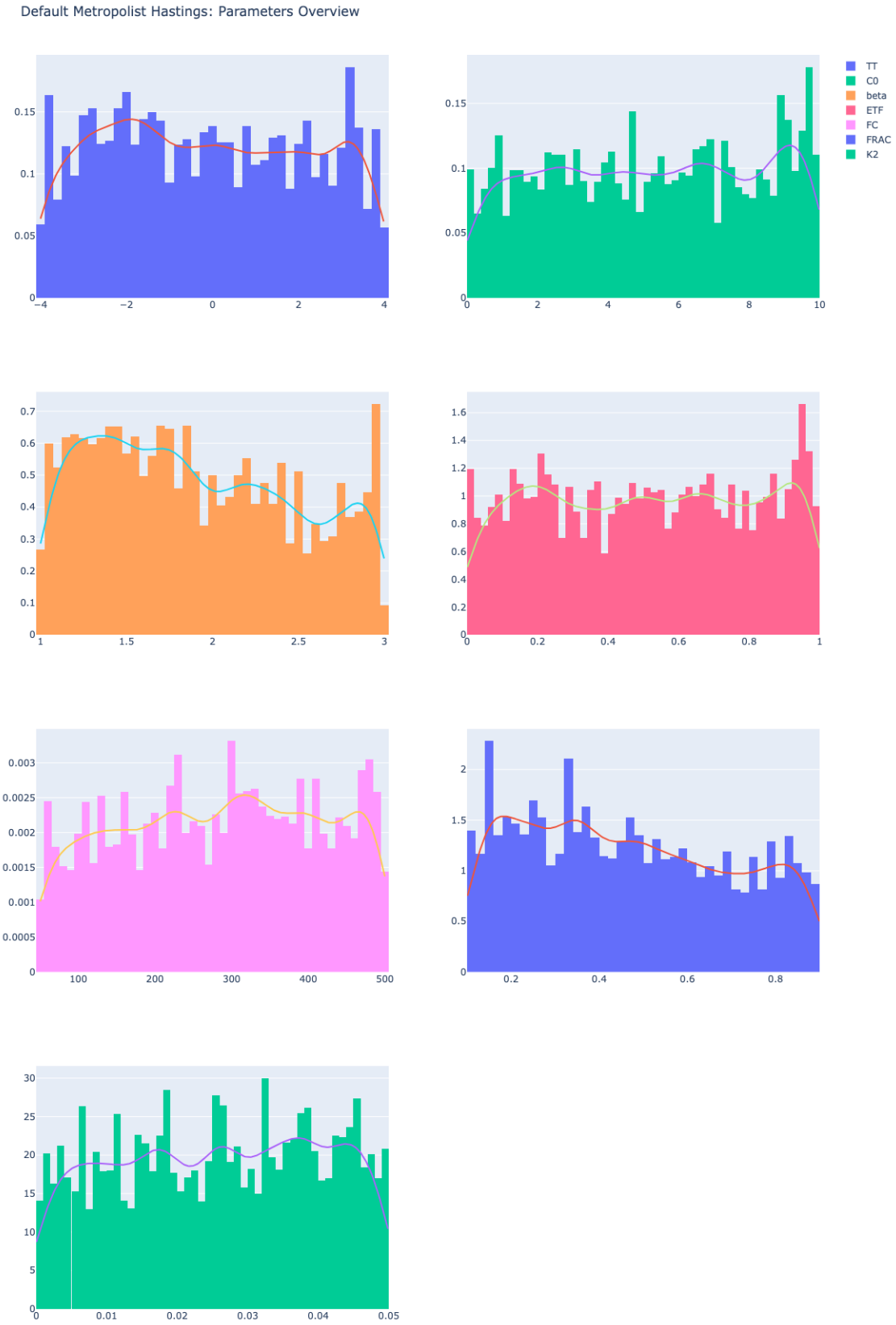


Figure 4.1.: Overview of the posterior distribution of the parameters calibrated by the default Metropolis Hastings algorithm

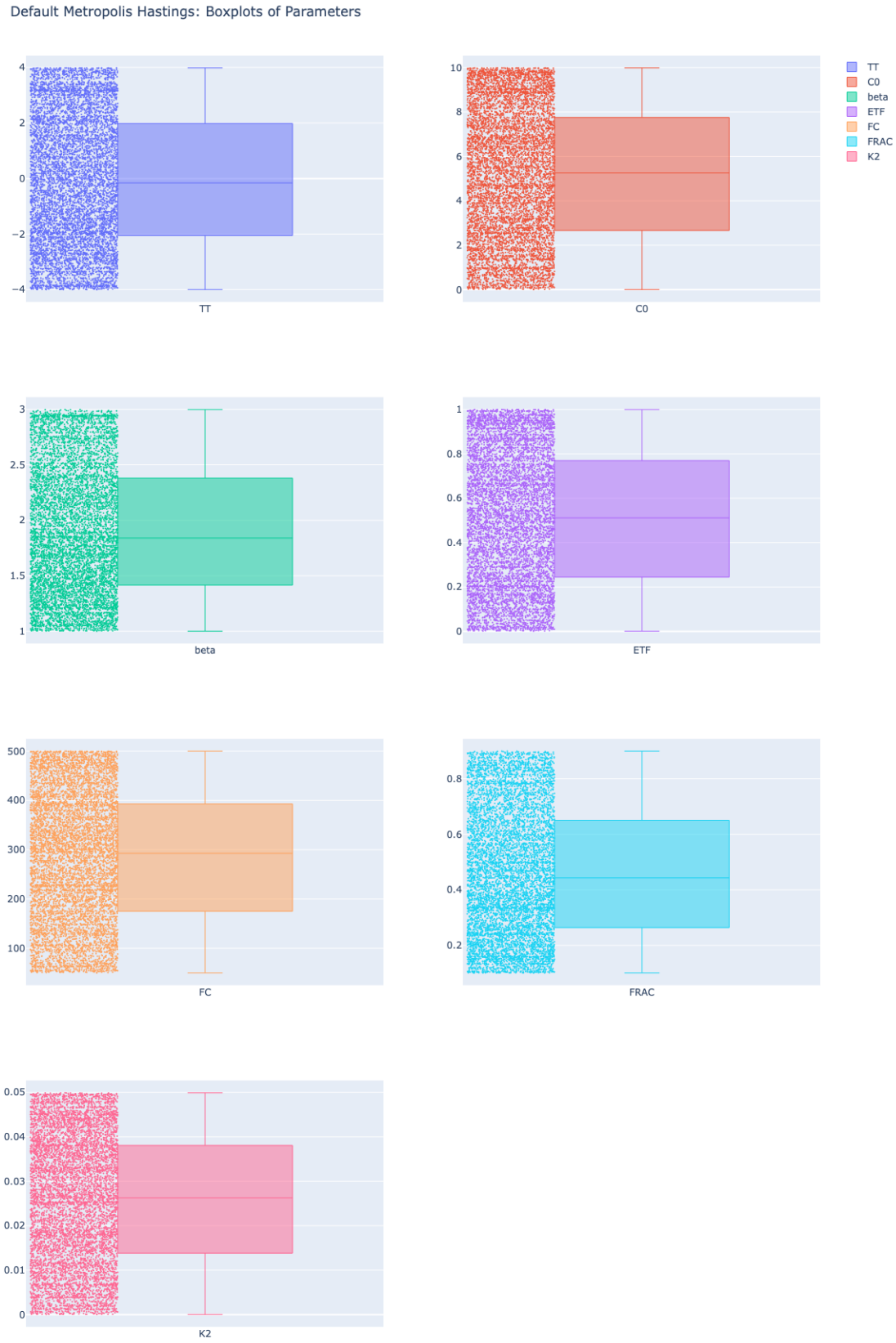


 Figure 4.2.: Boxplots of the generated posterior samples of each parameter calibrated by the default Metropolis Hastings algorithm

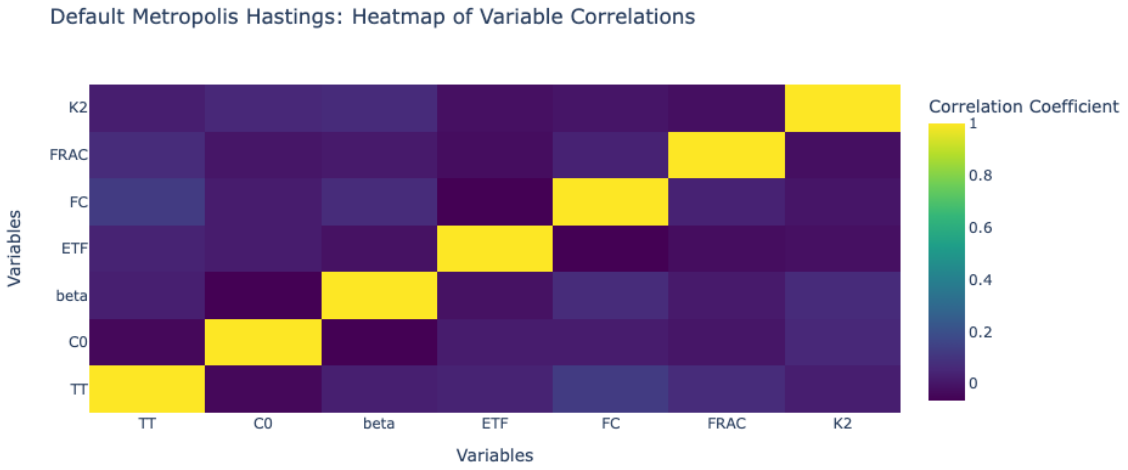


Figure 4.3.: Heatmap between parameters that are calibrated by the default Metropolis Hastings algorithm

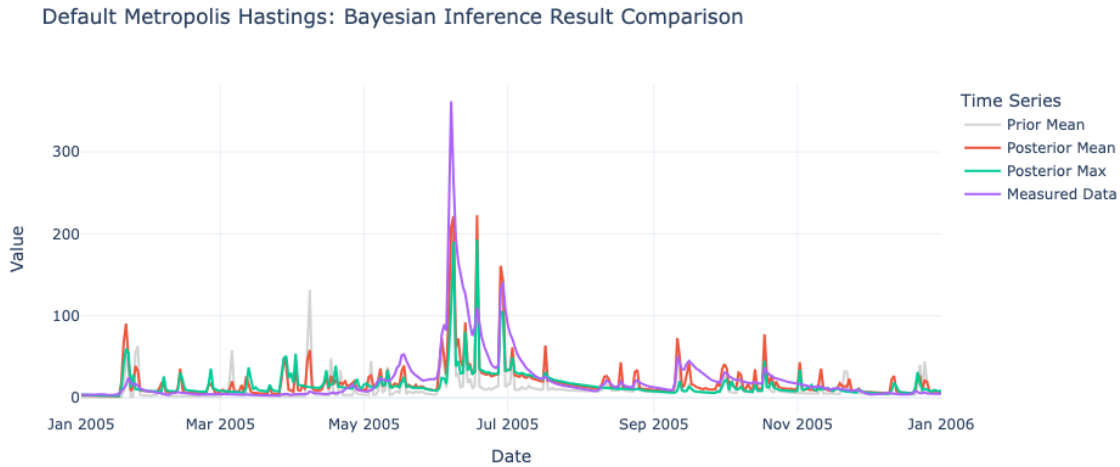


Figure 4.4.: Comparison of Bayesian inference results of the default Metropolis Hastings

are out of bounds, which is not helpful for the calibration. For example, if a generated point, which is accepted, is out of bounds, the p variable in the algorithm will be set to 0, which causes invalid values like negative infinity to occur in the calculation. If these values are sampled and carried on, values that are further from the bounds may be going to be sampled, which leads to mistakes in the result. Therefore, measures need to be taken to avoid these errors from happening. Three implementation variants against this issue have come up. These are ignoring, boundary aggregation, and reflecting boundary.

The ignoring method is the default method and the most straightforward: Any points that are generated outside of the bound are going to be eliminated since they are seemingly useless for the sake of uncertainty quantification. It is extremely important to mention that instead of taking another sample from the iteration, the last generated sample needs to be carried on. Since the sample out of bounds is supposed to be an impossible case, the acceptance rate in that point needs to be treated as 0, which means that this point is directly rejected. For multivariate distribution, a set of data points needs to be rejected entirely at once if one single data point is out of bounds since the acceptance rate otherwise would be different. Otherwise, the detailed balanced condition could be violated, as the transition probabilities would not be symmetric anymore for every single parameter.

The boundary aggregation method is different in treating the out-of-bounds samples, in which it transforms the sampled data that are out of bounds. Instead of reflecting the data that is out of bounds, we simply sample the upper bound or the lower bound and carry on from there. If a new sample is generated based on the normal distribution that is centered around the out-of-bounds sample, there is less or equal to fifty percent chance that the newly generated point is inside the range.[Do08] The main benefit of this method is that there is still a minimum of fifty percent chance that the next sample is kept inside the bound in cases where the samples are out of bounds. For multivariate distribution, a single data point in each dimension can be handled individually, however, the acceptance rate needs to be calculated based on the transformed sample points.

The posterior that is derived from the boundary aggregation method is shown in Figure 4.5. A few samples are getting aggregated on both sides of the boundaries. If we ignore these two bars on both sides, the distributions of the rest of the samples of each parameter still resemble uniform distributions, which does not provide a lot of information regarding the actual distribution of the parameter.

The boxplot of these samples is shown in the Figure 4.6. An obvious takeaway from this chart is that all of the boxplots have a lower first quantile and a higher third quantile, whereas the median is retained almost at the same place. The disposition of both of these quantiles is the result of the aggregation of samples on both sides of the boundaries.

The reflect boundary method also transforms the data instead of ignoring it but in a different way than the aggregation method. Due to the symmetry of the transition kernel, the possibility of sampling an out-of-bounds sample is the same as the possibility of sampling the point symmetric over the mean of the Gaussian normal transition kernel.[Do08] In this case, the acceptance rate of the transformed sample point is not changed, which has no interference with the algorithm itself and future samples. For multivariate distribution, a single data point in each dimension can be handled individually and the acceptance rate does not need to be recalculated due to the symmetry of the transition kernel distribution.

We now take a look at the posterior distribution of the samples. It is shown in Figure 4.7. From the graph, we can see that the posterior distributions sampled from this version

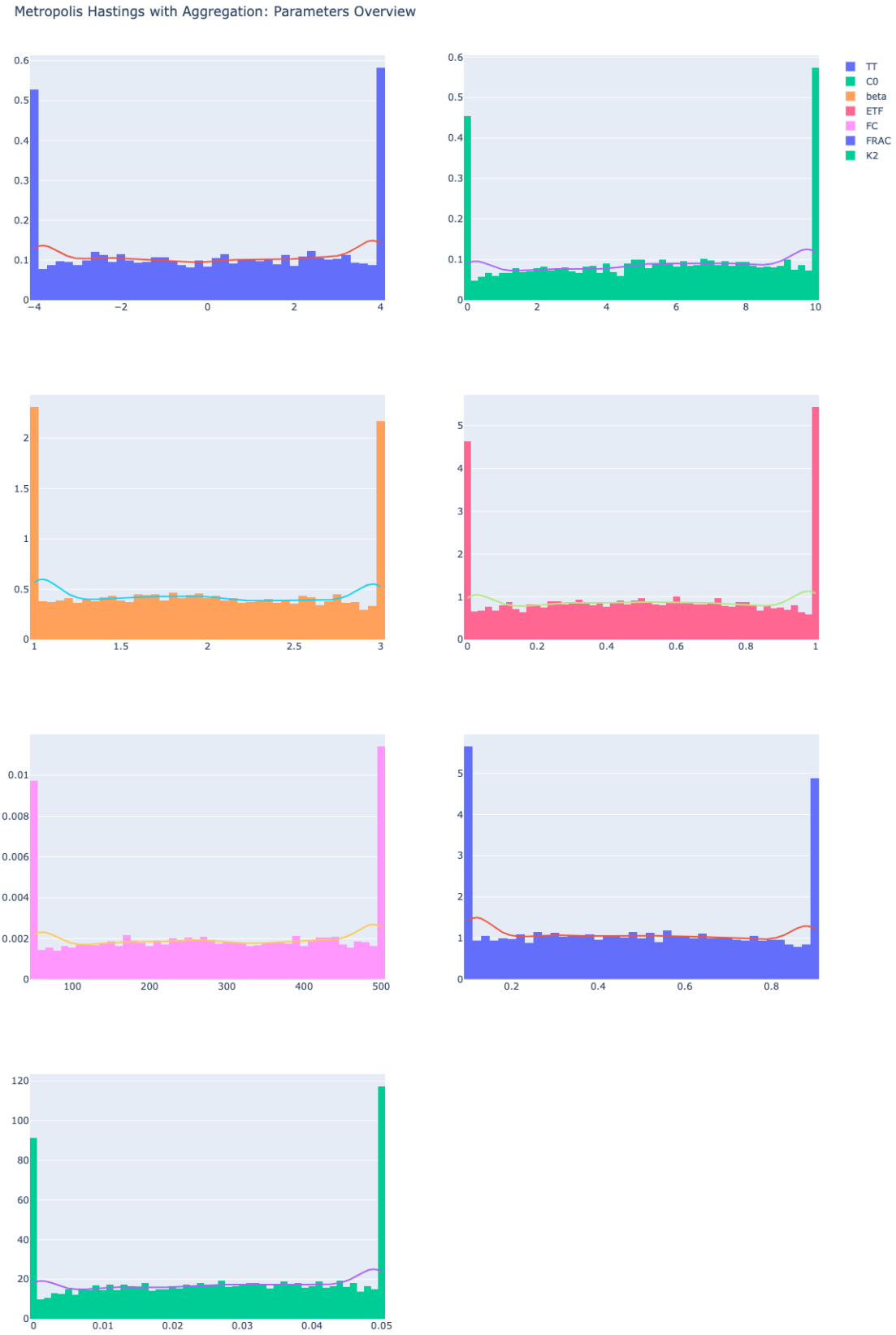
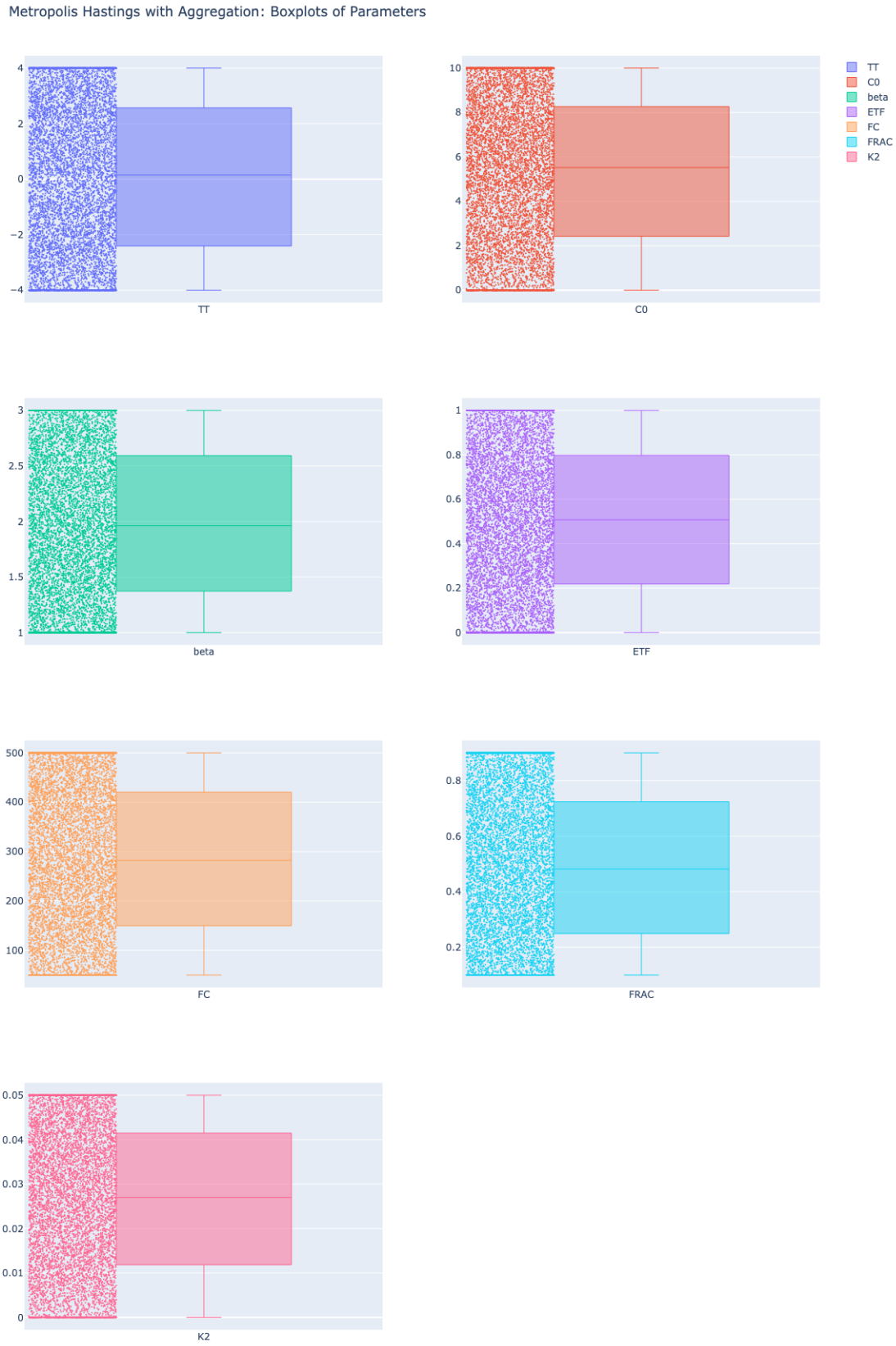


Figure 4.5.: Overview of the posterior distribution of the parameters calibrated by the Metropolis Hastings algorithm that samples the bound value if the sample is out of bounds



26 Figure 4.6.: Boxplots of the generated posterior samples of each parameter calibrated by the Metropolis Hastings algorithm that samples the bound value if the sample is out of bounds

of Metropolis-Hastings look very much different from the others. All of these posterior distributions resemble normal distributions, with one peak somewhere in the middle. For some parameters like C0, beta, ETF, and TT, the boundary is shifted, which means that no samples from the region near the boundaries are generated. Since all of the samples that are out of bounds are reflected, the reflected samples compensate the holes of the nonreflected samples, so that it gives rise to a normal distribution like posterior.

The boxplots of the parameters are shown in Figure 4.8. Due to the normal distribution like posterior, all of the boxplots shrink by a certain amount, with some having lower upper bounds or upper lower bounds. This is an apparent result, since the shape of the normal distribution focuses on the mean, whereas the sampling probabilities of samples that are further from the mean are lower. This property can be presented by the boxplots.

To find out which of these three variants delivers the best result, each of these three versions is separately executed, while all of the rest input parameters are set to the same value or instance. The different metrics that are mentioned in the section above are then calculated and visualized using a bar chart so that the values can be compared.

The result is shown in Figure 4.9. The first impression of the bar chart is that the accuracy of the actual inferred results is pretty similar among all three versions, with the ignoring and the aggregate methods performing only slightly better. For efficiency, however, the ignoring performs better than both of the other methods by a huge margin. The reason behind it is obvious: the ignoring operation is way more efficient than the reflecting boundary and aggregation, which involves mathematical operations. Therefore, the default ignoring method is the clear winner here and should be retained for further model executions.

4.5.2. Sampling Kernel

The sampling kernel plays a crucial role in the accuracy and efficiency of the performance. Since we stick to the normal distribution in this paper, the standard deviation is the only value that needs to be explored. As suggested, the default value of the standard deviation is set to 6 because it is the largest possible number that fits in the use case of the hydrology model. However, the final result would also be different if the standard deviation is less than the optimal one. In this case, the movement of the samples is going to be relatively centered local, since the points in the vicinity of the mean are more likely to be sampled. To test multiple scenarios and their behaviors, three values for the standard deviation are going to be tested in the following execution: the range interval over 8, over 10, over 12, over 18, and over 24.

TODO

4.5.3. Likelihood Functions

At the start of the chapter, we have already discussed the importance of the role played by standard deviations in the likelihood functions. As mentioned before, 1 is selected for the default value. However, the choice of the standard deviation does impact the final result, since it exerts an influence on the sampling probability of the values based on their distance from the mean. If the standard deviation is set lower, the sample that is away from the mean will receive less probability. If the standard deviation is set higher, that sample will not receive that little probability, which allows more tolerance to be present in the calculation.

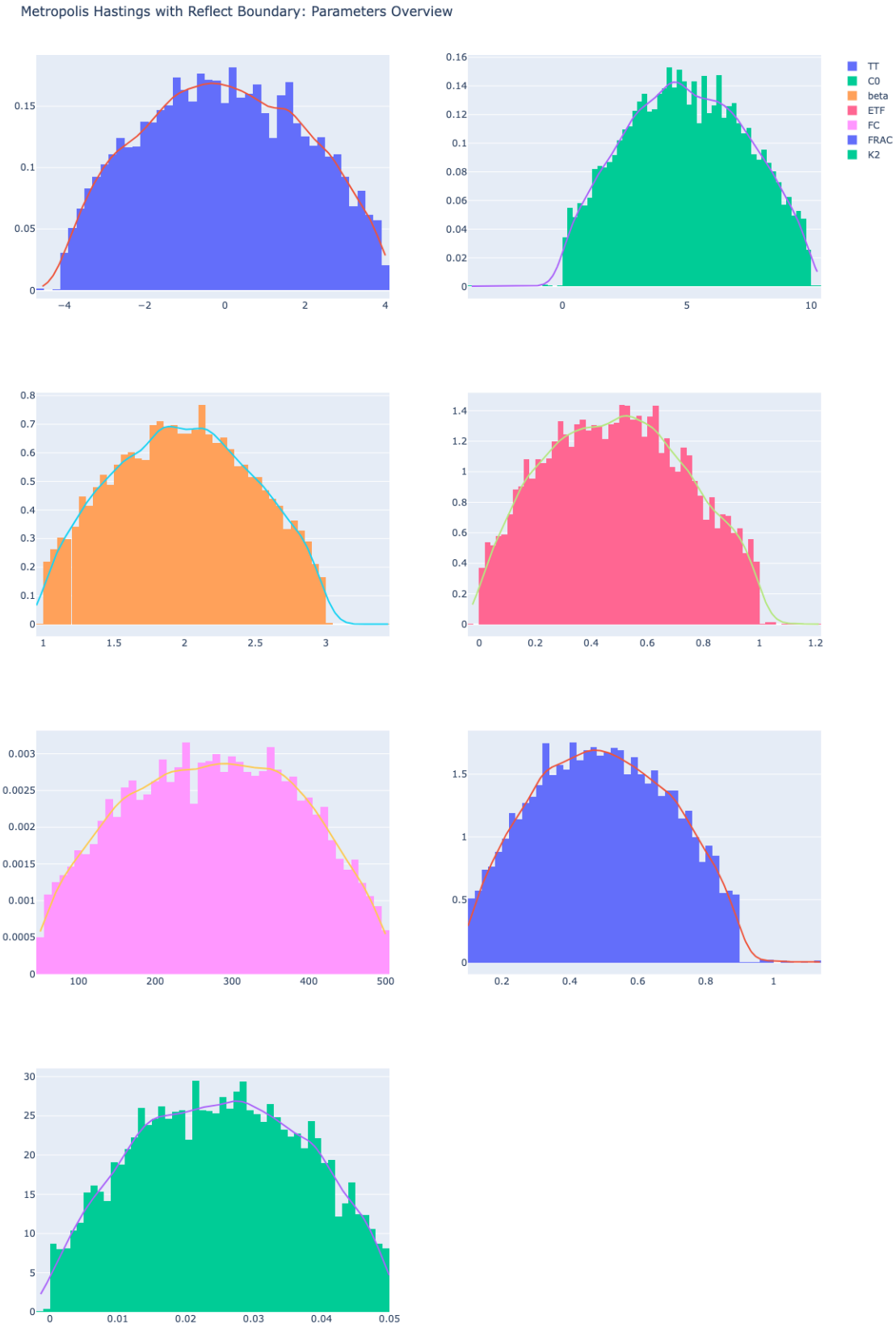


Figure 4.7.: Overview of the posterior distribution of the parameters calibrated by the Metropolis Hastings algorithm that reflect the samples into the inside of the range if the they are out of bounds

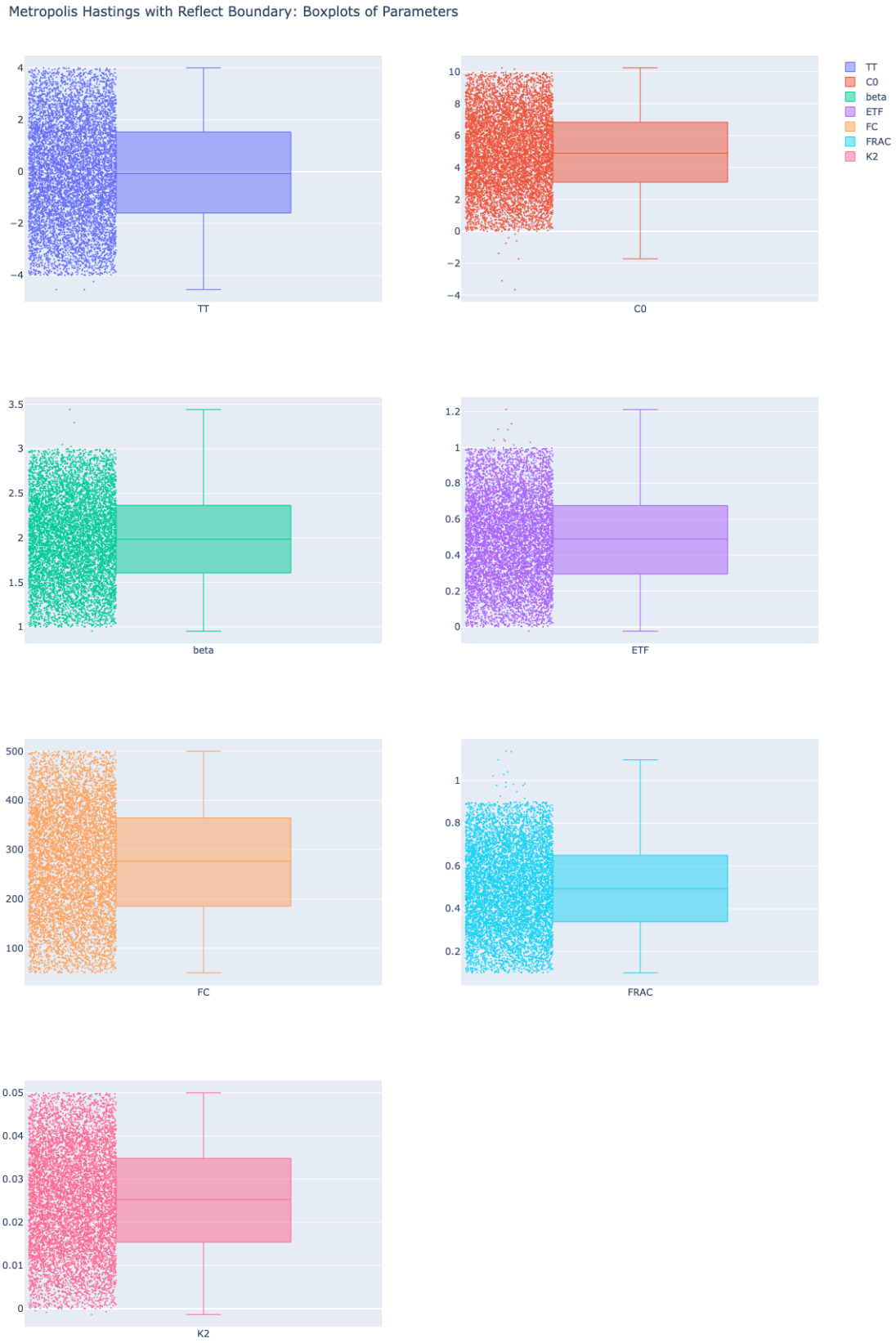


Figure 4.8.: Boxplots of the parameters calibrated by the Metropolis Hastings algorithm that reflect the samples into the inside of the range if the they are out of bounds

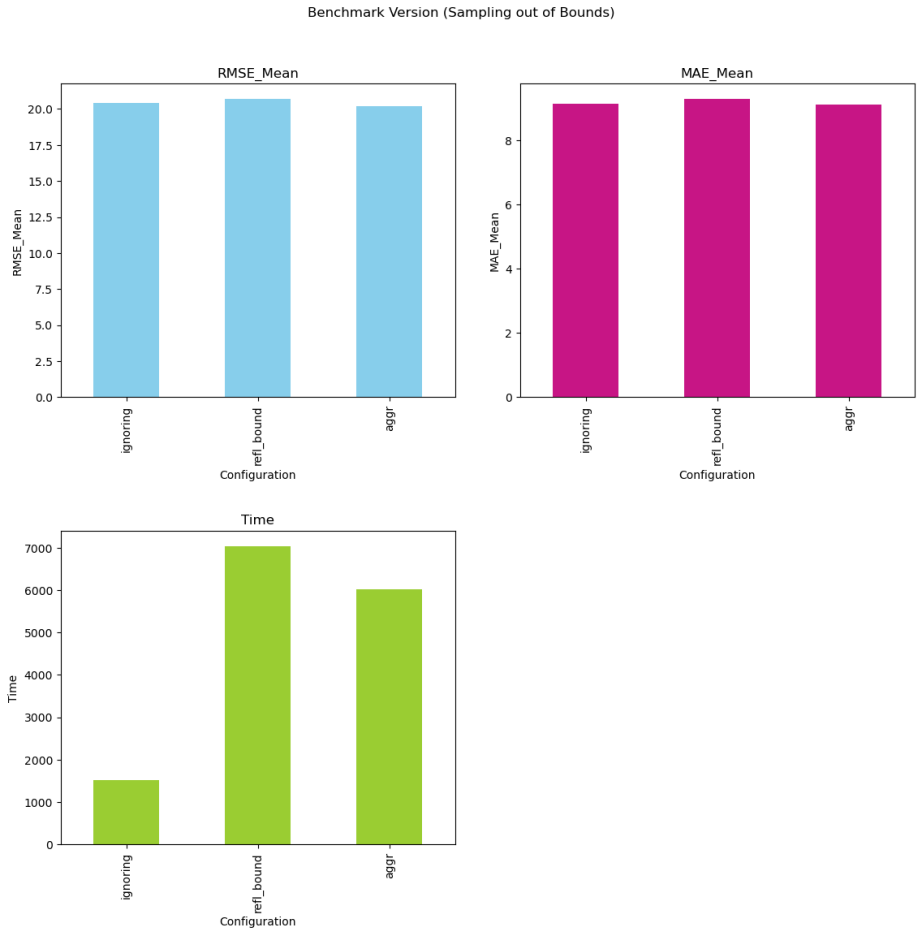


Figure 4.9.: Comparison of the accuracy and the efficiency of different Metropolis Hastings algorithms

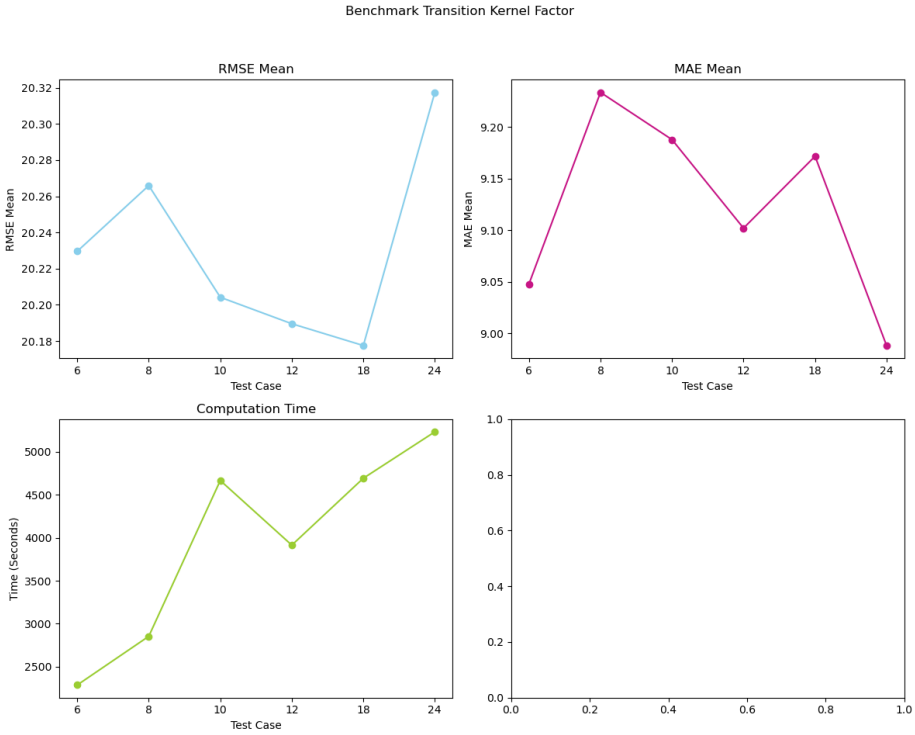


Figure 4.10.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the sampling kernel standard deviation

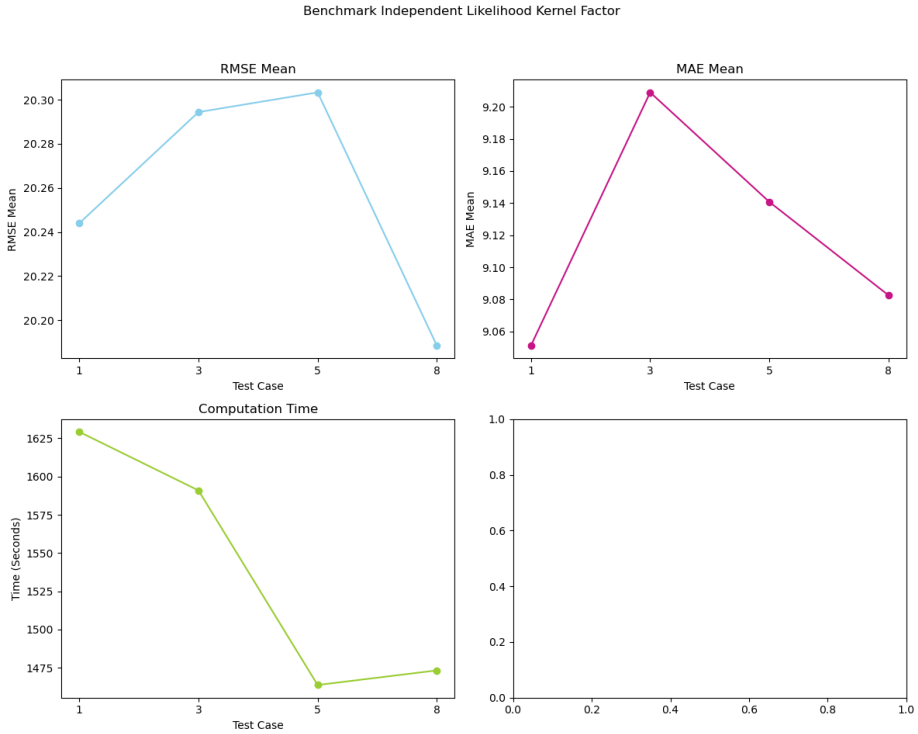


Figure 4.11.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the likelihood function standard deviation

However, the value cannot be too large, otherwise, the tolerance level will be too high for the likelihood function to give out a meaningful solution. Thus, for the test values, we go down from 1 down to 8, which is an interval of values that still might generate meaningful calculations. The selected values for testing are 1, 3, 5 and 8.

As we can derive from Graph 4.11, the metrics for the mean also do not differ that much from each other. In this case, the conclusion could be drawn that the standard deviation does not have a huge impact on the actual inferred result. For the inferred maximum time series, the discrepancy is also not obvious, even though the test case 5 shows the most instability throughout the posterior. Nevertheless, using the test case 5 results in the most efficient calculation, whereas using the test cases 1 and 2 will require slightly more computation time.

The above-computed likelihood function takes in a value as the standard deviation so that each sample is independently observed from another. However, this standard deviation can potentially be even more optimized if the standard deviation is based on some value that is meaningful for the final calculation. If there is somehow a correlation between the likelihood function and the observed data, which provides an absolute exact value, we might derive the information regarding the correlation between the inferred data and the observed data, so that the calculated likelihood could be more precise. Therefore, we introduce here the dependent likelihood function, which takes in the observed data as the standard deviation. Since the observed data might be too large for the likelihood function to deliver meaningful results, an alternative way is to take a certain factor of the observed data as input. Here,

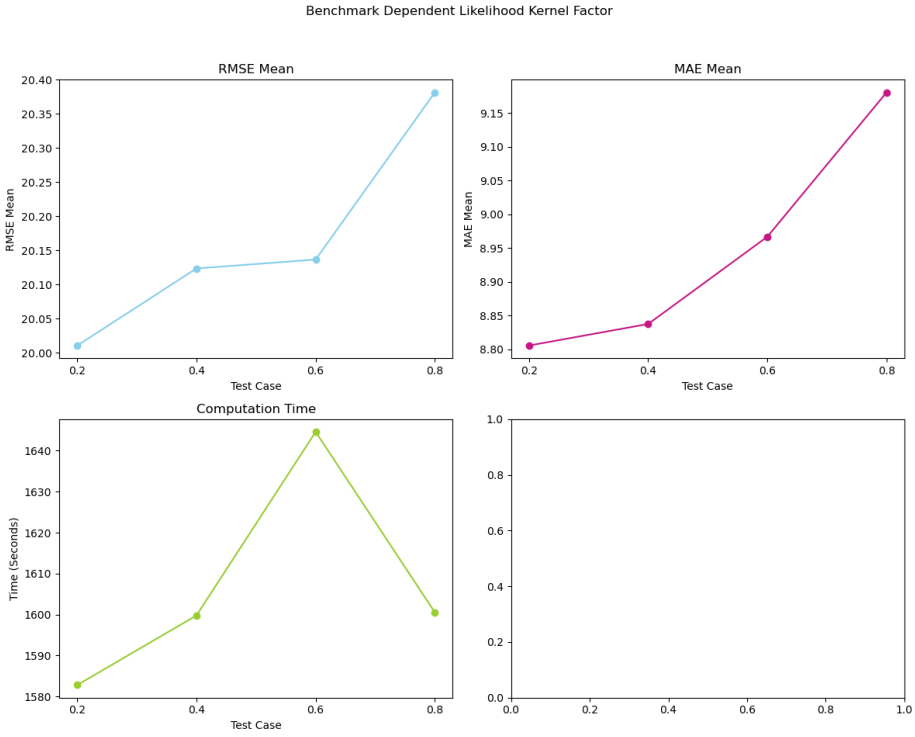


Figure 4.12.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the dependent likelihood function standard deviation

several factors are going to be tested, including 0.2, 0.4, 0.6 and 0.8. The goal is to observe the correlation between the accuracy and the value as it increases.

The result is shown in Figure 4.12, where we can see a pattern: the accuracy of the inferred data decreases as the factor increases. As for the computation time, the case of 0.6 requires the most time to perform computation, and is, however, still pretty similar to the rest. Therefore, the case 0.2 should be the optimal choice.

Comparing the dependent and the independent version of the likelihood functions, the dependent version with a standard deviation of 0.2 times the measured data outperforms the independent version with a standard deviation of 5, despite a slightly longer run time, which is reasonable due to the mathematical computation of the standard deviation. The incorporation of exact data in the likelihood function does provide a more accurate result.

4.5.4. Alternative Implementation of Probability Acceptance Rate

After we explore both of the input parameters that are responsible for the calculation of the acceptance rate, we move on to the selection of the acceptance rate. Due to the multi-variate property of the HBV-SASK model, the calculated acceptance rates will be formed into an array. To transform this array into a value, there are two choices: One option is that we take the mean of all the values as an acceptance rate, so that the final acceptance rate could more generally represent all of the individual acceptance rates by parameter. Another option

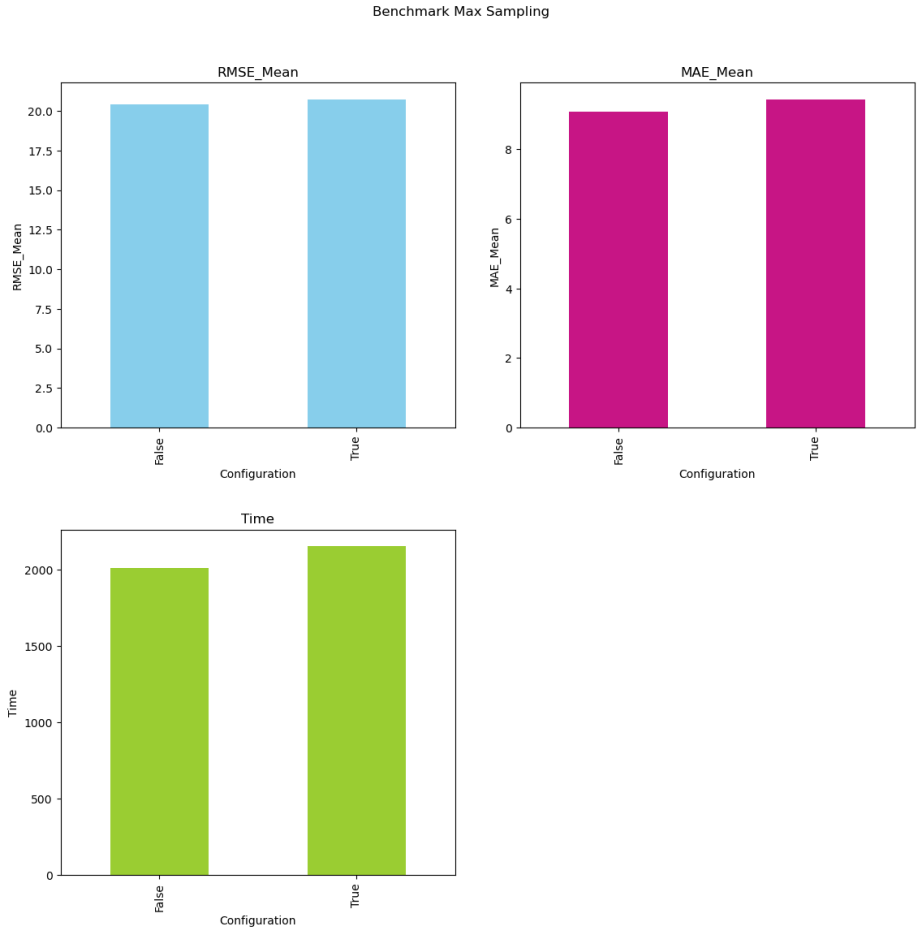


Figure 4.13.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the implementation of probability acceptance rate calculation

is to take the maximum value of the entire acceptance rate array. On the one hand, we can improve the efficiency of the algorithm, since the calculation of the mean is avoided. On the other hand, some dimensions might be easier to sample from than others due to less complexity. Using the maximum acceptance rate gives us therefore the insight of the entire parameter space, where the parameter with the best performance decides the acceptance rate. However, the maximum acceptance rate might be misleading if the distribution of the acceptance rate array is too widespread, which results in the complete opposite of efficiency and accuracy.

We execute the algorithm in both versions, with the rest of the parameters being identical. Graph 4.13 suggests that for the HBV-SASK model, the mean sampling method does not only deliver a slightly better performance in accuracy but also more stability and most importantly: better efficiency. This shows that the acceptance rate in each iteration in the array might have far different values so the max sampling method cannot deliver good enough results. Therefore, we stick to the original mean sampling method.

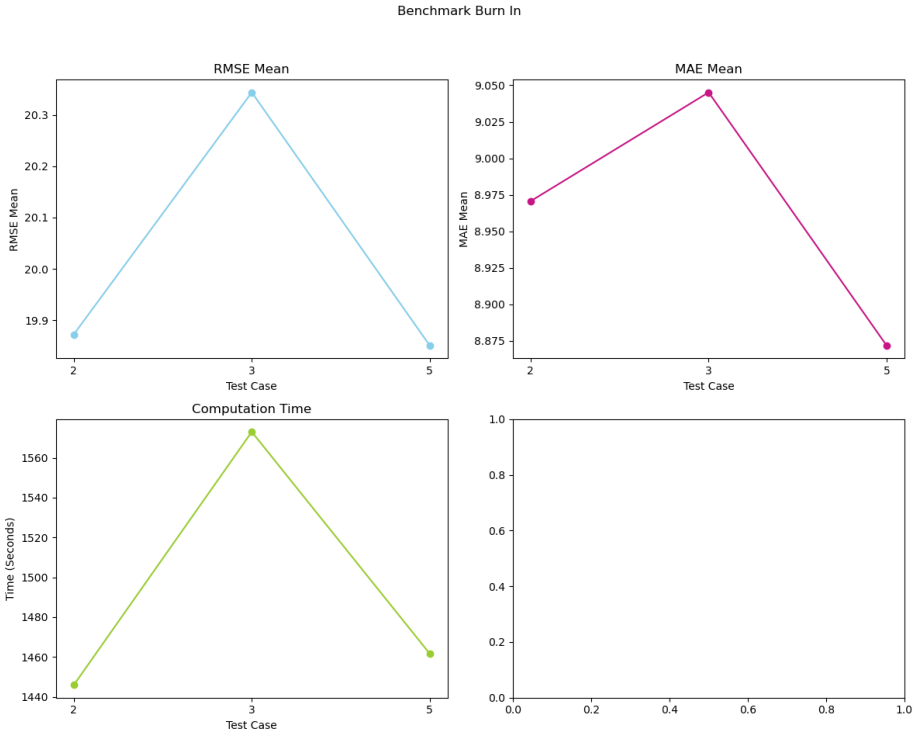


Figure 4.14.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on length of burn in phase

4.5.5. Burn In Phase

Another crucial part of all Markov chain Monte Carlo algorithms is the discard of the burn-in phase. For Metropolis-Hastings, it is no exception. Determining a general optimal burn in phase leads to an optimization of the result since it is the process of removing generated samples that do not follow the stationary distribution and are unstable. The algorithm was previously executed with a burn-in phase of 20 percent, which means that the first twenty percent of the generated sample data are discarded. However, a comparison to other values of the burn-in phase would give us an overview of how effective the 20 percent is and whether it is enough. The values that are used to compare are 33 and 50 percent. On the graph, the values 2, 3, and 5 denote the denominator of the burn in phase, as they should be interpreted as $\frac{1}{2}$, $\frac{1}{3}$ and $\frac{1}{5}$.

Graph 4.14 infers that the case 50 percent and the case 20 percent show relatively similar accuracy and efficiency. However, because removing half of the samples might be a bit too much and the similarity in performance, 20 percent is a better choice. Therefore, for the rest of the execution phase, we keep discarding the first 20 percent of the samples as the burn-in phase.

4.5.6. Effective Sampling Size

Effective sampling size is a new concept that has not been introduced in this paper until here. It generally measures the number of independent samples equivalent to the correlated

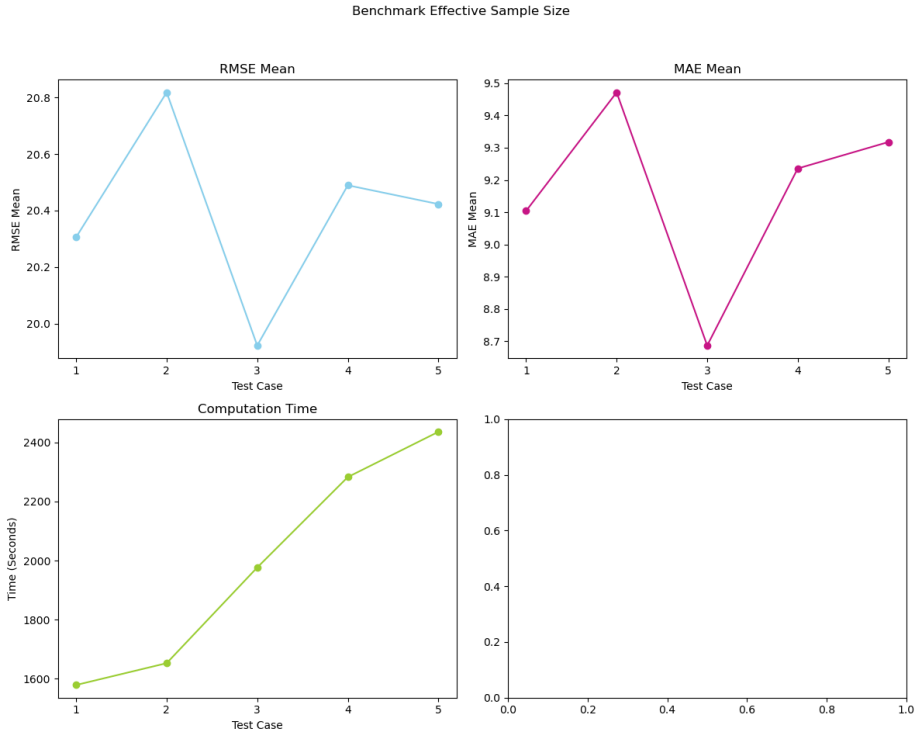


Figure 4.15.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the selection of effective sample size

samples obtained from the sampling process, which means that it provides us with the number of samples that are independent of each other.[BBP14] The most basic way to implement an effective sampling size in the Markov chain Monte Carlo algorithms is to skip a couple of samples and only regard every several samples in our sample space. The reason to do this is that in Markov chain Monte Carlo algorithms, every sample is dependent on the last sample generated. However, this dependency might have a higher degree of influence, for instance: the newly generated sample is most likely to lie inside of a certain range depending on the standard deviation. Therefore, not considering the sample directly after another might lead to a certain level of independence, which gives rise to more generalization of the sampling space.

For testing purposes, we compare the algorithm that does not include the effective sampling size feature with algorithms that only consider every second, third, fourth, and fifth sample for the sampling space. The result is shown in Figure 4.15. The efficiency is proportional to the value set for the effective sample size, whereas no pattern could be found for the accuracy aspect. Even though only considering the third sample is relatively less efficient than considering every second sample or even not implementing this feature, it provides better accuracy than all the other cases and relatively better stability due to the low RMSE and MAE of the maximum time series. Therefore, it would be wise for us to keep retaining only the third value from the sample space in the later execution of the Metropolis-Hastings algorithm.

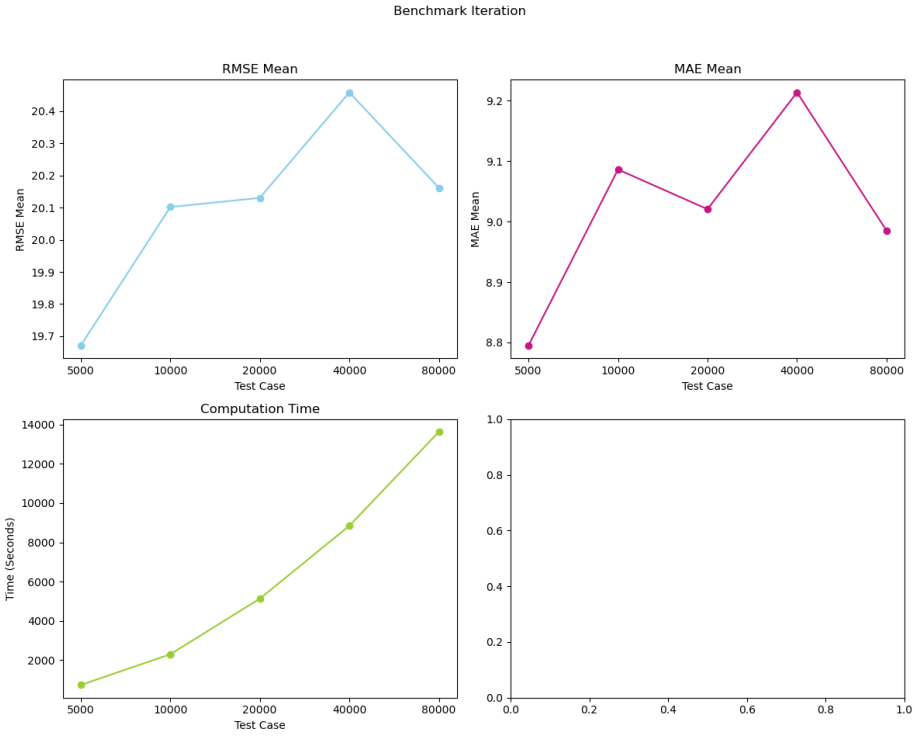


Figure 4.16.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the number of iterations

4.6. Iteration

Another crucial part of the Metropolis-Hastings algorithm is the amount of iterations. More iterations mean that we can gather more samples. To understand whether the amount of samples that are gathered is enough or not, comparing the accuracy with other numbers of iterations is needed. Until now, we performed the Metropolis-Hastings algorithm with 10000 iterations. Now we compare this number to other iterations like 5000, 20000, 40000, and 80000 to find out which number of iterations would deliver the best result and give a decent efficiency.

The result is shown in Figure 4.16, where it is clear to observe that the computation time grows proportionally to the number of iterations. The case of 5000 delivers the best accuracy and efficiency but might lead to too small of a sample space due to the removal of the burn in period and effective sample size. Of all of the rest cases, they share a similar range of accuracy. However, due to the efficiency of run time, more than 10000 iterations will be an overkill. Therefore, we continue to execute the algorithm with 10000 iterations.

4.6.1. Initial States

The last input parameter that needs to be explored is the initial states. The initial states should not have that much of an effect on the accuracy, but rather a great influence on the efficiency.[GRS95] A better initial state would allow the algorithm to get rid of the burn-in

phase and sample from the stationary distribution sooner, which reduces calculation burdens. It also allows the sampling kernel to discover more samples from the optimal ranges.

Several possible initial states should be taken into consideration. The most general one is the random initial state, used when there is little information regarding the distribution available. To do this, we sample a random state from the posterior and start from here. For testing purposes, however, we randomize 1000 samples and take the mean of them to maximize generalization and randomization. The lower and the upper boundary as the initial values would also work, which requires no initialization at all. Other possible values are derived from prior and posterior distributions. We try the first quantile, the mean, and the third quantile of the prior so that we can figure out whether an optimal starting value is coincidentally near these points. However, the focus point should be on the following three initial states: the first and the third quantile of the posterior distribution as well as the median of the posterior distribution. In the third section of this chapter, we generate a primary result that provides a general result of the posterior distribution. To start the entire algorithm from an inferred posterior state might result in better entry into the algorithm since the starting states are already proven to be very possible on the stationary distribution. Instead of taking the mean of the posterior, we select the median because it represents the half position of the entire posterior distribution, whereas the mean only represents the middle value. It is expected that the most desirable solution comes from one of the proposals of the posterior.

We execute the model with all these different initial states and receive the results shown in Figure 4.17. The RMSE and MAE of the mean time series over all of the test cases prove that the performance is not influenced by the initial states. However, the efficiencies of the algorithms with different initial states do have a massive difference between them. The maximum initialization performs well, as well as the third quantile of the prior, the third quantile of the posterior, and the median of the posterior. After checking the numerical statistics, the median of the posterior delivers the most efficiency as expected. Therefore, it will be set as the default initial state from now on.

4.7. Result Comparison

After evaluating all of the input parameters, we will compare both sets of input parameters by using the Monte Carlo simulation. Using the set of input parameters that deliver the best performance by accuracy metrics in the exploration phase, which will, later on, be called the tuned input parameters, the model will be executed 1000 times, where the RMSE and the MAE of the result time series mean and maximum will be calculated. These results are going to be then compared to those of the models that run on the knowledge-based set of input parameters so that the set of input parameters that delivers better results will be selected to represent the basic Metropolis-Hastings method.

We first draw the histogram and the KDE plot for all of the parameters. It is shown in Figure 4.18. We can observe that the distributions fluctuate more than the parameters from the model that uses the knowledge-based input parameters. In this case, some of the parameters still show irregularity, whereas some of the distributions do show some certain level of resemblance to normal distribution, such as C0, FC, FRAC, and K2. Similar to the case before, the probability of sampling values near both boundaries is relatively low.

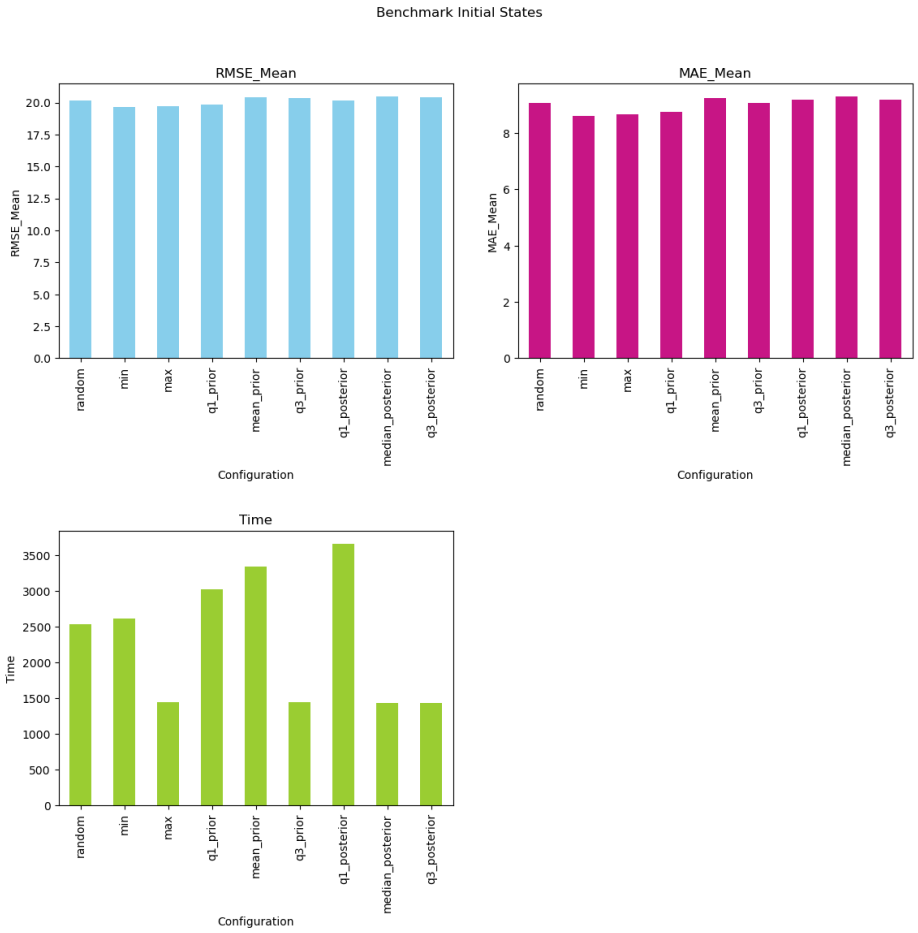


Figure 4.17.: Comparison of the accuracy and the efficiency of Metropolis Hastings algorithms based on the selection of initial states

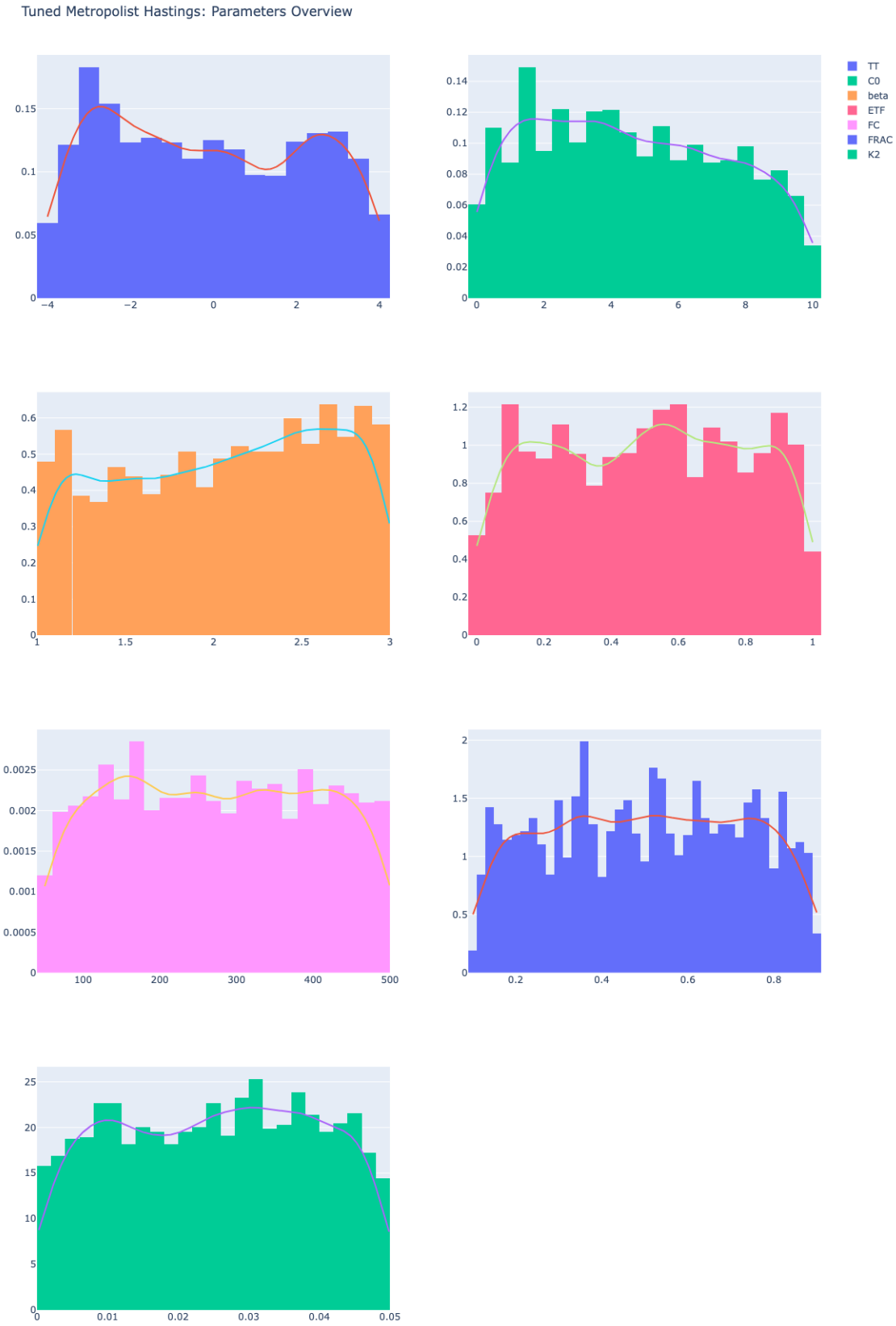


Figure 4.18.: Overview of the posterior distribution of the parameters calibrated by the Metropolis Hastings algorithm with tuned input parameters

The boxplots of these parameters are shown in Figure 4.19. We can see that the ranges of most of the sampled parameters are different from the knowledge-based version. The beta and the C0 parameters have moved completely towards the lower bound, whereas the FRAC parameter has moved upwards. The ETF parameter retained its lower bound but had a lower upper bound. In contrast, The TT parameter retained its upper bound, however had a lower bound.

The heatmap of the parameter space in Figure 4.20 shows that there are still little to no correlations between each pair of parameters.

From these generated samples, we retrieve the following result of the Bayesian inference problem shown in Figure 4.21.

To evaluate the result, we keep using the metrics that were calculated before for the model using the knowledged-based set of input parameters, namely RMSE and MAE. The same calculations are executed on the mean and the maximum of the inferred time series. The RMSE of the posterior mean is 21.974609013782757 and the MAE of the posterior mean is 11.457657543376751, whereas the RMSE of the posterior max is 24.458078992931487 and the MAE of the posterior max is 14.532783129590447. The model using the tuned input parameters performs better than the model using the knowledge-based input parameters in some metrics, but not the others. Besides, there is randomness in the Monte Carlo algorithm,[KU88] which contributes to slightly different values in every single execution.

Therefore, to generalize the accuracy of the result, we run both models 100 times and gather the mean of all the metrics of the results.

Metric	**Knowledge-Based Posterior Mean**	**Tuned Posterior Mean**
RMSE	22.122504129857315	22.124942509212538
MAE	11.400067417022779	11.600318945622558

From this table, the knowledge-based input parameters achieve an all-around better performance, but not by much. While the RMSE does not differ from each other that much, the knowledge-based input parameters provide a slightly lower MAE. This might be the case that the inferred time series run by the model using the tuned input parameters performs slightly poorer in predicting extreme data points.

Therefore, we select the knowledge-based input parameters as the default input parameters for the Metropolis-Hastings algorithm for further usage in this paper.

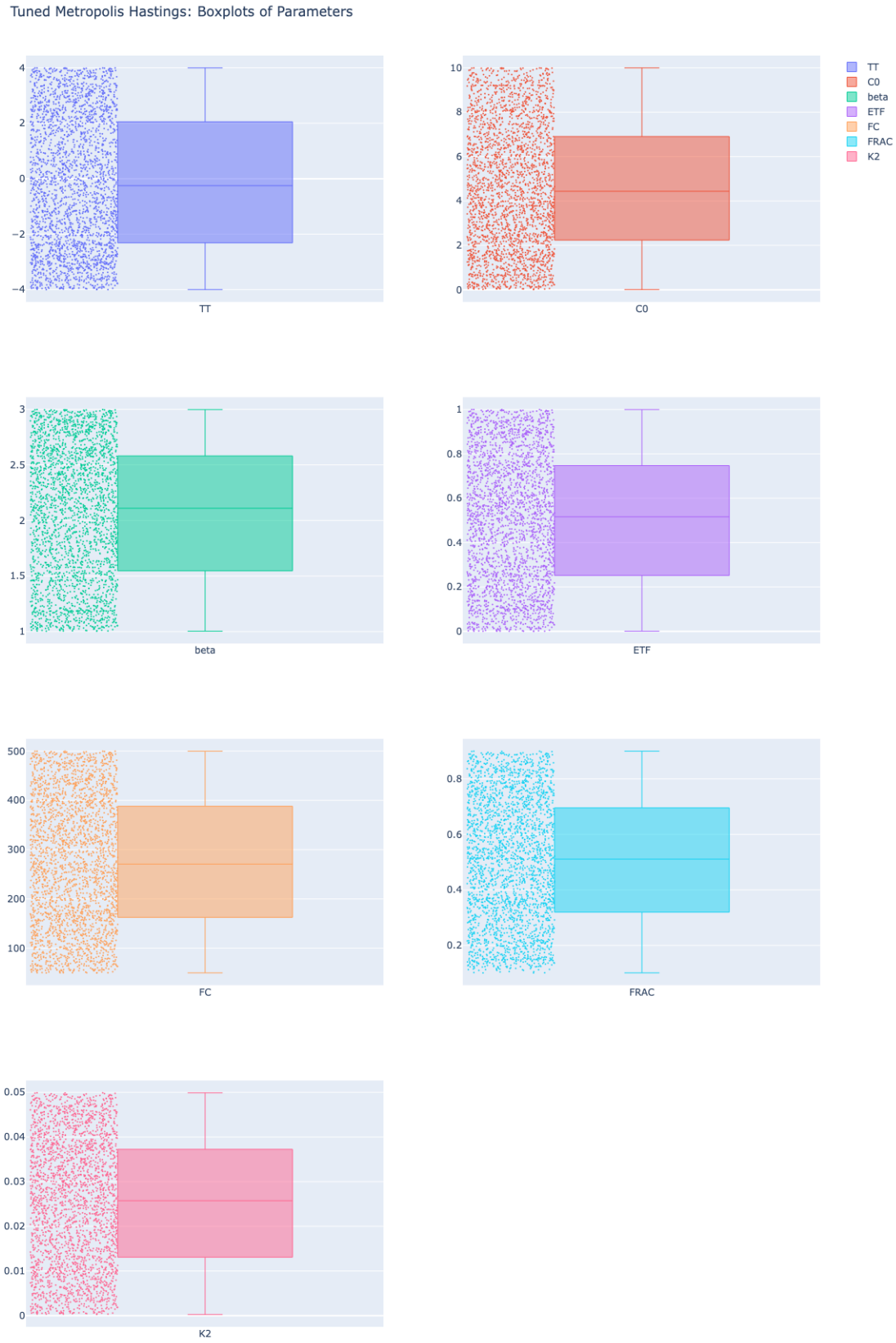


Figure 4.19.: Boxplots of the generated posterior samples of each parameter calibrated by the Metropolis Hastings algorithm with tuned input parameters

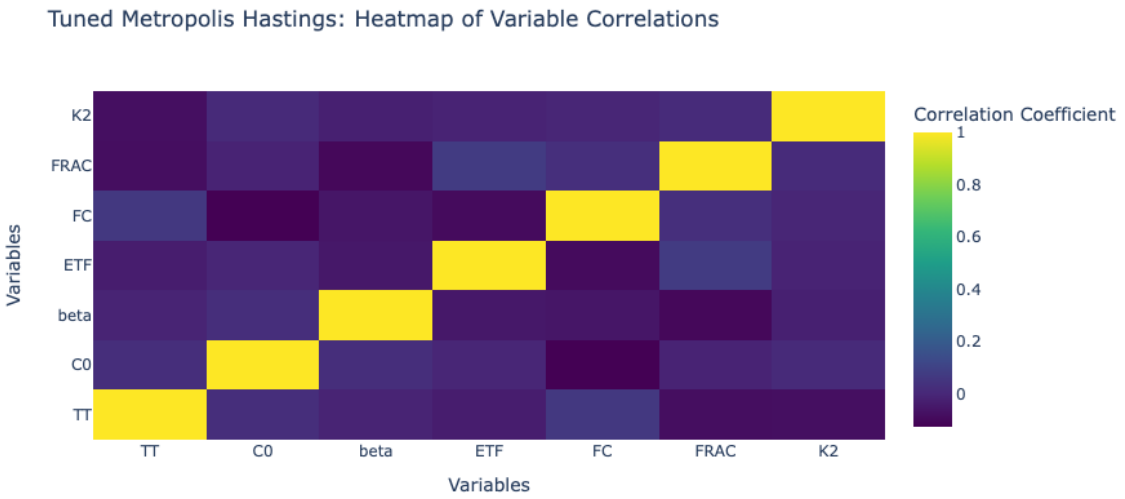


Figure 4.20.: Heatmap between parameters that are calibrated by the Metropolis Hastings algorithm with tuned input parameters

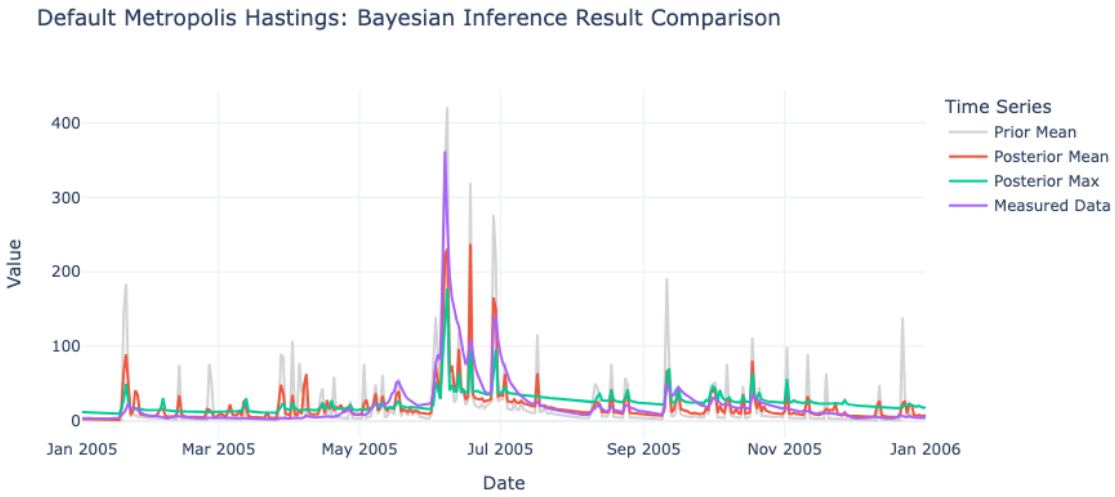


Figure 4.21.: Comparison of Bayesian inference results of the Metropolis Hastings with tuned input parameters

5. Parallel Metropolis Hastings

5.1. General Idea of Parallel Metropolis Hastings

Explain: Convergence

5.2. Numbers of Chains

5.3. Evaluation of Individual Test Cases of Chain Numbers

5.3.1. One Single Chain

5.3.2. Two Chains

5.3.3. Four Chains

5.3.4. Five Chains

5.3.5. Eight Chains

5.3.6. Ten Chains

5.4. Comparison of Parallel Metropolis Hastings with different Chain Numbers

6. Introduction

Write some useful intro. Here are tips along the way:

6.1. Tips

6.1.1. How to Describe

When listing several points you have three basic options:

- itemize
- enumerate
- description

1. itemize
2. enumerate
3. description

itemize short, unordered

enumerate short ordered

description listing of descriptions. Also nice for longer ones.

6.1.2. How to Quote

"This is a quote!"

- Citations to a source can be made like this `\cite{gratl17task}` = [**?**]

  Always join text and the citation with a non-breaking space: `text~\cite{foo}`.
- Referencing Sections, Figures, Tables, Formulas: `\autoref{sec:intro}` = Chapter 6.
- Footnotes for url or further notes: `\footnote{\url{https://www.top500.org}}` = [1]

6.1.3. How to Math

Use the align environment for equations especially if you want to align them somehow.

$$1 + 1 \neq 3 \tag{6.1}$$

$$\left(\frac{10}{1}\right) - 9 = 1 \tag{6.2}$$

[1] `https://www.top500.org`

6.2. Environments

6.2.1. How to Figure

Anything can also be put in multiple columns.

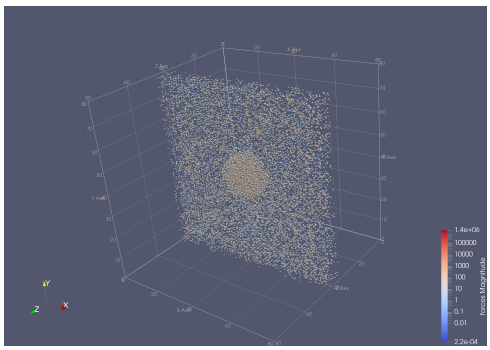


Figure 6.1.: Some Caption. Always also include a source if it wasn't created by you!
Source: [?]

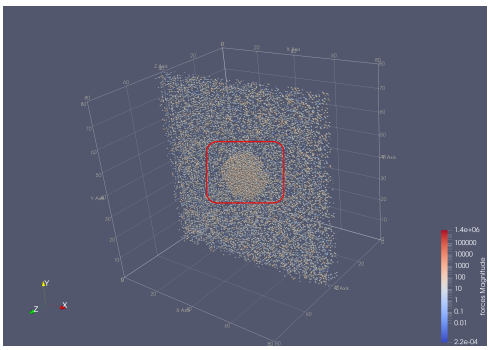
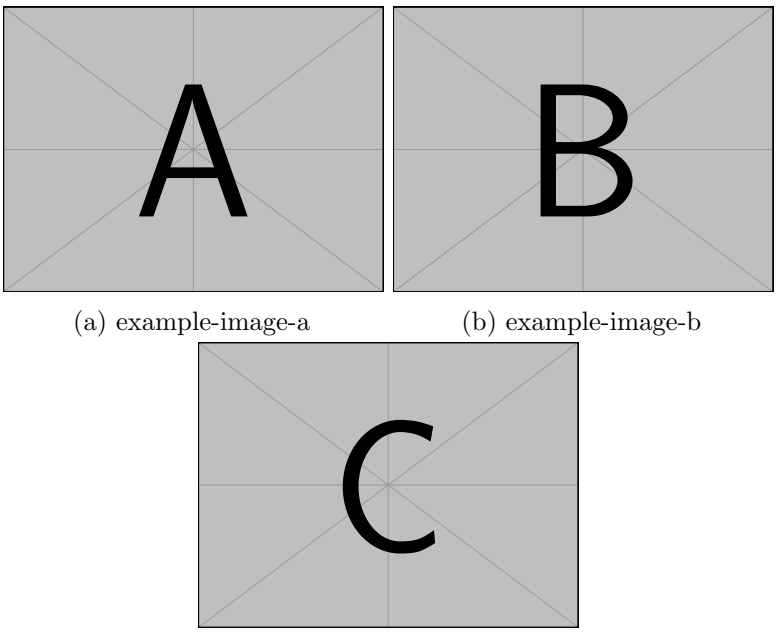


Figure 6.2.: Figures can be drawn on or completely generated with tikz.

Subfigures If grouping of several pictures seems reasonable, think about using subfigures. This often comes in handy with plots.



(a) example-image-a (b) example-image-b

(c) example-image-c

Figure 6.3.: One caption to describe them all.

6.2.2. How to Algorithm

Algorithm 2: Bogosort

Input: data array
Output: data sorted

```
// Checks if array is sorted
Function is_sorted(data):
    for i ← 0 to data.size() - 1 do
        if data[i] > data[i+1] then
            return false
    return true
// actual algorithm
Function bogosort(data):
    while not is_sorted(data) do
        random.shuffle(data)
```

Figure 6.4.: some description what is happening

6.2.3. How to Code

```
1 void runner(int type, void *data){
2     switch(type)
3         case taskType1:
4             // do stuff using data
5         case taskType2:
6             // do other stuff using data
7 }
```

Listing 6.1: General form of a typical runner() function.

6.2.4. How to Table

bla left	bla centered over two lines	bla right
bla left	bla centered	cell spanning two rows
cell spanning two columns		

Table 6.1.: Fancy table that can contain line breaks and extended cells.

Part II.

Appendix

A. Some more stuff

For everything that does not really belong in the thesis but is good to mention.

List of Figures

List of Tables

Bibliography

[AFDJ03] Christophe Andrieu, Nando Freitas, Arnaud Doucet, and Michael Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50:5–43, 01 2003.

[And14] Sweeney D.J. Williams T.A. et al. Anderson, D.R. *Statistics for Business and Economics. 12th Edition.* West Publishing Company, 2014.

[Atz10] Paul Atzberger. Strategies for improving the efficiency of monte-carlo methods. 09 2010.

[B+95] Sten Bergström et al. The hbv model. *Computer models of watershed hydrology.*, pages 443–476, 1995.

[Bar67a] George A Barnard. *The use of the likelihood function in statistical practice*, volume 1. 1967.

[Bar67b] George A Barnard. The use of the likelihood function in statistical practice. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 27–40. Univ of California Press, 1967.

[BBP14] James Berger, MJ Bayarri, and LR Pericchi. The effective sample size. *Econometric Reviews*, 33(1-4):197–217, 2014.

[BGJM11] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo.* CRC press, 2011.

[Bha88] Gyan Bhanot. The metropolis algorithm. *Reports on Progress in Physics*, 51(3):429, 1988.

[Bla06] Martin Bland. Mean and standard deviation. 2006.

[BSSW+23] J Barido-Sottani, O Schwery, RCM Warnock, C Zhang, and AM Wright. Practical guidelines for bayesian phylogenetic inference using markov chain monte carlo (mcmc) [version 1; peer review: 3 approved, 1 approved with reservations]. *Open Research Europe*, 3(204), 2023.

[BT11] George EP Box and George C Tiao. *Bayesian inference in statistical analysis.* John Wiley & Sons, 2011.

[Cas07] Chad Casarotto. Markov chains and the ergodic theorem. 2007.

[CD14] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[CG95] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.

[DLT+17] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

[Do08] Chuong B Do. The multivariate gaussian distribution. *Section Notes, Lecture on Machine Learning, CS*, 229, 2008.

[DS13] M.H. DeGroot and M.J. Schervish. *Probability and Statistics*. Pearson custom library. Pearson Education, 2013.

[Gel02] Andrew Gelman. Prior distribution. *Encyclopedia of environmetrics*, 3(4):1634–1637, 2002.

[Gop98] R.A. Gopinath. Maximum likelihood modeling with gaussian distributions for classification. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, volume 2, pages 661–664 vol.2, 1998.

[GR18] Hoshin V Gupta and Saman Razavi. Revisiting the basis of sensitivity analysis for dynamical earth system models. *Water Resources Research*, 54(11):8692–8717, 2018.

[GRS95] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[Har03] Hanns L. Harney. *Bayes' Theorem*, pages 8–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Her08] Reginald W Herschy. *Streamflow measurement*. CRC press, 2008.

[HMVDW+20] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[Hod22] T. O. Hodson. Root-mean-square error (rmse) or mean absolute error (mae): when to use them or not. *Geoscientific Model Development*, 15(14):5481–5487, 2022.

[KK17] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144, 2017.

[KTB13] Dirk P Kroese, Thomas Taimre, and Zdravko I Botev. *Handbook of monte carlo methods*. John Wiley & Sons, 2013.

[KU88] Andrei N Kolmogorov and Vladimir A Uspenskii. Algorithms and randomness. *Theory of Probability & Its Applications*, 32(3):389–412, 1988.

[Moo97] Christopher Z Mooney. *Monte carlo simulation*. Number 116. Sage, 1997.

[MPS11] Wes McKinney, Josef Perktold, and Skipper Seabold. Time series analysis in python with statsmodels. In *SciPy*, pages 107–113, 2011.

[PB95] José C Pinheiro and Douglas M Bates. Approximations to the log-likelihood function in the nonlinear mixed-effects model. *Journal of computational and Graphical Statistics*, 4(1):12–35, 1995.

[QKVT18] Matias Quiroz, Robert Kohn, Mattias Villani, and Minh-Ngoc Tran. Speeding up mcmc by efficient data subsampling. *Journal of the American Statistical Association*, 2018.

[SR20] Razi Sheikholeslami and Saman Razavi. A fresh look at variography: measuring dependence and possible sensitivities across geophysical systems from any given data. *Geophysical Research Letters*, 47(20):e2020GL089829, 2020.

[SRH19] R. Sheikholeslami, S. Razavi, and A. Haghnegahdar. What should we do when a model crashes? recommendations for global sensitivity analysis of earth and environmental systems models. *Geoscientific Model Development*, 12(10):4275–4296, 2019.

[Ste20] John Steele. Analytic maximum likelihood for the parameters of the gaussian and bernoulli distributions, as well as the parameters of a linear model using ordinary least squares. 2020.

[VS13] Douglas N. VanDerwerken and Scott C. Schmidler. Parallel markov chain monte carlo. 2013.

[Wan09] Ryan Wang. Markov chain monte carlo. *MARKOV CHAIN MONTE CARLO*, 2009.

[Was21] Michael L Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[Wei02] Eric W Weisstein. Normal distribution. *https://mathworld. wolfram. com/*, 2002.

[Wes97] Mike West. Time series decomposition. *Biometrika*, 84(2):489–494, 1997.

[WSB23] Sebastian Wolf, Linus Seelinger, and Michael Bader. Parallel uncertainty quantification with fused simulations. 2023.

[We18] Weglarczyk, Stanisław. Kernel density estimation and its application. *ITM Web Conf.*, 23:00037, 2018.