



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

On the selection of appropriate benchmark functions for optimization algorithms

Master's thesis in Computer science and engineering

Lucas Ruud

MASTER'S THESIS 2021

On the selection of appropriate benchmark functions for optimization algorithms

LUCAS RUUD



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

On the selection of appropriate benchmark functions for optimization algorithms
LUCAS RUUD

© LUCAS RUUD, 2021.

Supervisor: David Issa Mattos, Department of Computer Science and Engineering
Examiner: Gregory Gay, Department of Computer Science and Engineering

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

Typeset in L^AT_EX
Gothenburg, Sweden 2021

On the selection of appropriate benchmark functions for optimization algorithms
LUCAS RUUD
Department of Computer Science And Engineering
Chalmers University of Technology

Abstract

In the field of software engineering optimization is an important aspect when it comes to ensuring that the software that is being developed is able to efficiently perform the necessary operations. Creating software that fast and reliable is required to be able to solve problems in a reasonable amount of time and is also something that gives a competitive edge in industry. There exist many different kinds of optimization algorithms that can help achieve this objective. To evaluate and compare these optimization algorithms, benchmark functions are used. However, there exists many different benchmark functions with different characteristics and selecting a good set of these to fairly evaluate an optimization algorithm thus becomes difficult due to the many aspects there are to consider. Selecting an appropriate set of benchmark functions is also an area that has not been sufficiently researched as of yet. A few problems that accompany the selection process is being able to create a set of benchmark functions that are of varying difficulty levels and at the same time don't take an unreasonable amount of time to execute. This is the overarching problem that this thesis looks into. Based on the evaluation of the benchmark functions conducted, using item response theory and multi level models the results show that it is possible to create a good subset of benchmark functions from a larger set. The impact of the amount of dimensions considered by the optimization algorithm does not affect the results in this specific case. The benchmark functions can also be clustered into different difficulty levels. These results can be used in, for example, the creation of new optimization algorithms by being able to select an appropriate set of benchmark functions that are around the same difficulty levels as the optimization algorithm being developed. This makes the development process of new optimization algorithms faster and the resulting algorithms would then be of higher quality then the other options available.

Keywords: optimization, benchmark, IRT, Bayesian.

Acknowledgements

I would like to thank my supervisor David for his advice and support throughout this master thesis.

Lucas Ruud, Gothenburg, August 2021

Contents

List of Figures	xiii
List of Tables	xix
1 Introduction	1
1.1 Optimization in SE	2
1.2 Benchmarking optimization algorithms	3
1.3 Purpose of the thesis	4
1.4 General- and application benchmark research	5
2 Background	7
2.1 Goals of benchmarking	7
2.2 The traits of the benchmark suite	11
2.2.1 Comprehensive	11
2.2.1.1 The topology of benchmark functions	11
2.2.2 Representative	12
2.2.3 Tunable	12
2.2.4 Known solutions	12
2.3 Why benchmarking?	15
2.4 Usage of statistics in benchmarking	15
2.4.1 Frequentist statistics	15
2.4.2 Bayesian statistics	16
2.4.3 Item response theory	17
3 Theory	19
3.1 Bayesian statistics	19
3.1.1 Bayes' theorem	19
3.1.2 Markov Chain Monte Carlo	19
3.1.3 Highest posterior density interval (HPDI)	20
3.1.4 Benefits over frequentist statistics	20
3.1.4.1 Null hypothesis significance testing (NHST)	22
3.1.5 Bayesian models	22
3.1.5.1 Binomial model	22
3.1.5.2 Multi level models	22
3.1.6 Priors	23
3.1.7 Diagnosing the models	24
3.1.7.1 Trace plots	24

3.1.7.2	\hat{R} (R-hat) values	24
3.2	Item response theory	24
3.2.1	IRT models	24
3.2.2	Item and test information	27
4	Method	31
4.1	Research questions	31
4.2	Data collection	32
4.2.1	Data collection for the sub setting and clustering of benchmark functions	32
4.2.2	Data collection for the analysis of dimensions	35
4.2.3	The system used for the data collection	36
4.3	Benchmark functions and algorithms	37
4.4	Data analysis	37
4.5	Threats to validity	38
5	Selection of benchmarks with IRT	39
5.1	Preparing the data	40
5.2	The IRT model	40
5.3	Selection of priors	40
5.4	Diagnosing the convergence of the MCMC	40
5.4.1	Trace plots for the difficulty parameter	41
5.4.2	Trace plots for the discrimination parameter	42
5.4.3	Trace plots for the ability parameter	42
5.4.4	\hat{R} values	42
5.5	Extracting the results	43
5.6	Results	43
5.6.1	Part A	43
5.6.2	Part B	45
5.7	Discussion	46
6	Increasing the dimensions used by the benchmarks	53
6.1	Preparing the data	53
6.2	Modeling the probability of success	53
6.3	Selection of priors	54
6.4	Diagnosis of the results from the stan model	54
6.4.1	Trace plots	54
6.4.2	\hat{R} values	54
6.5	Extracting the results	55
6.6	Results	55
6.7	Discussion	58
7	Clustering the benchmarks according to difficulty level with IRT	59
7.1	Extracting the results	59
7.2	Results	59
7.3	Discussion	61

8	Conclusion	63
8.1	Future work	65
	Bibliography	67
A	Appendix 1: The benchmark functions	I
A.1	The benchmark functions used when selecting a subset of and clustering benchmark functions	I
A.2	The optimization algorithms used when selecting a subset of and clustering benchmark functions	VI
A.3	The optimization algorithms used when increasing the number of dimensions	VII
B	Appendix 2: Difficulty parameter trace plots	IX
C	Appendix 3: Discrimination parameter trace plots	XXI
D	Appendix 4: Item information curves	XXXIII
E	Appendix 5: Benchmark difficulty parameter distributions	XLV
F	Appendix 6: Tables	LIII
G	Appendix 7: RQ2 trace plots	LXIX
H	Appendix 8: Clustering of benchmark functions	LXXI

List of Figures

2.1	Topology (top) and contour (bottom) plots of the Bird function. The darker areas in the contour plot indicate deeper valleys.	13
2.2	Topology (top) and contour (bottom) plots of the Keane function. The darker areas in the contour plot indicate deeper valleys.	14
3.1	The impact of the difficulty and discrimination parameters in the item information curves. In the top part of the figure the discrimination parameter is set to three and in the bottom part the difficulty parameter is set to zero.	26
3.2	Item information curves and the test information curve. In the item information figure, a is the discrimination value and b is the difficulty value.	28
5.1	Example of a good trace plot. The values on the y-axis are the values sampled and the values on the x-axis are which sample “took” a certain value along with the total number of samples. The beginning of the plot is not included as these samples were used as warm up. . .	41
5.2	Trace plot showing the chain convergence for the algorithms	42
5.3	Curve showing where the maximum information gain from a test suite composed of 30 selected benchmark functions meant to be close to the algorithms ability levels	47
5.4	Curve showing where the maximum information gain from a test suite composed of 30 selected benchmark functions meant to create a good general benchmark suite	47
5.5	Curve showing where the maximum information gain from a test suite composed of all the selected benchmark functions	48
5.6	Four sample benchmark suites created by randomly selecting 30 benchmark functions	50
6.1	HPDI plots for the algorithms and the dimensions	56
6.2	HPDI plots for the random effect s	57
7.1	Example showing some benchmark functions divided into their respective difficulty levels. The solid black lines show the limits for the division of the difficulty levels. “Benchmark median” is the median difficulty level of the specific benchmark function.	60

B.1	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	IX
B.2	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	X
B.3	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	X
B.4	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XI
B.5	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XI
B.6	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XII
B.7	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XII
B.8	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XIII
B.9	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XIII
B.10	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XIV
B.11	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XIV
B.12	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XV
B.13	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XV
B.14	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XVI
B.15	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XVI
B.16	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XVII
B.17	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XVII
B.18	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XVIII
B.19	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XVIII
B.20	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XIX
B.21	Trace plot showing the chain convergence for the difficulty parameter for the benchmarks	XIX
C.1	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXI

C.2	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXII
C.3	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXII
C.4	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXIII
C.5	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXIII
C.6	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXIV
C.7	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXIV
C.8	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXV
C.9	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXV
C.10	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXVI
C.11	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXVI
C.12	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXVII
C.13	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXVII
C.14	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXVIII
C.15	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXVIII
C.16	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXIX
C.17	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXIX
C.18	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXX
C.19	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXX
C.20	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXXI
C.21	Trace plot showing the chain convergence for the discrimination parameter for the benchmarks	XXXI
D.1	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXIII
D.2	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXIV

D.3	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXIV
D.4	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXV
D.5	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXV
D.6	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXVI
D.7	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXVI
D.8	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXVII
D.9	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXVII
D.10	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXVIII
D.11	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXVIII
D.12	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXIX
D.13	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XXXIX
D.14	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XL
D.15	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XL
D.16	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XLI
D.17	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XLI
D.18	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XLII
D.19	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XLII
D.20	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XLIII
D.21	Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent	XLIII
E.1	Distribution of the difficulty parameters for the benchmarks	XLV
E.2	Distribution of the difficulty parameters for the benchmarks	XLVI
E.3	Distribution of the difficulty parameters for the benchmarks	XLVI
E.4	Distribution of the difficulty parameters for the benchmarks	XLVII
E.5	Distribution of the difficulty parameters for the benchmarks	XLVII
E.6	Distribution of the difficulty parameters for the benchmarks	XLVIII
E.7	Distribution of the difficulty parameters for the benchmarks	XLVIII

E.8	Distribution of the difficulty parameters for the benchmarks	XLIX
E.9	Distribution of the difficulty parameters for the benchmarks	XLIX
E.10	Distribution of the difficulty parameters for the benchmarks	L
E.11	Distribution of the difficulty parameters for the benchmarks	L
E.12	Distribution of the difficulty parameters for the benchmarks	LI
E.13	Distribution of the difficulty parameters for the benchmarks	LI
G.1	Trace plot showing the chain convergence for the algorithms	LXIX
G.2	Trace plot showing the chain convergence for the dimensions	LXX
G.3	Trace plot showing the chain convergence for s	LXX

List of Tables

F.1	Summary values of the discrimination and difficulty level parameters for the benchmarks	LIII
F.2	Summary values of the discrimination and difficulty level parameters for the benchmarks of research question one part A	LXIV
F.3	Summary values of the discrimination and difficulty level parameters for the benchmarksof research question one part B	LXV
F.4	Summary values of the ability level of the algorithms	LXVII
F.5	Summary values of the results from the model from research question 2	LXVII
H.1	Clustering of the benchmark functions	LXXI

Terminology used in this thesis

In this section the specific terminology used in this thesis is explained. The following words and concepts are used frequently throughout the thesis and they might not be used in the exact same way or have the same meanings in other areas. Having this section will help reduce the chances for misunderstandings due to specific terms.

1. **Optimization algorithm:** An algorithm designed to find the optimal solution to a given problem. Can be subjected to various constraints such as time, number of iterations etc. How good an optimization algorithms solution is, is governed by a fitness function.
2. **Benchmark function:** A function that serves as a problem for the optimization algorithm to solve. Investigating how the optimization algorithm performs on the benchmark function can show, for example, how long it took to find a solution, the quality of the solution etc.
3. **Test bed:** A collection of benchmark functions
4. **Benchmark suite:** A collection of benchmark functions
5. **Difficulty level:** How difficult a certain benchmark function is perceived to be. This can refer to, for example, how long, on average, this function takes to be solved by an optimization algorithm or how many dimensions it has.
6. **Dimension:** The certain thing that is being optimized, higher dimension on a benchmark function means that more things are being optimized at the same time. This basically means that a benchmark function includes several variables at the same time.

1

Introduction

Optimization is something that has come to be a very important aspect in today's society, be it maximizing revenue and minimizing costs in a company or scheduling the activities of a day for a private person [1]. It is of course no surprise then that optimization is a very important aspect of software engineering (SE) as well.

Making the software development cycle more streamlined or being able to efficiently maintain software after its release are just two high level examples of how optimization plays a part in SE. You want the software to find the most optimal solution to the problem it is trying to solve. There are a vast array of optimization algorithms available that have different ways of going through data and finding the most optimal solution to a problem. Choosing the best optimization algorithm for your software and problem would then mean that you need to take into account the different abilities and performance of the different algorithms.

There are multiple ways of comparing algorithms to see how their performance matches up. One way of comparison could be to run the algorithms on the same or at least similar problems. This would ensure that the results from the algorithms actually are comparable since the problems they are used on are of the same difficulty. Another benefit with this is that if the algorithms are used on real world problems the results would also be very representative of how the algorithms actually performs if it is deployed into action. However, a downside of this would be that the algorithms could take a very long time to find a solution if a problem is especially difficult to solve. This time aspect could be even more of an issue if you are developing a new optimization algorithm and want to see how it compares to already existing algorithms out there. If this is the case then you probably want to not have to wait for hours to get results, find out that they are worse than a compared algorithm, modify your algorithm and get new results. This would take a lot of extra time. Since this is the case, it would be more time efficient to evaluate on more artificial problems of varying difficulty. This would sacrifice some of the connection to the real world but gaining time, allowing the new algorithm to be developed faster.

Going from this, another way of investigating the performance of different optimization algorithms and how they compare to each other are benchmark functions. Benchmark functions fall into the area of artificial problems since these are all problems created by people and can be as difficult or easy as desired. This means that you can control many aspects of the problem that the algorithms is supposed to be evaluated on which means that you can more precisely see what aspects the algorithms in question are good at and less good at. However, there are also a lot of different benchmark functions available to choose from and making a selection of

benchmark functions to efficiently and fairly compare the algorithms then becomes pretty difficult. The difficulties of the benchmark functions are widely different, they may or may not be able to take into account multiple dimensions, they take varying amounts of time to come to a solution etc. These are just a few aspects that add to the problem of selecting a set of benchmark functions to compare or evaluate algorithms difficult.

Using benchmark functions would then fall into the category of laboratory experiments. We are basically setting up a very controlled environment since we are running the algorithms on benchmarks on a certain computer [14]. Experiments on computers can be controlled to a large degree because you have the ability to choose what programs are running and even the components used, which makes it easier for others to replicate the experiment. So we can closely control the each variable in the computer and thus see what parameters affect the results in what way. Closely determining what affects the results, and in what way, gives an indication regarding what the best areas are for the algorithms to be used in. A problem with this though is that since the results are collected in a very controlled environment, they may not entirely reflect the algorithms actual abilities in real world problems. Since the algorithms are going to be used on problems that are undoubtedly not as artificial as the benchmarks are the results might not be entirely truthful. But as mentioned previously, what is gained is time when the algorithms are being developed leading to better feedback of the algorithms abilities and to the development of better algorithms.

1.1 Optimization in SE

Optimization has been investigated previously in the field of SE with many articles falling into this category. There is also an annual conference on search based software engineering (SBSE) where the newest research gets additional exposure [50]. As shown below, this research has had an impact and can have further impact in the field.

Basios et al. [2] proposes an optimization framework to solve the Darwinian Data Structure Selection (DS2) problem. They applied their tool on Google Guava and the results from the tool showed a 9% improvement on execution time, 13% improvement on memory consumption, 4% improvement on CPU usage separately and 27.45% of the final solutions provides improvement without sacrificing other objectives.

A study performed by Lukasczyk et al. [3] shows how optimization algorithms can be used beneficially in the context of software engineering. They use a genetic algorithm, which is an optimization algorithm inspired by natural processes like evolution, to automatically generate a test suite for Python. The optimization algorithm creates a test suite with tests that come close to a specified fitness metric, with regards to certain constraints such as maximum allowed time etc. The better score a set of test cases get the better it is with regards to what you are trying to cover. In the case of this study they are focusing on branch distance but you can just as easily substitute this to something else, like line coverage or path coverage etc. This is a lot more efficient than manually creating test cases and checking if

they meet some specified criteria.

A similar study by Bruce et al. [4] also looks into automating test generation. This study uses an ant colony optimization algorithm and focuses on branch coverage instead of branch distance. The results from this study shows that this approach of using an ant colony optimization algorithm provided a very high branch coverage (70%-100%) in a very short amount of time (1s - 120s) for the programs they used to test it. Testing is also a part of software engineering that takes a lot of time and money for every software project. By being able to automatically create tests cases by using an optimization algorithm helps with shortening the development time and increasing the quality of the test suite in regards to the coverage criteria you are aiming for.

A fourth study, that was a replication study by Al-Subaihin et al. [5], looked into how efficient it would be to use another genetic algorithm to cluster already existing software. They used mobile apps in the app store for testing and found that while it could be costly to use this method, at certain configurations this approach outperformed the approach they used for comparison, which was a hierarchical clustering. Having a way of efficiently and accurately cluster different software could be very useful for other studies in software engineering. So looking into using optimization algorithms in this context to see if they can outperform current clustering approaches is something that has relevance.

The studies mentioned above are just some examples, there exist many more and research in this area is ongoing.

1.2 Benchmarking optimization algorithms

Benchmark functions are, simply put, mathematical equations. These equations have, for example, a maximum value and finding this could be one of the objectives of an optimization algorithm. Measuring, for example, how long it takes for optimization algorithms to find this value gives a basis for comparison between them for that benchmark function. The optimization algorithm produces a result to the benchmark function by trying different input values and investigates the output value from the benchmark function. This value can then be compared to a known best value to see how well a optimization algorithm is at finding optimal solutions. This is also the way the results from the optimization algorithms and benchmark functions were investigated in this thesis. A very simple example can be made with equation 1.1. This can be considered to be a benchmark functions (although a very simple one), with a maximum value of four at an x value of zero. Finding the x value that maximizes the outcome value could be one thing that an optimization algorithm is supposed to do. Measuring the time it take for an optimization algorithm to come up with a solution, the amount of value tried, how close the solution is to the optimal solution etc. are just a few things that could be used when comparing optimization algorithms.

$$f(x) = -x^2 + 4 \tag{1.1}$$

The benchmark functions can be of varying difficulty and take different amounts

of time to solve. Equation 1.1 would probably be solved very quickly, while others will probably take more time. The results from running optimization algorithms on benchmark functions can thus be used to determine the effectiveness of an algorithm or to compare it to others. If we know that one algorithm finds a solution to a certain benchmark function in two minutes and another one finds a solution in one minute, then the second algorithm is probably the better choice. There are of course other things that can be measured, like checking how close to the exact solution (if one is known) an algorithm is and how often a solution is found etc.

The previously mentioned genetic algorithms and the ant colony optimization algorithm both fall in the category of black-box optimization algorithms. This means that the optimization algorithm tries to find the best value possible using the allotted resources. The black-box optimization algorithms can also only observe the result from at any given point, but they do not have access to derivatives of it.

During the development of black-box optimization algorithms, researchers utilize benchmark functions to verify the performance of the algorithms and to compare different competing algorithms. As hinted towards, benchmark functions are functions that test how well a specific optimization algorithm performs which can then be put into comparison with other optimization algorithms to see which performs the best. Clearly if one optimization algorithm works better than another, you would probably want to use the better one. The results from the comparison of these algorithms in benchmark functions traditionally guide the selection of these algorithms in future applications, such as the ones mentioned above. This also puts focus on the benchmark functions themselves as different benchmark functions may give results of different quality when it comes to how good the solution is or the time taken etc. for the optimization algorithm. Collecting several different benchmark functions that work on the same type of data is known as a benchmark suite. Another thing when it comes to using benchmark functions is that there exist many different choices (over 180 as seen in the survey by Jamil et al. [6]). This means that there exist several different options to choose from when an optimization algorithm is evaluated but usually an optimization algorithm is only tested using a few benchmark functions which are chosen rather arbitrarily. This affects the reliability of the results from the evaluation of an optimization algorithm which could fool someone to think that an optimization algorithm is better or worse than it really is which could then affect the results of, for example, the study it is used in. The data that is collected from a benchmark function also usually determines what the optimization algorithm in question is going to be used for in software engineering. So if the data is misrepresentative the optimization algorithm may end up being used for purposes that are not the best for it. Currently, there are also no studies that look into quantitatively assessing different benchmark suites. This is supported by the fact that the study by Bartz et al. [7], while providing best practices for choosing benchmark functions, does not raise any quantitative assessment of them.

1.3 Purpose of the thesis

This thesis is aiming at evaluating different benchmark functions using statistical theories to create a benchmark suite that is more suitable to discriminate algorithms

in benchmark comparisons. Because the results of the benchmark suites are directly tied to how the optimization algorithms are applied to software engineering problems this could lead to problems if the benchmark suite itself gives low quality results. This could happen if the benchmark suite is not well suited to discriminate different algorithms. By evaluating the benchmark functions in the suites, these problems could be mitigated which would ultimately lead to better solutions to the software engineering problems that the optimization algorithms are applied to.

Evaluating benchmark suites qualitatively is the main purpose of this study and as mentioned this can affect the area of software engineering positively. However, the method used in this thesis is can also be used in other software engineering areas. This means that another purpose of this study is to provide a clear view of the method used so that other researchers in the field of software engineering can potentially benefit from this as well.

1.4 General- and application benchmark research

For the purpose of this thesis it is relevant to differentiate between general benchmark research and application benchmark research. This thesis focuses on the general benchmark research, meaning that it will look into the benchmark functions in their own right and not in the context of any specific application. The techniques used in this thesis are suitable for general and application benchmark research. But again, this thesis looks into qualitatively analyzing benchmark functions in the general case and does not consider how they could be used in contexts such as, test generation or other application research areas.

Bartz et al. [7] gives some qualitative reasons as to why benchmark functions are good to use to investigate the performance of algorithms.

By using benchmark functions, the results from this can be extrapolated and generalized to a wider area. The general performance of an algorithm will be known beforehand, helping with choosing how and where the algorithm can be used in different areas. If using a specific application to investigate the performance of an algorithm, the results might not be as easily generalizable. An algorithms performance on a particular application gives information regarding how well the algorithm works for that application. However, since there are multiple things that come together to form this application there are many more variables that can affect the results of the algorithm. This makes it more difficult to say if an algorithm will work just as well on another application.

Since an application is composed of many different components which gives many different parameters that can affect the result of an optimization algorithm the problem as a whole might also be less understood. This is not a problem for benchmark functions as they are very well defined as mathematical expressions. This increases the overall understanding of the problem itself, and the interaction between the benchmark function and the optimization algorithm.

If evaluating an optimization algorithm on an application we will get information regarding the algorithms performance on the application itself and nothing more. While this may be sufficient depending on your intentions another advantage of using benchmark functions arises here. Evaluating an optimization algorithm using

a benchmark function might lead to new insights into the algorithm itself and serve as a source for new works in the field of optimization algorithms.

Bartz et al. [7] also gives general guidelines on choosing suitable benchmark functions for the research that is to be conducted. You could use their research to come up with a suitable suite of benchmark functions to test or compare an optimization algorithm. This thesis is similar in the way that it is trying to specify what the different benchmark functions do well, less well, their difficulty etc. By doing this, the results from this thesis are meant to be used in a similar manner, where you would be able to use the results to pick suitable benchmark functions to compare optimization algorithms.

2

Background

Benchmark functions in computer science is a way of determining how “good” an algorithm is. “Good” in this context can refer to a number of different aspects of the algorithm and the specifics of the benchmark study determines which aspects are being investigated. For example, aspects that might be of interest could be, the time take, how close the found solution is to the best solution, the number of values the algorithm tried before selecting a solution etc.

An example of using a benchmark function can be made with the Ackley function with two dimensions defined in equation 2.1.

$$f(x, y) = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos 2\pi x + \cos 2\pi y)} + e + 20 \quad (2.1)$$

This function has a global minimum at $(0, 0)$, however it also has a lot of local minimum points as well. If an optimization algorithm has as an objective to find the global minimum of this benchmark function it will need to be able to have some way of differentiating a global minimum from a local one. Getting trapped in a local minimum would probably be a bad sign for the optimization algorithm and thus it might not be the best suited one to choose. The optimization algorithm “tests” different values to try to find the best solution to the benchmark function. It employs different strategies to select values and where to “go” (the next value to try) after one value has been tried. As mentioned, many different things can be used to compare the optimization algorithms. We can use the aspects mentioned before to compare optimization algorithms to see how each one performs in each of these aspects and make a choice based on this information. Since we know the global minimum of this benchmark function we simply need to see how far of each algorithms solution was from this value and we can decide which one is better. Measuring the time taken for an algorithm to find a solution is another thing that simple thing that could indicate which algorithm is a better choice. After an algorithm has found a potential solution to the benchmark function it will probably try some other value nearby to see if it is better or not and take another step from there based on if it is better or not. The amount of times the algorithm steps to find another value could be another interesting thing to compare and consider.

2.1 Goals of benchmarking

Bartz et. al. [7] presents 16 different possible goals when performing benchmark studies. These goals will each be elaborated a bit upon.

Basic assessment of performance and search behavior One of the most basic goals is to examine the general performance of an optimization algorithm. This means that the algorithm is run on a problem instance and the results are recorded and analyzed. The more times an algorithm is run on a problem the more reliable the results are. There are many different aspects in the results that can be evaluated to determine if the algorithm in question is performing well or not. However, the main purpose for this goal is to determine if the algorithm being investigated works well for the type of problems that the benchmark presents.

Algorithm comparison Another major goal, which is closely related to the goal in this thesis, is to compare different optimization algorithms to each other. Comparing algorithms will help with understanding what the strengths and weaknesses of each algorithms are. This is useful for determining when to use a certain algorithm, or if there exists a better algorithm to select for the problem you are trying to solve. This is also helpful when designing a new optimization algorithm. Getting information regarding how the new algorithm compares to others during the development process will make it possible to make changes that improves the performance of the new algorithm and make it more competitive in terms of being selected over others.

Competition Determining a “winner” between one or more algorithms can also be a goal of benchmarking. If a set performance measure and a set of particular problems are given, from the optimization algorithms used we can then select a winner, which would be the algorithm that performed the best under the given conditions. This provides a very good way of selecting the best algorithm to use for a given problem which is especially useful for real world applications [32]. There are however both positives and negatives about this goal. The way of comparing algorithms and then selecting one or the other has come under some criticism when it comes to the underlying scientific method used [33]. The “winning” algorithm may be the one that is very tuned to the particular conditions and thus overfit. But seeing this as a competition can also inspire new ideas for algorithms to emerge and better ways of selecting appropriate algorithms for particular problems.

Assessment of the optimization problem There exists problems where the optimum solution is not known and in other problems there is limited information available. For these kinds of problems we do can’t really know if the optimization algorithms solutions are good or not. Therefore, the quality of the solution to these problems would need to be evaluated through simulations or experiments. Benchmarking can then be used to analyze and visualize the problem to gain insight about it.

Illustrating algorithms’ search behavior To understand how an optimization algorithm actually functions you could try looking at the algorithm itself, the internals of it. However this is still very technical and might not give a satisfying answer. Benchmarking can be used here to visualize how the optimization algorithm works

when it is trying to find the solution. This means that it provides another way of understanding the algorithm than just mathematically.

Testing invariances It is argued that algorithms should be invariant in regards to certain aspects like scaling and translation of function values [34] increasing dimensions [35] or search space rotation. Benchmarking can be used to empirically show if algorithms have these invariances.

Algorithm tuning Optimization algorithms usually have some parameters that can be tuned in order to change the behavior of the algorithm slightly. For example, if an algorithm can choose from multiple different "paths" to take at any one point, there might exist a parameter that determines how many "paths" can be considered at the same time and changing this would lead to the algorithm behaving differently. Tuning these kinds of parameters in different ways should eventually lead to the algorithm performing optimally for the give problem. Tuning of algorithms can be done both manually and automatically and there exist different tools that tune parameters [37] [38] [39]. Doing all the tuning manually can take a very long time due to the number of parameters and the different values they can assume [36]. Benchmark functions can be used here to evaluate the performance of an algorithm after a parameter has been changed and thus determine if the change is worth it or not. This requires some form of specific metric to be chosen as being improved, like time to find a solution etc. It also gives insight into how the algorithm responds to smaller changes. If the performance decreases by a large amount after changing one parameter another algorithm that has a less drastic response to the same change might be preferable. This response to the change in parameters is know as robustness and is an important aspect to consider when tuning an algorithm [40].

Understanding the influence of parameters and algorithmic components Tuning an algorithm determines what the different parameters in the algorithm should be set to in order to achieve the best performance for the given problem. This goal refers to finding out why the parameters affect the algorithm in the way they do. Fully understanding this would require more statistical tools for a more comprehensive analysis. There exist tools for doing this as well like the sequential parameter optimization package available in R [41].

Characterizing algorithms' performance by problem (instance) features and vice versa This goal refers to understanding the relationship between the optimization algorithm and the problem it is trying to solve. This means that you are trying to connect aspects of the problem with the performance of the algorithm to understand why an algorithm works well or not for a specific type of problems. The specific aspects that are investigated varies, but examples include high levels aspects like the amount of dimensions and the topology of the problem space etc. Low level aspects can be derived through exploratory sampling and include modality and ruggedness [42].

Performance regression The results from benchmarking an optimization algorithm can also be analyzed in order to predict how well an algorithm would perform on other types of problems. This has advantages when it comes to selecting an appropriate optimization algorithm for a problem, as well as how the algorithm should be configured. Performance regression is something that also exist in the machine learning area, however, in this context performance regression is also known as transfer learning [43].

Automated algorithm design, selection, and configuration When we are able to connect an optimization algorithms performance with certain aspects of the problems and the results from benchmarking an algorithm can be used to predict the performance of the algorithm on other problems, we can use these results to design, select or configure an algorithm for the current problem. This means that one goal is to collect this data in order to use it to establish rules so that the best optimization algorithm can be chosen for the current problem. These rules can be human interpretable or generated automatically [44] [45].

Cross-validation and complementation of theoretical results Results in the context of optimization usually does not represent the actual performance of an algorithm given a specific dimension, time to complete etc. Usually the results are expressed in terms of runtime bounds, which give probabilistic results rather than concrete results [46]. To help with this, a benchmarking study can be perform in order to complement the already existing results.

Source of inspiration for theoretical studies The results from a benchmarking study can serve as inspiration for new research. Especially if the results are surprising and does not go with what we had originally believed the results to be. Benchmarking studies then work as a way of getting these potentially new research subjects into light. An example of this can be made with the (1+1) evolutionary algorithm where a study was performed to serve as a compliment to another study that had already looked into the performance of this algorithm [47].

Benchmarking as intermediary between theory and practice The previous two goals together show another potential goal for benchmarking studies, which is that it can serve as a good intermediate between the mathematical realm of algorithm design and the practical realm of actually using the algorithms in reality. Using benchmarking in experimental studies makes the results more comparable [48] and thus helps with determining if one algorithm is better than another.

Source code validation If we have a problem that we know a solution to we can use an optimization algorithm to see if the algorithm produces the expected result. If it doesn't then this means that the source code of the algorithm might be faulty and possibly need to be redesigned.

Algorithm development Benchmarking an optimization algorithm while it is still being developed is a good way of getting a preliminary view of its performance,

strengths and weaknesses etc. This means that we can improve an algorithm before it is released by reducing weaknesses and generally increasing its performance. It could also show if the algorithm is actually worth developing depending on the results collected. This way of benchmarking can also lead to a loop of improvement. For example, a study found evidence that dynamically choosing parameters for an algorithm can be beneficial over static choice [49]. Knowing this, a loop could be established where algorithms were modified and then investigated to find out if it was better or not.

2.2 The traits of the benchmark suite

When benchmarking an algorithm, to get the most reliable data possible, you would then want a suite of problems that are comprehensive, representative and tunable and have known solutions [7]. There also already exist several different benchmark suites for different categories of algorithms that have been compiled over the years.

2.2.1 Comprehensive

To get a good understanding regarding the performance of algorithms they need to be evaluated on problems of different difficulty. Having a set of problems that span from “very easy” to “very hard” difficulty would then cover this. Difficult is however something subjective to the algorithms themselves and a difficult problem for one might be easy for another. A set of problems that is able to show both the strengths and weaknesses of an algorithm is to be desired.

2.2.1.1 The topology of benchmark functions

When speaking of difficulty in relation to optimization algorithms, the topology of the benchmark functions is something that is closely tied to this.

Each benchmark function can be viewed as a landscape of sorts and the optimization algorithm tries to find the highest peaks and the lowest valleys there are in this landscape. The general shape of this landscape is something that can make it difficult for an optimization algorithm to solve it as it may get “trapped”. Different algorithms have different ways of solving problems and this means that they are differently adept at finding the peaks and valleys of the landscape.

$$f(x, y) = \sin(x)e^{(1-\cos(y))^2} + \cos(y)e^{(1-\sin(x))^2} + (x - y)^2 \quad (2.2)$$

$$f(x, y) = -\frac{\sin^2(x - y)\sin^2(x + y)}{\sqrt{x^2 + y^2}} \quad (2.3)$$

Two examples of the topology of two different benchmark functions can be seen in figures 2.1 and 2.2. These two figures are visual representation of the equations shown in equation 2.2 and 2.3 respectively. These two figures exemplify why it can be difficult for optimization algorithms to find the optimal solutions to problems. The top image of these two figures show the overall landscape the algorithm is searching and the bottom image show how deep the landscape is at certain points.

It is in these valleys where the algorithm can get stuck if it is not well equipped to search the landscape and if this happens it may not find the best solution. The two figures also show that the landscapes can look very different from each other and have different aspects that makes them difficult. Figure 2.1 shows that the overall landscape is somewhat “bent” with several smaller valleys. While figure 2.2 shows that the overall landscape is flat with several smaller valleys and valleys that get very deep near the center. Finding a “good” solution to any of these benchmark functions means that the minimum or maximum value of the function is found (depending on what you are interested in). The shape of the topology makes it more or less difficult for the best solutions to be found and could be problematic for different algorithms depending on how they can deal with things like getting stuck in a valley etc. Of course each benchmark function can be visualized in this way and each produce a different landscape, some will be simple and some will be more complex.

Thinking about this landscape intuitively, the more complex it is the more problematic it will be for optimization algorithms to find a solution. It will need to have some interesting shape though because if it is entirely flat the best solution is already known and every algorithm will arrive at the same solution, making the benchmark function pointless. Aspects that affect the optimization algorithms ability to find solutions include modality, smoothness, presence of plateaus and the general global structure and having information about these would help with selecting an algorithm that performs well on this landscape [25]. Most of the time however, this information regarding the landscape is not available.

2.2.2 Representative

The results gathered from a benchmarking study are only meaningful if the problems selected are close in relation to the problems that the algorithm will actually be used on. So any claim made at the end of a study will be dependent on how representative the problems are.

2.2.3 Tunable

Being able to change smaller things about the problem, for example, the number of dimensions will make it easier to see how an algorithm reacts to these changes and provide more comprehensive results.

2.2.4 Known solutions

If there is a known best solution to a problem, there is no reason not to compare the results from the algorithms to this. This would give very good information regarding the performance of the algorithm. There are however problems where the best solution is not known so this is not always a possibility. In these cases, comparing the performance of the algorithm to a previously known best performance would be another option.

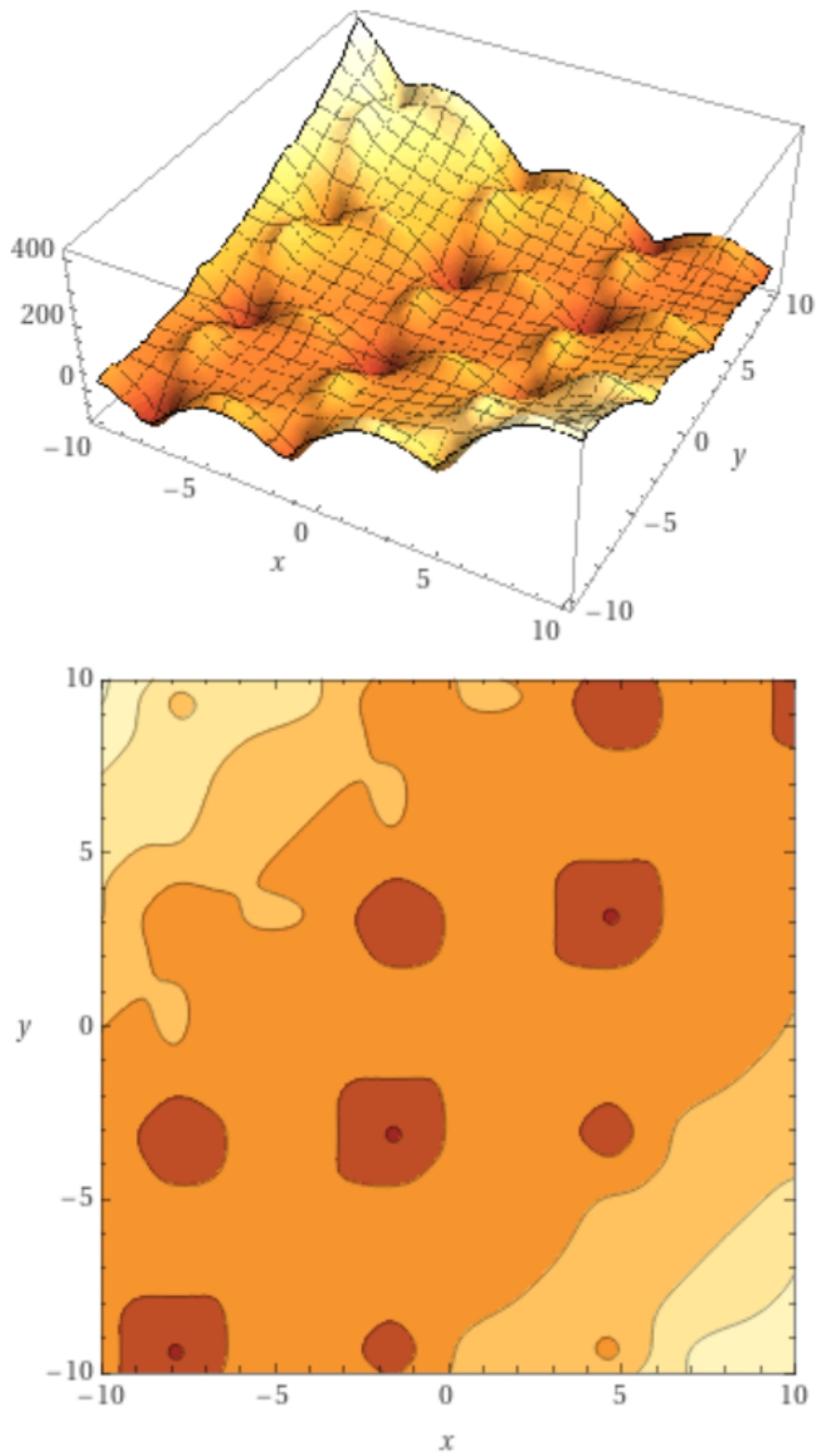


Figure 2.1: Topology (top) and contour (bottom) plots of the Bird function. The darker areas in the contour plot indicate deeper valleys.

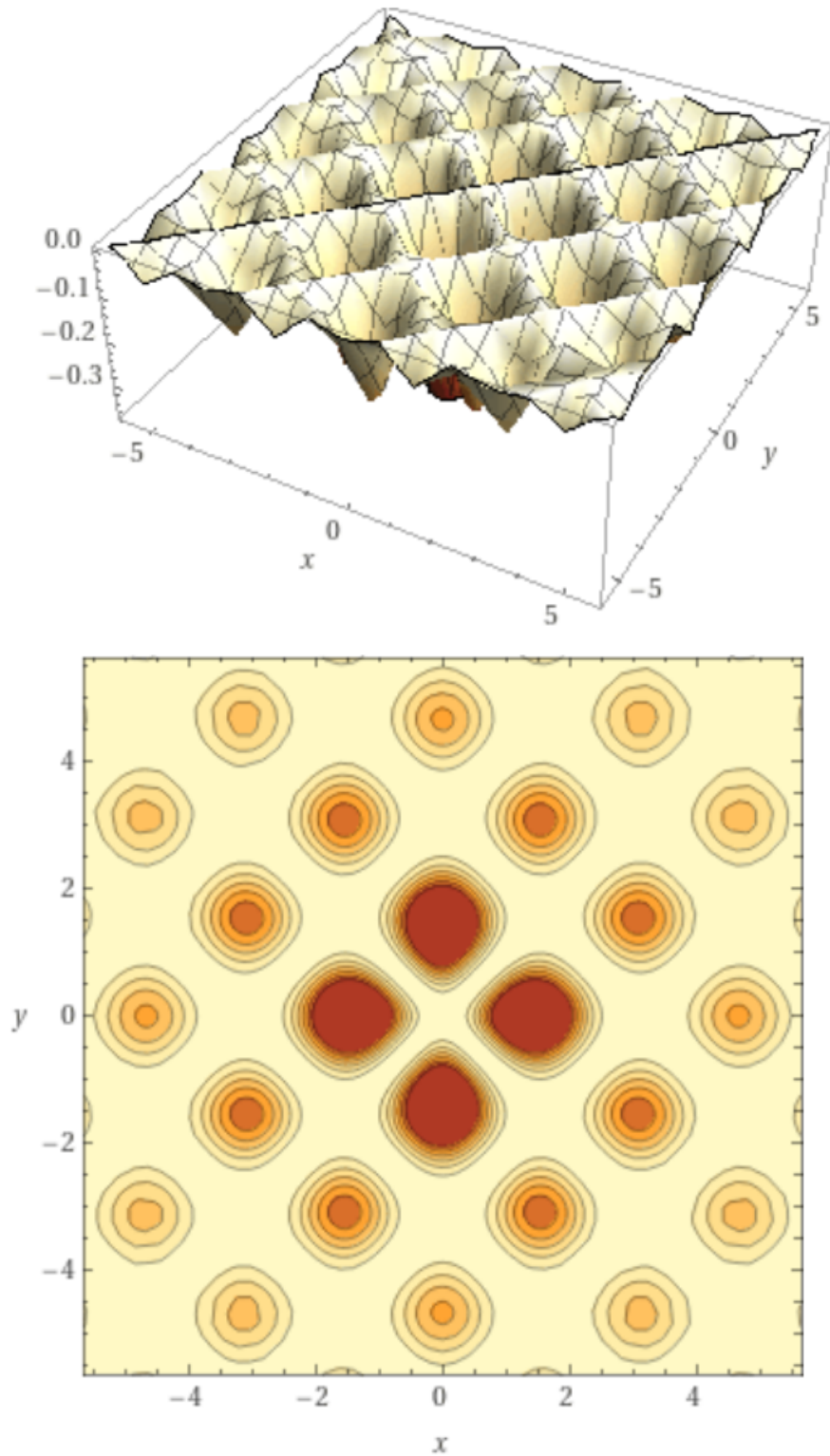


Figure 2.2: Topology (top) and contour (bottom) plots of the Keane function. The darker areas in the contour plot indicate deeper valleys.

2.3 Why benchmarking?

Even seemingly easy optimization problems could take a very long time for an optimization algorithm to solve. Combining this with the fact that optimization is a big part in software engineering and in general, then it makes sense to find an algorithm that performs the best. In this context, the best algorithm means the best possible candidate that is available to choose. There is no one best solution for every problem available, but there might exist several solutions to the same problem and choosing the best one from these is preferable.

When algorithms are tested to find out how well they perform we want to be certain that the results that we get are due to the actual algorithm and not due to some other random effect. For example, if the computer used to test the algorithm is slow the resulting time for the algorithm to find solutions might be higher than usual. One way of getting around this issue is to use controlled experiments where variables that might interfere with the results are held constant. This helps with getting reliable results and also helps other researchers who are interested in replicating the experiment on their own [8].

Mersmann et al. [26] also investigates what properties of the benchmark functions landscape affect how easy it is to solve. The properties that they note as influential are multi-modality, global structure, separability, variable scaling, search space homogeneity, basin size homogeneity, global to local optima contrast and plateaus.

2.4 Usage of statistics in benchmarking

This also means that there is a heavy use of statistics in order to actually be able to say that one algorithm is, for example, better than another or meets a certain performance criteria. In a study by Bartz et al. [7] there is shown a graph of how to choose a fitting statistical test based on the data that has been collected, a similar but more expansive graph can also be found in the book *Statistical Rethinking* [9, p. 2]. The options in this graph are however all frequentist approaches which is a commonly used paradigm in data analysis, however, this study will instead make use of Bayesian statistics.

The usage of statistics in computer science and in relation to evaluating algorithms is not something new either. Many studies have been performed in this field where the results have a need to be statistically verified. The vast majority of these studies use the frequentist paradigm of statistics and not the Bayesian paradigm, making it relatively new to this field.

2.4.1 Frequentist statistics

Barr et al. [15] essentially discuss benchmarking algorithms and the importance of having a rigorous experimental framework when investigating algorithms to achieve a certain quality of the results. They also present a framework to follow and highlight that using statistics can show what factors contribute to the result and which do not.

García et al. [16] goes deeper into the statistical aspects and investigates which specific statistical technique works best to analyze evolutionary algorithms behavior on optimization problems. The focus is to compare parametric statistical tests against the non-parametric statistical tests.

Lacoste et al. [17] investigates the area of machine learning algorithms and proposes a new method based on non-parametric test to determine if one algorithm is actually better than another. The method verifies that the amount of data available is actually enough to support a claim such as “algorithm one is better than algorithm two”.

What all of these studies have in common is that they all focus on the usage of frequentist statistics to arrive at the solutions. This also shows that the area of frequentist statistics is well established in the field of computer science, and in general science as well. There is good reason for this but, as will be discussed, the frequentist area has some problems that might not make it into the best possible choice.

2.4.2 Bayesian statistics

Bayesian data analysis (BDA) has been proposed as a replacement for the frequentist approach [21]. The reason for this is that it addresses the previously mentioned problems and provides an easier way of interpreting the data [22]. However, despite this fact, BDA is not widely used in the field of benchmarking.

Calvo et al. [23] investigates the usage of Bayesian data analysis in a practical environment. They use BDA to investigate the performance of eleven evolutionary algorithms over a set of 23 discrete optimization problems. They also bring up the frequentist statistics paradigm and argue why BDA has advantages in this sort of research. Due to the fact that BDA is not that used in research they also go into some details regarding the general workflow of using BDA, providing good insight for future research to be performed using BDA.

Mattos et al. [22] investigates optimization algorithms using BDA to answer several questions about them. They also discuss the advantage of using BDA over frequentist statistics and gives a bit of an overview of BDA in general and how it is used. This makes it interesting as an entry point into BDA as it explains the main concepts behind it.

Furia et al. [24] goes gives a in depth comparison between frequentist and Bayesian statistics and argue that in the field of empirical software engineering Bayesian statistics should be used more than it is today. This is complemented with a re-analysis of two previously conducted studies using frequentist statistics to further contrast the two approaches. The downside of Bayesian statistics is also taken into account pointing out some of the problems that come with the Bayesian approach.

While BDA is, as mentioned, not widely used currently, the mentioned studies shows that it is not entirely overlooked. There are also arguments being made trying to show that it can work better than the frequentist approach if the downsides of that approach is investigated.

2.4.3 Item response theory

Item response theory (IRT) aims to connect the difficulty of a question with the abilities of a person to get a probability of the individual correctly answering the question [10]. The higher the abilities of the person, the higher the probability they have of answering the question correctly.

IRT has been around since the 1950s [11] and has seen use in, for example, education. This is a logical application area of IRT as students frequently takes tests and IRT can then be used in order to make sure that a test is designed in a good way. IRT has however not seen much usage in software engineering.

In the context of the thesis topic however it is interesting to look into how well IRT works when comparing benchmark suites. It would work in a similar way to how it works in education, the optimization algorithms would be the students and the benchmark functions would be the test. This would then tell us how well the benchmark function works. So to clarify this a bit more formally, the optimization algorithm is the test taker, a benchmark function is an item and a benchmark suite is a test. From this it becomes apparent that using IRT in the context of benchmarking makes sense conceptually as there exists clear connections between the concepts of IRT and benchmarking which makes it easier to use IRT to evaluate benchmark functions.

3

Theory

3.1 Bayesian statistics

Bayesian statistics works by first setting up a prior estimation of how we expect the data we analyze to be distributed, which may or may not be based on previous knowledge. This prior is then updated for every value that we observe to form a set of plausabilities that tell us how probable a certain value is to appear [9, p. 28-30].

3.1.1 Bayes' theorem

The main mathematical concept behind Bayesian statistics is Bayes' theorem shown in equation 3.1

$$P(h|d) = \frac{P(d|h) \cdot P(h)}{P(d)} \quad (3.1)$$

In equation 3.1 $P(h|d)$ can be read as “the probability of h given that d is true”. In other words, this is a way of calculating the probability of an event occurring given that some other event has already occurred. In the context of this thesis d is the data, h is the hypothesis for the data and this means that $P(h|d)$ is the probability of the hypothesis being true given the data collected [22].

The concepts in equation 3.1 are described below [22].

1. $P(h|d)$: The probability of the hypothesis h being true given the data d. This is also known as the posterior probability distribution for each hypothesis we are estimating.
2. $P(d|h)$: The probability of getting data d given that hypothesis h is true. This is also known as the likelihood of the data given certain conditions, in this case a hypothesis.
3. $P(h)$: The probability of our hypothesis being true. This is also known as a prior.
4. $P(d)$: The probability of observing the data d. This is also known as the marginal likelihood and is held constant due to the difficulty of computing it [27].

3.1.2 Markov Chain Monte Carlo

Bayes' theorem provides a way to “update” our beliefs depending on the observed data and generates the posterior distribution. The actual computations are more complicated though and uses a class of sampling algorithms called markov chain

monte carlo (MCMC) [13]. MCMC generates thousands of credible values from the posterior distribution. These values will then be used to approximate the posterior distribution. The more values generated the better this approximation will be and the more reliable the final results will be [27]. Generating more values will also increase the time taken, so this creates a balancing act where you want to generate enough values for the posterior to be approximated well while at the same time not taking too long.

The collection of samples works the following way, first a “landscape” is set up by the log posterior created by the data collected. On this “landscape” a “particle” moves around. This “particle” moves around the “landscape” and the position of this “particle” basically represents the samples collected. The “landscape” has peaks and valleys and the particle is more likely to spend more time in some areas and thus collect more samples here.

While there exist several different MCMC algorithms, the particular one utilized in this thesis is the Hamiltonian No U-Turn Sampler (NUTS). The NUTS algorithm eliminates the need for user configuration of parameters while still being able to perform well [28]. It is also available in Stan which can be used in R, the language used to analyze the data in the thesis [29]. As the name implies, this particular algorithm eliminates U-turns. When a MCMC algorithm traverses the “landscape” created by the benchmark function and collects values it might end up in the same general area if it uses some sort of “random walk” behavior. This means that the values collected will not be representative of the entire problem and the optimal solution might not be found. The NUTS algorithm notices when it starts to turn around towards its starting position and makes adjustments to where it goes next to avoid returning to the starting area [28]. This way more of the landscape can be explored leading to more representative values being collected.

3.1.3 Highest posterior density interval (HPDI)

When a posterior distribution has been created using MCMC the model has served its purpose and we can now start to ask questions regarding the distribution. The highest posterior density interval denotes the narrowest interval that contains the specified probability mass [9, p.56-57]. How wide this interval is will impact how sure we can be of the impact of the parameters. If the HPDI is very narrow, we can be pretty sure of the effect of the parameter and if it is very wide we can be less sure [52]. One particular thing to note is if the HPDI crosses zero the effect of a parameter would be inverted (positive values vs negative values). If the HPDI is very wide but strictly positive or negative we would not be able to be sure about how big the effect of a parameter is but we would at least be able to say that it affects the result in some way. If the HPDI spans both sides of zero we can’t even say for certain how the parameter affects the result one way or another.

3.1.4 Benefits over frequentist statistics

The reason that Bayesian statistics is used is because there exist some shortcomings with the frequentist approach. A few of these shortcomings are: misinterpretation of

the actual meaning of the p-value, lack of separation between the sample size and the effective sample size, ignoring the magnitude and uncertainty, lack of information regarding the null hypothesis and misinterpretation of the meaning of confidence intervals, among others, and Bayesian statistics has the potential to solve some of these issues [7]. Despite this fact, the Bayesian approach is not as wide spread as the frequentist one for the analysis of benchmark experiments.

A cornerstone in the frequentist approach to statistics is the use of p-values. This means that a certain value threshold is set, for example five percent (0.05), and the p-value is compared to this. This is used in combination with null hypothesis testing meaning that the null hypothesis is accepted or rejected depending on the value of the p-value. The common interpretation is that if the resulting p-value is less than the threshold then the result from the study is viewed as true. However, the p-value can't actually say that something is true just because it is less than some predetermined threshold, it simply states that the results may need further investigation [18]. This means that the p-value can be misused and potentially overstate the importance of the results.

The acceptance threshold is also something that is chosen arbitrarily. The reason that five percent is so often seen in research utilizing p-values is simply because of historic reasons. There is nothing to indicate that an acceptance level of five percent is the best value, it might as well be three percent. After all, three percent is smaller than five percent so this would be an even more significant result. But since this threshold is used to compare a p-value that has no meaning, the acceptance threshold is equally meaningless [19][20].

When discussing confidence intervals at some percentage, it might seem intuitive to interpret these as this is the probabilities of the values within the confidence interval. But the correct interpretation is that this is the fraction of true values that fall within the confidence interval from an infinite series of repeated experiments [24]. Studies may use confidence intervals as basis for conclusions that, because of this misunderstanding, are wrong.

Another thing to mention is that the p-value does not separate between the effective sample size and the actual sample size (the total number of samples taken). An explanation for this is that if an effect is small then the data needed to confirm this effect needs to be large. However, this also means that you could prove just about anything if more data is added. Conversely if the data size is small it might not be enough to show the effect at all [19].

There is also no information in the p-value regarding the magnitude of the effect or the uncertainty of it. These two aspects are important to factor in when conclusions are to be made from the acquired results. So the null hypothesis can be rejected even though the magnitude of the effect is small and it might be rejected even though the uncertainty interval is wide [19]. A small magnitude in the results will lead to conclusions being made based on an effect that, in reality, does not impact the results much at all. A wide uncertainty means that the results can't be trusted. Especially if the uncertainty interval stretches over both sides of zero, in this case the actual effect might have the opposite function than the perceived one.

When the p-value falls within the acceptance threshold the null hypothesis can be rejected, when the p-value does not fall within the acceptance threshold the null

hypothesis can't be rejected (we fail to reject it). However, this does not necessarily mean that what the null hypothesis states is true. All the other possible hypothesis regarding the current research could still be true [20].

3.1.4.1 Null hypothesis significance testing (NHST)

NHST is part of the frequentist paradigm and to give a bit more information a very brief explanation of the basics of this will be given. Nickerson [51] gives some explanation regarding the basic concepts of NHST. As indicated by the name NHST makes use of a null hypothesis, this is something that we generally want to reject or disprove in order to be able to give credibility to the alternate hypothesis, which would be whatever it is that is trying to be proved. These two hypotheses might sometimes be referred to as H_0 for the null hypothesis and H_1 for the alternate hypothesis. Using NHST we get a value p which is used in conjunction with some predefined significance level, often represented by α (a common value for this is 0.05 or 5%), to determine if we can reject the null hypothesis. If the value of p is less than the value of α the null hypothesis can be rejected. If p is greater than α then we fail to reject the null hypothesis. This leads to the possibility of two kinds of errors, named type I and type II errors. A type I error happens when the null hypothesis is rejected when it is true and a type II error happens when the null hypothesis is not rejected when it is false.

3.1.5 Bayesian models

Bayesian statistics make use of mathematical models to analyze the given data. The following types models are the ones that are used in this thesis to answer the research questions posed in section 4.1. While the following models are used in this thesis they were all taken from existing literature.

3.1.5.1 Binomial model

This thesis will look into the probability of success since this will probably lead to more easily interpretable results given the available time. The model used will be a binomial model. Binomial models deal with outcomes that are either success or fail which makes it a good choice. The model can also be extended with the dimensions of each benchmark which will make it easy to observe the effects that increasing dimensions have in the overall success rate. The success rate of the benchmark functions will be the metric that is used to give an answer to this research question. The success rate in this case means that we see if an optimization algorithm produces a solution that is within a certain distance from the best solution, in this case we use a one decimal precision level.

3.1.5.2 Multi level models

Multi-level modeling (MLM) is a concept in statistics that is also present in Bayesian statistics. Normally when creating models using Bayesian statistics the parameters

of the model will have set values for things like the standard deviation and the average. When using MLMs these parameters can in turn have parameters themselves which have their own specific distributions [9, p.358-359].

The reason this is done is to give the model a sort of memory. Take the example of ordering coffee at several cafés and estimating the time it takes to get it [9, p.355-356]. Each café will take a different amount of time to process your order so we can't assume that they are all equal, but cafés in general are similar so we can't just forget about the time it took at another café. Using the distribution of waiting times is a way of solving this. When a new value for the waiting times is observed the estimates for both the individual café and the whole population are updated.

Using MLMs carries with it benefits such as improved estimates for repeat sampling, improved estimates for imbalance in sampling, estimates of variation and avoiding averaging and retaining variation [9, p. 356].

3.1.6 Priors

From Bayes' theorem in equation 3.1 we know how to update our beliefs about the hypothesis and thus create a posterior. One thing to note about this is that when a new value is added in order to update our beliefs, the previous posterior is now considered to be the prior for the new posterior that is created with the information from the new data value. This raises the question about what to do with the very first value observed and this is where priors enter.

For every parameter in the model used there needs to be a prior present to represent the initial probability distribution so that the machine performing the calculations has a starting point [9, p.34]. When a new value is observed our beliefs will update and the result will be a combination of the data point and the prior [55].

Priors can be weakly informative, strongly informative or somewhere in between [52]. This is basically saying how much the data is constrained by the priors. A weakly informative prior will constrain the data less than a strongly informative one will. Having a weakly informative prior means that the prior will be "overridden" more easily by a smaller amount of data points, while a strongly informative prior will require more data points to be "overridden" [53] [54]. The more strongly informative the priors are the better it is for the results, but often weakly informative priors are used [9, p.35]. However, even a weakly informative prior is better than a prior that gives virtually no information and sometimes it can help with mending things like diverging chains [9, p.258].

How is a good prior chosen then? There is no real way to calculate what a good prior should be, instead this is up to the researcher to decide. This means that the selection of priors is affected by the researchers experience with the topic and their subjectivity. It would however not be scientifically acceptable to choose a prior that is, in some way, biased since this would make the results questionable [27]. An example can be made with the proportion of land and water on the earth [9, p.28-30]. We know that there is about one third land and two thirds water on the earth and this information could be used as a prior. Of course, it is not always that we have this prior information so we can't always have a prior that's as good as that. So a weakly informative prior often takes place in stead in these cases.

3.1.7 Diagnosing the models

After using a model to analyze a set of data, the results from the model will need to be diagnosed to see if there were any problems during the analysis process. This is an important thing to do since problems during the analysis process can lead to unreliable results which in turn can lead to the wrong conclusions being made. This thesis will make use of two methods of diagnosing the results from the models trace plots and the \hat{R} values.

3.1.7.1 Trace plots

This diagnosis method is a more visual way of inspecting the results of the models. As the name suggests this method creates a trace plot of the stan models showing how well the different chains used in the analysis mixes. We want the chains to mix well, in other words the different lines in the plot should all be very entangled, not separated from each other as well as stay around the same area. The chains staying in the same area means that they stay on the same path in the posterior distribution and the chains mixing well means that each successive sample is not highly correlated to each other [9, p.253] Informally speaking the trace plots should be similar to a “fuzzy caterpillar”. This means that the chains in the trace plot should all overlap and not show any signs of a pattern, this gives a trace plot that resembles a “fuzzy caterpillar”.

3.1.7.2 \hat{R} (R-hat) values

Similar to the trace plots, the \hat{R} values also indicate whether the chains in a stan model mixed well or not. However, this is less visual in nature since it is only a single value. It is, in essence, a complicated way of quantifying the convergence of chains numerically [13]. A good \hat{R} value approaches one from above and even an \hat{R} value of 1.01 can be seen as suspicious. Aiming at an \hat{R} value less than this is to be desired but in practice having an upper limit of 1.05 still works [9, p.250-258].

3.2 Item response theory

IRT was originally developed in the field of educational research to investigate how the latent traits of students could be evaluated by an exam. Since its creation, it has undergone changes and seen use in other fields as well [30].

3.2.1 IRT models

There exist several models that deal with different problems and applications [30]. The model used in this thesis is the two-parameter logistic (2PL) model but the one- (1PL) and three-parameter (3PL) models will be discussed as well. They are divided into one-, two- and three-parameter due to the fact that they consider the parameters difficulty, discrimination and guessing to different degrees.

The 3PL model considers all three parameters and is described in equation 3.2.

$$p_i(\theta) = c_i + \frac{1 - c_i}{1 + e^{-a_i(\theta - b_i)}} \quad (3.2)$$

The parameters in equation 3.2 have explanations as described below [31].

1. $p_i(\theta)$: The probability of an individual with ability level θ to correctly answer a certain question.
2. c_i : This represents the probability of an individual with very low ability guessing correctly on a question.
3. a_i : This indicates the steepness of the probability curve. The higher the value, the more a small change to ability will affect the probability of a correct answer.
4. b_i : The difficulty level of a certain item.

The 3PL model can be simplified into the 2PL model by removing the guessing parameter c_i . This means that we are removing the probability of guessing correctly in any way. In this context it means that it is assumed that an algorithm can't just guess a solution to a benchmark function. Practically this is done by setting the guessing parameter c_i to zero, this means that we can simplify equation 3.2 into equation 3.3.

$$p_i(\theta) = \frac{1}{1 + e^{-a_i(\theta - b_i)}} \quad (3.3)$$

This can then further be simplified into the 1PL model by removing the discrimination parameter a_i . This means that the slope of the curve gets a constant value and all changes in ability level affect the probability of answering correctly the same. The value chosen for a_i to be fixed to will obviously affect the model depending on what it is, but say that we assume that a_i has a constant value of one, this will allow us to simplify equation 3.3 into equation 3.4.

$$p_i(\theta) = \frac{1}{1 + e^{-(\theta - b_i)}} \quad (3.4)$$

Using these models with Bayesian statistics works pretty much like using Bayesian statistics normally. The model used is coded in Stan using R but it is also using a package in R called “cmdstanr”, this is the only change from using Bayesian and R without IRT. After this the model is sampled using the NUTS algorithm described earlier to create the posterior distribution.

Figure 3.1 shows how the difficulty parameter and the discrimination parameter look like in a 2PL model. The top part shows that easy problems can be solved by test takers with low ability level and more difficult problems require a higher ability level. This makes sense intuitively and the plot also shows how the probability changes according to the difficulty level. This is however all dependent on the discrimination parameter, which in this case is set to three. Changing the discrimination parameter would also change how much an increase in the ability level affects the probability of solving the problem. The bottom part shows how the discrimination parameter affects the increase or decrease in probability based on the ability level. The higher the discrimination, the more drastic will the change in probability be when the ability level change. This part has a difficulty parameter that is set to zero in order to show the discrimination effect.

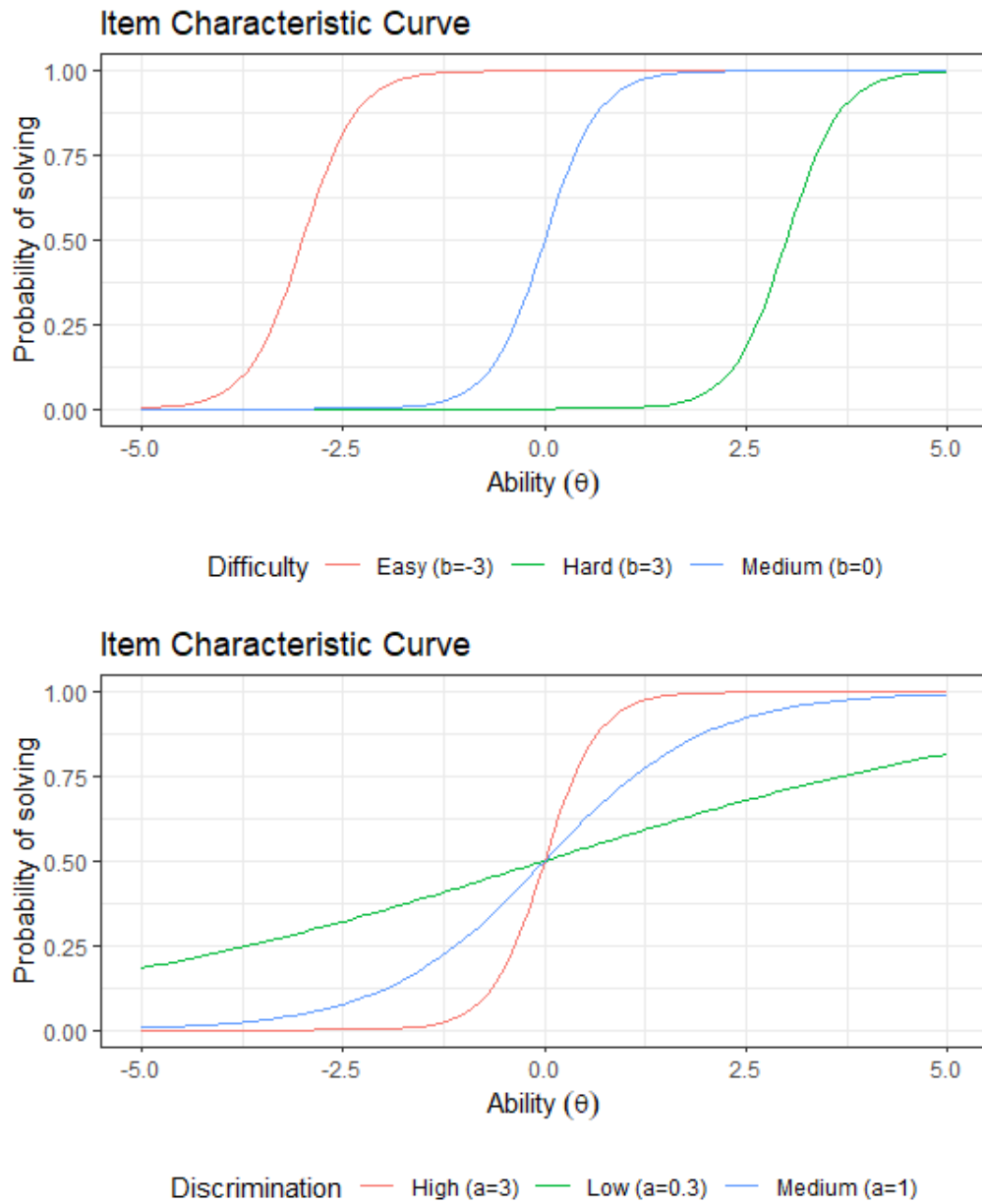


Figure 3.1: The impact of the difficulty and discrimination parameters in the item information curves. In the top part of the figure the discrimination parameter is set to three and in the bottom part the difficulty parameter is set to zero.

We used Bayesian statistics in combination with IRT and a two-parameter model. The two parameters that were used were the difficulty and discrimination parameters. Using these parameters made it easy to differentiate the benchmarking functions so that relevant information could be extracted and analyzed.

3.2.2 Item and test information

Using IRT we can get information regarding how much information is gained by a test and at what ability level this information gain is located. This information is calculated using equation 3.5 [30].

$$\begin{aligned} I_i(\theta) &= \frac{[\frac{\partial}{\partial \theta} P_i(\theta)]^2}{P_i(\theta)Q_i(\theta)} \\ Q_i(\theta) &= 1 - P_i(\theta) \end{aligned} \tag{3.5}$$

For the 2PL model equation 3.5 can be simplified to equation 3.6 [30].

$$I_i(\theta) = a_i^2 P_i(\theta) Q_i(\theta) \tag{3.6}$$

Using equation 3.6 we can visualize the item information curves for different difficulty and discrimination values, this can be seen in the top part of figure 3.2.

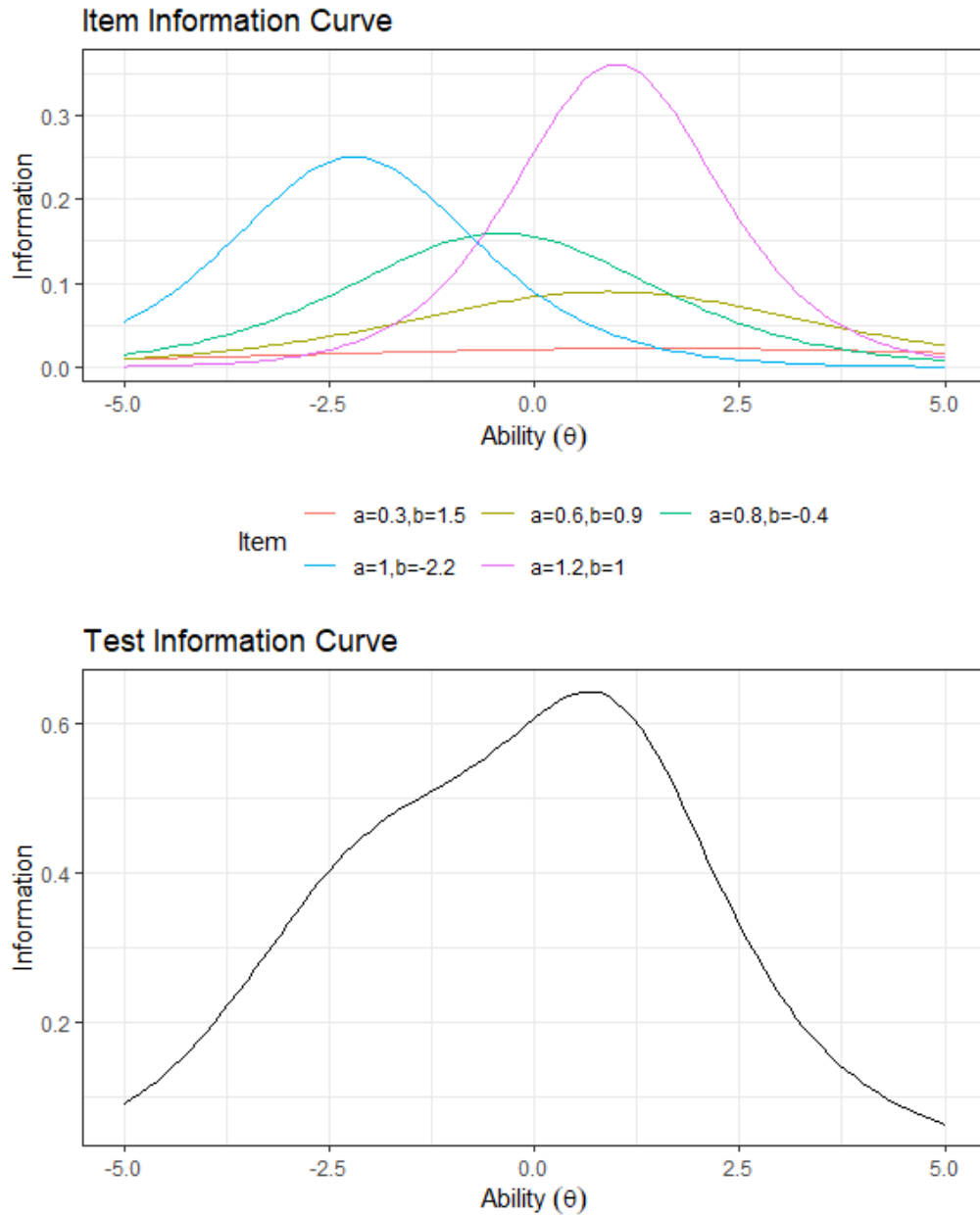


Figure 3.2: Item information curves and the test information curve. In the item information figure, a is the discrimination value and b is the difficulty value.

The item information curves shown in figure 3.2 shows how much information is gained at what levels of ability. This information can be combined together so we get the information for the entire test, which can be seen in the right part of figure 3.2. This will tell us at what ability levels this test is best applied in order to get the most information about the test takers. The equation used to get this test information curve is described in equation 3.7 [30].

$$I(\theta) = \sum_i I_i(\theta) \quad (3.7)$$

What equation 3.7 means is, basically, that all of the individual item information

curves are summed up and the result is the test information curve. This also means that we can design a test that better assesses the latent traits at specific ability levels by selecting specific item information curves that increase the information gained in those areas [30]. The areas where the most information is gained is also the areas where the standard error of measurement, as seen in equation 3.8, is low [30].

$$\text{SEM}(\theta) = \frac{1}{\sqrt{I(\theta)}} \tag{3.8}$$

4

Method

The purpose of this thesis is to evaluate benchmark suites and through this be able to select appropriate benchmark functions for the problem at hand. This chapter presents the research questions that are to be answered to achieve the purpose of the thesis and goes deeper into how these research questions are answered.

The data collected for each research question was first modified so that only data relevant to answer the current question was kept. More concretely, this means that certain columns in the data set were removed, added or had their names changed to make them more intuitive. With the data in an acceptable state it was analyzed using the Bayesian and IRT models and R. R was chosen due to the fact that we had previous experience with it and it is a known language for data analysis and as such it is relatively easy to find help online with potential problems. The results from the analysis process were diagnosed using trace plots and \hat{r} values in order to determine if any problems were encountered that would make the data unreliable. If no problems were found we moved on to investigating the results themselves in order to answer the current research question. Finally, at this point we simply present the results gathered and discuss them and what they mean for the research question.

4.1 Research questions

The following research questions are what this thesis is trying to answer.

1. What is a subset of benchmark functions from the two benchmark suites, BBOB [12] and from the Jamil et al. [6] study, that gives good results? In this case good results relate to the information gain at various ability levels. A subset of 30 benchmark functions will be selected to investigate this and this can be answered in different contexts as described below.
 - (a) Can we create a subset of benchmark functions that work well for the optimization algorithms chosen in terms of information gain at the ability levels that the algorithms are located? The ability levels of the optimization algorithms are related to the difficulty levels of the benchmark functions. It is basically a measure of each algorithms' "skill" and algorithms with an ability level similar to a benchmark functions difficulty level should be able to provide a good solution to these.
 - (b) Can we create a subset of benchmark functions that generally have a high area of information gain over several ability levels?
2. When comparing algorithms using scalable benchmark functions (benchmark functions that can take into consideration several dimensions at the same time)

from these two suites, do the results change if the amount of dimensions that we consider is increased?

3. Can we cluster these benchmark functions in different difficulty levels according to the difficulty levels in IRT?

By being able to select a subset of benchmark functions that provide roughly the same results as all of the functions in selected suites will reduce the time it takes to run all of the benchmark functions. This means that the amount of time required to benchmark an optimization algorithm could be reduced from days to hours (depending on the amount of benchmark functions chosen and their difficulty). This in turn has the effect of, for example, speeding up the development of a new optimization algorithm or more quickly be able to choose a good optimization algorithm for the problem at hand.

Knowing if the time required for an optimization algorithm to solve a benchmark function increases as the amount of dimensions increases is another important factor to consider when selecting benchmark functions. This is especially true if there is a need to use more than one dimension in the benchmark functions to properly evaluate optimization algorithms. To exemplify what dimensions refer to in this context we can look at equation 4.1 which is the Sphere benchmark function. D in this benchmark function, which can be arbitrarily big, is the number of dimensions considered and basically means the number of variables that are included in the benchmark function. Even though multiple dimensions can be considered there is still only one benchmark function producing results. For example, if two dimensions are used the optimal solution is $(0, 0)$, if three dimensions are used the optimal solution is $(0, 0, 0)$ etc. Optimization algorithms would also produce solutions in the same format which then could be compared to the optimal solutions to see how good the solution is.

$$\sum_{i=1}^D x_i^2 \quad (4.1)$$

Clustering the benchmark functions according to their difficulty levels makes it easier to choose functions based on where the test information curve needs to be improved. This means that it is easier to create benchmark suites that test the ability levels of interest.

4.2 Data collection

In order to be able to answer the posed research questions as good as possible given the available time two different data sets were collected and analyzed.

4.2.1 Data collection for the sub setting and clustering of benchmark functions

To answer research questions one and three the data to be analyzed was collected by running the selected benchmark functions and optimization algorithms in an already existing system and collecting the results from the different runs. Each optimization

algorithm was run ten times per benchmark function and 10000 function evaluations was performed per dimension. We used 247 benchmark function and these can be found in Appendix A, while the optimization algorithms used can be found below.

1. ABC
2. Bat
3. CuckooSearch
4. DifferentialEvolution
5. FireflyAlgorithm
6. GeneticAlgorithm
7. GWO
8. NelderMead
9. PSO
10. RandomSearch
11. SimulatedAnnealing

Using an existing system made the data collection process easier as the only thing to do in this step would be to select the benchmark functions that should be analyzed and which algorithms to use, which will allow for more time to analyze and compare the data. The specific data collected is described below.

1. BestArm (the result of the algorithm).
2. The number of function evaluations.
3. The current algorithm.
4. The current benchmark function.
5. The euclidean distance. This is the distance between the solution found by the optimization algorithm and the closest global minimum point.
6. The true reward difference. This is the distance between the minimum value the benchmark function can assume and the minimum value the optimization algorithm found.
7. The cumulative regret. Regret here means how much an optimization algorithms' solution differed from a benchmark functions optimal solution. The cumulative regret is the sum of all of these errors over all the runs of an algorithm on a benchmark function.
8. The time taken for an optimization algorithm to find a solution to a benchmark function.
9. If the benchmark function uses continuous data or not.
10. The benchmark functions' differentiability. If we can derive the function at any point within a range of values.
11. The benchmark functions' separability. A benchmark function is considered separable if n variables can be written as a sum of n functions using one variable and not if the variables are not independent [6].
12. The benchmark functions' scalability. If we can increase the dimensions of the benchmark function, in other words, if we can consider more variables in the benchmark function.
13. The benchmark functions' modality. If the benchmark function has many peaks it is multi-modal and if there is only one peak it is uni-modal.
14. The number of dimensions for the current benchmark function.

15. If the optimization algorithm managed to find a optimal solution to the benchmark function.
16. If the benchmark function is part of the BBOB set or not.
17. The benchmark functions' base class. This relates to if the function is scalable or not, in other words, if it can consider more dimensions. If it is scalable the base class tells what the base function is, an example can be made with the "AckleyN6" function which has the base class "Ackley."
18. The standard deviation of the noise level used.
19. The maximum number of function evaluations. This is the number of times an optimization algorithm tests a certain value in a benchmark function to see if this is better or worse than the previously found value.
20. The maximum number of function evaluations per dimension. This is the number of times an optimization algorithm tests a certain value in a benchmark function for each dimension included to see if this is better or worse than the previously found value.
21. The number of function evaluations per dimension. This is the number of evaluations the optimization algorithm actually used per dimension and may be smaller than or equal to the maximum number of function evaluations per dimension but not larger.
22. The precision of the solution in regards to solving with a specified number of decimals. In this case the number of decimals were: zero, one, three and six.
23. If the optimization algorithm solved the problem without having the budget run out at a specific precision level. The precision level again refers to the optimization algorithms solution being within a certain number of decimals from the optimal solution and the number of decimals were: zero, one, three and six.
24. The simulation number (zero through nine depending on which run it is).

Even though all of this data was collected, only a subset of it was used to answer the research questions. The reason for this is simply because there is no need to use all of the data collected to answer the research questions. The data used to get the results is specified below, note that this data was only used to answer research questions one and three since for research question two another data set that had already been collected was used.

1. The current algorithm.
2. The current benchmark function.
3. The standard deviation.
4. The maximum number of function evaluations per dimension.

Additionally, four more columns were created and used.

1. The algorithm ID.
2. The benchmark function ID.
3. How many times each optimization algorithm was run on each benchmark, in this case it is ten.
4. The number of times an optimization algorithm found a solution at one decimal precision for the benchmark function.

4.2.2 Data collection for the analysis of dimensions

For research question two, the same data that was collected before could be used here as well. However, another data set was used instead, the BBOB-2019 data set. Since research question two investigates the impact of different number of dimensions when comparing algorithms, it makes sense to use a data set that has information that includes as many dimensions as possible. The data collected for research question one and three only includes information for dimensions up to ten, while the BBOB-2019 includes information for dimensions up to 40 (2, 3, 5, 10, 20 and 40), in other words, up to 40 different variables were included in these benchmark functions. This makes the BBOB-2019 the more appropriate choice in here as it includes information on more dimensions and these results should be able to answer the research question since the number of dimensions is the only thing being considered. The specific algorithms used can be found below.

1. adapt-Nelder-Mead-scipy-2019
2. Adaptive_Two_Mode
3. BFGS-scipy-2019
4. CG-scipy-2019
5. COBYLA-scipy-2019
6. DE-scipy-2019
7. L-BFGS-B-scipy-2019
8. Nelder-Mead-scipy-2019
9. Powell-scipy-2019
10. RS-4-initIn0
11. RS-5-initIn0
12. RS-6-initIn0
13. TNC-scipy-2019

Each algorithm was run 15 times per benchmark function and dimension (so a total of 90 runs per algorithm). The specific reason as to why it was done in this way is not known since we were not involved in its collection. Using another data set here was deemed better as it had more information regarding benchmark functions using many dimensions. This both saved time and also makes it easier for others to check the results as the data set is available online. Another thing to note about this data set is that the benchmark functions used are not known. However, this is not necessary information in this case as it is the impact of increasing dimensions that is being investigated and this is possible to do without knowing the specific benchmark functions used. The specific data that had been collected in this data set is described below.

1. The optimization algorithm ID (the name of the algorithm).
2. The number of dimensions.
3. The benchmark function ID (numeric value 1-24).
4. The worst recorded result.
5. The worst result actually reached.
6. The best result actually reached.
7. The mean result actually reached.
8. The median result actually reached.
9. The number of runs.

10. The number of successes.
11. The budget. The total number of evaluations that the algorithm used when it tried to solve the problem. This varies quite a bit depending on optimization algorithm, benchmark function and dimensions.

From this list the following data is actually used to answer the second research question.

1. The optimization algorithm ID (the name of the algorithm).
2. The number of dimensions.
3. The benchmark function ID (numeric value 1-24).
4. The number of runs.
5. The number of successes.
6. The budget.

Additionally, two more columns were created and used.

1. The total number of runs.
2. The algorithm ID. This is different from the previous one as this is a numeric ID.

In order to assess the impact of more dimensions, the number of successful outcomes will be examined in relation to the increasing number of dimensions. Examining what happens to the number of successful outcomes as the number of dimensions increases will allow us to come to a conclusion regarding the impact of dimensions. This will be examined using an odd ratio which will be explained in more detail in section 6.6.

4.2.3 The system used for the data collection

As mentioned before, the system used for the data collection was something that already existed and did not require any extra work in the context of the thesis. The system itself is written in Python and runs on Docker and Google cloud. The evolutionary algorithms included come from the NiaPy library (<https://github.com/NiaOrg/NiaPy>) and the pycma library (<https://github.com/CMA-ES/pycma>), both available for Python. In addition to this, the “random search” algorithm has its own implementation in the system.

When gathering data using the system, the user can define parameters as described below.

1. Number of repetitions for each algorithm in each benchmark.
2. Noise (normal distributed noise) level (standard deviation). When a result is found for a benchmark function the noise level is considered to alter the result found depending on how severe the noise level is. This is basically a way of introducing a bit more randomness when getting the results from the tool and a way of trying to make the problems more realistic.
3. Maximum number of function evaluations per dimension.

After the system has collected the data, the results of each repetition are then saved in a csv file.

The system has been used to collect data for research before such as for the paper written by Mattos et. al. [22]. For more information regarding the specifics of

the code of the system one can look through the repository available at GitHub (<https://github.com/davidissamattos/hypercomp>).

As a minor note here regarding this tool, as with any other tool, if there is some sort of flaw with it that affect the results that it produces, the results of this entire thesis will be affected. This is something that we will have to accept however as it would be unreasonable to perform this study without it.

4.3 Benchmark functions and algorithms

The benchmark functions came from the study by Jamil et al. [6] and from the BBOB test suite [12]. All of the benchmark functions included in these two sources were included in this thesis. The optimization algorithms were selected from the NiaPy library because this library was the largest and provided the easiest integration. In order to answer the questions posed in section 4.1 specific data needs to be collected during the data collecting process. The data that was collected from the benchmark functions is described in section 4.2.1 and section 4.2.2.

4.4 Data analysis

The analysis of the data will be conducted using R, a language for data analysis. R includes a library for using Stan, which is a tool for building and inspection Markov chain Monte Carlo (MCMC) estimates [9, p. 241]. This is the technique that will generate the probability distribution using the data collected before and it is from this that we get the final results that we will use to answer the research questions. The underlying math of the models and the tools used to analyze the data is something that has already been investigated in the past and shown to work. Because of this we will simply be using ones that already exist and not focus on creating new math or tools.

Research questions one and three can be answered using the same analysis and simply looking at different results from it. Bayesian statistics are used, however the main difference here is that an IRT model will be used. The data used is prepared by doing some simple column operations, like removing columns that are not used and creating new ones to make it easier to extract the results later, like creating another identification column or something similar. The model was executed using four parallel chains with 2000 iterations and 1000 of these were used for warm up. This simply means that 2000 samples were collected per chain and that the chains sampled simultaneously (because the chains were parallel) to speed up the process and not have to wait for one chain to complete before another one could start. We used four chains because the computer used had four cores and thus each chain could use one core. The warm up is basically the same as running a car engine a bit before driving the car. It readies the system for the sampling process. After the model has run using the data, the results can be viewed in different formats, for example, as a graph or as a table, to make easier to present and comprehend. At this stage the relevant results can be examined in order to give answers to both research question one and three.

For research question two Bayesian statistics is once again used, however this time a binomial model will be used instead of an IRT model. The reason for this was to make it easier to answer the research question and thus save a bit of time. It was also not possible to collect information regarding how the dimensions affected the results of the optimization algorithms using the IRT model used for research question one and three. The rest of the analysis process is the same as for the other two research questions. The data is prepared, the model is used with the data and the results are examined to find an answer. The model was executed using four chains, 30000 iterations with 2000 of these being used as warm up. Again, this means that 30000 samples were collected per chain and that all chains sampled simultaneously.

4.5 Threats to validity

Since we are only considering a small amount of optimization algorithms and benchmark functions compared to all the choices that exist, this study will not be very comprehensive. Adding more optimization algorithms and benchmark functions would be beneficial as it would give more accurate results regarding the ability and difficulty levels of the optimization algorithms and benchmark functions. This could then affect the results we get in all of our research questions.

When we are clustering the benchmark functions into different difficulty level, this is very much up to the subjective definition of how the different difficulty levels are defined. Using a definition other than the one that is used here could produce different results and lead to a different division of the benchmark functions.

Using the tool that we use to collect data is another potential threat to the results presented here. The reason for this is that if there exist some kind of flaw in the tool that affects the quality results of the data collected all follow up analysis will also be affected. This could then lead to inaccurate results being presented and conclusions that are not valid.

The three research questions presented here gives information regarding how benchmark functions can be differentiated and selected. However, these are not the only aspects that exist. So it is a bit restrictive regarding what to consider when selecting appropriate benchmark functions and considering other aspects could lead to different benchmark functions being a better choice.

When investigating the effects of the dimensions on the optimization algorithms another data set is used, which is another threat to validity. A different data set was used from the first one as this made the data collection a lot faster. However, using the original data set instead of this new one could lead to other results being found and different conclusions being made. An upside to this is obviously that more dimensions were able to be included when the data set was analyzed and since it is available online, it is also easy for the results to be verified by others.

5

Selection of benchmarks with IRT

This research question aimed at investigating if it was possible to select an appropriate subset of benchmark functions for the ability levels of the optimization algorithms used. To reiterate, the ability levels of the optimization algorithms are related to the difficulty levels of the benchmark functions and is basically a measure of each algorithms' "skill" at solving the benchmark functions. We are able to view to ability levels of each of the algorithms in the same process as when we are identifying the benchmark functions' difficulty levels. Because we used the optimization algorithms to identify the difficulty levels of the benchmark functions we also have access to the ability levels of the optimization algorithms as this was identified by R when we ran our model and we got information regarding how many times the optimization algorithms were able to produce acceptable solutions to the benchmark functions. With this information we can plot the ability levels of the optimization algorithms as well as the difficulty levels of the benchmark functions. Two parts of this research question was created, the first part, to see if it was possible to create a subset of benchmark functions with a high area of information gain around the same level that the benchmark suite containing all the benchmark functions has. The second part, to see if it was possible to create a subset of benchmark functions with a high level of information gain over a wide interval of ability levels.

If there exists a subset of benchmark functions from the two suites used that can produce similar results, in terms of information gain in the context of IRT, as all of the functions in both suites, it could be preferable to use this smaller subset as it would save time.

IRT will be used to answer this research question as it provides a very intuitive way of deciding if a benchmark suite works well for the chosen optimization functions. The abilities of each of the optimization algorithms are evaluated along with the difficulty level and discrimination value of each benchmark function. Using the difficulty levels and the discrimination levels of the benchmark functions we can get a graph showing at which ability level the function gives the most information. Using all of these graphs for the individual benchmark functions we can choose a subset of them to maximize the information gained at a certain ability level. This would then lead to a benchmark suite that gives a lot of information regarding an optimization algorithms performance and how it compares to others.

5.1 Preparing the data

The data used to answer this question was the data collected from the BBOB suite [12] and from the Jamil et al. study [6]. The first step is to make some minor changes to the column names to make them more intuitive from what they were originally called. We also add a few columns creating and showing the ID number of each optimization algorithm and benchmark function. Additionally, two more columns are added showing the number of times each algorithm was evaluated on each benchmark function along with the number of times it successfully reached a solution. This data is required for the IRT model.

5.2 The IRT model

As previously mentioned, to answer this research question, IRT will be used. The IRT model used is a 2PL IRT model described below.

$$y_{i,p} = \text{Binomial}(N_{i,p}, \mu_{i,p}) \quad (5.1)$$

$$\mu_{i,p} = \frac{\exp(a_i \cdot (b_i - \theta_p))}{1 + \exp(a_i \cdot (b_i - \theta_p))} \quad (5.2)$$

$$a_i \sim \text{Half-Normal}(0, 3) \quad (5.3)$$

$$b_i \sim \text{Normal}(0, 3) \quad (5.4)$$

$$\theta_p \sim \text{Normal}(0, 3) \quad (5.5)$$

In the model above the parameters have the following explanations:

1. $y_{i,p}$: This is the result of the binomial.
2. $\mu_{i,p}$: The 2PL IRT equation
3. a_i : The discrimination parameter for the benchmark functions.
4. b_i : The difficulty parameter for the benchmark functions.
5. θ_p : The ability parameters for the optimization algorithms.

5.3 Selection of priors

The priors for a_i , b_i and θ_i are all weakly informative. Putting a stronger prior here could bias the results negatively since there is not enough information to confidently say that the prior should be more constrained to some value. A very wide prior could have a negative impact on the results as well. A weakly informative prior is in this case the best option then as it helps to constrain the data during the analysis process while not giving information that is wrong.

5.4 Diagnosing the convergence of the MCMC

The results from the IRT model are checked by looking at the trace plots for the different benchmark functions along with the \hat{R} values.

5.4.1 Trace plots for the difficulty parameter

The full list of the trace plots for difficulty parameter for the benchmark functions can be found in Appendix B. The reason for this is that there exists a trace plot for each one of the almost 250 benchmark functions.

The trace plots were checked manually to see if all the plots indicated that the chains from the MCMC mixed well. Figure 5.1 shows an example of what a good trace plot should look like. What we check is if the chains are stationary, in other words that the chains are “straight” in the plot and not making weird turns and such. We also do not want to see anything that could resemble a pattern and want the chains to look random. The values on the y-axis should also not have any major outliers, all the values in the chains should be roughly around the same level and thus “mixing” with each other. We want all of the chains we use to achieve all the things that we mentioned and if they do we can say that the trace plot show that no problems were encountered. The example shown in figure 5.1 exhibit all of the characteristics that we just mentioned. All of the chains are centered around values from around one to two, there are a few outliers but these are not major. There is also no discernible pattern in the chains and they all look random.

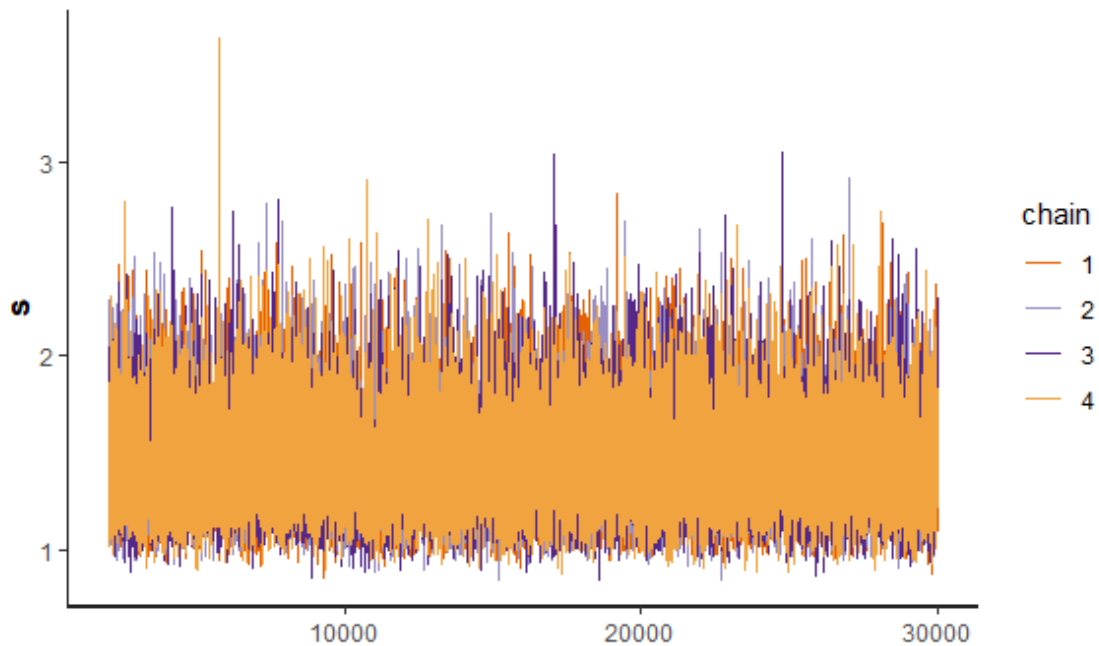


Figure 5.1: Example of a good trace plot. The values on the y-axis are the values sampled and the values on the x-axis are which sample “took” a certain value along with the total number of samples. The beginning of the plot is not included as these samples were used as warm up.

Looking through the trace plots for this research question we can see that all of them also show the same characteristics, indicating that no problems were encountered during the evaluation of the model.

5.4.2 Trace plots for the discrimination parameter

For the same reason as in section 5.4.1 the trace plots for the discrimination parameters for the benchmark functions are not included here, but instead can be found in Appendix C.

As with for the difficulty parameter trace plots, the trace plots for the discrimination parameters indicate no problems.

5.4.3 Trace plots for the ability parameter

We generate a trace plot of how well the chains mixed when analyzing the ability parameter of the algorithms and end up with the following.

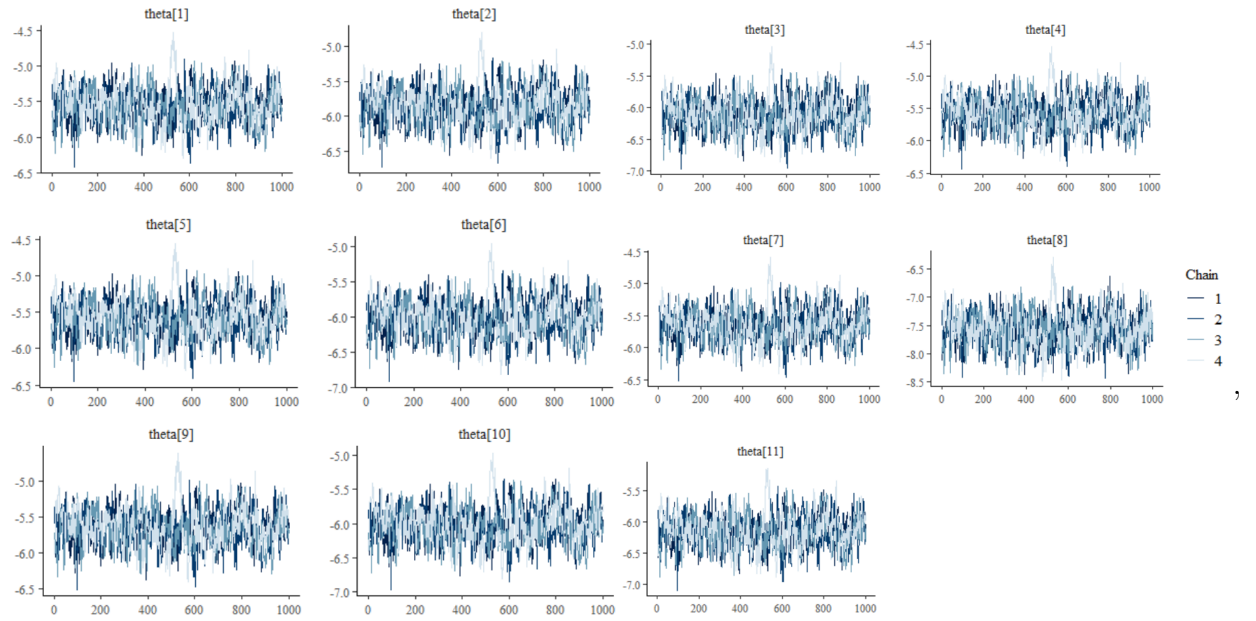


Figure 5.2: Trace plot showing the chain convergence for the algorithms

Figure 5.2 shows that all of the chains for all of the algorithms mixed well. In other words, there is no divergence in the chains so the result can be trusted. The graphs correspond to each algorithm used, the names of the algorithms used can be found in Appendix A.

5.4.4 \hat{R} values

The full list of \hat{R} values can be found in Appendix F in table F.1. All the \hat{R} values here are less than 1.05 and even less than 1.01. This is another indication that all the chains mixed well and further supports the fact that the results can be trusted. We come to this conclusion since an \hat{R} value close to one indicates that the chains have converged to the target distribution [9, p.250]. This is similar to how we inspected the trace plots visually before and a value close to one means pretty much the same as if we see that the trace plots doesn't indicate any problems. A value close to one is basically indicating that the chains have converged at the same values [13]. This

is something that should happen as they should all be sampling values using the same posterior distribution created by the data gathered. The threshold of 1.01 is a value that is very close to one and this has come to be accepted over time as more knowledge has been gained [56]. One interesting thing to note however is that some \hat{R} values approach one from below and in section 3.1.7.2 we said that the \hat{R} values should approach one from above. This could potentially indicate a problem but if we compare the results of the \hat{R} with the trace plots we can come to the conclusion that it is not a problem in this case since all the trace plots look fine. This is likely just a coincidence as the \hat{R} is estimated from the number of iterations used when running the model. So sometimes it can be less than one by chance. The fact that all \hat{R} values that are less than one all start with 0.999 and that the rest are all below 1.01 indicate good independent convergence of the chains.

5.5 Extracting the results

After it was confirmed that there were no problems encountered by analyzing the trace plots, the results data could be analyzed. The test information curve was analyzed to see how the algorithms related to it and to see if the benchmark suite was good for the used algorithms. The item information curves were also taken into account as selecting a specific set of these will allow us to create a suite that is tailored to the algorithms in question while using fewer benchmark functions and thus saving time.

5.6 Results

The individual item information curves for each benchmark function are what will decide how the final test information curve will look like. This means that we will need to see how these curves look like in order to be able to select an appropriate subset of benchmark functions. The full list of item information curves can be found in Appendix D. Combining all of these curves we get the test information curve that shows the ability levels that the entire test suite is best suited to measure. To reiterate the two parts of this research question, part a deals with creating a subset of benchmark functions with the difficulty levels around the optimization algorithms ability levels, part b deals with creating a subset of benchmark functions with wide area of high information gain.

5.6.1 Part A

Overlaying the ability levels of the optimization algorithms that are being investigated will more clearly show if the benchmark suite works well for the selected algorithms. The algorithms to make up the benchmark suite supposed to be close to the algorithms ability levels were chosen based on their difficulty level. Since we can see from Appendix B that no benchmark function has the exactly the same difficulty level as the optimization algorithms ability level selections has to be made based on how close they get. Every benchmark function has its median difficulty

level at some value and a credible interval between 5% to 95% as seen in Appendix F. Sorting these benchmark functions based on the lower, five percent credible interval and choosing the 30 with the lowest values was how the benchmark functions were chosen here. The reasoning behind this was simply to come as close to the ability levels of the optimization algorithms as possible. The specific chosen benchmark functions are listed below.

1. Alpine1N10
2. BartelsConn
3. Bohachevsky1
4. Bohachevsky2
5. CosineMixtureN10
6. ElAttarVidyasagarDutta
7. EllipsoidalN2
8. GriewankN10
9. GriewankN6
10. PinterN10
11. PinterN6
12. Price4
13. QingN10
14. QingN6
15. QuinticN10
16. SalomonN2
17. Schwefel2d20N10
18. Schwefel2d20N6
19. Schwefel2d21N6
20. Schwefel2d22N10
21. Schwefel2d22N6
22. Schwefel2d36
23. StretchedVSineWave2N
24. Trefethen
25. Trigonometric2N10
26. Trigonometric2N2
27. Trigonometric2N6
28. Tripod
29. WhitleyN2
30. WWavyN6

The specific discrimination and difficulty values for these benchmarks can be found in Appendix F in table F.2. Combining the information curves of each of these benchmark functions gives the resulting test information curve as seen in figure 5.3. The overall objective here was to try to find a set of 30 benchmark functions that together have a high area of information gain at the same area that the optimization algorithms' ability levels. This would mean that the optimization algorithms should be able to show their true capabilities since the difficulty levels of the benchmark functions would match the ability levels of the optimization algorithms. Looking at this figure we can see that even though an attempt was made to come as close as possible to the optimization algorithms ability levels as possible, the suite still

is not close enough to be able to give any reliable results at the ability levels the algorithms are located at.

5.6.2 Part B

A similar selection approach as was used when selecting benchmark functions to be as close to the ability levels of the algorithms was used here as well. However, in this case the focus was at creating a benchmark suite that had a decently high level of information gain over a much wider interval of difficulty levels. A suite such as this would be able to give accurate results to a wide variety of optimization algorithms that have ability levels that are similar to the ability levels that the benchmark suite has a high information gain at. So the benchmark functions were chosen one by one to create a sort of plateau (or at least close to a plateau) with a decently high information gain. The higher the information gain the better as the SEM value will decrease as the information gain increases which means that we can be more sure of the results we would get from the created benchmark suite. However, an exact value of what is high is not defined and is based on subjectivity. From the benchmark functions chosen here we can however see that it is possible to increase the information gain in specific areas and continuing to do this would keep leading to a lower SEM value, making the results more and more reliable. The specific chosen benchmark functions are listed below.

1. AttractiveSectorN2
2. Bunkin6
3. Corana
4. DixonPriceN10
5. EggHolder
6. FreudensteinRoth
7. Hartman6
8. JennrichSampson
9. Keane
10. Langerman
11. LinearSlopeN10
12. LinearSlopeN6
13. Mishra10a
14. Mishra10b
15. Mishra11N10
16. Mishra1N10
17. Mishra1N6
18. Mishra2N10
19. Mishra2N6
20. Mishra3
21. Mishra7N6
22. Mishra9
23. QingN2
24. QuinticN6
25. SarganN10

- 26. SarganN6
- 27. Schwefel2d4N10
- 28. XinSheYang3N10
- 29. XinSheYang3N6
- 30. ZakharovN10

The specific discrimination and difficulty values for these benchmarks can be found in Appendix F in table F.3. The fact that it covers the ability levels that it does was randomly chosen, but an effort was made to try to keep it as close as possible to the ability levels of the optimization algorithms. Looking at figure 5.4 we can see that the peak of the information gain for the selected benchmark functions lie around the ability level minus four to minus one. As mentioned, the overall objective here was to see if it was possible to find a set of 30 benchmark functions that together have a high area of information gain over several ability levels, regardless of what these ability levels were. The algorithms used are not entirely necessary to answer the first research question but it is interesting to see how they compare to the created benchmark suite. Looking at the ability levels of the algorithms, these all have ability levels less than minus five. This means that this benchmark suite would not be suitable to evaluate the selected algorithms. The reason for this is that the benchmark functions included in the suite have the highest area of information gain at ability levels that are way higher than the ability levels of the algorithms. Any results collected by this benchmark suite for these algorithms would have a high SEM (meaning that there exists a wide area of values within which the actual result can be located) and therefore will be very unreliable. In this case, with the optimization algorithms available, the created benchmark suite would serve as a example of a general benchmark suite for the ability levels that it covers. However, for benchmarking the selected optimization algorithms it is worth it to look into other algorithms to include that cover the area as the ability levels of the optimization algorithms.

To summarize, for the first part, it was found that with the selected optimization algorithms and benchmark functions it is not possible to create a good benchmark suite that would for the chosen algorithms. The ability levels of the optimization algorithms are much lower than the difficulty levels of the benchmark functions and even creating a benchmark suite comprised exclusively of the benchmark functions with the lowest difficulty levels still yields a benchmark suite that has the most information gain at a point that is well above the algorithms ability levels.

For the second part, it was found that it was possible to create a benchmark suite that had a decently high information gain over several difficulty levels. The specific difficulty levels that the benchmark suite focuses on was chosen by random but looking at the difficulty parameter of all the benchmark functions one can assume that it would be possible to create a benchmark suite with decently high information gain over any of those difficulty levels.

5.7 Discussion

The individual item information curves all combined form the test information curve shown in figure 5.5

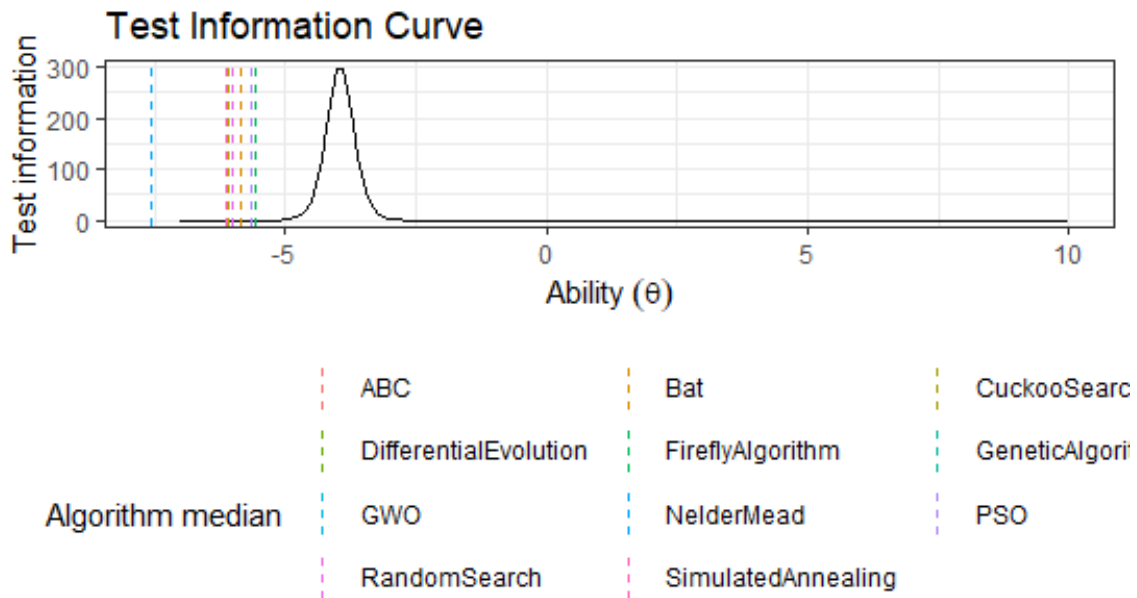


Figure 5.3: Curve showing where the maximum information gain from a test suite composed of 30 selected benchmark functions meant to be close to the algorithms ability levels

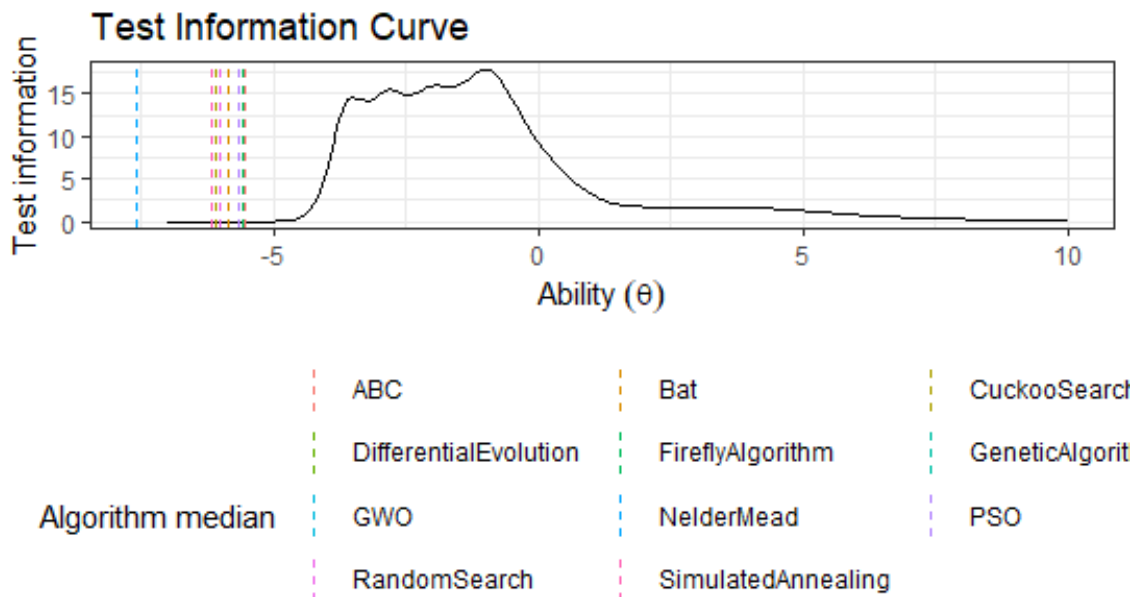


Figure 5.4: Curve showing where the maximum information gain from a test suite composed of 30 selected benchmark functions meant to create a good general benchmark suite

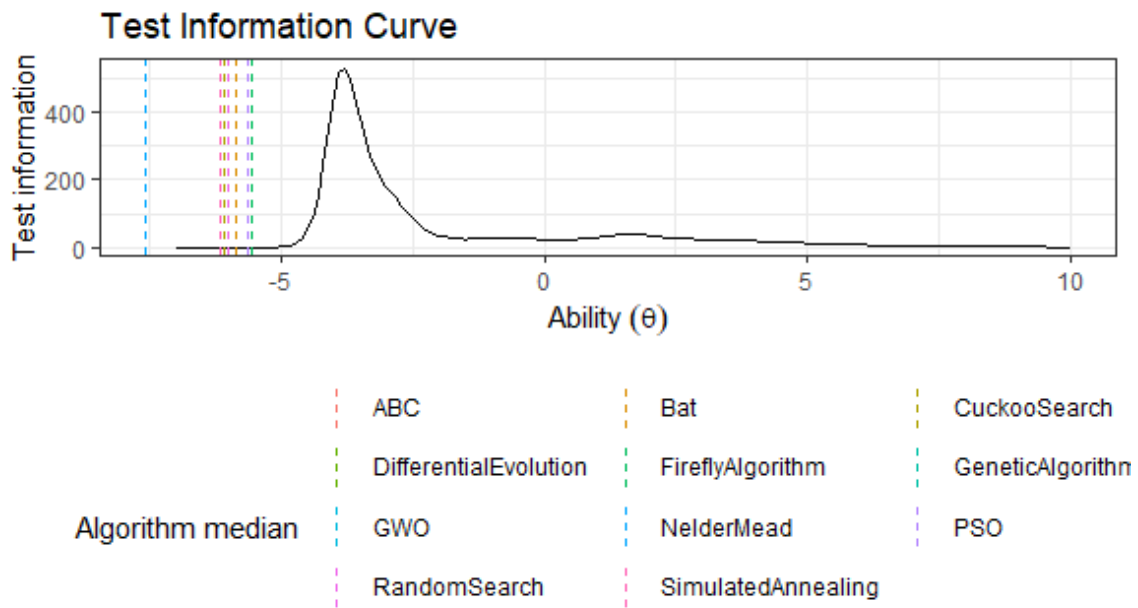


Figure 5.5: Curve showing where the maximum information gain from a test suite composed of all the selected benchmark functions

We can see that while this benchmark suite has a very large peak of over 450, it takes all the benchmark functions to create this test, which number around 250. The value of 450 can be used in equation 3.8 in order for the standard error of measurement to be known, in this case it equates to approximately 0.05. This is basically the value that is given to one standard deviation. Two standard deviations encompasses around 95% of the data distribution. This means that for the peak area around minus four, 95% of the values would be between -3.9 and -4.1 (two standard deviations on both sides of the mean value minus four). The higher the test information value, the lower the value for the standard deviation and the more concentrated the values will be. So we can be more certain about the ability level that provides the specific test information. There is no real way of determining what this value should be and it is up to the individual to set a good target value for the SEM.

By manually selecting item information curves to try to construct a benchmark suite that was able to give good quality results in regards to the ability levels of the chosen optimization algorithms the results were not very impressive. The resulting suite, as can be seen in figure 5.3, did not come close to the ability levels of the algorithms, even though the benchmark functions chosen were all on the lower side of the spectrum in terms of their difficulty levels. An appropriate suite would probably be able to be constructed if other benchmark functions were considered, but with those chosen in this thesis it is not possible.

We were also able to construct a benchmark suite that has a wide area of high information gain. So the benchmark suite covers a wide interval of ability levels, which means that an optimization algorithm is tested on a wide selection of difficulty

levels which gives good information on future problems an algorithm would do well on. It is also more likely that more optimization algorithms would be able to be properly benchmarked since the suite covers a wide area of difficulty levels. Since the information gain is also relatively high, we can be more sure that the performance of the optimization algorithms at the specific ability level. This area stretches from around ability levels of minus four to minus one and has an information gain of around 15 to 17.5 as seen in figure 5.3. These values represent the difficulty of the benchmark suite with lower values meaning easier and higher values meaning more difficult. Exactly what is easy and difficult depends on the optimization algorithms used since one algorithm might have a very easy time solving a problem while another algorithm might struggle to solve the same problem. While this benchmark suite does not have a peak near the same magnitude it does cover the same ability levels pretty well. Another big advantage is that it only uses 30 benchmark functions which should lead to a suite that is much more time efficient. We can also compare the manually created test suite by letting the computer randomly pick and choose 30 benchmark functions to form a benchmark suite, the results of this can be seen in figure 5.6.

These four samples are all similar that they all have a peak at the same area, around minus four, and they are also similar to the benchmark suite consisting of all the benchmark functions. The fact that they all have a similar peak is not surprising since most of the individual item information curves have peaks in this area, making it more likely for the combined benchmark suite to also have a peak here. Compared to the manually created benchmark suite these suites does not have the same “width” when it comes to the information gain in relation to ability level. So while they are able to give more information at their peaks than the manually constructed suite, they will not be as good at assessing a wide range of ability levels. So these benchmark suites will probably not make for good tests since they only focus in on this one particular area. The algorithms being analyzed will most likely have different ability levels and as such the information gained from these benchmark suites will not be much if they fall outside of this peak area. This shows that while it is possible to create a random suite of 30 benchmark functions that are similar to the entire original suite that they are chosen from this might not be the best thing to do. Using information regarding how the benchmark functions perform and at what difficulty levels can lead to the creation of a benchmark suite that is better at analyzing and comparing different optimization algorithms.

So what is a subset of benchmark functions that provides similar results as all the benchmark functions included in the two benchmark suites used? The answer to this is not definitive but we have shown that good subsets can be created. Figure 5.3 shows one possible configuration of benchmark functions that can be used to create a suite that both measures the test takers abilities well and at the same time use a lot less benchmark functions so that the time taken is reduced. Random selection of individual benchmarks is also a possibility as they provide decently good results in this case but they are still not as good as the manually selected benchmark suite. However as figure 5.3 shows, it is not possible to create a good benchmark suite for the optimization algorithms chosen using the chosen benchmark functions in this thesis.

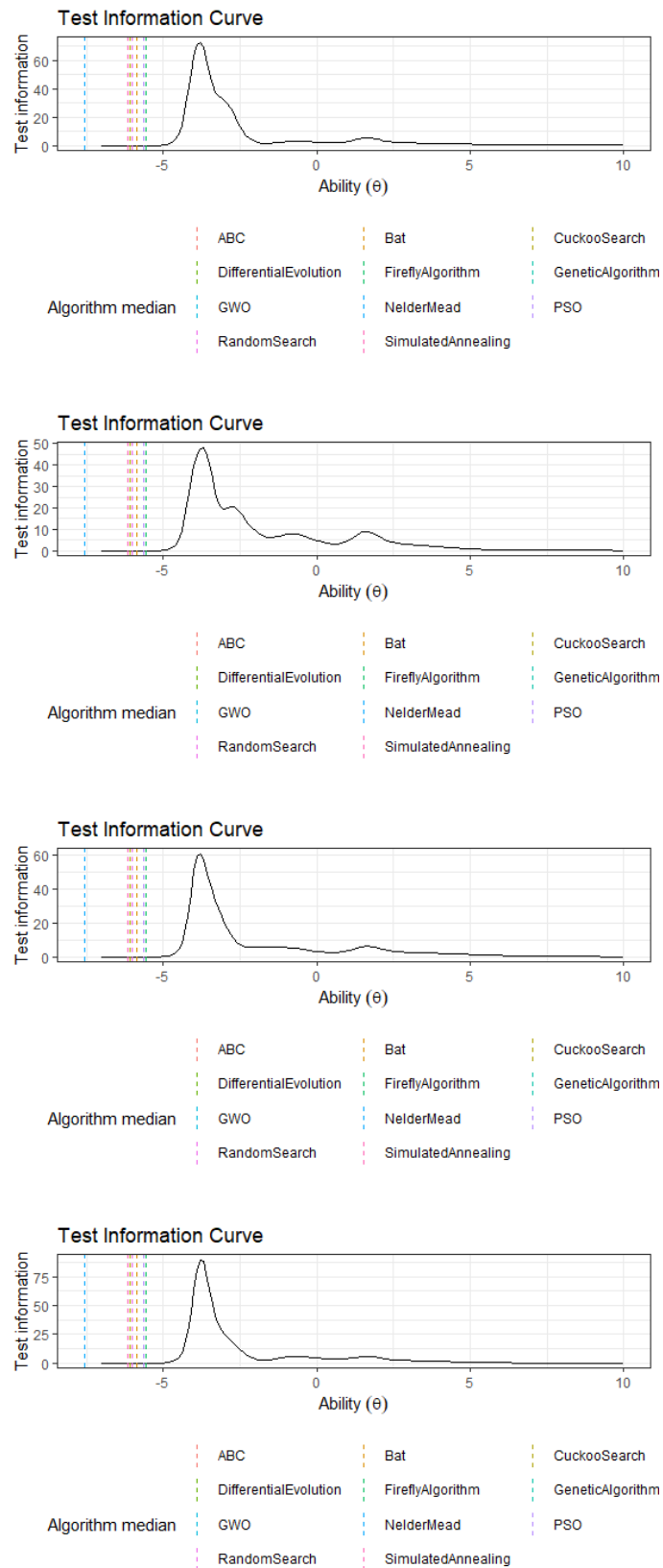


Figure 5.6: Four sample benchmark suites created by randomly selecting 30 benchmark functions

Knowing, with confidence, at what ability level an optimization algorithm works well and less well is valuable information when it comes to selecting an optimization algorithm to solve a certain problem. If an optimization algorithm is selected to solve a problem that is beyond its ability level the results will not be good. Being able to visualize at what ability levels the benchmark suite gives good quality results means that we will know at what ability levels an optimization algorithm works well and not. This also gives a good way of identifying at which ability levels the benchmark suite does not give good results that can then be improved by adding benchmark functions that increase the information gain at this specific ability level. This will also enable someone to create an appropriate benchmark suite to evaluate an optimization algorithm that is currently being developed. By including less benchmark functions in such a suite while still maintaining a reasonable level of information gain will allow for fast benchmarking of the optimization algorithm in development while the results are trustworthy, speeding up the development in comparison with using an existing benchmark suite. This all leads to a better way of choosing which optimization algorithms to use than just through guesswork or using what someone else used. The selection of suitable optimization algorithms for the task at hand then leads to improved performance in the system in which they are used.

Something important to note regarding this research question is that it is only dealing with a small subset of benchmark functions. Having more benchmark functions to choose from would make it possible to create suites that have other areas of high information gain. This could possibly enable one to create a benchmark suite that gets a lot closer to the ability levels of the optimization algorithms used. More benchmark functions could have been added to this study, however this would have increases the time needed to complete the study and due to time constraints this was not possible.

6

Increasing the dimensions used by the benchmarks

This research question aimed at investigating if increasing the number of dimensions considered in by a benchmark function affected the results.

Benchmark functions can have different amounts of dimensions which mean that the optimization algorithm using that benchmark function will have to try to optimize several things at the same time (several variables at the same time). The benchmark function is still only one equation but it has some way of considering multiple dimensions. An example would be the Sphere function mentioned in section 4.1. This can be contrasted to equation 2.2 and equation 2.3 where x and y are the only two variables able to be considered and there is no way of “adding” more variables. Having an algorithm that can find the best possible solution with consideration to two or more factors is something that can be quite useful. Therefore it is also important to know if an algorithm can produce reliable results while taking several factors into account. However, as the amount of dimensions increases it is also possible that the chances for the algorithm to successfully find a solution decreases. Knowing if the amount of dimensions affects the result of the optimization algorithms and in which way, will help in the selection of benchmarks to create an appropriate suite.

6.1 Preparing the data

Another data set was used to answer this question due to the fact that this new data set had information on a lot more dimensions than the first data set did. This also means that the algorithms used are different than the ones used to answer research question one and three. It also means that the new data set is also structured differently in terms of columns and the column names. This did not have much of an impact though and the only real modification of the data set that was required was the addition of a few columns just to make it more intuitive to work with.

6.2 Modeling the probability of success

The model used to investigate the probability of success is a binomial model. As mentioned in section 3.1.5.1, binomial models deal with success or fail outcomes, which is basically what the probability of success is, the amount of successful tries out of the total number of tries.

The actual model is the following.

$$y = \text{Binomial}(N, p) \quad (6.1)$$

$$p = \text{logit}^{-1}(a_{\text{benchmark},i} \cdot s + a_{\text{algorithm},j} + b_{\text{dimension},j} \cdot x_{\text{dimension},j}) \quad (6.2)$$

$$a_{\text{benchmark},i} \sim \text{Normal}(0, 1) \quad (6.3)$$

$$a_{\text{algorithm},j} \sim \text{Normal}(0, 5) \quad (6.4)$$

$$b_{\text{dimension},j} \sim \text{Normal}(0, 5) \quad (6.5)$$

$$s \sim \text{Exponential}(0.1) \quad (6.6)$$

In the model above the parameters have the following explanations:

1. y : This is the result of the binomial.
2. p : The probability of a successful outcome, in this case this means the probability of an optimization algorithm to find a solution within one decimal precision of the optimal solution.
3. $a_{\text{benchmark},i}$: The mean effect of the benchmarks.
4. $a_{\text{algorithm},j}$: The mean effect of the algorithms.
5. $b_{\text{dimension},j}$: The slope of the dimension.
6. $x_{\text{dimension}}$: Predictor for the dimensions.

6.3 Selection of priors

The priors for $a_{\text{benchmark},i}$, $a_{\text{algorithm},j}$, $b_{\text{dimension},j}$ and s are all weakly informative. The reason for this is the same as discussed in 5.3, there is not very much prior information regarding the data so we select a weakly informative prior to give the model a little help at least.

6.4 Diagnosis of the results from the stan model

Similar to sections 5.4.1 and 5.4.2 the full list of trace plots for the benchmarks can be found in Appendix G.

6.4.1 Trace plots

All the trace plots for the algorithms, dimensions and the random effect s seem to have mixed well. There is no indication of any worrisome parts in the trace plots and this is further indicated by looking at the \hat{R} values.

6.4.2 \hat{R} values

The full list of \hat{R} values can be found in Appendix F in table F.5. All the \hat{R} values are less than both 1.05 and 1.01 which indicates that all the chains mixed well. There are a few \hat{R} values that approach one from below but as mentioned in section 5.4.4 this is probably not a problem given that the trace plots for the chains all look good and all values that are below one start with 0.9999. This in combination with the fact that all the other \hat{R} values are good indicates that there is no problem.

6.5 Extracting the results

The trace plots showed no sign of problems so the resulting data could be analyzed without any new measures. Looking at the HPDI intervals for the different optimization algorithms along with the HPDI interval for the variables for dimension and s allows us to see if the amount of dimensions considered actually has an impact on the probability of success.

6.6 Results

Using HPDI plots we can get information regarding the success probability of each algorithm used in relation to the amount of dimensions in the benchmark function. This information is plotted using an odds ratio, this means that the number of successes is divided by the number of failures. So values close to zero means that an algorithm never or almost never succeeds.

Looking at the HPDI plots in figure 6.1 we can see that most of the algorithms have a pretty low odds ratio, meaning that they have a very low chance of succeeding at finding a solution to the benchmark functions. The two obvious exceptions to this is the Powell-scipy-2019 algorithm and the Adaptive_two_mode algorithm, which both have a higher median and wider credible interval. Looking at the HPDI plot for the effect of the dimensions, these all increase the odds ratio by quite a bit. Interestingly we can see that the Powell-scipy-2019 algorithm and the Adaptive_two_mode algorithm are affected the least by the increase in dimension.

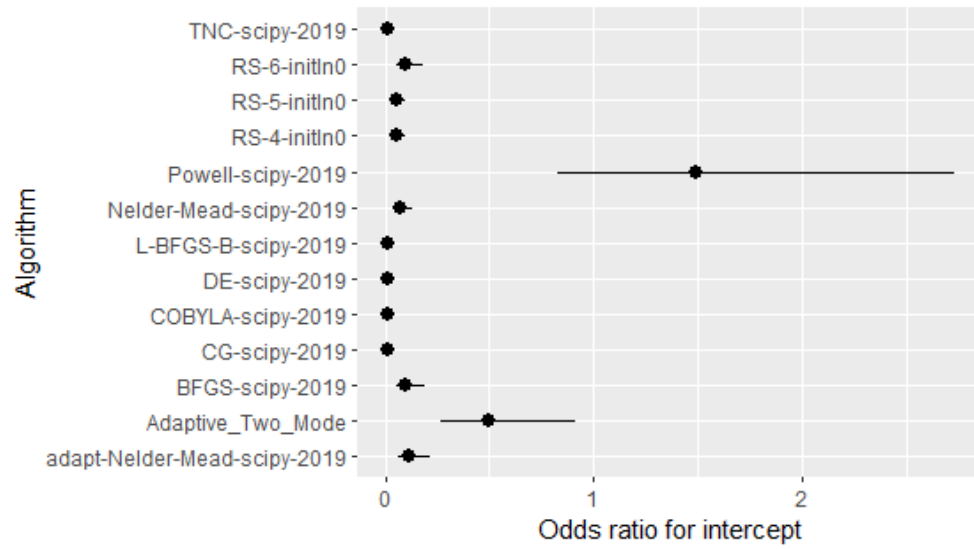
The effect of the benchmarks in 6.1 is also interesting. The effect of the dimensions on the benchmarks all has a pretty focused confidence interval and the overall spread is not very wide, ranging from around 0.85 to around 1.05.

The HPDI plot of figure 6.2 shows the random effect of each benchmark. We can see that this has a pretty high median and a wide confidence interval, ranging from around 1 to about 1.9. This means that while the random effect can have a more substantial impact it can also have less of an impact. The lowest part of the confidence interval is also located close to one which would mean that there is basically no effect at all. Also to note that the span from 1 to 1.9 is not very large in general so the random effect does not have much of an effect.

Looking at the plots in figure 6.1 we can what the effect of an increase in the dimensions does to the probability of success of an algorithm. The top hand plot shows that almost all of the algorithms used have a very low chance of finding a solution to the problems presented by the benchmark functions.

The bottom plot show what the effect of the dimension parameter is for each of these algorithms. This plot is interesting since they are all close to one which, at first glance, might lead someone to believe that the dimensions actually increase the probability of the algorithm to find a solution to the problem. However, the thing that should be kept in mind here is that this is an odds ratio. So values close to one means that the number of successes and failures are equal. Also, every single algorithm has a pretty low chance of solving the benchmark function in the first place as indicated by the top plot in figure 6.1. Because of this low chance of finding

HPDI interval for the algorithms



HPDI interval for the dimensions

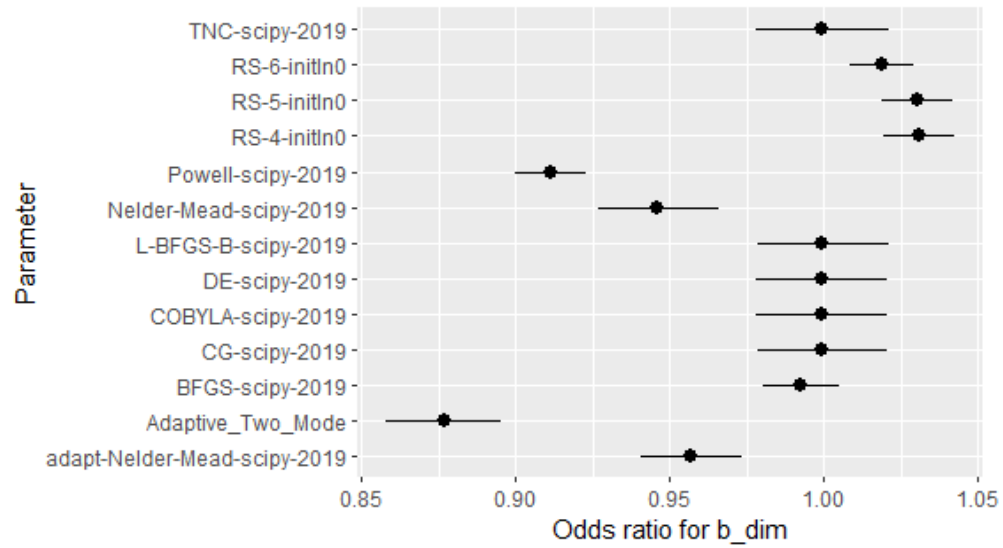
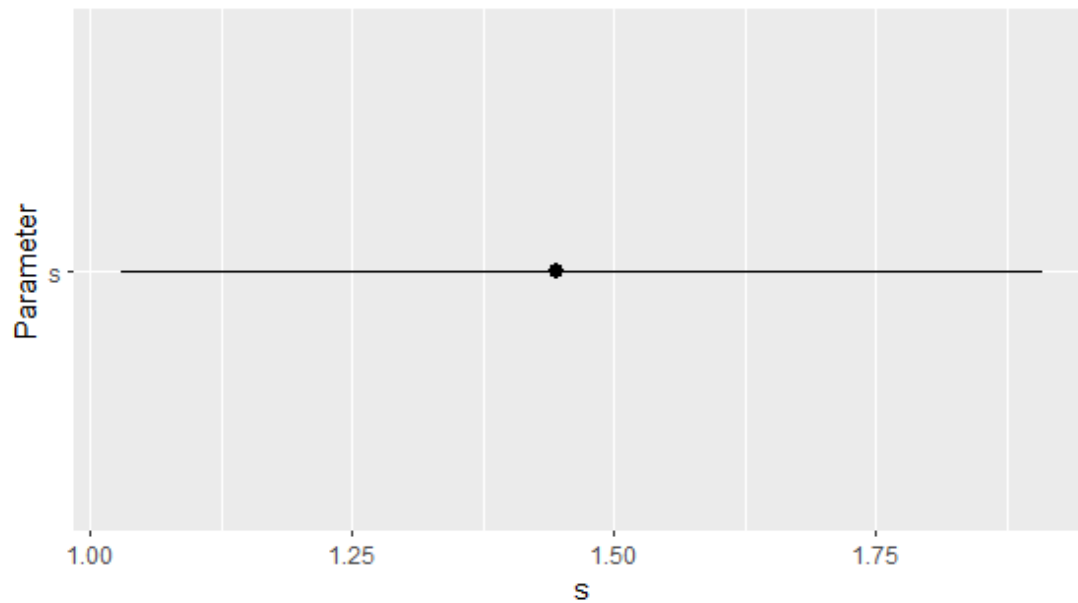


Figure 6.1: HPDI plots for the algorithms and the dimensions

HPDI interval for s **Figure 6.2:** HPDI plots for the random effect s

a solution, the increase in the number of dimensions essentially does not matter and this is what is shown in the bottom plot of figure 6.1 as well. Because the values are all very close to one this means that the effect of the dimension parameter is that it keeps the results in the same ratio as before. The algorithms would probably not find a solution to the benchmark function in the first place so adding additional dimensions to consider will then not have any effect since it only serves to make the problem harder. The span of values for the dimension parameter for each of the algorithms is also very narrow, meaning that they all pretty much have the same effect on each algorithm.

So in this case, increasing the number of dimensions that are considered by the algorithms does not impact the probability of the algorithms finding a solution to the benchmark functions in any way.

Summarizing, the results shows that there is almost no change in the odds ratio for the optimization algorithms success when the number of dimensions considered increases. Intuitively, if more things are being optimized at the same time one would think that it would be more difficult for the optimization algorithms to find a solution and thus the odds ratio would worsen. However, the optimization algorithms used have a much lower ability level than that of the benchmark functions difficulty levels. This means that even from the start the optimization algorithms had a very slim chance of actually finding a solution, so making the problem more difficult by adding more dimensions did not have much of an effect since the algorithms couldn't do that much worse from the start.

6.7 Discussion

It is quite feasible to want to optimize several things at the same time in an application. This means that it also makes sense to know how an increase in the dimensions to optimize affects the optimization algorithm. If the time take for an optimization algorithm to find a solution to a posed problem increases tenfold per each dimension added it is hardly worth using that algorithm. By comparing multiple optimization algorithms that could potentially be used to solve a problem with multiple things to optimize allows the developer to choose the best one. Much like in research question one this would lead to more efficient applications. Another interesting thing to bring up in this context is that intuitively one might expect that an increase in the dimensions considered leads to a lower chance of finding a solution. However, as shown by the results for research question two, this is not always the case. If an optimization algorithm already has a very low probability of solving a problem adding more dimensions will not change much. If a developer has a set of optimization algorithms that are being considered for use and benchmarking these would yield similar results as presented in section 6.6, the algorithm set might need to be reevaluated. It might also lead to a reevaluation of the underlying problem that was to be solved by the selected optimization algorithms. If the algorithms chosen performed very badly using several dimensions, then perhaps the problem is not understood as well as it could be.

Considering more dimensions would give a more comprehensive view of what the actual effect of the dimensions are. In this case of this thesis the optimization algorithms had an ability level that was far below that of the difficulty levels of the benchmark functions. So while these results show that there is no change when the number of dimensions goes up they should be considered in the context of this specific thesis and not as a general truth. More dimensions could have been considered as well as optimization algorithms and benchmark functions with similar ability and difficulty levels, however this would not fit in the time frame of the thesis.

7

Clustering the benchmarks according to difficulty level with IRT

Having a benchmark suite that is composed of benchmark functions that are of appropriate difficulty level when evaluating optimization algorithms is a key factor. If the benchmark functions are too easy the algorithms will probably always solve them with good results and thus no relevant information will be gathered. The same argument can be applied to a benchmark suite that consists solely of benchmark functions that are too difficult for the algorithms, but instead these will probably never be solved by the algorithms in question.

By using IRT here, we will be able to see what ability levels each benchmark function best tests individually and what ability level the entire benchmark suite tests. Using this information we should be able to select benchmark function based on the ability level we want to test and through this, be able to construct a benchmark suite that are appropriate for the optimization functions we want to investigate.

Since this research question is pretty much a continuation of the first research question in the way that the only thing different is that other things are investigated when extracting the results. Due to this fact the data preparation was already done and the diagnosis of the stan models were also the same.

7.1 Extracting the results

The results for this research question were acquired by plotting the difficulty parameters for the individual benchmark functions. Using zero as the reference point for “medium” difficulty we could define positive values (values greater than zero) to be more difficult and negative values (values less than zero) to be easier.

7.2 Results

The item information curves mentioned in section 5.6 are created by the difficulty and discrimination parameters of each benchmark. In order to cluster the benchmarks into different difficulty levels we need to look into how these difficulty parameters are distributed. The difficulty parameter distribution for each of the benchmarks can be found in Appendix E. Here we can see that a lot of the benchmarks

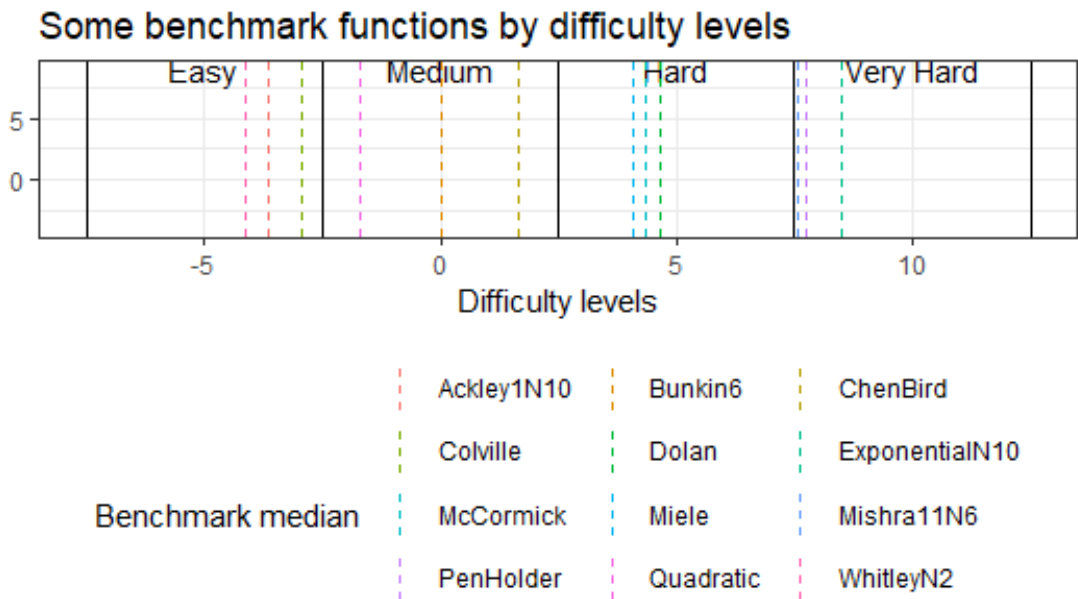


Figure 7.1: Example showing some benchmark functions divided into their respective difficulty levels. The solid black lines show the limits for the division of the difficulty levels. “Benchmark median” is the median difficulty level of the specific benchmark function.

have a difficulty level located close to minus five, however, there are still quite a few benchmark functions that have a positive difficulty parameter.

Looking at the figures in Appendix E and using zero as the value for “medium” difficulty, we can classify the benchmark functions into different difficulty levels. If we divide the difficulty levels into steps of five each and zero is the center of the difficulty “medium” then the difficulty levels between -2.5 and 2.5 are “medium”, difficulty levels between -7.5 and -2.5 are “easy”, difficulty levels between 2.5 and 7.5 are “hard” and difficulty levels between 7.5 and 12.5 are “very hard”. This is a very arbitrary division but it helps with the classification of the benchmarks. An example of this division can be seen in figure 7.1, the solid black lines represent the division between the difficulty levels.

With this we can conclude that most of the benchmarks are “easy” and afterwards there is quite a bit of variety in the difficulty levels “medium” and “hard” with only a few benchmarks reaching the “very hard” difficulty. This is supported by the test information curve in figure 5.5 where we can see that the peak of the information gain is located around ability level minus four. Cross referencing this with the plot of the difficulty levels in the figures in Appendix E we can see that the difficulty levels of most benchmarks are around the difficulty of minus four. By having this division of difficulty levels we can divide the benchmark functions into their respective general difficulty level, this result can be found in Appendix H.

Summarizing, the “difficulty” of each benchmark function was used in order to cluster the functions into “easy”, “medium”, “hard” and “very hard”. The difficulty

was defined through the use of IRT and the difficulty parameter of each benchmark function. This way made it easy to define intervals where the different difficulty levels were “located” and the benchmark functions could, through this, be clustered into different difficulty levels as shown in Appendix H.

7.3 Discussion

Using this way of classification we can come up with a benchmark suite that has a good spread of difficulties which makes the evaluation and comparison of optimization algorithms more reliable. If we also know the ability level of the algorithms that we are comparing, we can select specific benchmarks for these algorithms so that they are all evaluated in a way that more thoroughly tests their actual abilities. This clustering works especially well in combination with the visualization of the information gain at different ability levels. If one area of ability levels needs to be increased in terms of information gain we can easily identify benchmark functions that could be used for this purpose, if they are classified at some ability level already. Because this makes it easier to create a comprehensive benchmark suite it reduces the time needed to construct the suite and makes it easier and more reliable to determine if the benchmark suite gives a lot of information in the areas that we are interested in.

It is important to note that the difficulty levels set up here are purely based on subjectivity. There is no standard definition of difficulty levels such as “easy”. So while the definitions for the difficulty levels chosen here can be argued to be appropriate, it could also be argued that they are inappropriate. If more benchmark functions were to have been included the definitions for the difficulty levels might have needed to be reworked as they might not be good enough to sufficiently separate the benchmark functions in terms of difficulty. Having a definition of the different difficulty levels was necessary in this case and since there did not already exist one, we had to create one.

8

Conclusion

Optimization algorithms used in software engineering is a part of making sure that software can perform its specified purpose as good as possible. To evaluate optimization algorithms benchmark functions are used. In this thesis we investigate some of these benchmark functions to see how these can be chosen in a way that evaluates the optimization algorithm in question. Three aspects were considered, the division of a benchmark suite into a smaller but suitable subset, the impact of the number of dimensions considered and the division of benchmark functions into difficulty levels. The results show that it is possible to create a smaller subset of benchmark functions from a larger suite, that the dimensions considered does not make any difference to the results in this case and that it is possible to divide the benchmark functions into difficulty levels.

The results from this thesis show that it is possible to achieve better selection of benchmark functions, rather than just choosing them at random. Using IRT we can visualize at what difficulty levels the benchmark functions selected give the most information. Using this in combination with having benchmark functions clustered into different difficulty levels gives us an easy way of improving the information gain at any specific area of a benchmark function, provided that we have access to benchmark functions with a difficulty level in that area. At the same time, using IRT, we can visualize at what ability level the optimization algorithm in question is located. This makes it easy to select benchmark functions around the same difficulty level as the optimization algorithms' ability level allowing for thorough evaluation of the algorithms true capabilities.

The main property identified and investigated in this thesis is the difficulty levels of the benchmark functions. As pointed out, this is an important aspect when it comes to receiving reliable results from the benchmarking process. This can be used in combination with looking at the information gained at the difficulty levels of benchmark functions to create a suite that gives a lot of information about the optimization algorithms in question.

The results from the sub setting of benchmark functions show that we can create a benchmark suite consisting of fewer benchmark functions that still gives decent results at certain difficulty levels. This was accomplished using IRT and looking at the item information curves for the individual benchmark functions. This approach of analyzing how the benchmark functions and optimization algorithms perform in relation to each other and showing the results graphically or numerically makes it a lot easier for individuals to make informed choices regarding which benchmark functions to use. This is then better than just making selections at random as the suite created by randomly choosing benchmark functions is likely to not focus

around the ability levels of the optimization algorithms investigated. Analyzing how the dimensions affect the benchmark functions gives further information regarding which benchmark functions should be chosen as it reveals potential weaknesses in the benchmark functions when the dimensions increase. The success rate of the benchmark functions might drop dramatically as the dimensions increase, making them less usable. If optimization algorithms have a known ability level and we want to compare these we can make use of previous clustering of benchmark functions to be able to create benchmark suites quicker. If this clustering is done we do not need to analyze the benchmark functions all over again and can make use of the previous analysis and clustering of the benchmark functions and quickly create a benchmark suite that is suitable for comparing the optimization algorithms.

The purpose of this thesis was to evaluate benchmark functions and create a more suitable benchmark suite for discriminating optimization algorithms. The results found shows that it is very possible to achieve this using the methods discussed. From the benchmark functions included in this thesis we can create benchmark suites that cover a wide area of difficulty levels and thus tailor a benchmark suite to the specific ability levels of the optimization algorithms we are interested in investigating. The number of benchmark functions required to create a good suite is not very many, thus reducing the time needed to run all of the functions, and we know at what difficulty levels the benchmark functions give the most information. These two things makes it very simple to create good quality benchmark suites for any benchmark functions as long as the ability level of the optimization algorithms in question are known.

Having a choice of several optimization algorithms available a benchmark suite can be used to identify which one is the best to use for the current application that is being developed. Of course, this means that the benchmark suite needs to be able to evaluate all of the algorithms in a fair way, giving accurate information regarding the algorithms performance. Using the best optimization algorithm available makes the application perform better and this can help with achieving certain specifications regarding performance, security etc. Aspects like these can be very important if software is being developed for a client that needs to meet a certain level in these areas. The better a certain application performs the better it will also be in terms of competitiveness on the market. Applying the findings of this thesis to select benchmark functions in such a way that the best optimization algorithm is chosen helps achieve both of these things and while it is not the only thing that helps make software good it is a contributing factor.

As mentioned earlier, development of a new optimization algorithm is also something that can benefit from these findings. A benchmark suite can be created quickly, focusing on a certain difficulty level where the new optimization algorithm is supposed to operate. Using fewer benchmark functions the suite would take less time to use. The results from this could then be used to influence the direction of development for the new algorithm. This would lead to being able to create new optimization algorithms faster and being more sure that they are able to compete with existing ones in terms of performance.

Looking back at some of the previous research mentioned in section 1.1 we can get some ideas of where the benchmarking of optimization algorithms, and the findings

of this thesis, can come to have use in.

Automatic test generation is one such area. As mentioned, testing is something that is very expensive and time consuming and automating the process certainly helps. Using an optimization algorithm that performs better than another one has the potential to reduce the overall time taken while not compromising the results, leading to a faster testing process. Using the way of creating a suitable benchmark suite, using IRT and investigating ability and difficulty levels in combination with the amount of information gained at these levels, described in this thesis would allow for the selection of the optimization algorithm to use to be more informed and thus improve the testing process.

General software development can also benefit from the findings in this thesis. The study by Basios et al. [2] lead to increasing the efficiency of Google Guava through a solution they developed themselves. Benchmarking an optimization algorithm used in a certain software could indicate if there exists other options that might be better to use, leading to the software performing better overall. A similar argument can be made if a new optimization algorithm is being developed. Benchmarking this algorithm with a carefully chosen benchmark suite would allow for the optimization algorithm to become better, and thus potentially lead to a big performance increase in the software that it is used.

8.1 Future work

This thesis only uses a small subset of the available benchmark functions available, by conducting a similar study to this one and clustering the benchmark functions into difficulty levels would make future construction of benchmark suites a lot easier. Of course this would also require some sort of general definition of the difficulty levels to be created and spread as well.

A more thorough study of what the actual effects of the number of dimensions is, is another area that can be expanded upon. In this thesis we found that they did not matter much, however in this case the optimization algorithms abilities were far below the difficulty levels of the benchmark functions. It would be more interesting to see how the results would change if the optimization algorithms and the benchmark functions were of similar ability and difficulty level.

The main aspect considered when selecting benchmark functions in this thesis has been the difficulty levels of the benchmark functions in relation to the ability levels of the optimization algorithms. While this is an important aspect to consider to create a benchmark suite that fairly evaluates and/or compares benchmark functions, there exist other aspects as well. Another study investigating how to select benchmark functions in relation to these other aspects could lead to an even better understanding about how to best choose benchmark functions for your current purpose.

Bibliography

- [1] X.-S. Yang, “Optimization and metaheuristic algorithms in engineering”, *Metaheuristics in water, geotechnical and transport engineering*, pp. 1-23, 2013
- [2] M. Basios, L. Li, F. Wu, L. Kanthan and E. T. Barr, “Optimizing darwinian data structures on google guava”, in *International Symposium on Search Based Software Engineering*, Springer, 2017, pp. 161-167
- [3] S. Lukascyk, F. Kroiß, and G. Frasier, “Automated unit test generation for python,” in *International Symposium on Search Based Software Engineering*, Springer, 2020, pp. 181-187
- [4] D. Bruce, H D. Menéndez, and D. Clark, “Dorylus: An ant colony based tool for automated test case generation,” in *International Symposium on Search Based Software Engineering*, Springer, 2019, pp. 171–180
- [5] A. A. Al-Subaihin and F. Sarro, “Exploring the use of genetic algorithm clustering for mobile app categorisation,” in *International Symposium on Search Based Software Engineering*, Springer, 2020, pp. 181-187
- [6] M. Jamil and X.-S Yang, “A literature survey of benchmark functions for global optimisation problems,” *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013
- [7] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. Lopez-Ibanez, K. M. Malan, J. H. Moore, *et al.*, “Benchmarking in optimization: Best practice and open issues,” *arXiv preprint arXiv:2007.03488*, 2020
- [8] T. Hothorn, F. Leisch, A. Zeileis, and K. Hornik, “The design and analysis of benchmark experiments,” *Journal of Computational and Graphical Statistics*, vol. 14, no. 3, pp. 675-699, 2005
- [9] R. McElreath, *Statistical rethinking: A Bayesian course with examples in R and Stan.*, CRC press, 2015
- [10] L. F. Cardoso, V. C. Santos, R. S. K. Francês, R. F. Prudêncio, and R. C. Alves, “Decoding machine learning benchmarks,” in *Brazilian Conference on Intelligent Systems*, Springer, 2020, pp. 412-425
- [11] C. Zanon, C. S. Hutz, H. H. Yoo, and R. K. Hambleton, “An application of item response theory to psychological test development,” *Psicologia: Reflexão e Crítica*, vol. 29, 2016
- [12] algorithms-bbob,
<https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob>, Accessed: 2021-02-16
- [13] D. I. Mattos, and É. M. S. Ramos, “Bayesian paired-comparison with the bpcs package,” *arXiv preprint arXiv:2101.11227*, 2021

- [14] K. -J. Stol, and B. Fitzgerald, “The abc of software engineering research,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 1-51, 2018
- [15] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. Resende, and W. R. Stewart, “Designing and reporting on computational experiments with heuristic methods,” *Journal of Heuristics*, vol. 1, no. 1, pp. 1-32, 1995
- [16] S. García, D. Molina, M. Lozano, and F. Herrera, “A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: a case study on the CEC’2005 special session on real parameter optimization,” *Journal of Heuristics*, vol. 15, no. 6, pp. 617-644, 2009
- [17] A. Lacoste, F. Laviolette, and M. Marchand, “Bayesian comparison of machine learning algorithms on single and multiple datasets,” in *Artificial Intelligence and Statistics*, PMLR, 2012, pp. 665-675
- [18] R. L. Wasserstein, A. L. Schirm, and N. A. Lazar, *Moving to a world beyond “ $p < 0.05$ ”*, 2019
- [19] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon, “Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis,” *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2653-2688, 2017.
- [20] J. Gill, “The insignificance of null hypothesis significance testing,” *Political research quarterly*, vol. 52, no. 3, pp. 647-674, 1999.
- [21] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian data analysis*. CRC Press, 2013
- [22] D. I. Mattos, J. Bosch, and H. H. Olsson, “Statistical Models for the Analysis of Optimization Algorithms with Benchmark Functions,” *arXiv preprint arXiv:2010.03783*, 2020
- [23] B. Calvo, O. M. Shir, J. Ceberio, C. Doerr, H. Wang, T. Bäck, and J. A. Lozano, “Bayesian performance analysis for black-box optimization benchmarking”, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2019, pp. 1889-1897
- [24] C. A. Furia, R. Feldt, and R. Torkar, “Bayesian data analysis in empirical software engineering research,” *IEEE Transactions on Software Engineering*, 2019
- [25] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge, “Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges,” *Information Sciences*, vol. 317, pp. 224-245, 2015
- [26] O. Mersmann, M. Preuss, and H. Trautmann, “Benchmarking evolutionary algorithms: Towards exploratory landscape analysis,” in *International Conference on Parallel Problem Solving from Nature*, Springer, 2010, pp. 73-82
- [27] J. K. Kruschke, “Bayesian estimation supersedes the t test.,” *Journal of Experimental Psychology: General*, vol. 142, no. 2, p. 573, 2013.
- [28] M. D. Hoffman, and A. Gelman, “The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo.,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1593-1623, 2014
- [29] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Ridell, “Stan: a probabilistic programming language.,” *Grantee Submission*, vol. 76, no. 1, pp. 1-32, 2017

-
- [30] D. I. Mattos, L. Ruud, J. Bosch, and H. H. Olsson, “On the Assessment of Benchmark Suites for Algorithm Comparison,” *arXiv preprint arXiv:2104.07381*, 2021
 - [31] F. Martínez-Plumed, R. B. Prudêncio, A. Martínez-Usó, and J. Hernández-Orallo, “Item response theory in AI: Analysing machine learning classifiers at the instance level,” *Artificial Intelligence*, vol. 271, pp.18-42, 2019
 - [32] V. Beiranvand, W. Hare, and Y. Lucet, “Best practices for comparing optimization algorithms,” *Optimization and Engineering*, vol. 18, no. 4, pp. 815-848, 2017.
 - [33] J. N. Hooker, “Testing heuristics: We have it all wrong,” *Journal of heuristics*, vol. 1, no. 1, pp. 33-42, 1995.
 - [34] W. Vrieling, D. van den Berg, *et al*, “Fireworks Algorithm versus Plant Propagation Algorithm,” in *IJCCI*, 2019, pp. 101-112
 - [35] M. de Jonge and D. van den Berg, “Parameter sensitivity patterns in the plant propagation algorithm,” *submitted at the time of writing*, 2020
 - [36] A. E. Eiben, J. E. Smith, *et al*, *Introduction to evolutionary computing*, Springer, 2003, vol. 53
 - [37] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623-2631
 - [38] J. Bergstra, D. Yamins, D. D. Cox, *et al*, “Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms,” in *Proceedings of the 12th Python in science conference*, Citeseer, vol. 13, 2013, pp. 20
 - [39] R. S. Olson, and J. H. Moore, “TPOT: A tree-based pipeline optimization tool for automating machine learning,” in *Workshop on automatic machine learning*, PMLR, 2016, pp. 66-74
 - [40] I. Paenke, J. Branke, and Y. Jin, “Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 405-420, 2006.
 - [41] T. Bartz-Beielstein, L. Gentile, and M. Zaefferer, “In a nutshell: Sequential parameter optimization,” *arXiv preprint arXiv:1712.04076*, 2017.
 - [42] K. M. Malan, A. P. Engelbrecht, “A survey of techniques for characterising fitness landscapes and some possible ways forward,” *Information Sciences*, vol. 241, pp. 148-163, 2013.
 - [43] S. J. Pan, and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345-1359, 2009.
 - [44] J. Liu, A. Moreau, M. Preuss, J. Rapin, B. Roziere, F. Teytaud, and O. Teytaud, “Versatile black-box optimization,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 620-628.
 - [45] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*, Springer Nature, 2019.
 - [46] B. Doerr and F. Neumann, “Theory of evolutionary computation: Recent developments in discrete optimization,” 2019

- [47] B. Doerr, C. Doerr, and J. Lengler, “Self-adjusting mutation rates with provably optimal success rules,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 1479-1487.
- [48] M. Müller-Hannemann and S. Schirra, *Algorithm Engineering*, Springer, 2001.
- [49] G. Karafotias, M. Hoogendoorn, and Á. E. Eiben, “Parameter control in evolutionary algorithms: Trends and challenges,” *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 167-187, 2014.
- [50] SSBSE,
<https://ssbse.info/>, Accessed: 2021-06-07
- [51] R. S. Nickerson, “Null hypothesis significance testing: a review of an old and continuing controversy,” *Psychological methods*, vol. 5, no. 2, p. 241, 2000.
- [52] J. K. Kruschke and T. L. Liddell, “Bayesian data analysis for newcomers,” *Psychonomic bulletin & review*, vol. 25, no. 1, pp. 155-177, 2018.
- [53] J. K. Kruschke, “Bayesian data analysis,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 1, no. 5, pp. 658-676, 2010.
- [54] J. K. Kruschke and T. L. Liddell, “The Bayesian New Statistics: Hypothesis testing, estimation, meta-analysis, and power analysis from a Bayesian perspective,” *Psychonomic Bulletin & Review*, vol. 25, no. 1, pp. 178-206, 2018.
- [55] J. K. Kruschke, “What to believe: Bayesian methods for data analysis,” *Trends in cognitive sciences*, vol. 14, no. 7, pp. 293-300, 2010.
- [56] A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P. -C. Bürkner, “Rank-normalization, folding, and localization: An improved \hat{R} for assessing convergence of MCMC,” *arXiv preprint arXiv:1903.08008*, 2019

A

Appendix 1: The benchmark functions

A.1 The benchmark functions used when selecting a subset of and clustering benchmark functions

1. Ackley1N10
2. Ackley1N2
3. Ackley1N6
4. Ackley2
5. Ackley3
6. Adjiman
7. Alpine1N10
8. Alpine1N2
9. Alpine1N6
10. Alpine2
11. AttractiveSectorN2
12. BartelsConn
13. Beale
14. BiggsEXP2
15. BiggsEXP3
16. BiggsEXP4
17. BiggsEXP5
18. BiggsEXP6
19. Bird
20. Bohachevsky1
21. Bohachevsky2
22. Bohachevsky3
23. Booth
24. BoxBettsQuadraticSum
25. BraninRCOS
26. BraninRCOS2
27. Brent
28. BrownN10
29. BrownN2

30. BrownN6
31. BucheRastriginN10
32. BucheRastriginN2
33. BucheRastriginN6
34. Bunkin2
35. Bunkin4
36. Bunkin6
37. ChenBird
38. ChenV
39. Chichinadze
40. ChungReynoldsN10
41. ChungReynoldsN2
42. ChungReynoldsN6
43. Colville
44. Corana
45. CosineMixtureN10
46. CosineMixtureN2
47. CosineMixtureN6
48. CrossInTray
49. CsendesN10
50. CsendesN2
51. CsendesN6
52. Cube
53. Damavandi
54. DeckkersAarts
55. DeVilliersGlasser01
56. DeVilliersGlasser02
57. DixonPriceN10
58. DixonPriceN2
59. DixonPriceN6
60. Dolan
61. Easom
62. EggCrate
63. EggHolder
64. ElAttarVidyasagarDutta
65. EllipsoidalN10
66. EllipsoidalN2
67. EllipsoidalN6
68. Exp2
69. ExponentialN10
70. ExponentialN2
71. ExponentialN6
72. FreudensteinRoth
73. Giunta
74. GoldsteinPrice
75. GriewankN10

76. GriewankN2
77. GriewankN6
78. Hansen
79. Hartman3
80. Hartman6
81. HelicalValley
82. Himmelblau
83. Hosaki
84. JennrichSampson
85. Keane
86. Langerman
87. Leon
88. LinearSlopeN10
89. LinearSlopeN2
90. LinearSlopeN6
91. Matyas
92. McCormick
93. Miele
94. Mishra10a
95. Mishra10b
96. Mishra11N10
97. Mishra11N2
98. Mishra11N6
99. Mishra1N10
100. Mishra1N2
101. Mishra1N6
102. Mishra2N10
103. Mishra2N2
104. Mishra2N6
105. Mishra3
106. Mishra4
107. Mishra5
108. Mishra6
109. Mishra7N10
110. Mishra7N2
111. Mishra7N6
112. Mishra8
113. Mishra9
114. Parsopoulos
115. PathologicalN10
116. PathologicalN2
117. PathologicalN6
118. Paviani
119. PenHolder
120. Periodic
121. PinterN10

- 122. PinterN2
- 123. PinterN6
- 124. PowellSingular
- 125. PowellSumN10
- 126. PowellSumN2
- 127. PowellSumN6
- 128. Price1
- 129. Price2
- 130. Price3
- 131. Price4
- 132. QingN10
- 133. QingN2
- 134. QingN6
- 135. Quadratic
- 136. QuarticN10
- 137. QuarticN2
- 138. QuarticN6
- 139. QuinticN10
- 140. QuinticN2
- 141. QuinticN6
- 142. RastriginN10
- 143. RastriginN2
- 144. RastriginN6
- 145. RosenbrockModified
- 146. RosenbrockN10
- 147. RosenbrockN2
- 148. RosenbrockN6
- 149. RotatedEllipse
- 150. RotatedEllipse2
- 151. SalomonN10
- 152. SalomonN2
- 153. SalomonN6
- 154. SarganN10
- 155. SarganN2
- 156. SarganN6
- 157. Scaphfer1
- 158. Scaphfer2
- 159. Scaphfer3
- 160. Scaphfer4
- 161. SchafferF6
- 162. SchumerSteiglitzN10
- 163. SchumerSteiglitzN2
- 164. SchumerSteiglitzN6
- 165. Schwefel1d2N10
- 166. Schwefel1d2N2
- 167. Schwefel1d2N6

- 168. Schwefel2d20N10
- 169. Schwefel2d20N2
- 170. Schwefel2d20N6
- 171. Schwefel2d21N10
- 172. Schwefel2d21N2
- 173. Schwefel2d21N6
- 174. Schwefel2d22N10
- 175. Schwefel2d22N2
- 176. Schwefel2d22N6
- 177. Schwefel2d23N10
- 178. Schwefel2d23N2
- 179. Schwefel2d23N6
- 180. Schwefel2d26N10
- 181. Schwefel2d26N2
- 182. Schwefel2d26N6
- 183. Schwefel2d36
- 184. Schwefel2d4N10
- 185. Schwefel2d4N2
- 186. Schwefel2d4N6
- 187. Schwefel2d6
- 188. SchwefelN10
- 189. SchwefelN2
- 190. SchwefelN6
- 191. Shekel10
- 192. Shekel5
- 193. Shekel7
- 194. Shubert
- 195. SixHumpCamelBack
- 196. SphereN10
- 197. SphereN2
- 198. SphereN6
- 199. StretchedVSineWave2N
- 200. StyblinskiTang
- 201. SumSquaresN10
- 202. SumSquaresN2
- 203. SumSquaresN6
- 204. Table1
- 205. Table2
- 206. Table3
- 207. TesttubeHolder
- 208. ThreeHumpCamelBack
- 209. Trefethen
- 210. Trigonometric1N10
- 211. Trigonometric1N2
- 212. Trigonometric1N6
- 213. Trigonometric2N10

- 214. Trigonometric2N2
- 215. Trigonometric2N6
- 216. Tripod
- 217. Ursem1
- 218. Ursem3
- 219. Ursem4
- 220. UrsemWaves
- 221. VenterSobieszczzanskiSobieski
- 222. Watson
- 223. WayburnSeader1
- 224. WayburnSeader2
- 225. WayburnSeader3
- 226. WeierstrassN2
- 227. WhitleyN10
- 228. WhitleyN2
- 229. WhitleyN6
- 230. Wolfe
- 231. WWavyN10
- 232. WWavyN2
- 233. WWavyN6
- 234. XinSheYang1N10
- 235. XinSheYang1N2
- 236. XinSheYang1N6
- 237. XinSheYang2N10
- 238. XinSheYang2N2
- 239. XinSheYang2N6
- 240. XinSheYang3N10
- 241. XinSheYang3N2
- 242. XinSheYang3N6
- 243. ZakharovN10
- 244. ZakharovN2
- 245. ZakharovN6
- 246. Zettl
- 247. Zirilli

A.2 The optimization algorithms used when selecting a subset of and clustering benchmark functions

- 1. ABC
- 2. Bat
- 3. CuckooSearch
- 4. DifferentialEvolution
- 5. FireflyAlgorithm

6. GeneticAlgorithm
7. GWO
8. NelderMead
9. PSO
10. RandomSearch
11. SimulatedAnnealing

A.3 The optimization algorithms used when increasing the number of dimensions

1. adapt-Nelder-Mead-scipy-2019
2. Adaptive_Two_Mode
3. BFGS-scipy-2019
4. CG-scipy-2019
5. COBYLA-scipy-2019
6. DE-scipy-2019
7. L-BFGS-B-scipy-2019
8. Nelder-Mead-scipy-2019
9. Powell-scipy-2019
10. RS-4-initIn0
11. RS-5-initIn0
12. RS-6-initIn0
13. TNC-scipy-2019

B

Appendix 2: Difficulty parameter trace plots

The difficulty parameter trace plots for each of the benchmark functions are presented here. The number represents each benchmark function and their specific names can be found in Appendix A, in section A.1.

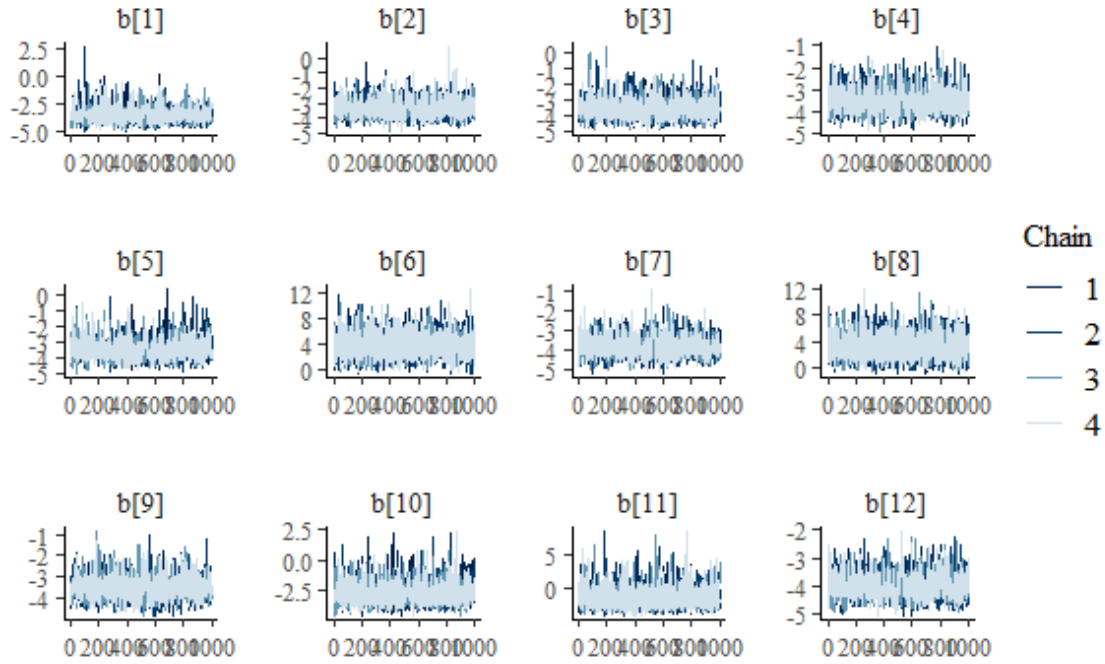


Figure B.1: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

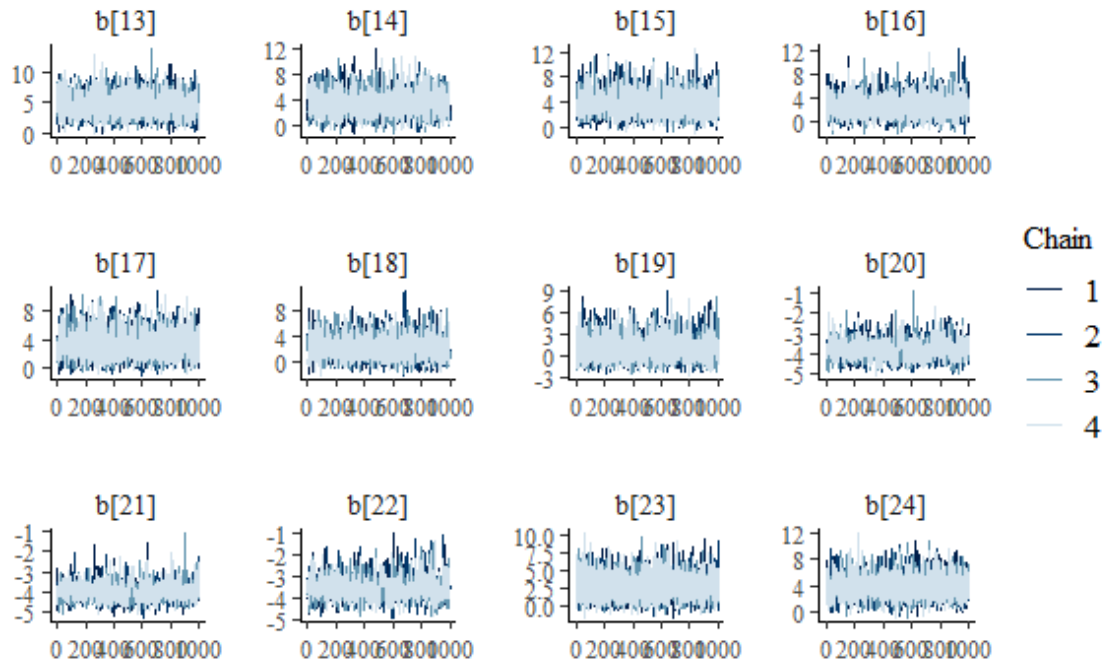


Figure B.2: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

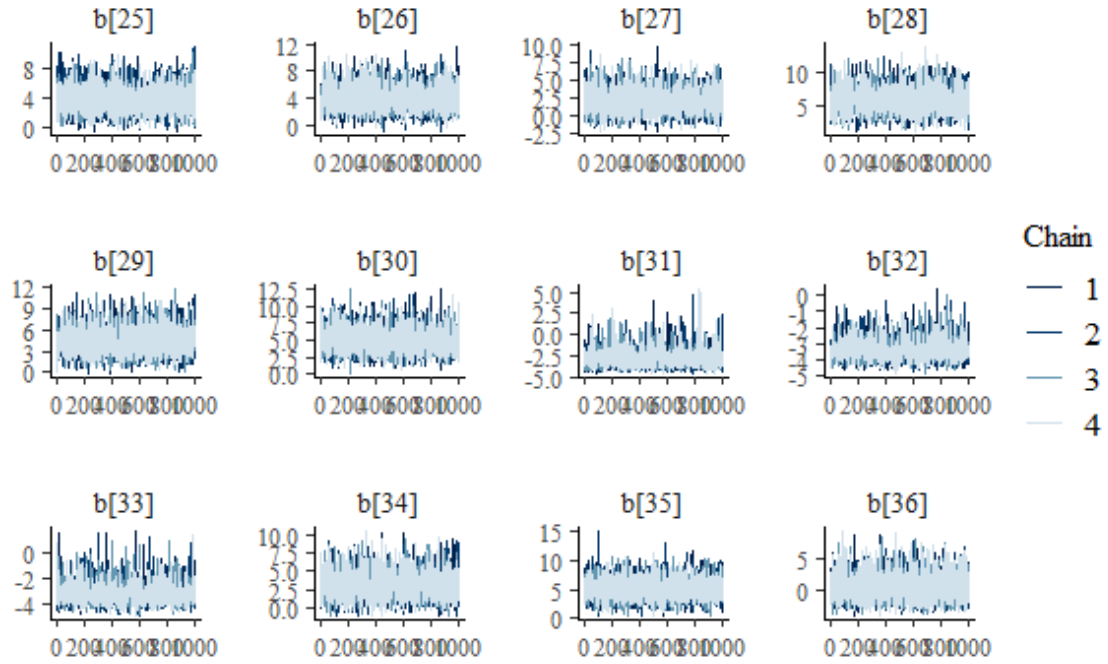


Figure B.3: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

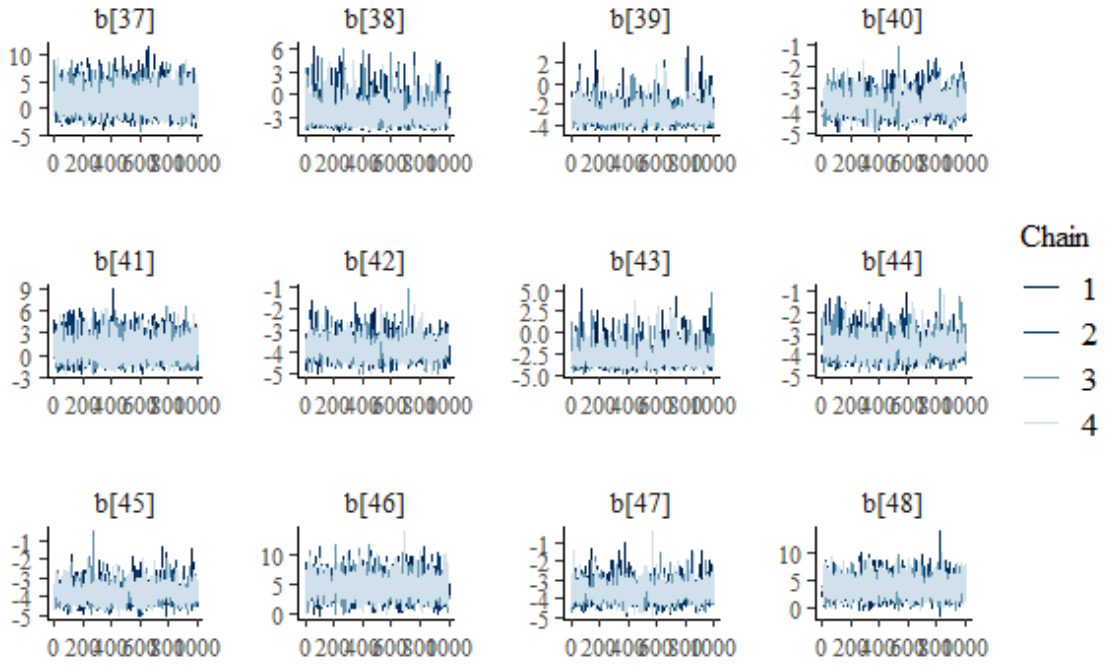


Figure B.4: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

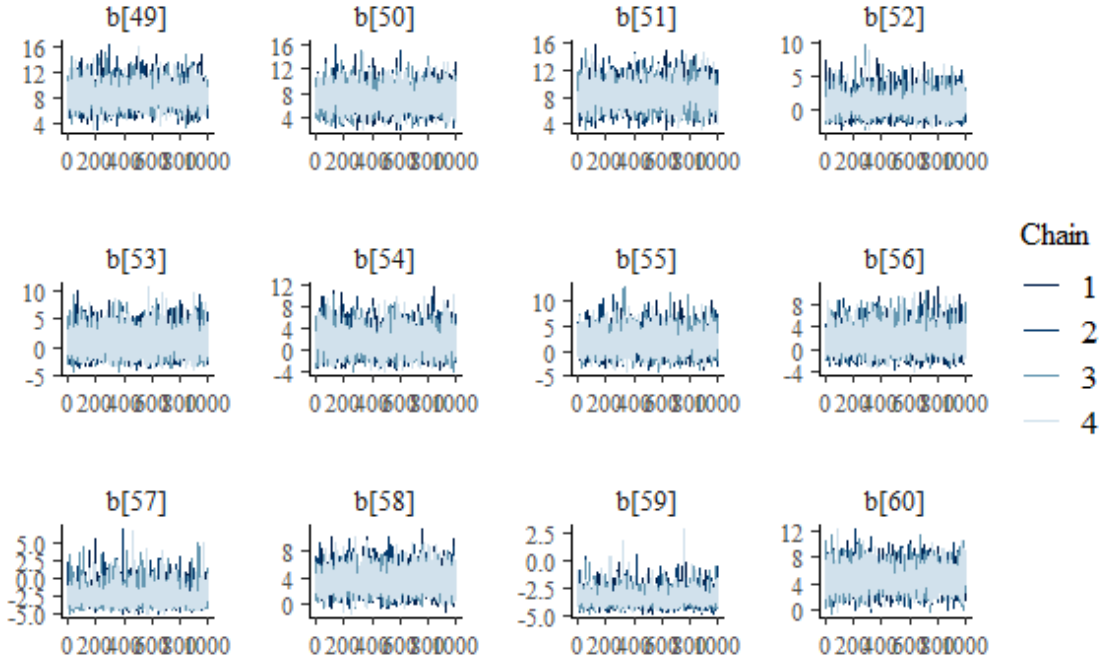


Figure B.5: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

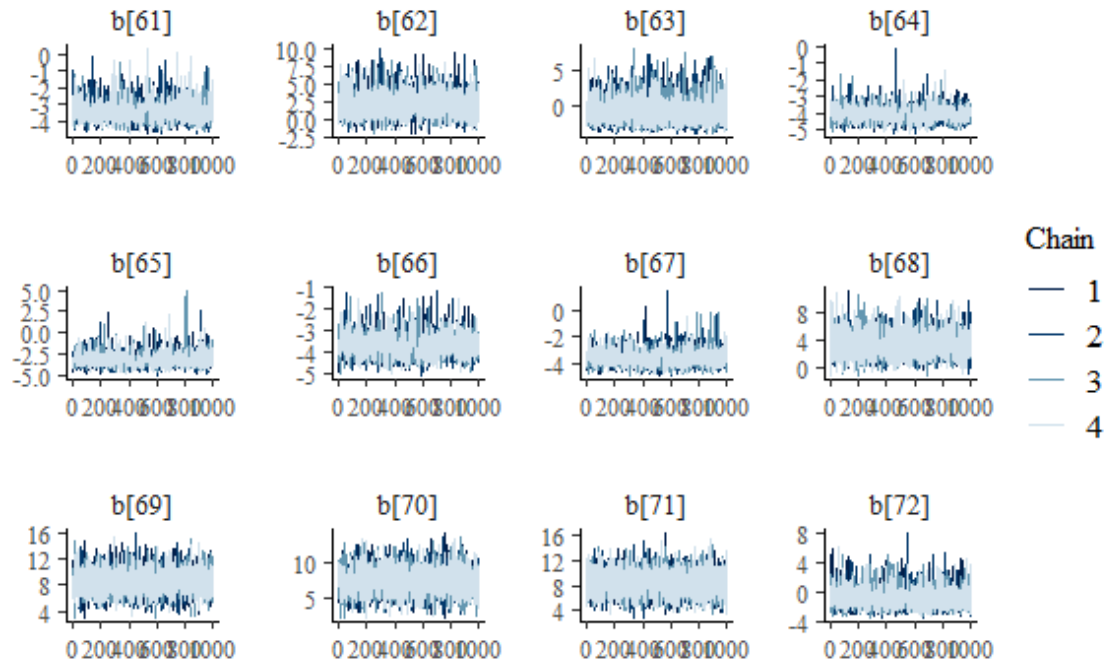


Figure B.6: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

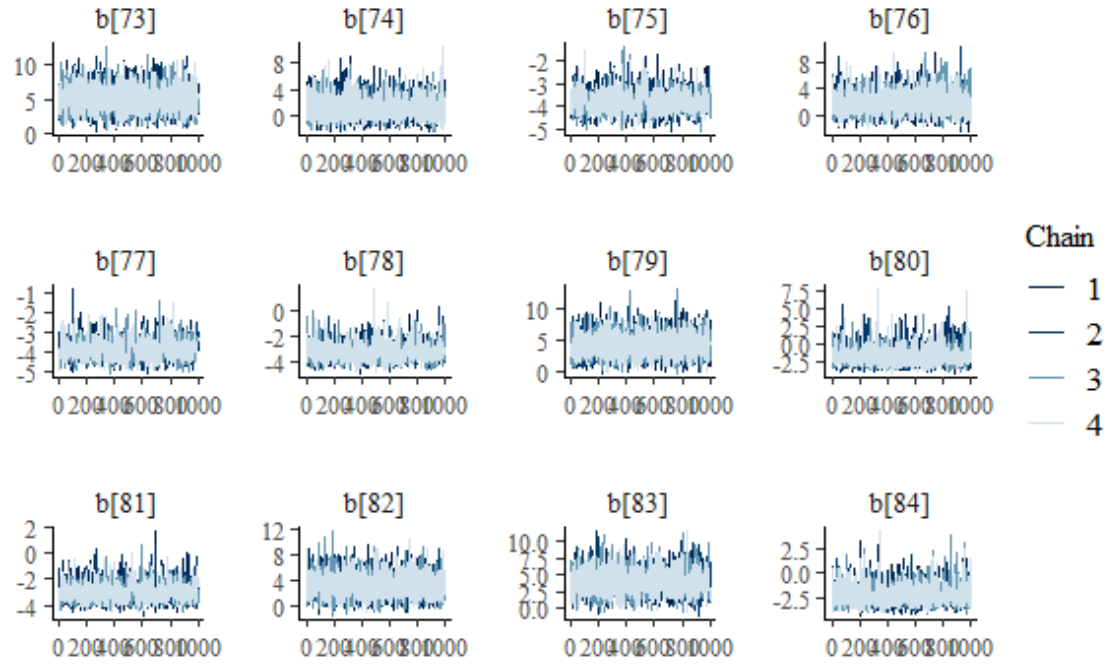


Figure B.7: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

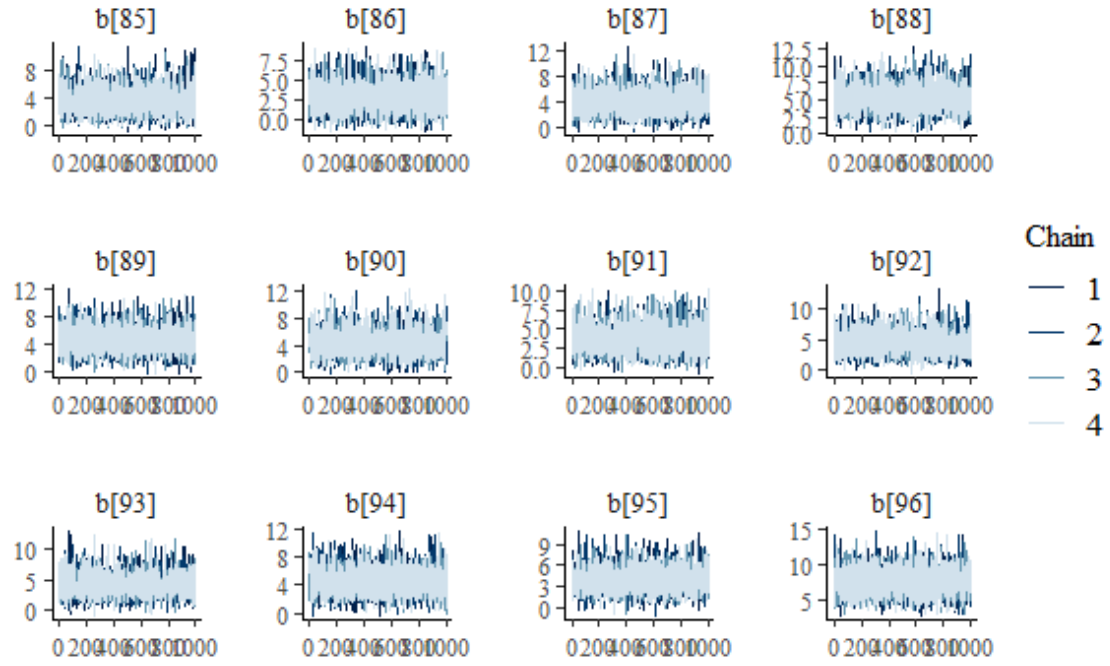


Figure B.8: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

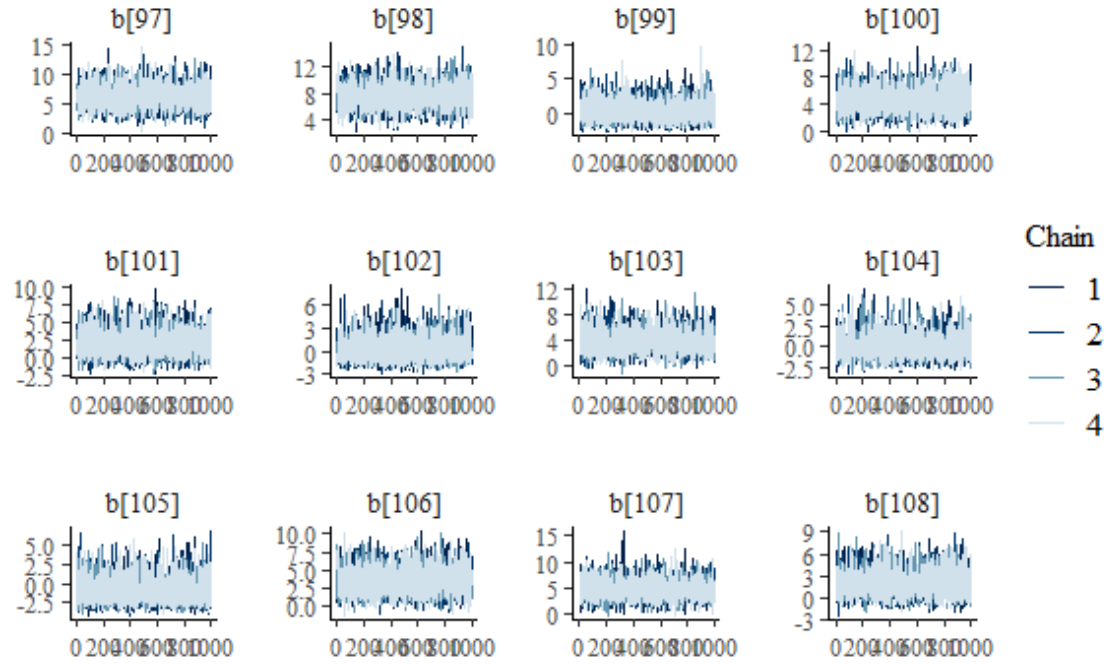


Figure B.9: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

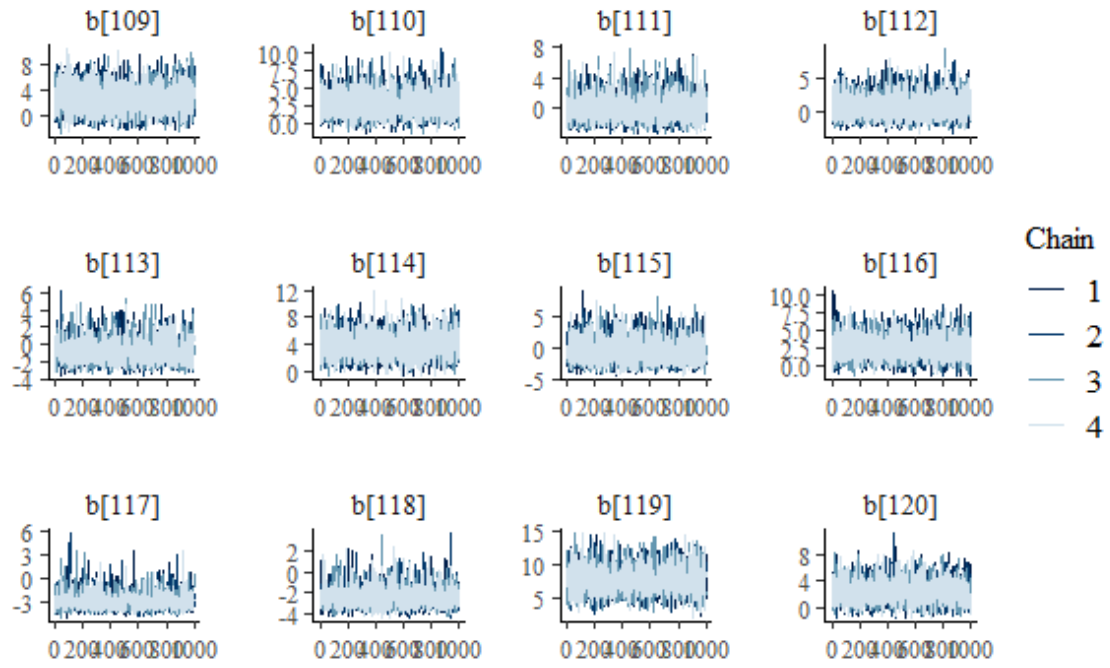


Figure B.10: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

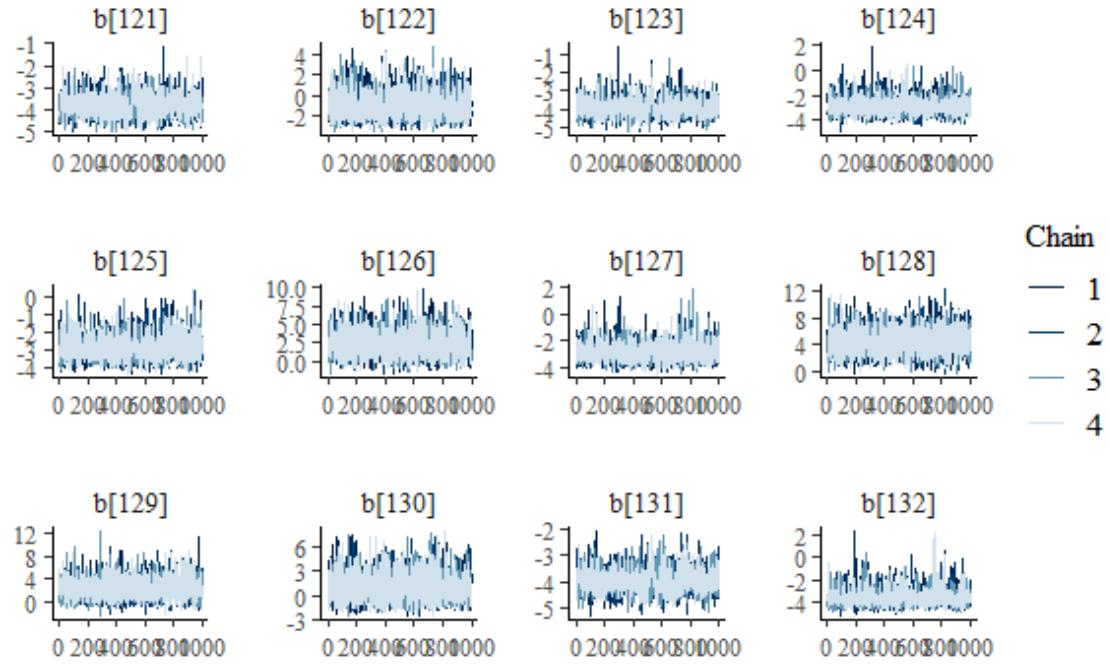


Figure B.11: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

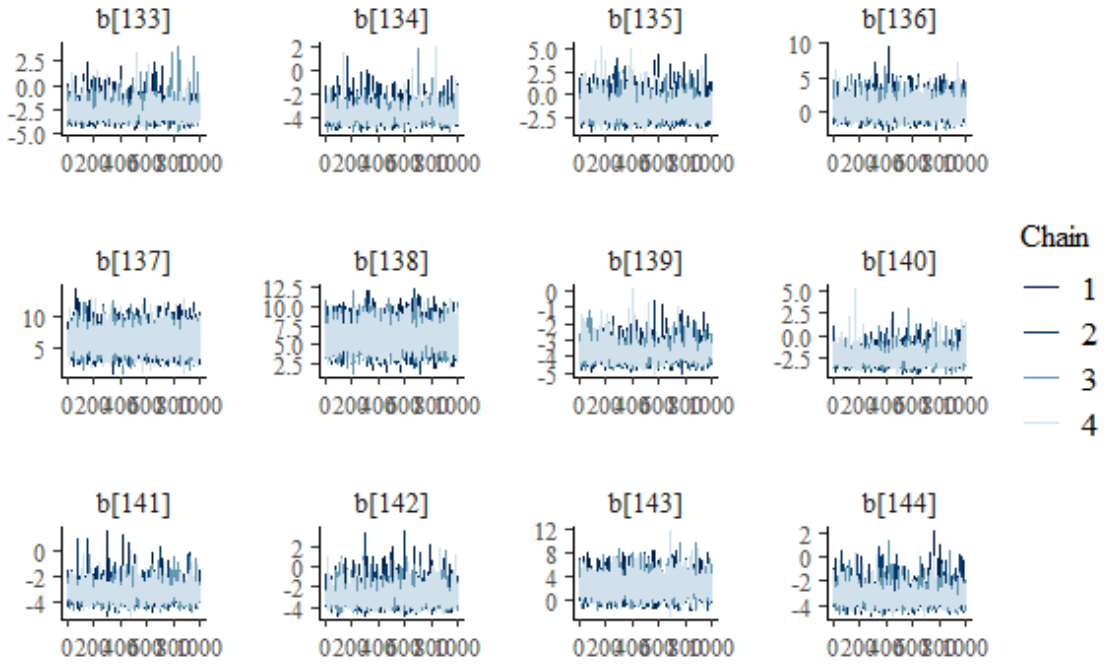


Figure B.12: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

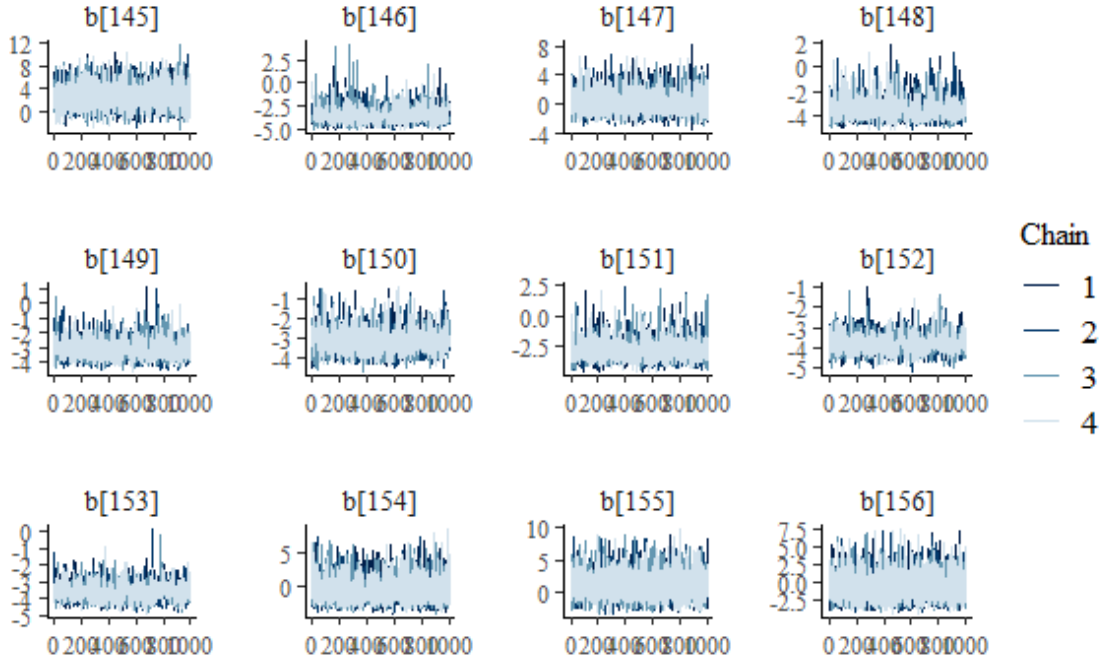


Figure B.13: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

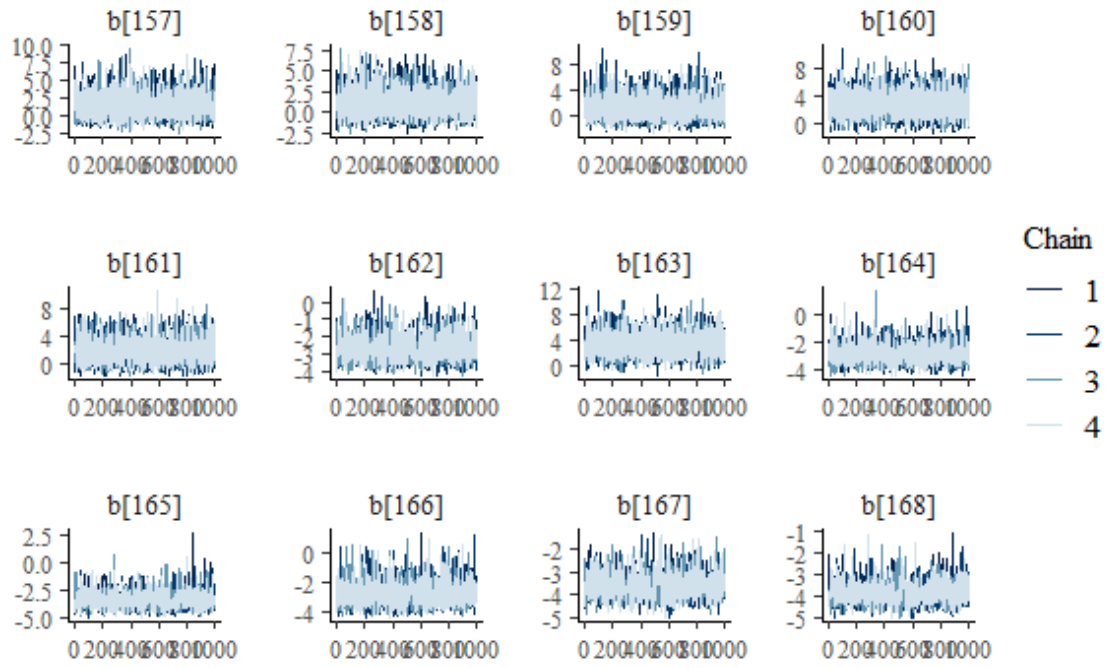


Figure B.14: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

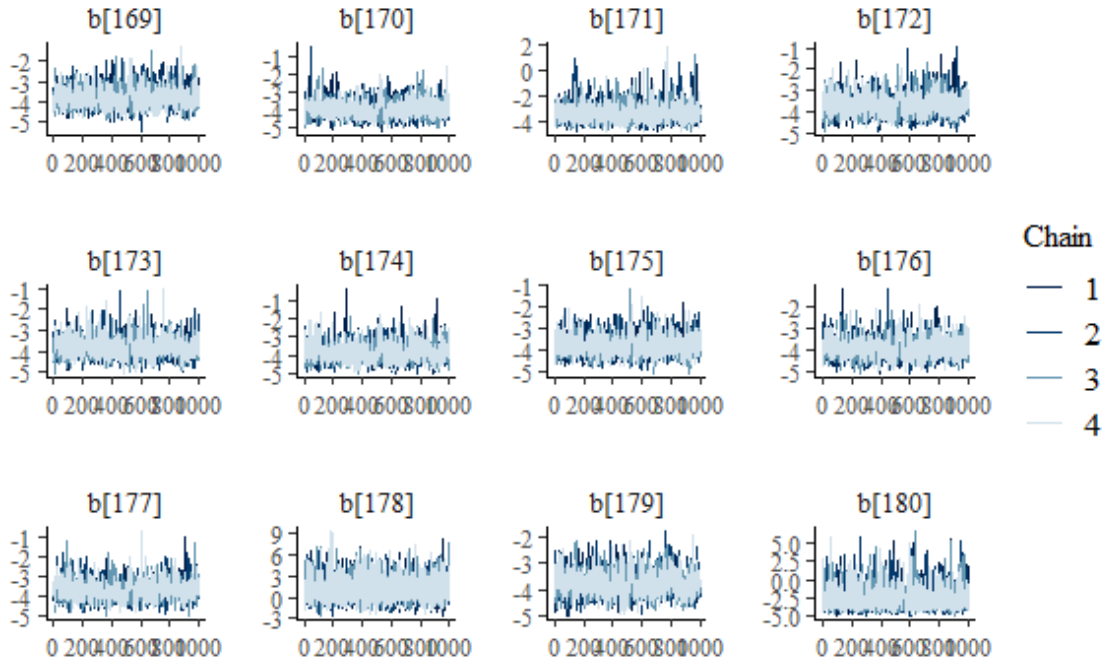


Figure B.15: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

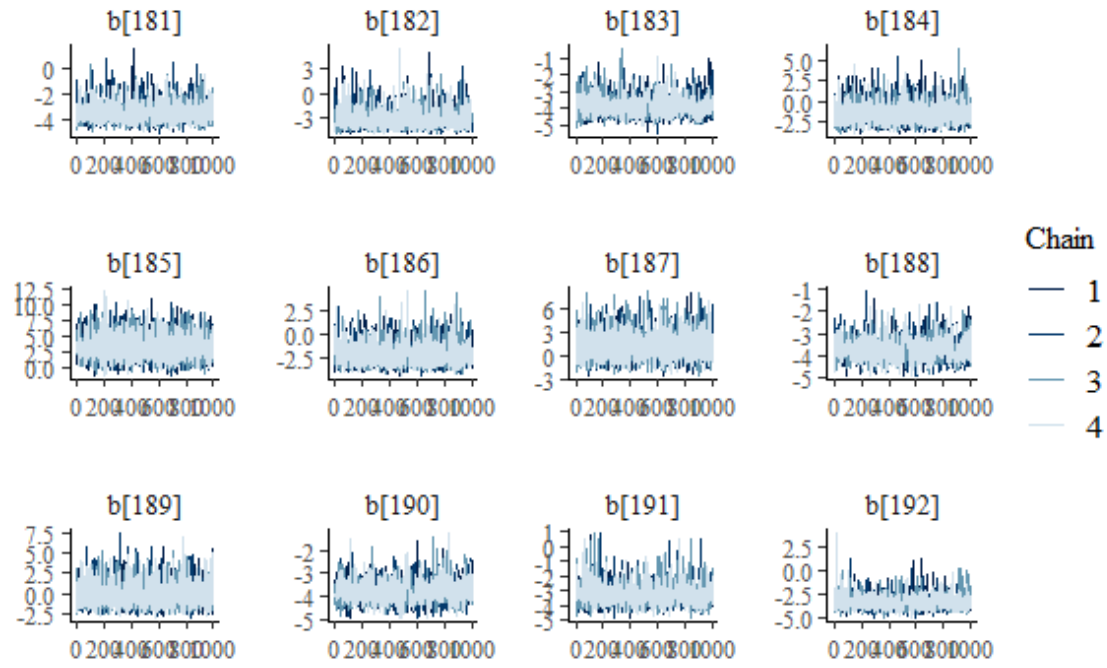


Figure B.16: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

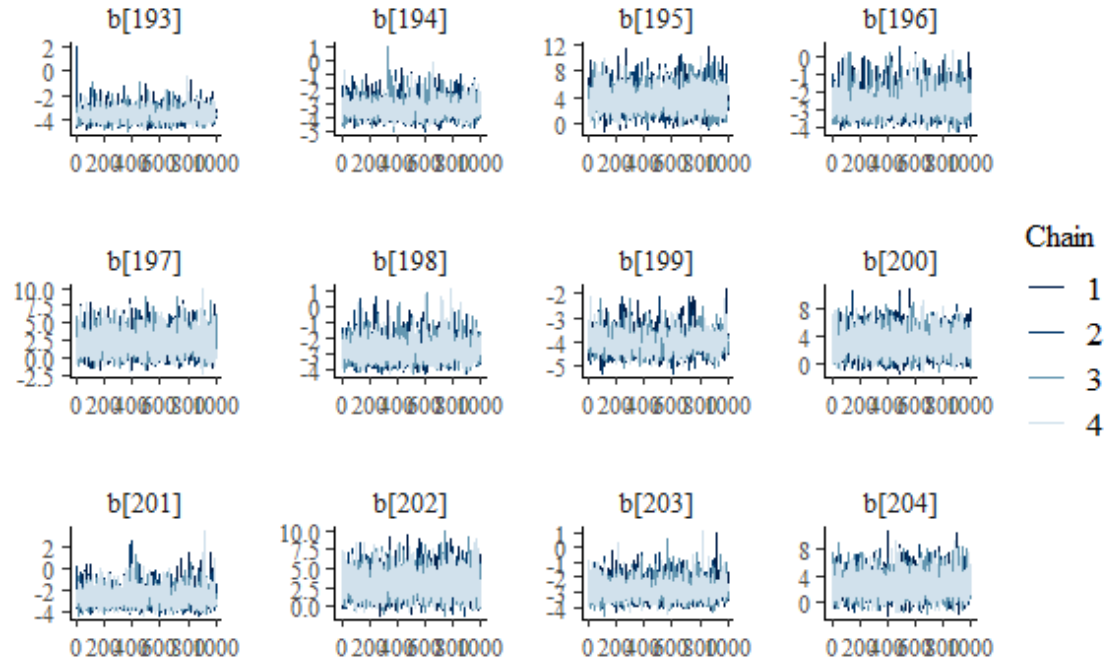


Figure B.17: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

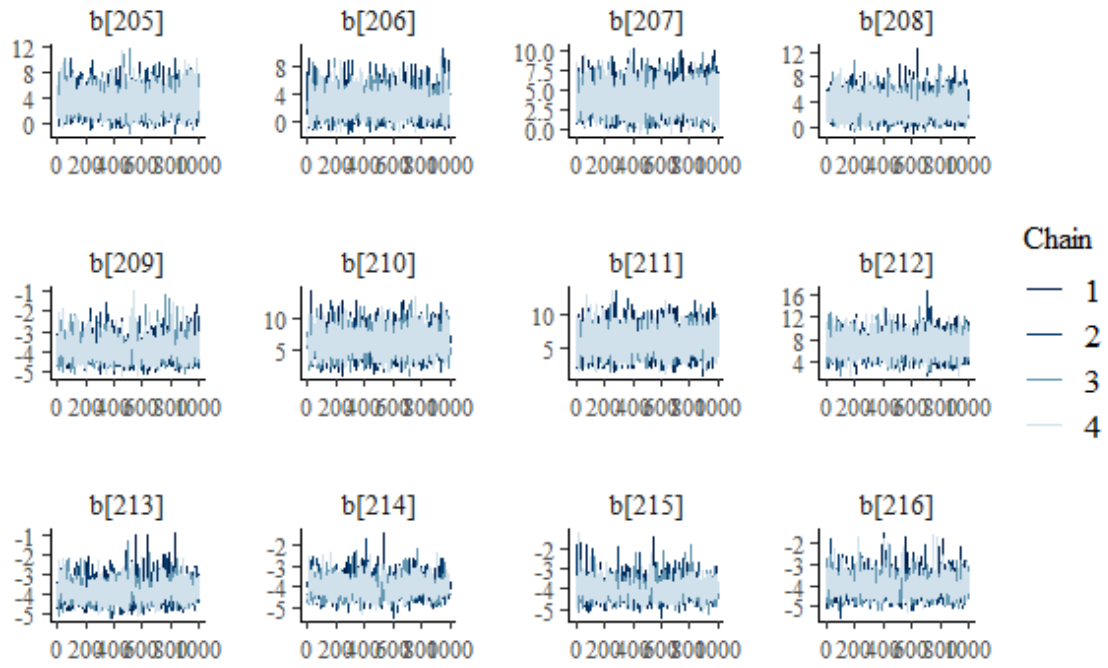


Figure B.18: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

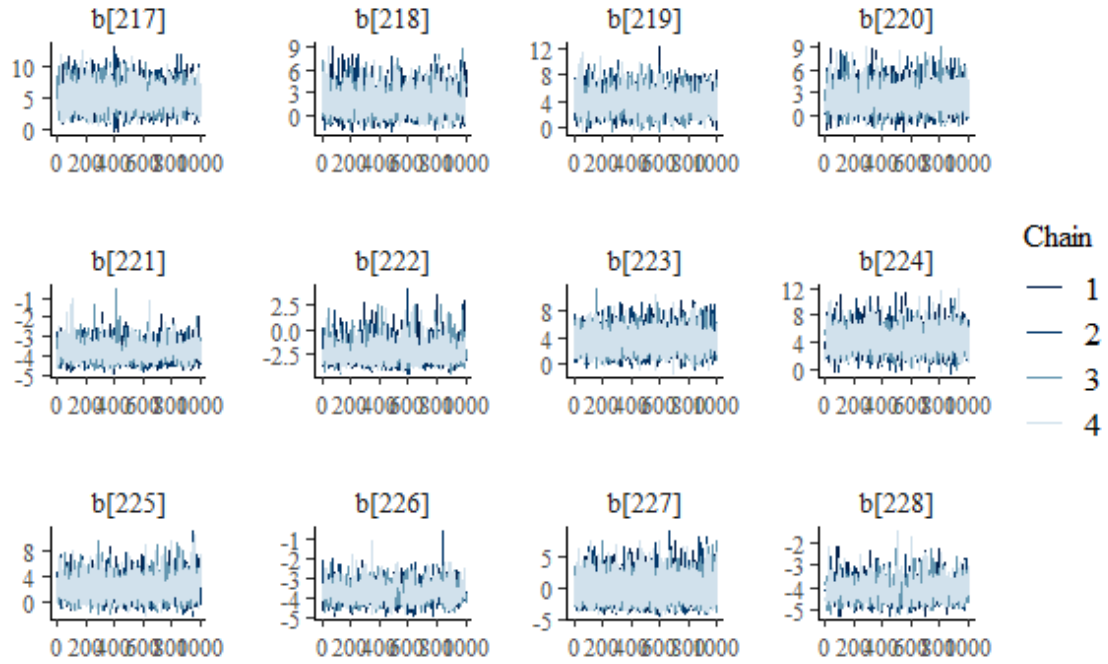


Figure B.19: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

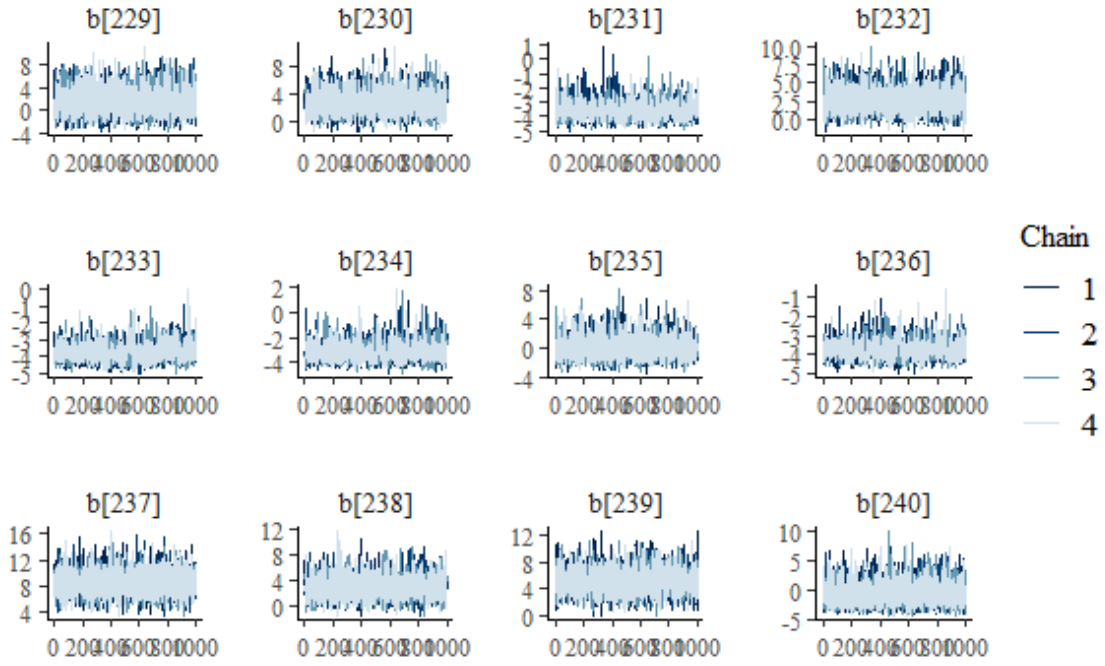


Figure B.20: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

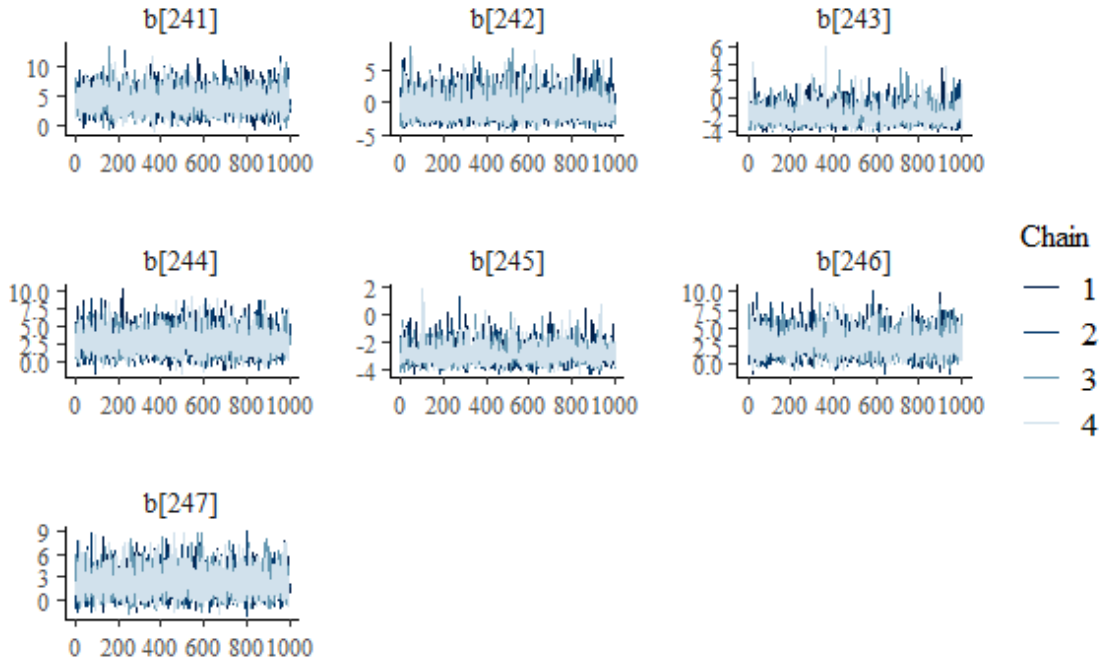


Figure B.21: Trace plot showing the chain convergence for the difficulty parameter for the benchmarks

C

Appendix 3: Discrimination parameter trace plots

The discrimination parameter trace plots for each if the benchmark functions are presented here. The number represents each benchmark function and their specific names can be found in Appendix A, in section A.1.

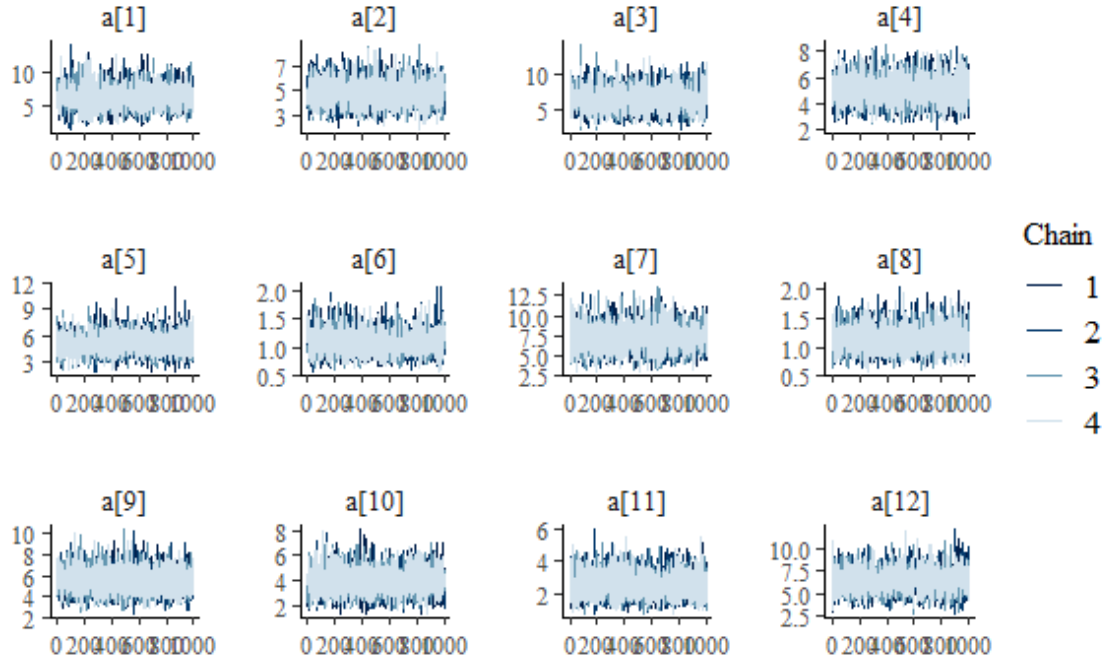


Figure C.1: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

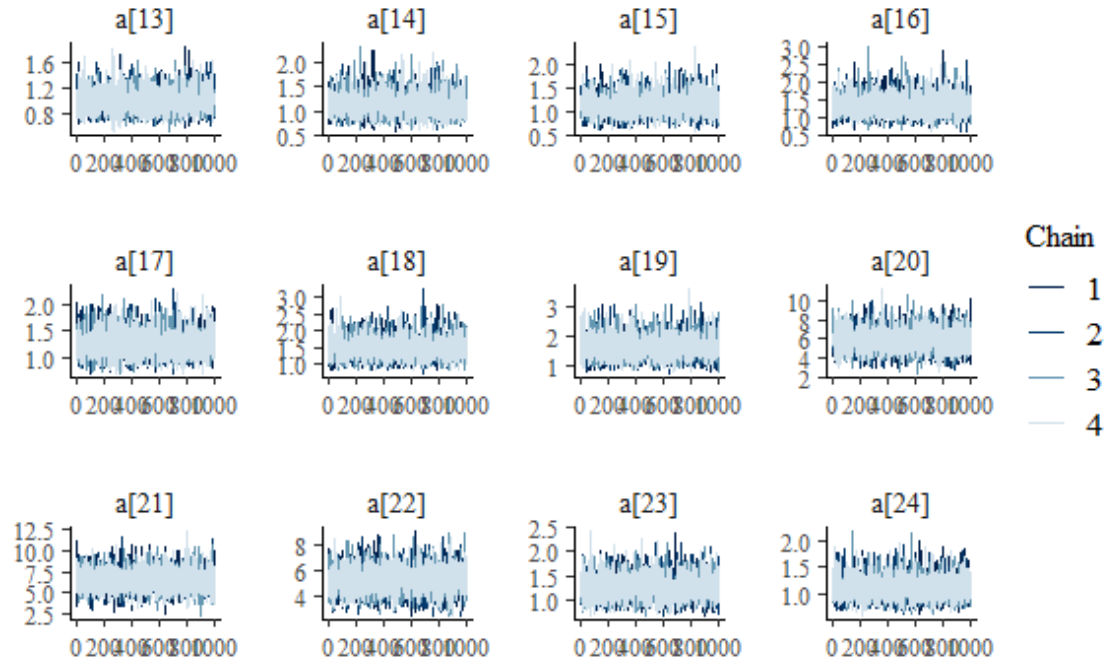


Figure C.2: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

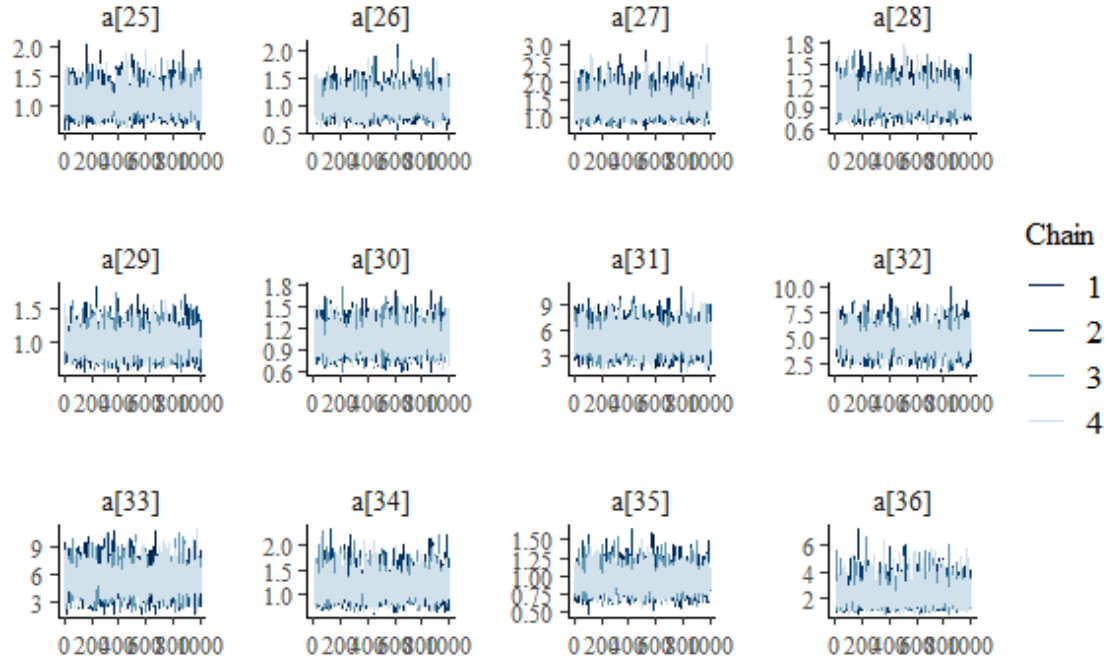


Figure C.3: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

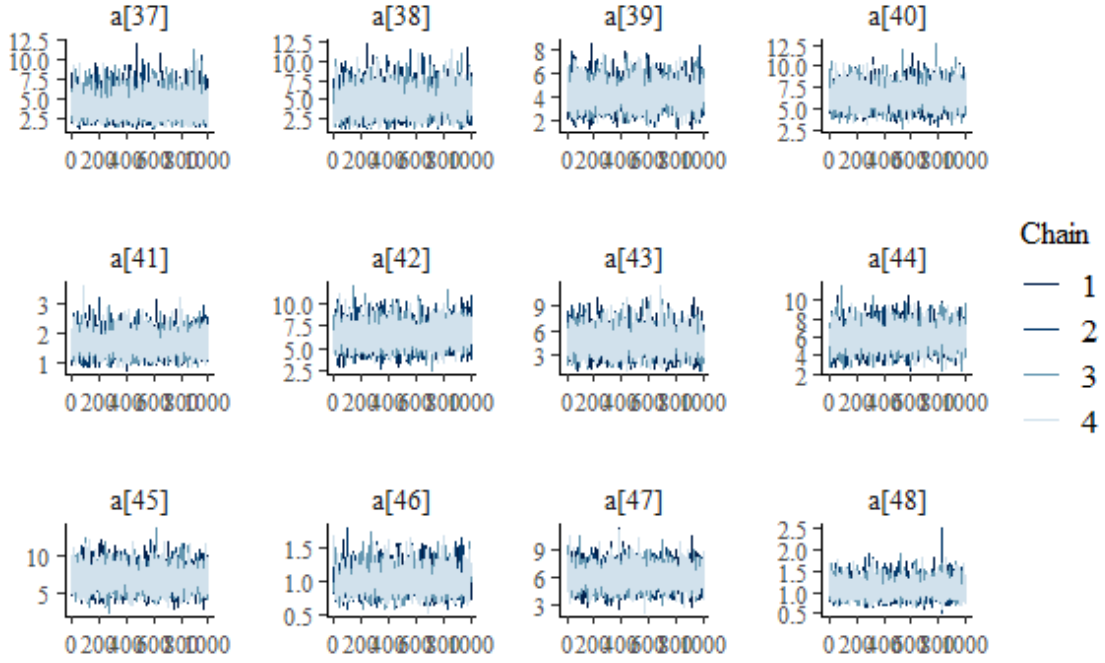


Figure C.4: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

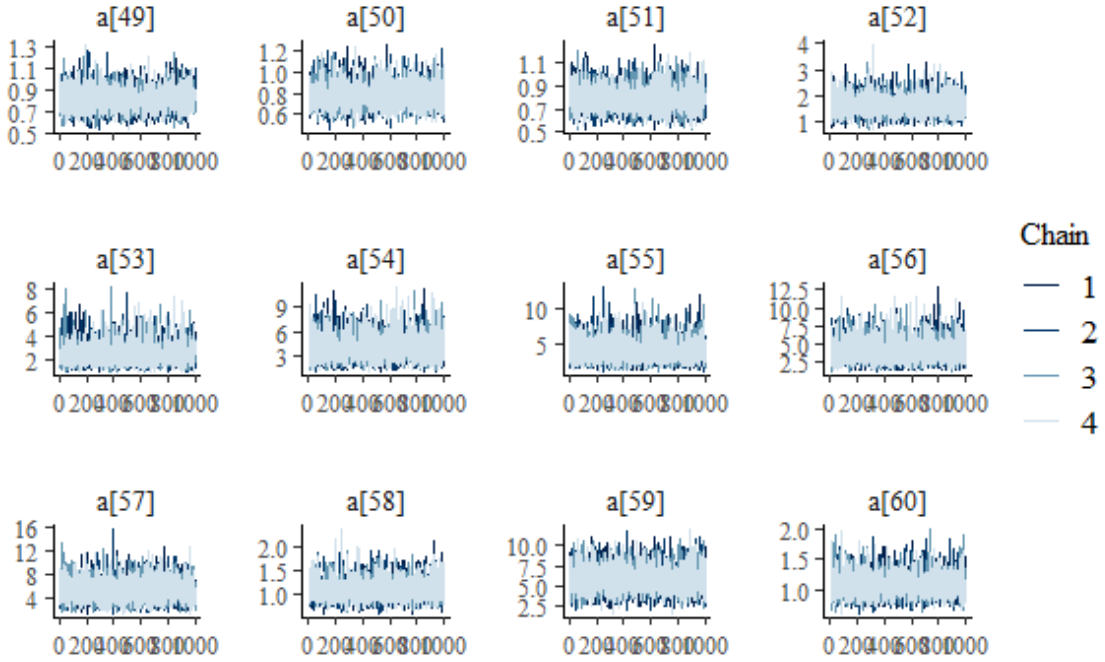


Figure C.5: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

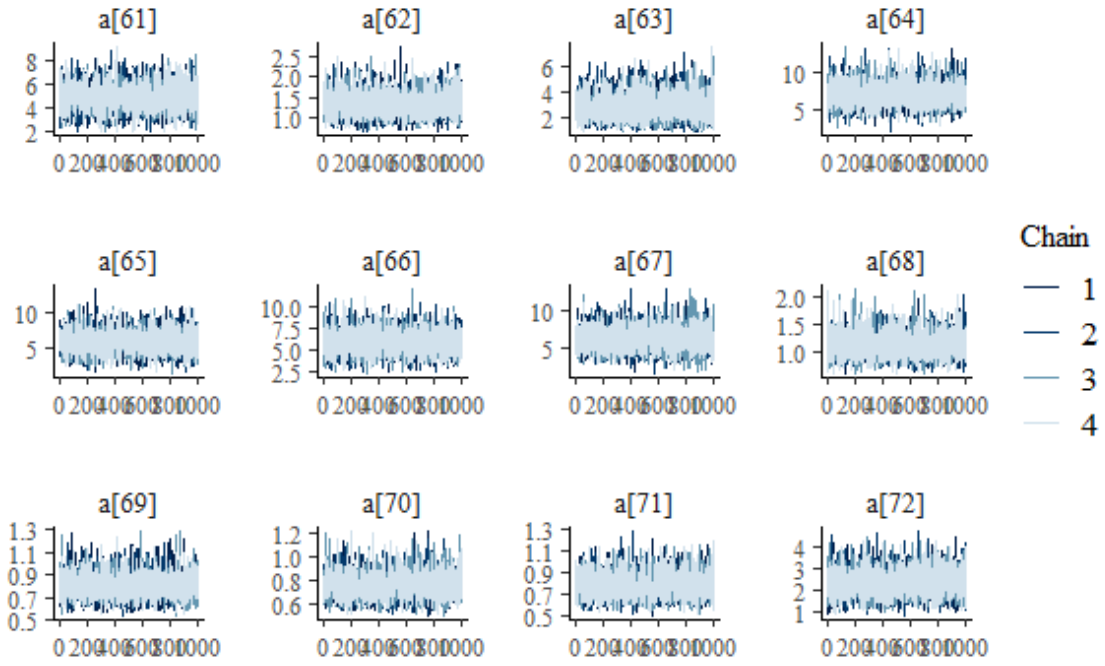


Figure C.6: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

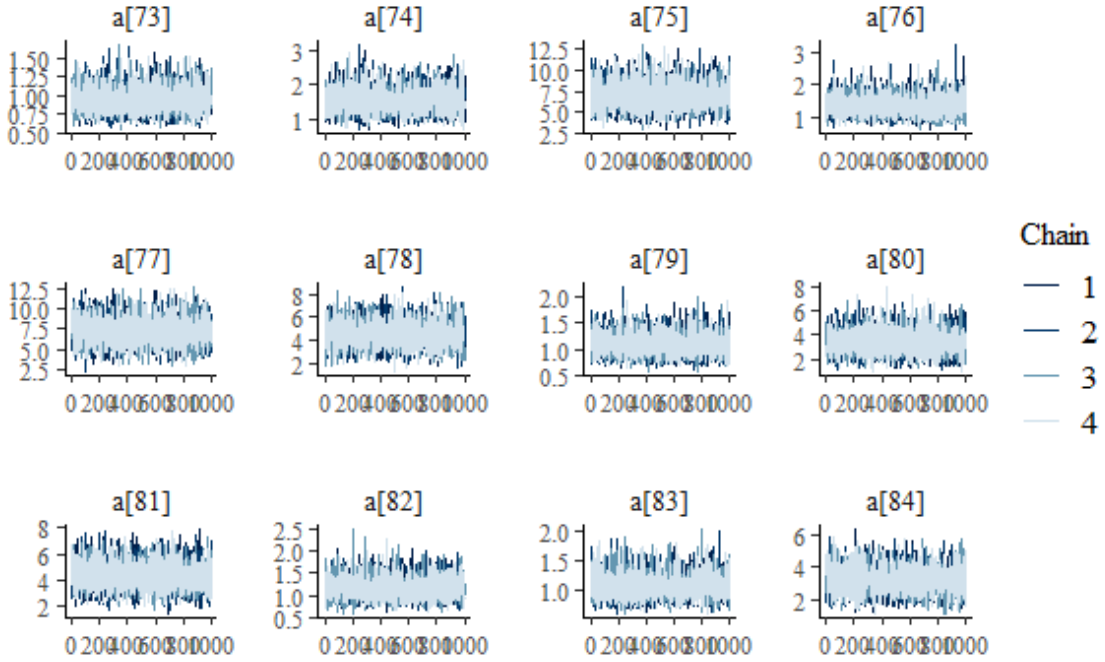


Figure C.7: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

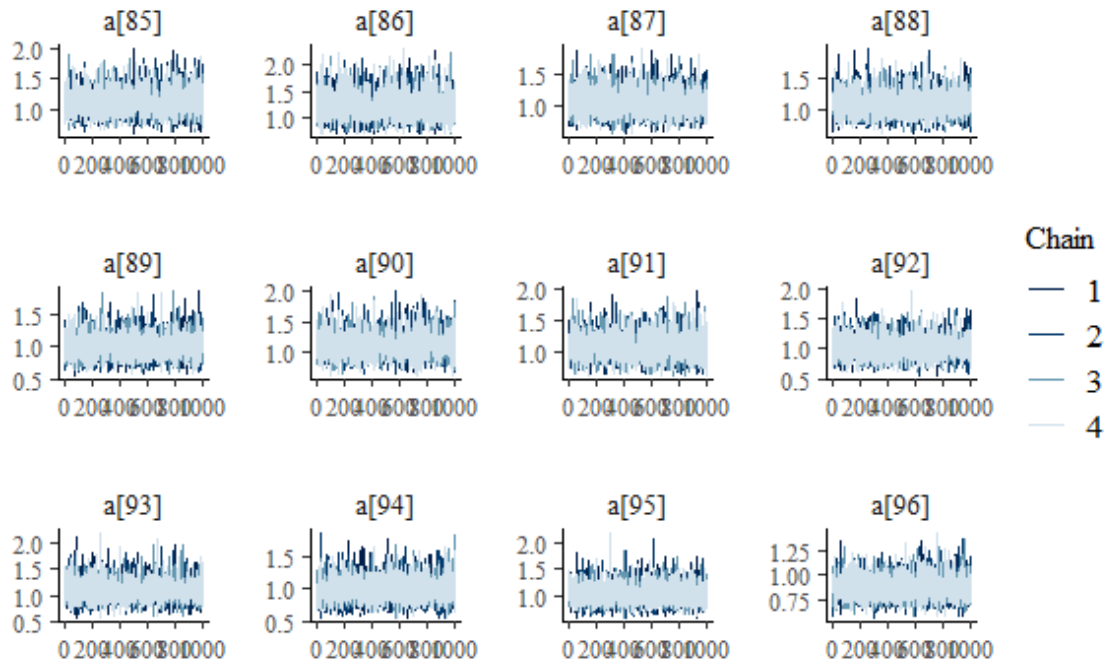


Figure C.8: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

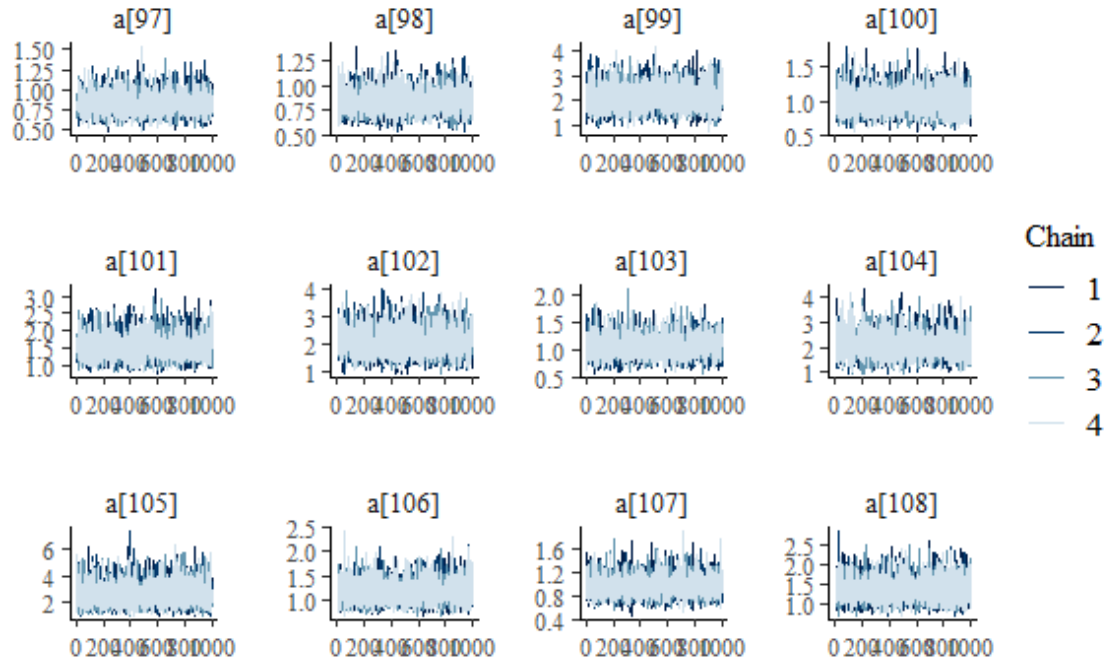


Figure C.9: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

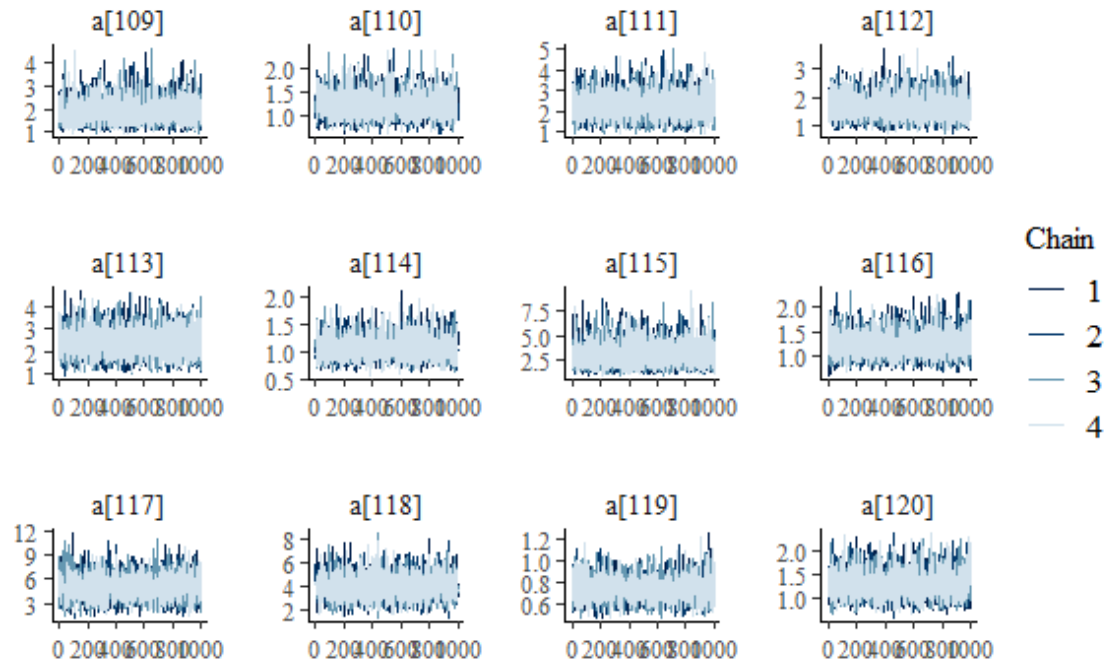


Figure C.10: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

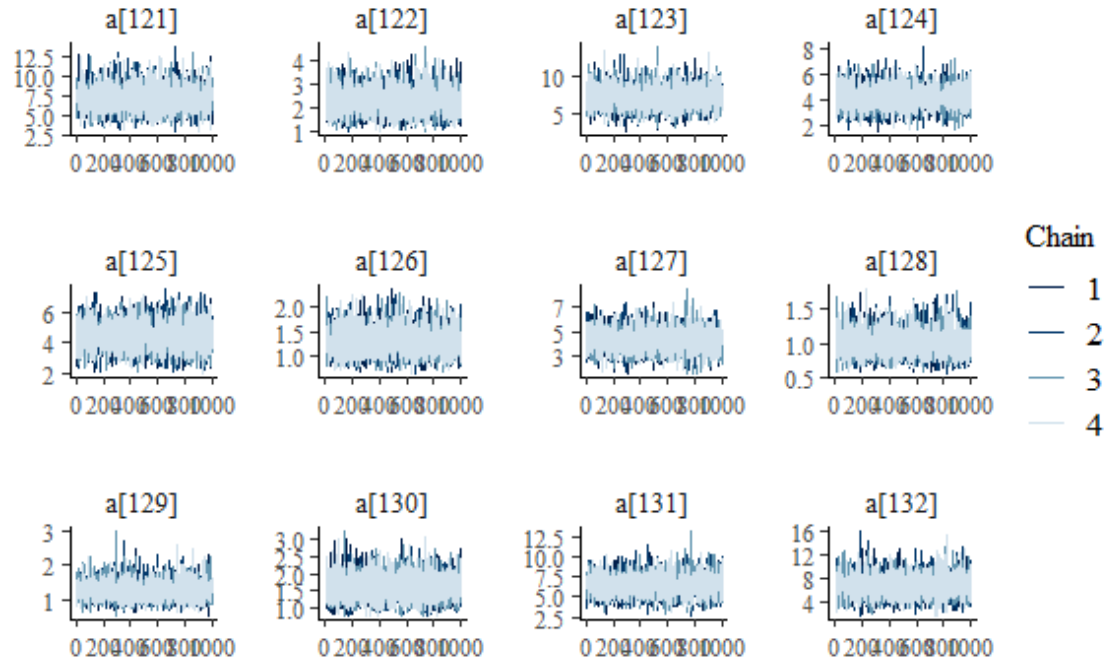


Figure C.11: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

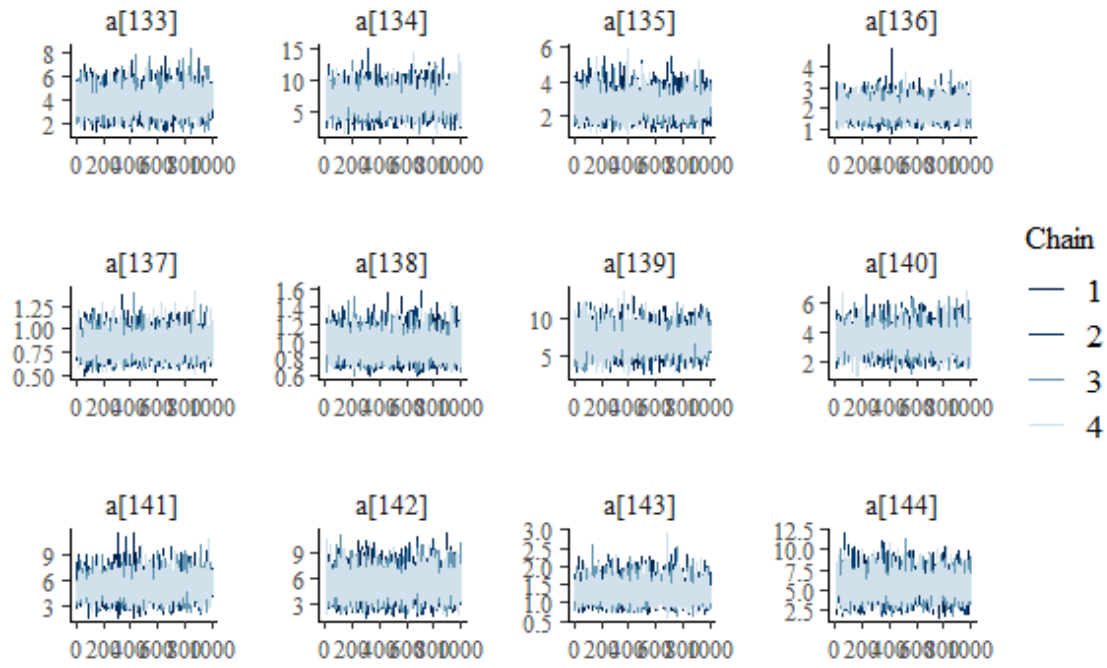


Figure C.12: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

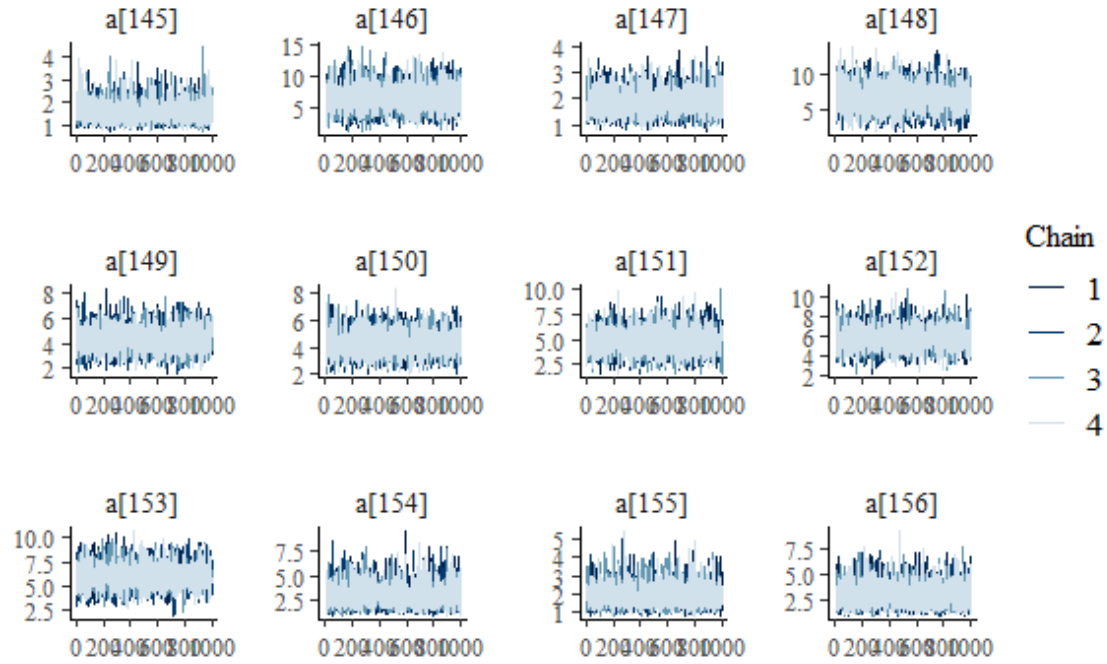


Figure C.13: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

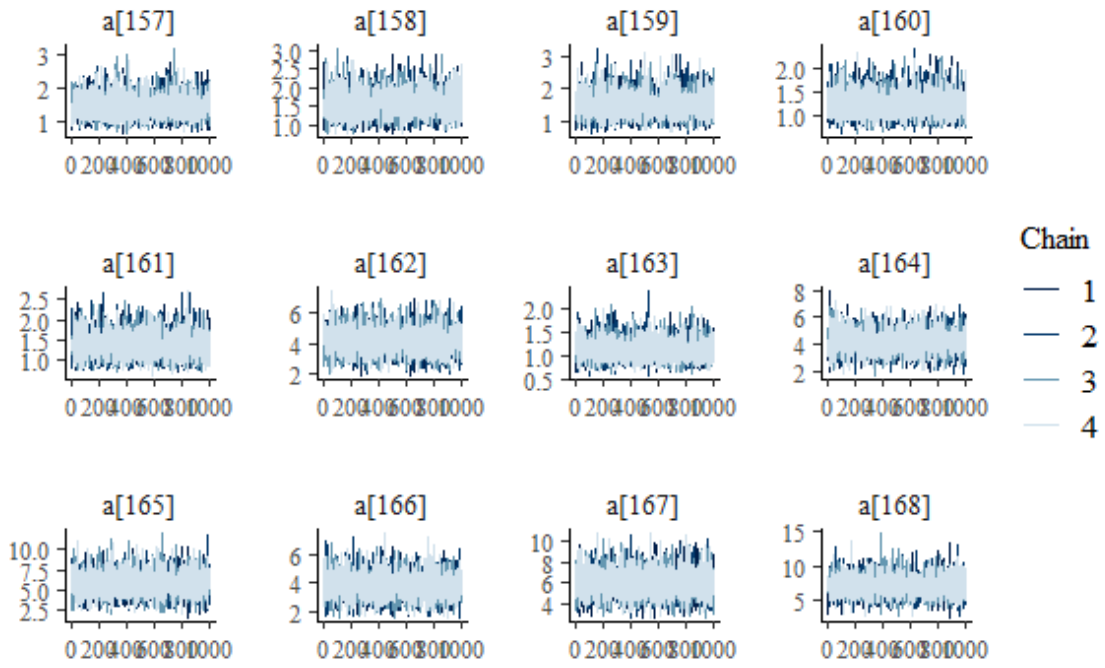


Figure C.14: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

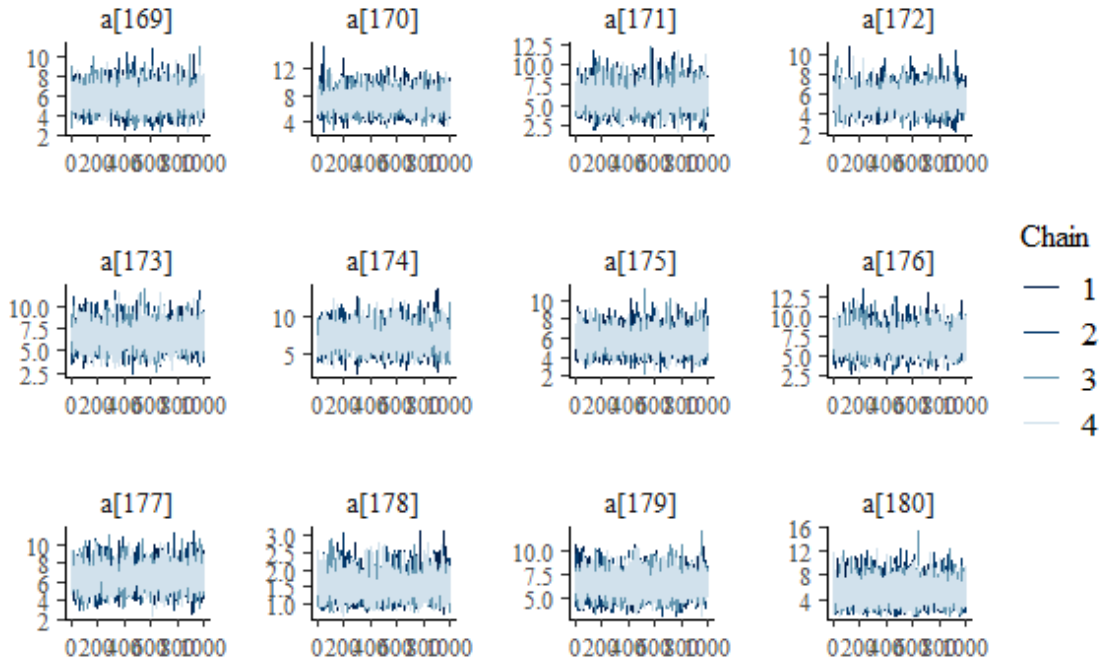


Figure C.15: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

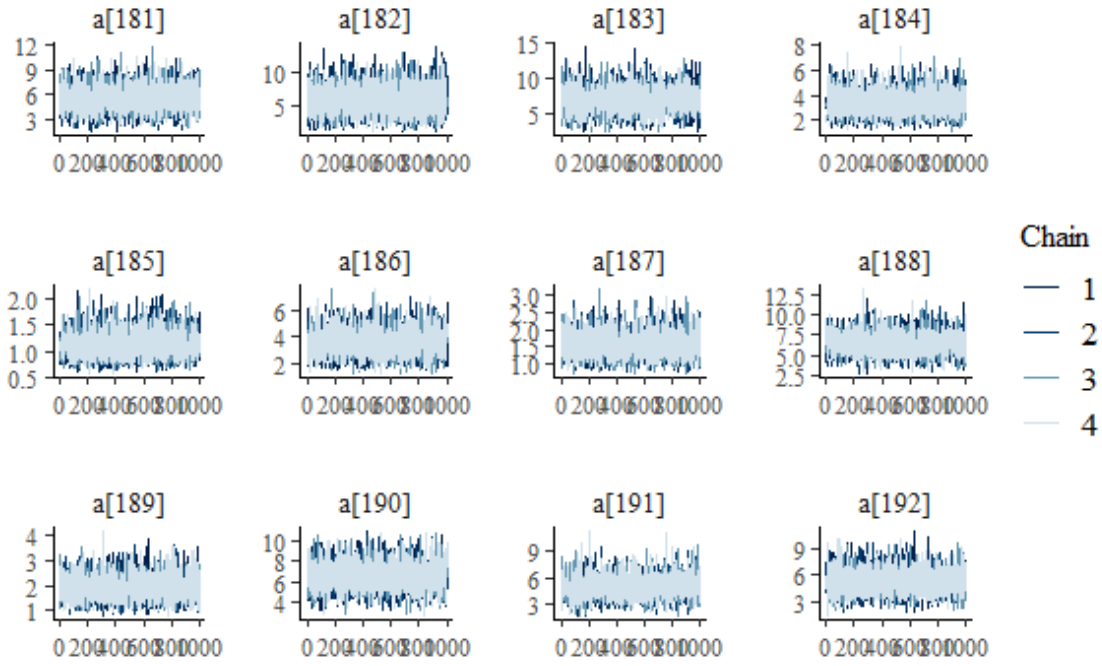


Figure C.16: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

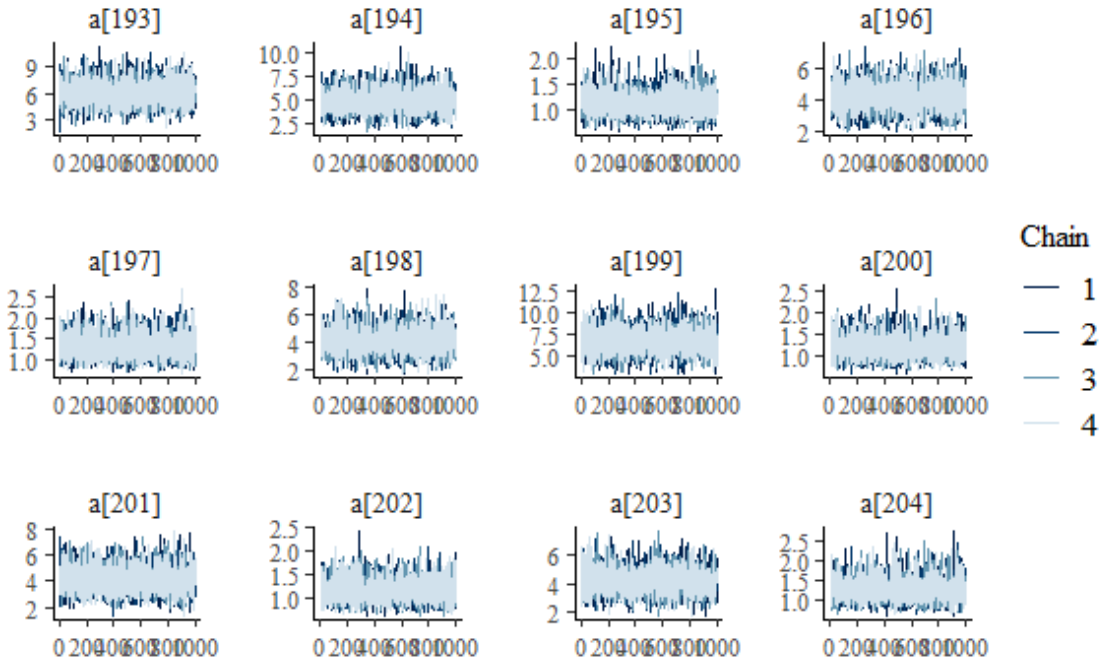


Figure C.17: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

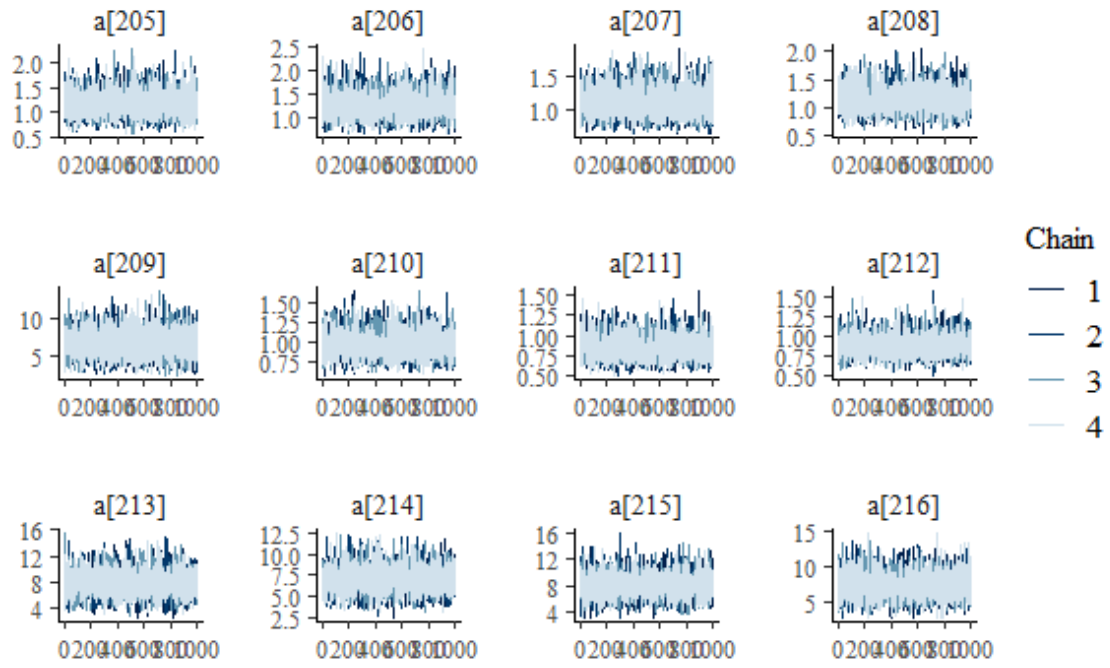


Figure C.18: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

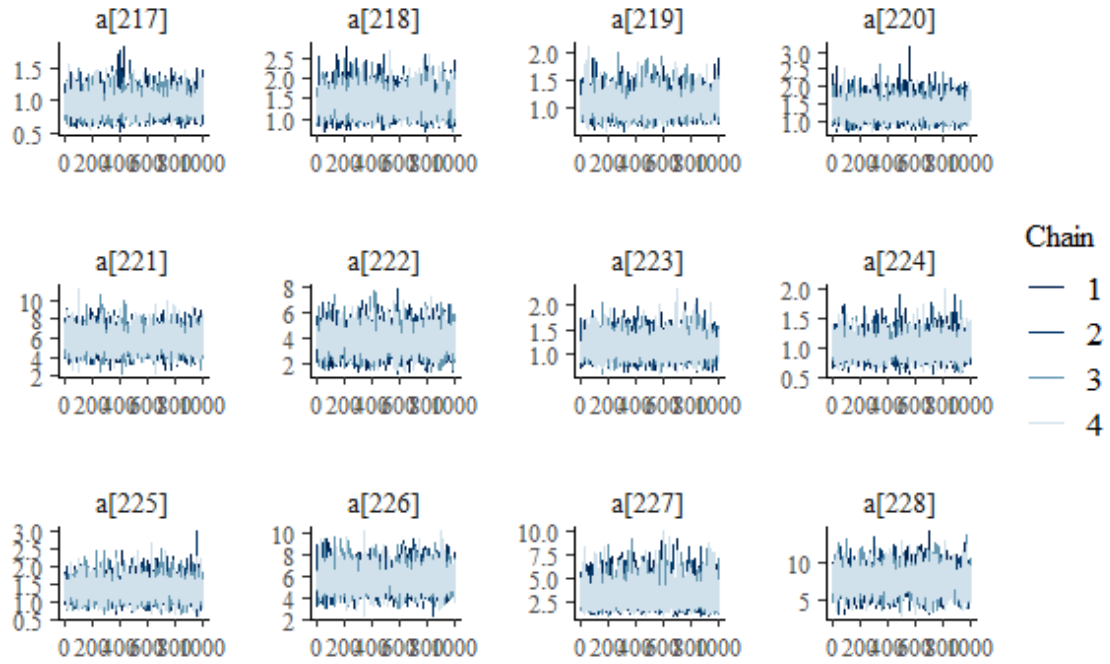


Figure C.19: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

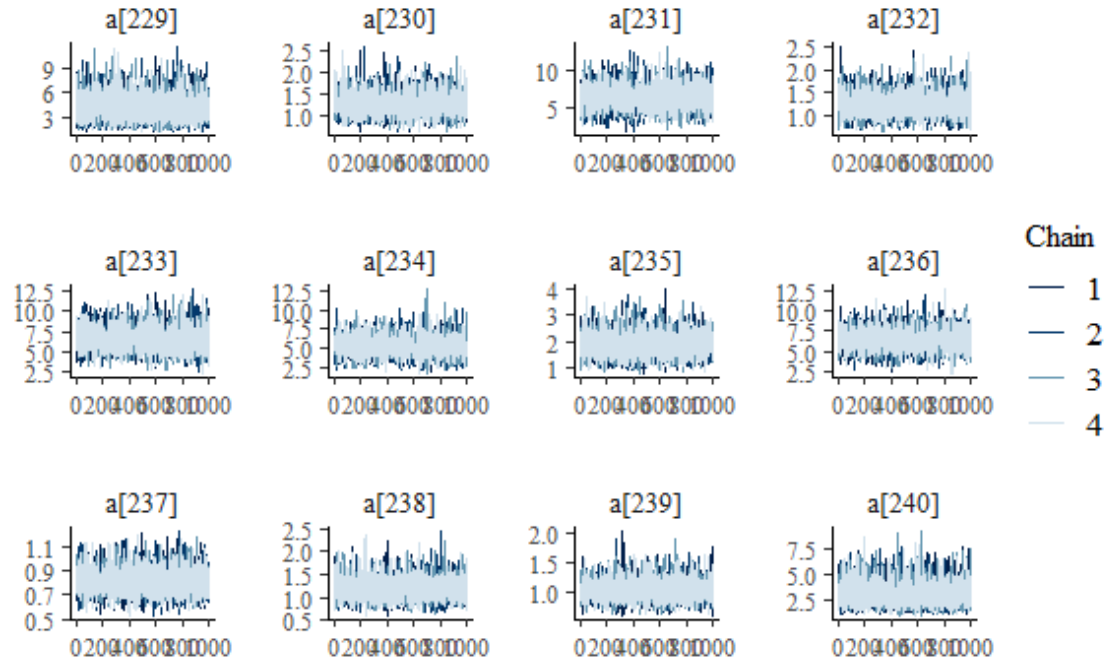


Figure C.20: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

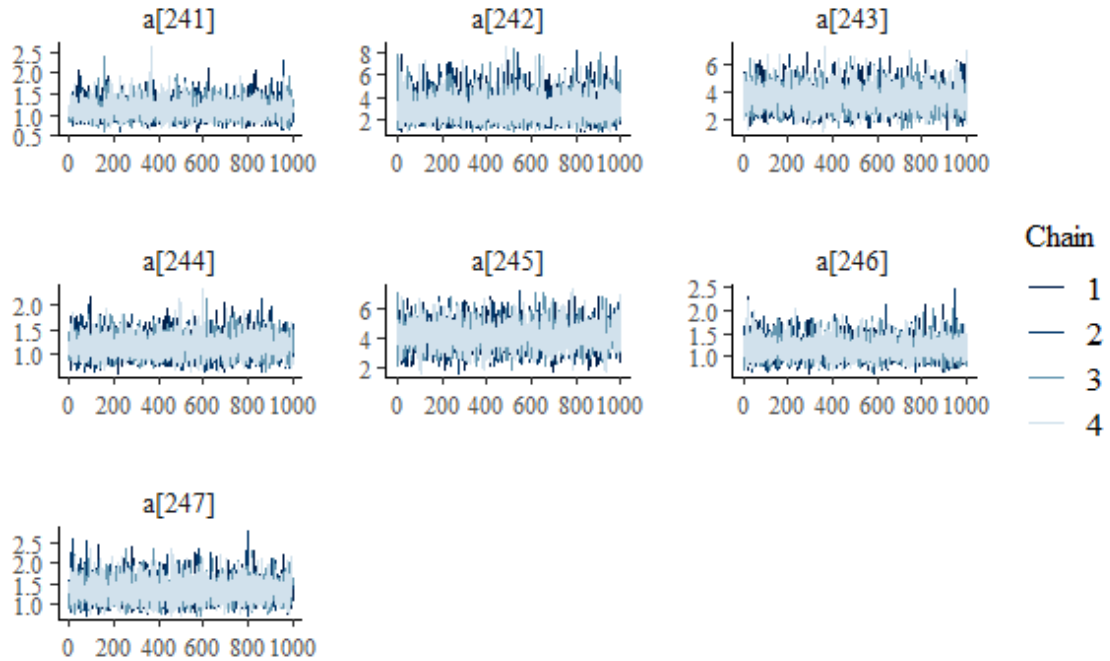


Figure C.21: Trace plot showing the chain convergence for the discrimination parameter for the benchmarks

D

Appendix 4: Item information curves

The item information curves for each of the benchmark functions are presented here. The number represents each benchmark function and their specific names can be found in Appendix A, in section A.1.

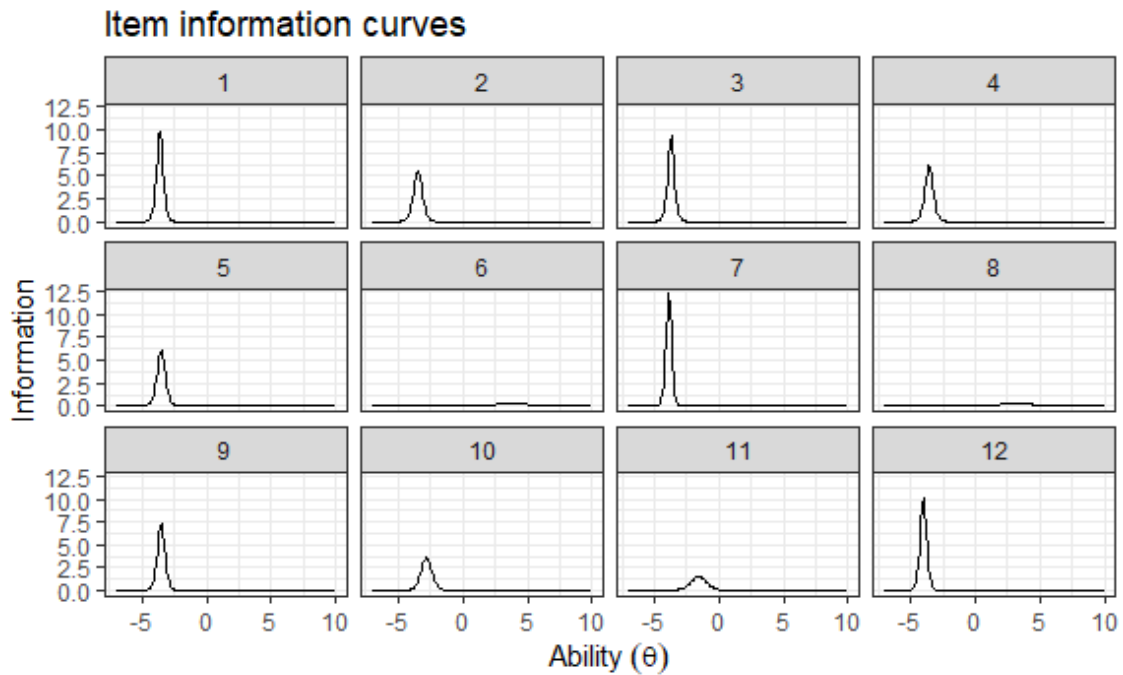


Figure D.1: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

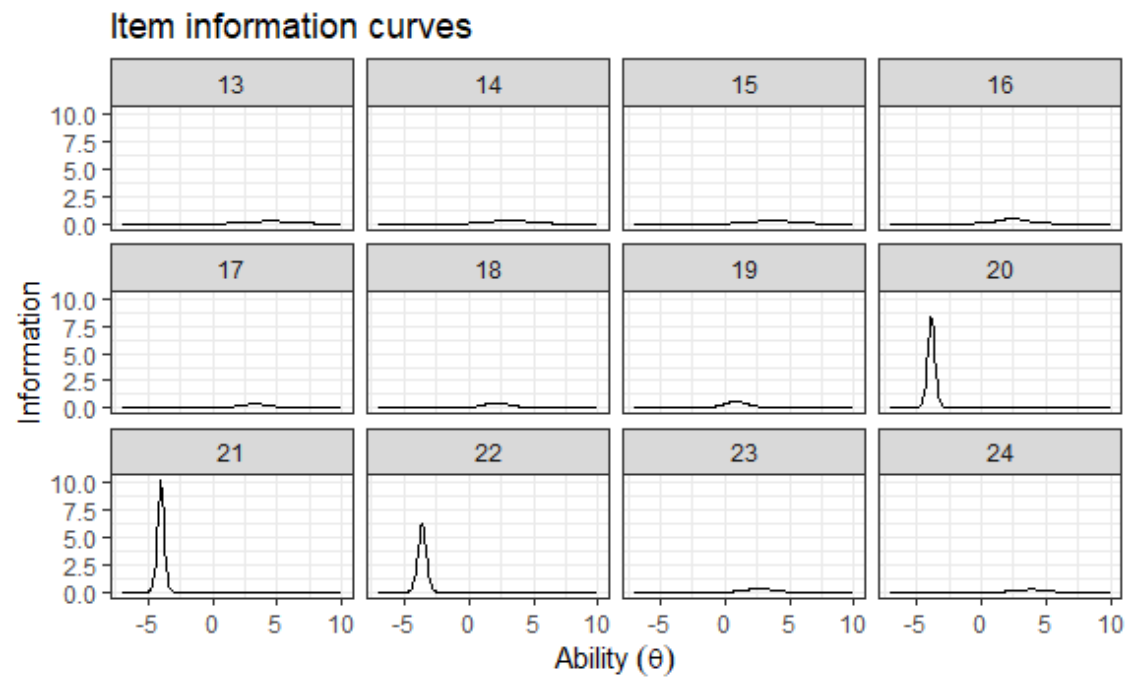


Figure D.2: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

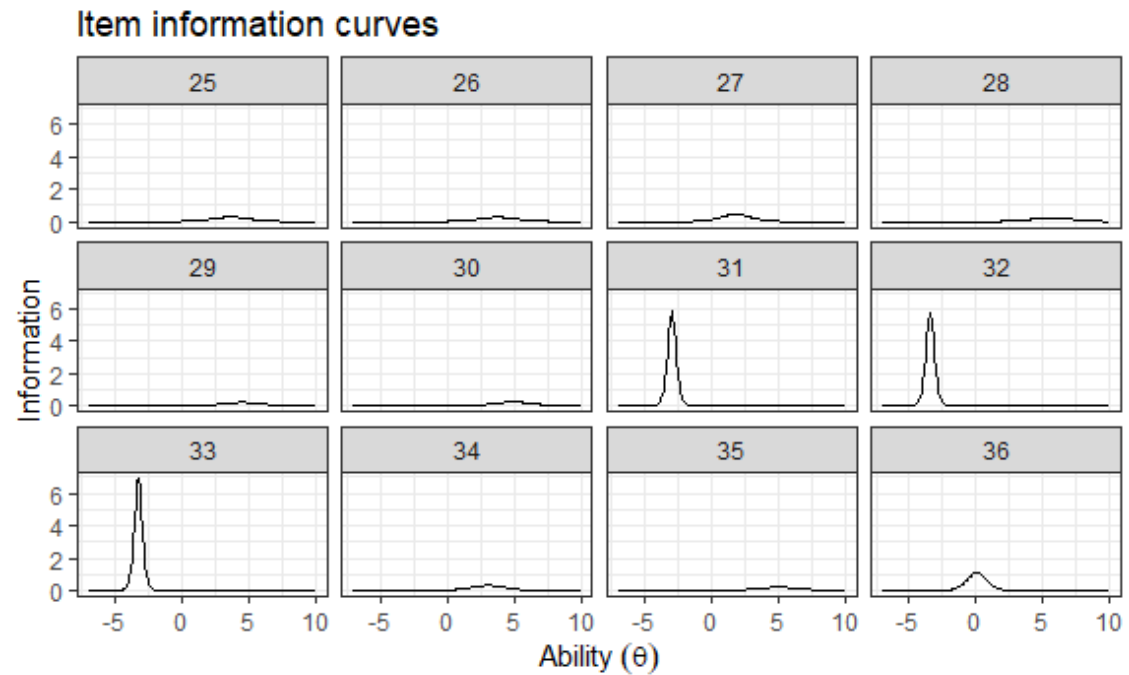


Figure D.3: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

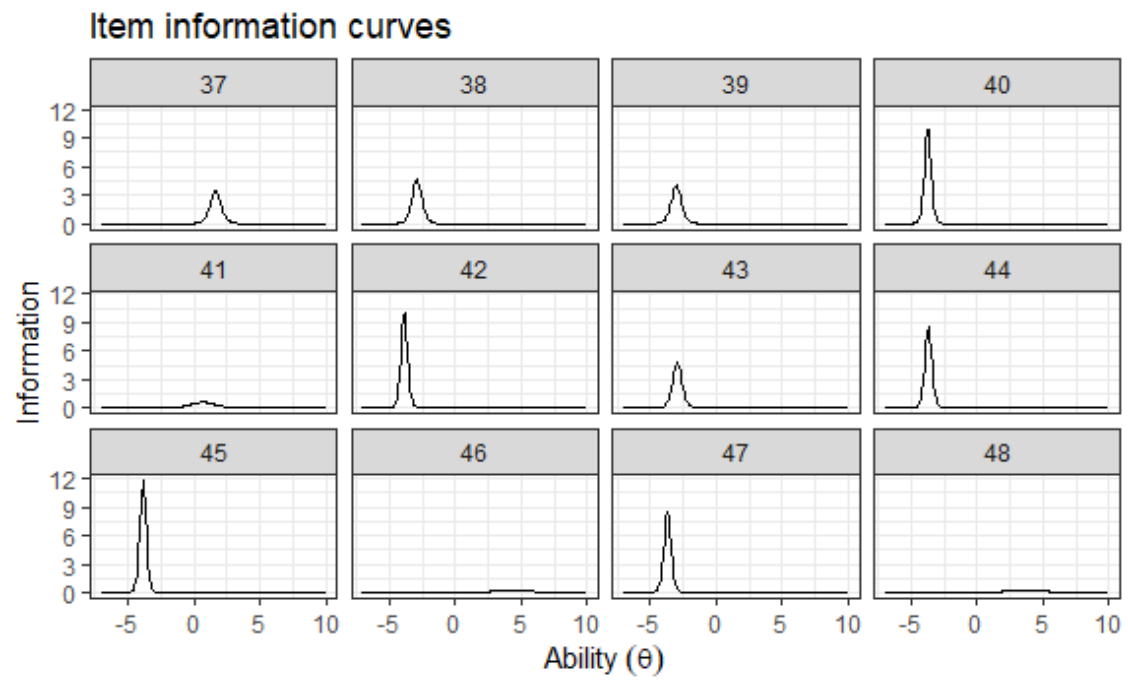


Figure D.4: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

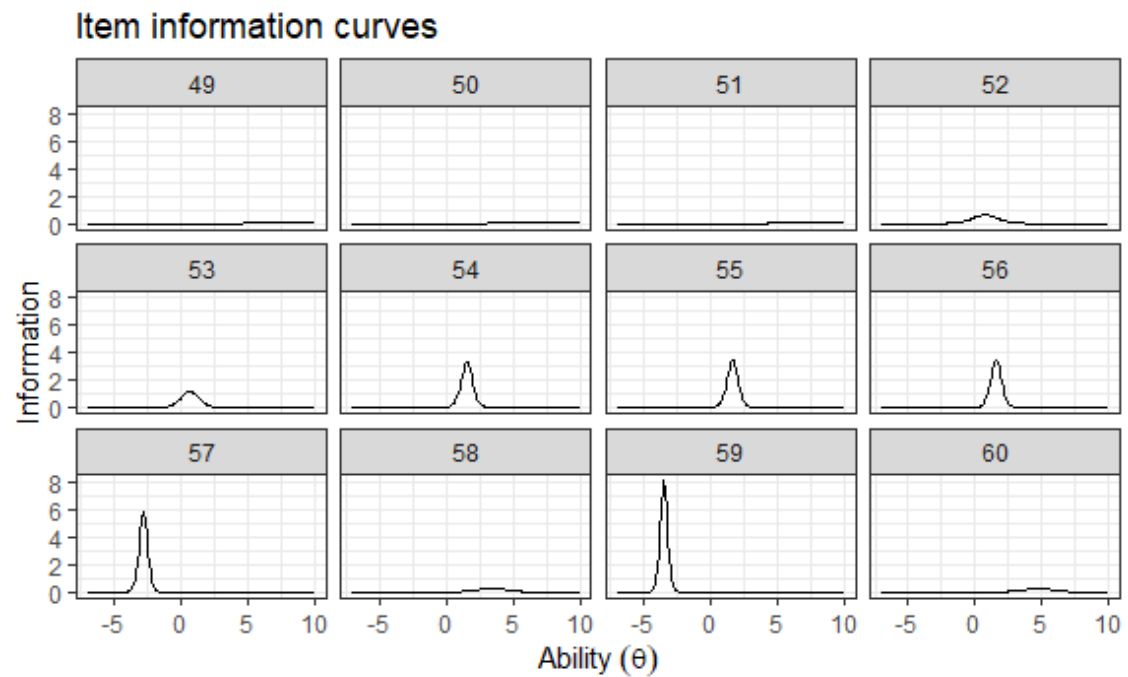


Figure D.5: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

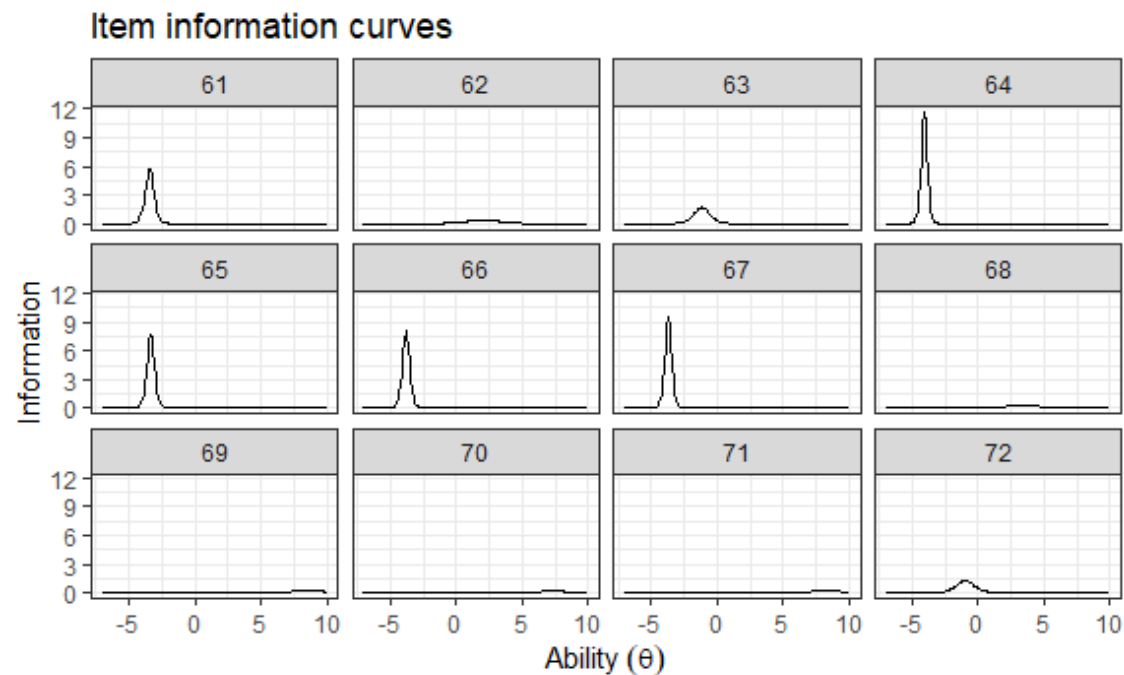


Figure D.6: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

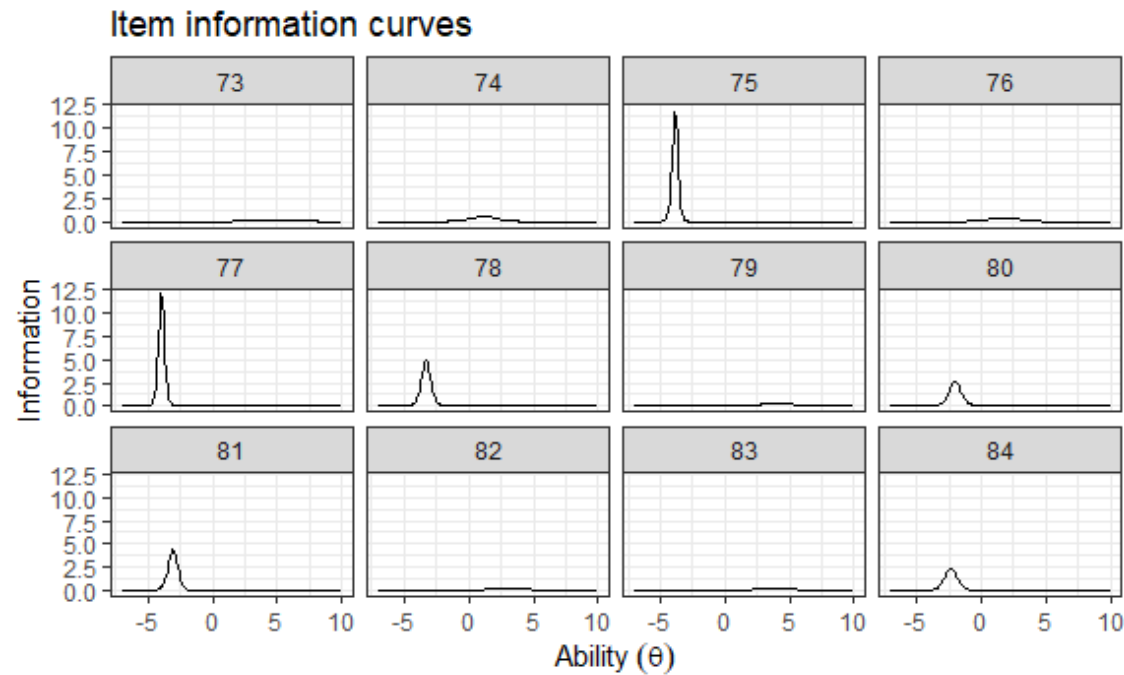


Figure D.7: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

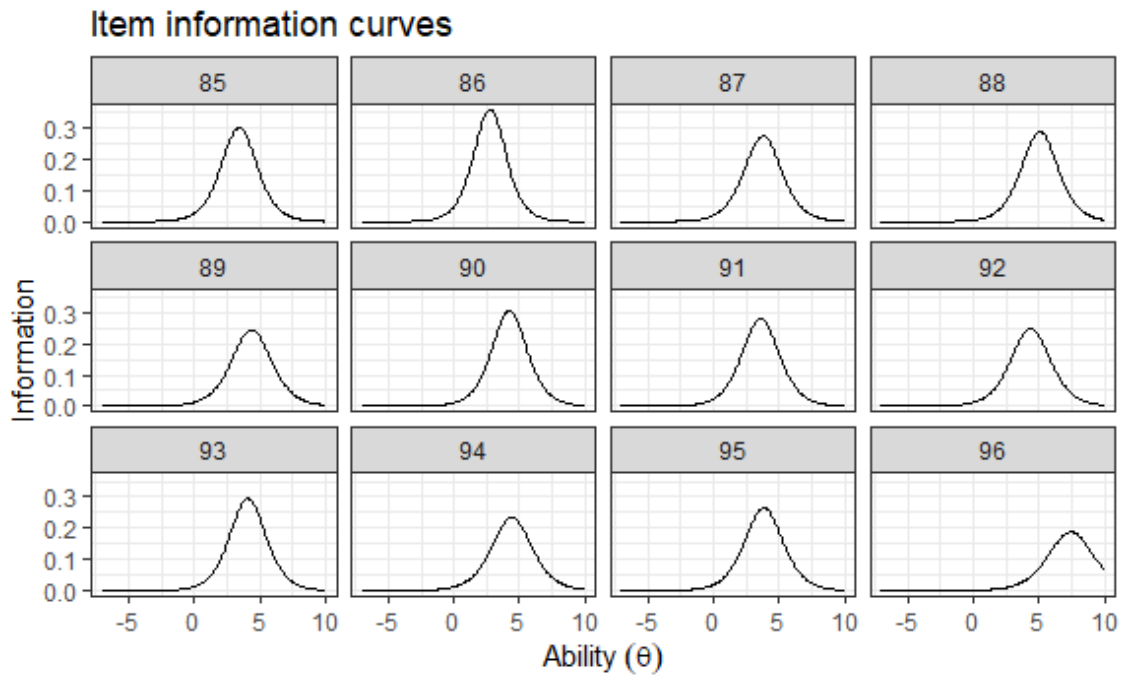


Figure D.8: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

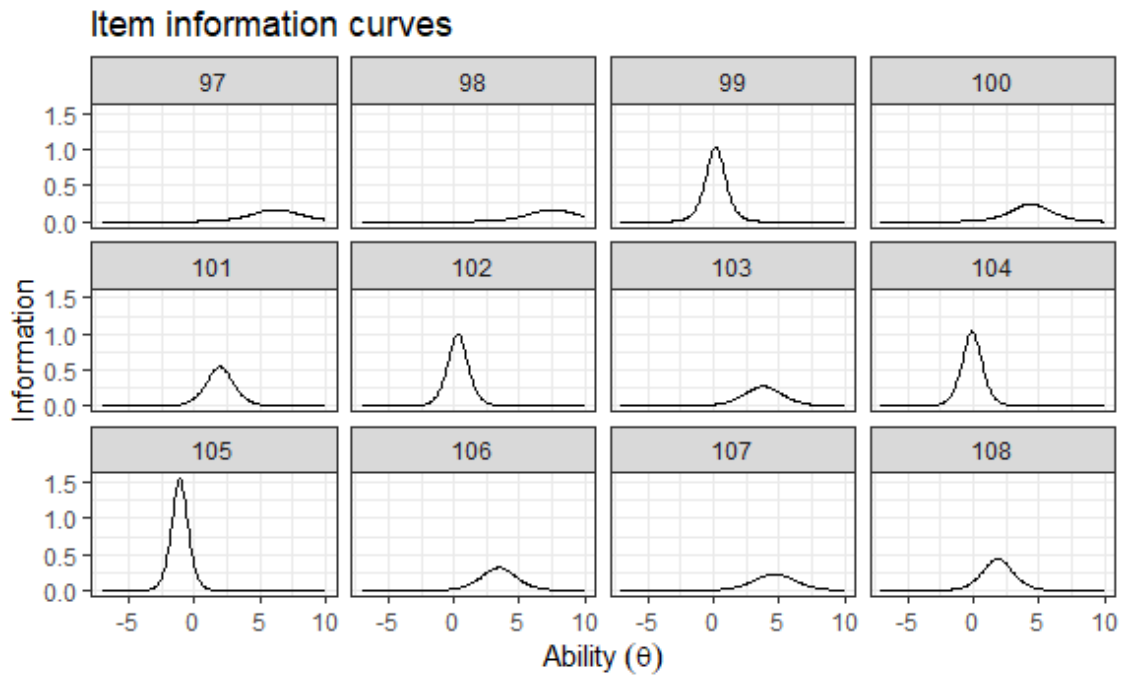


Figure D.9: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

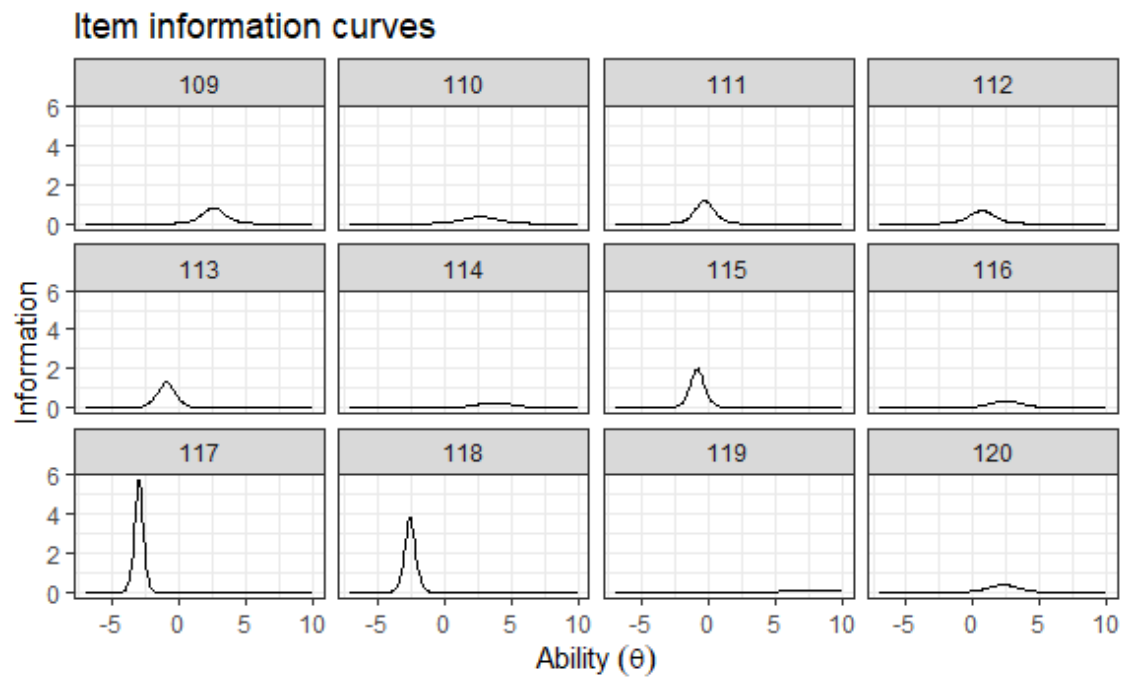


Figure D.10: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

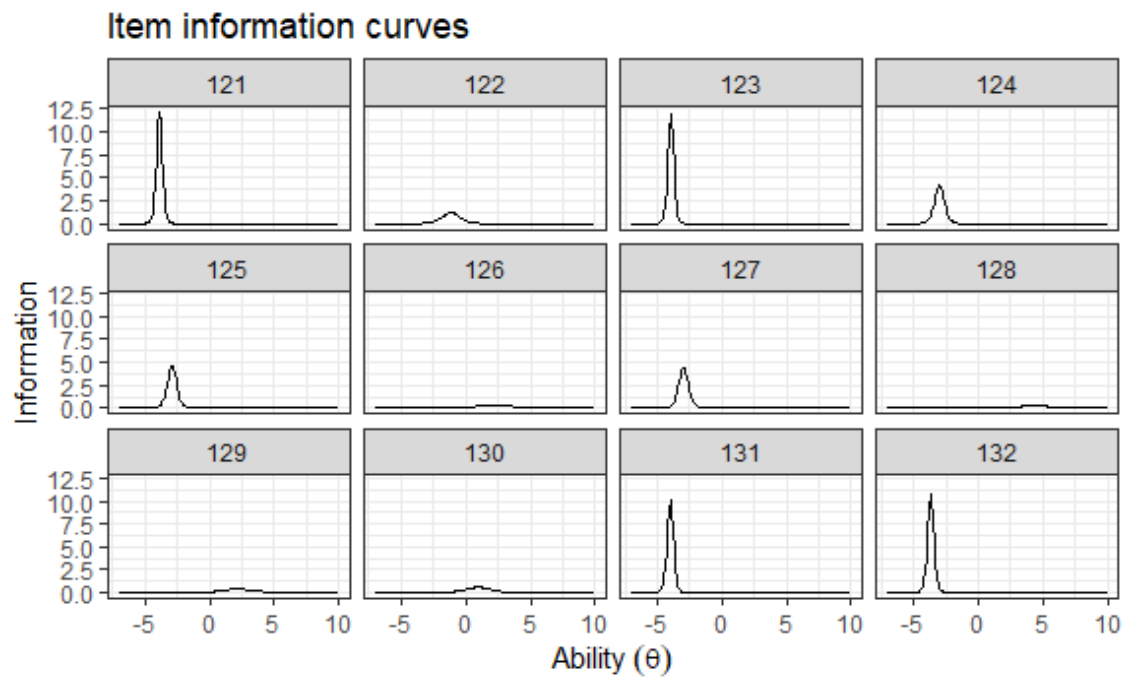


Figure D.11: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

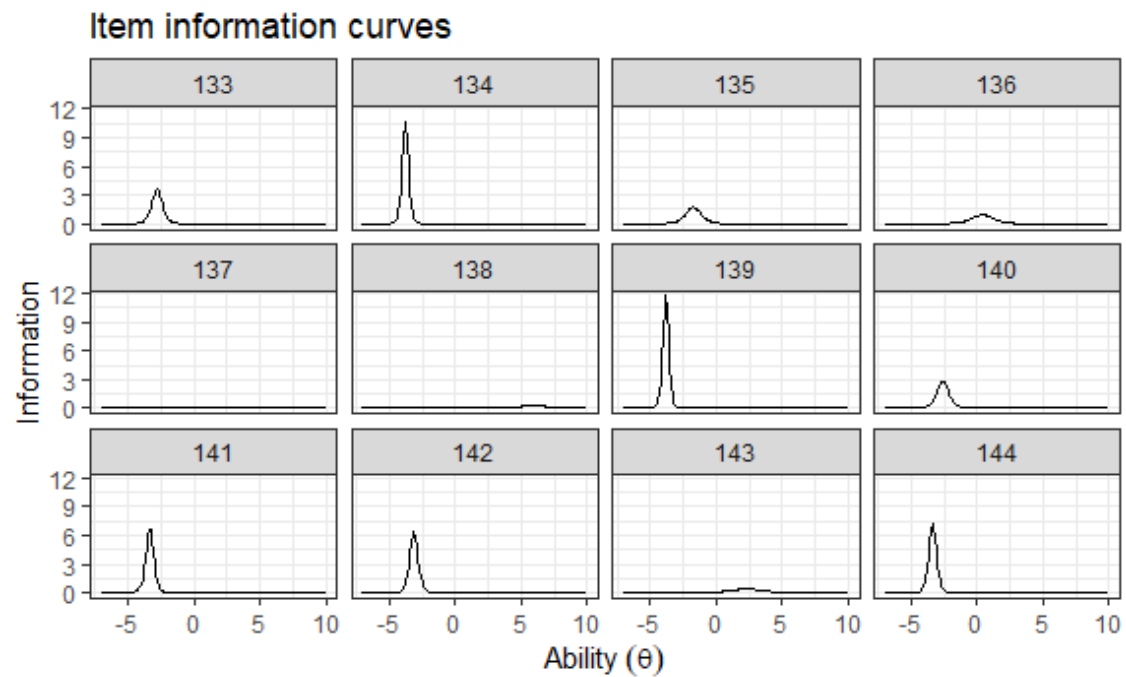


Figure D.12: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

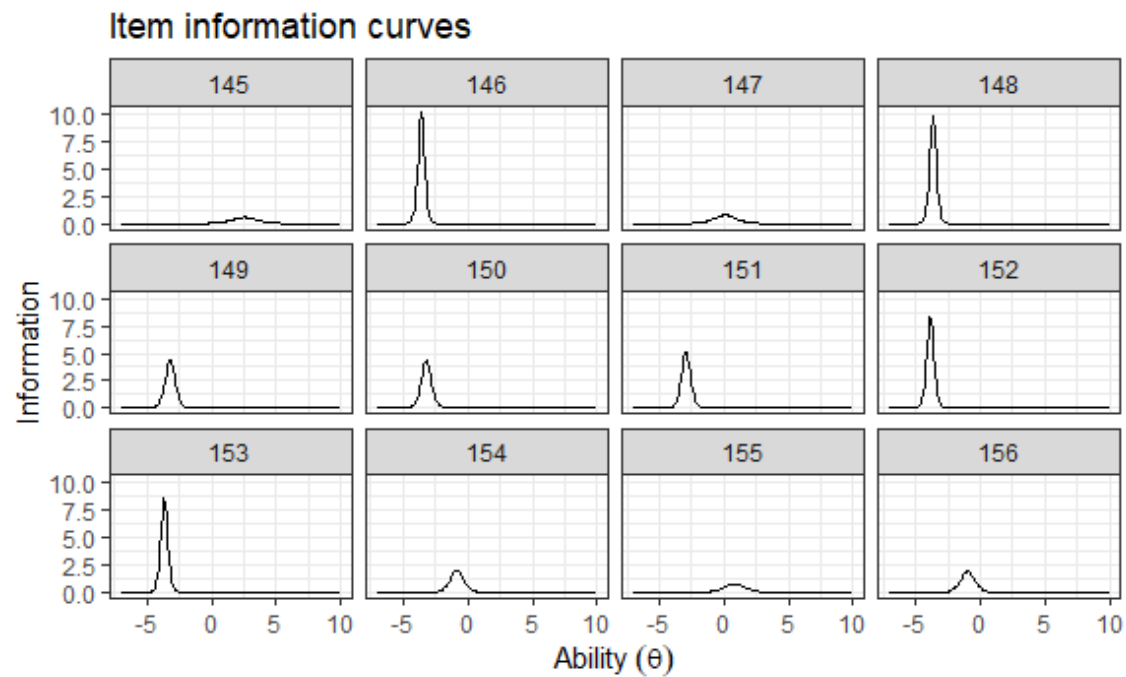


Figure D.13: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

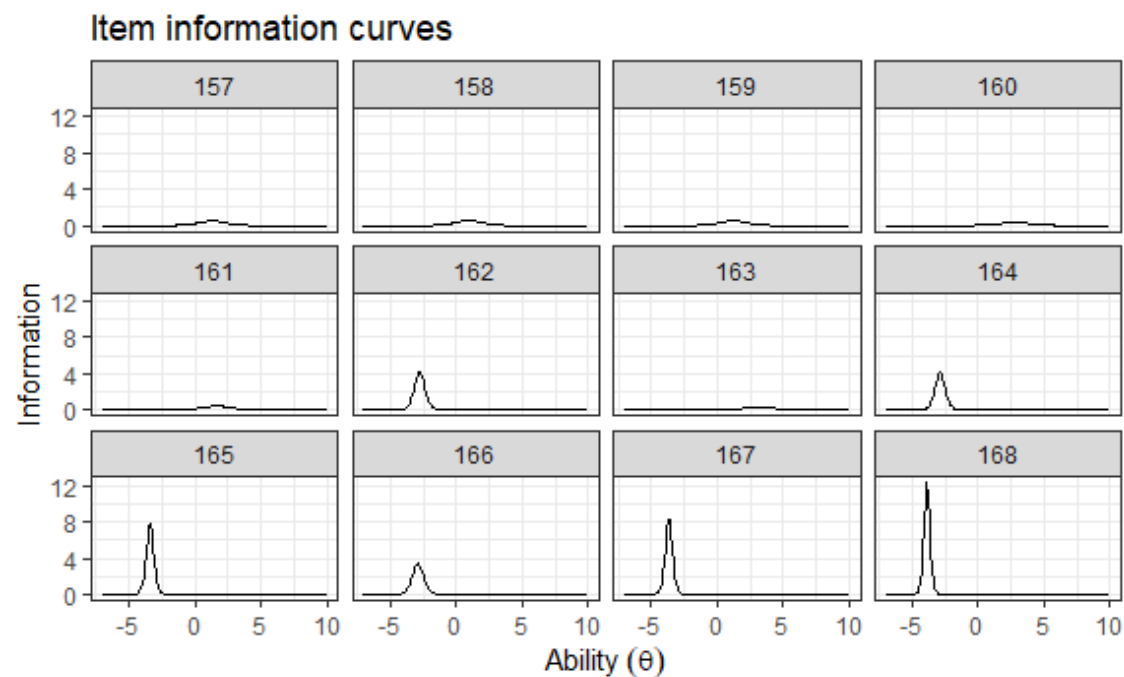


Figure D.14: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

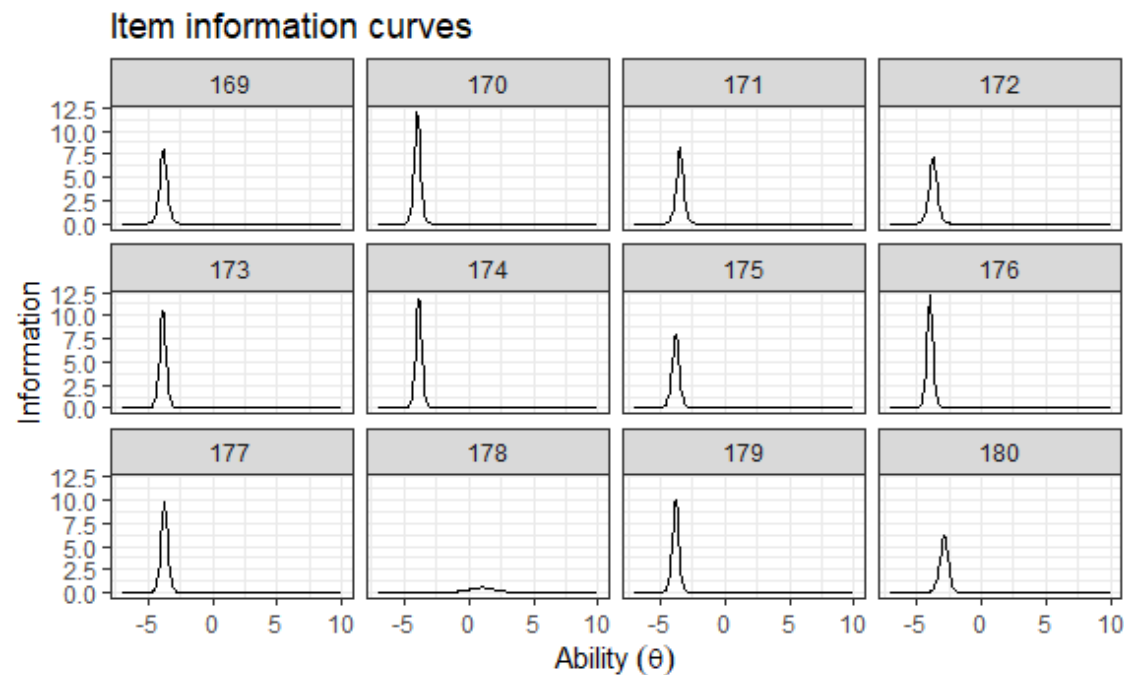


Figure D.15: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

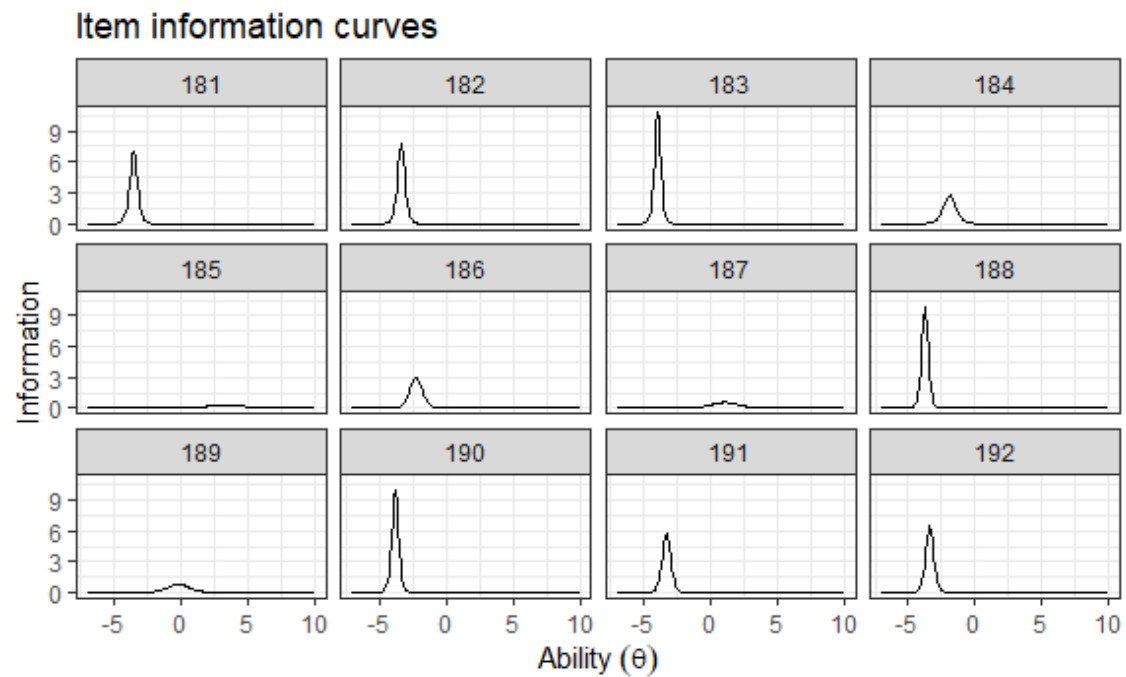


Figure D.16: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

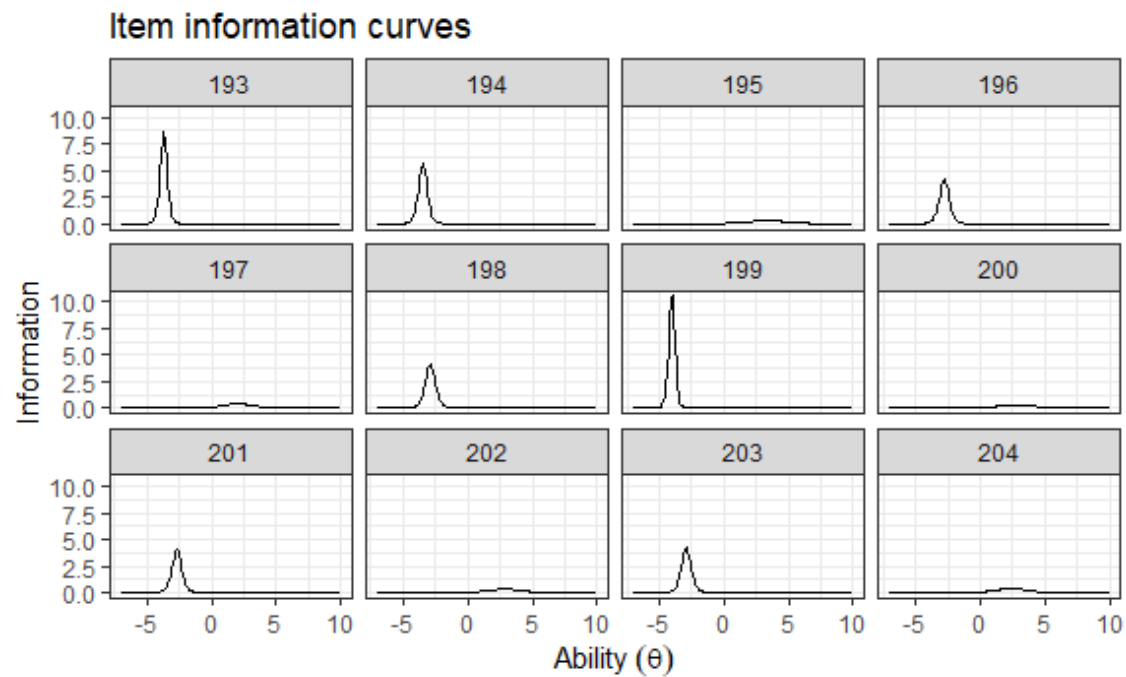


Figure D.17: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

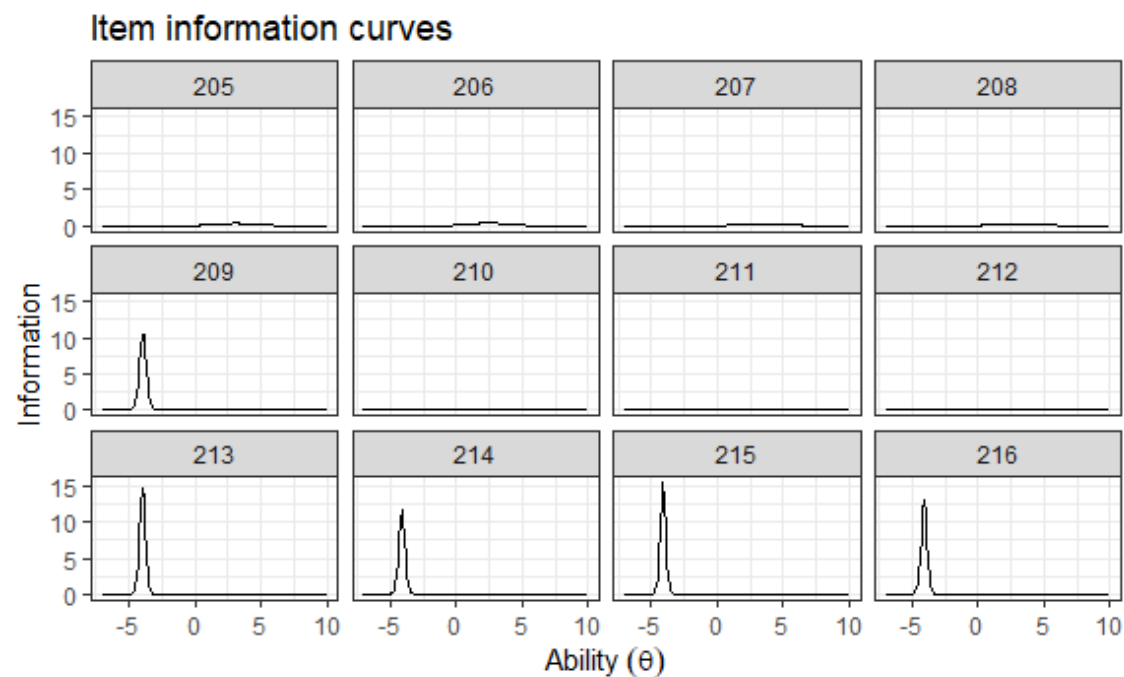


Figure D.18: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

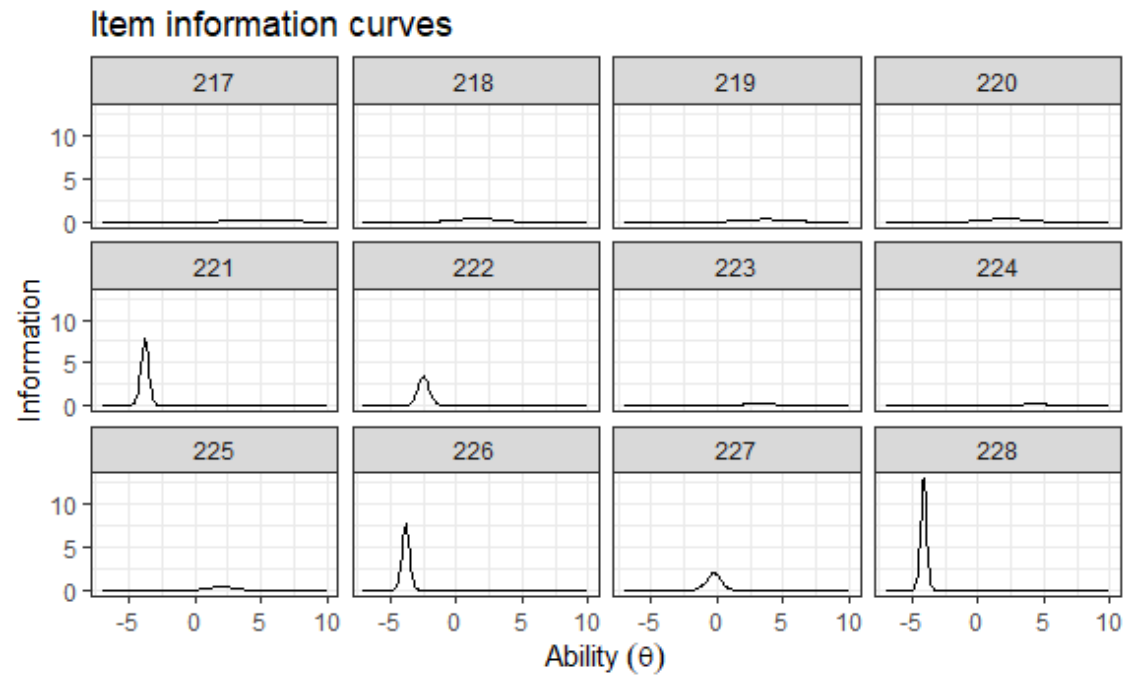


Figure D.19: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

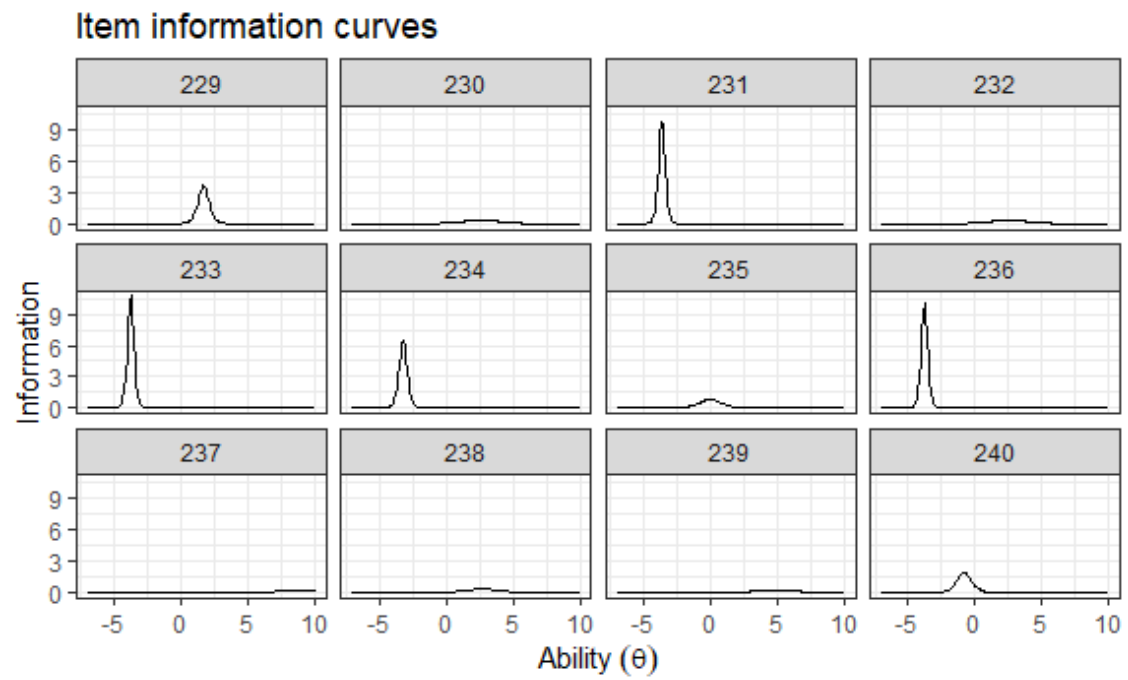


Figure D.20: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

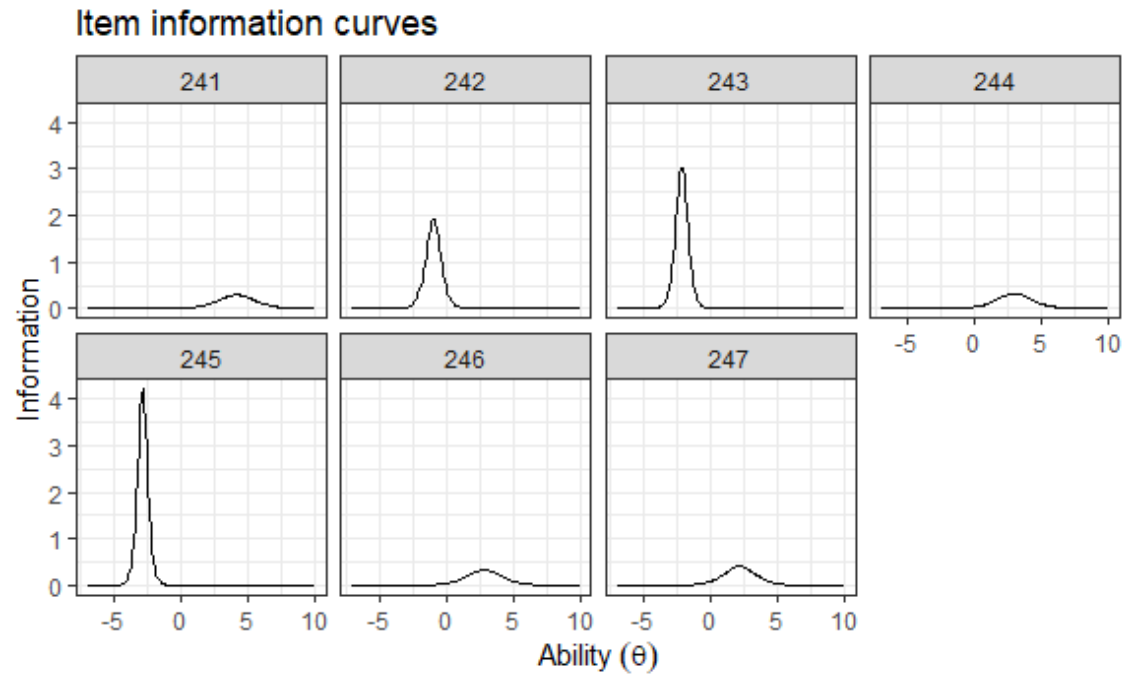


Figure D.21: Curve showing where the maximum information gain from the test item is located in relation to the ability of the respondent

E

Appendix 5: Benchmark difficulty parameter distributions

The difficulty parameter distributions for each of the benchmark functions are presented here.

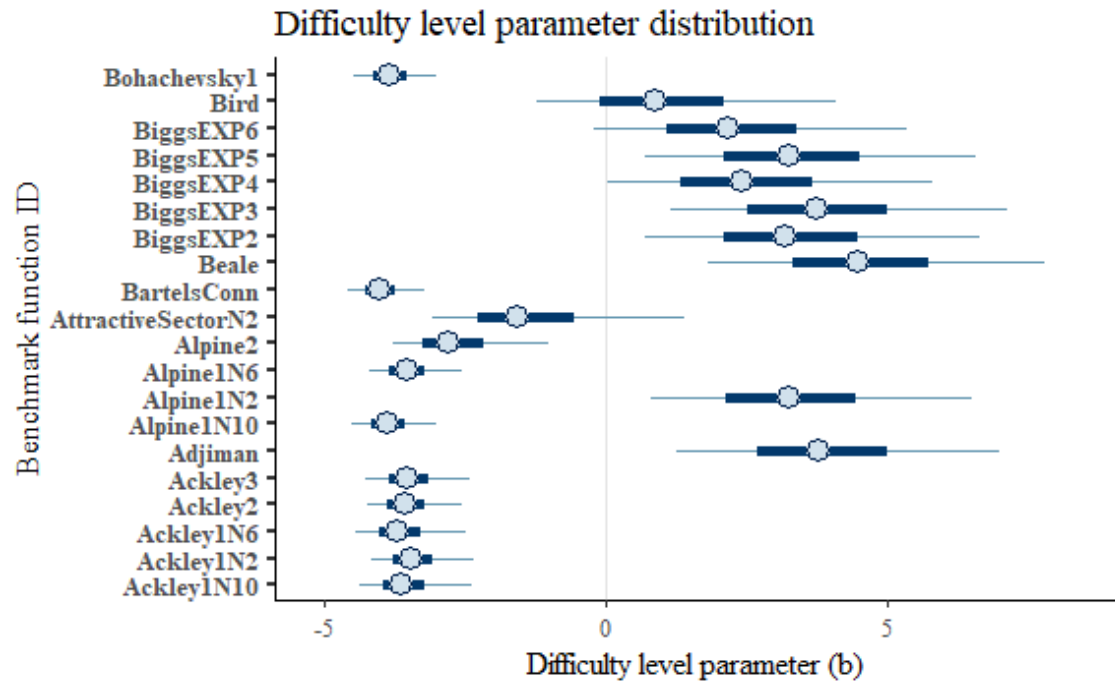


Figure E.1: Distribution of the difficulty parameters for the benchmarks

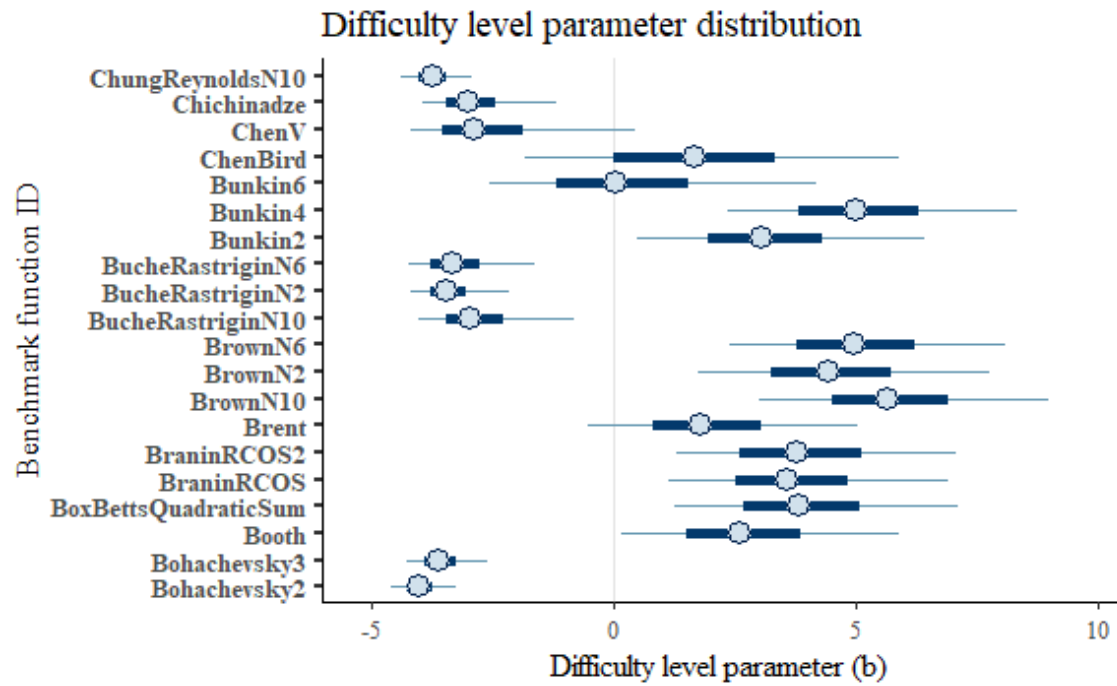


Figure E.2: Distribution of the difficulty parameters for the benchmarks

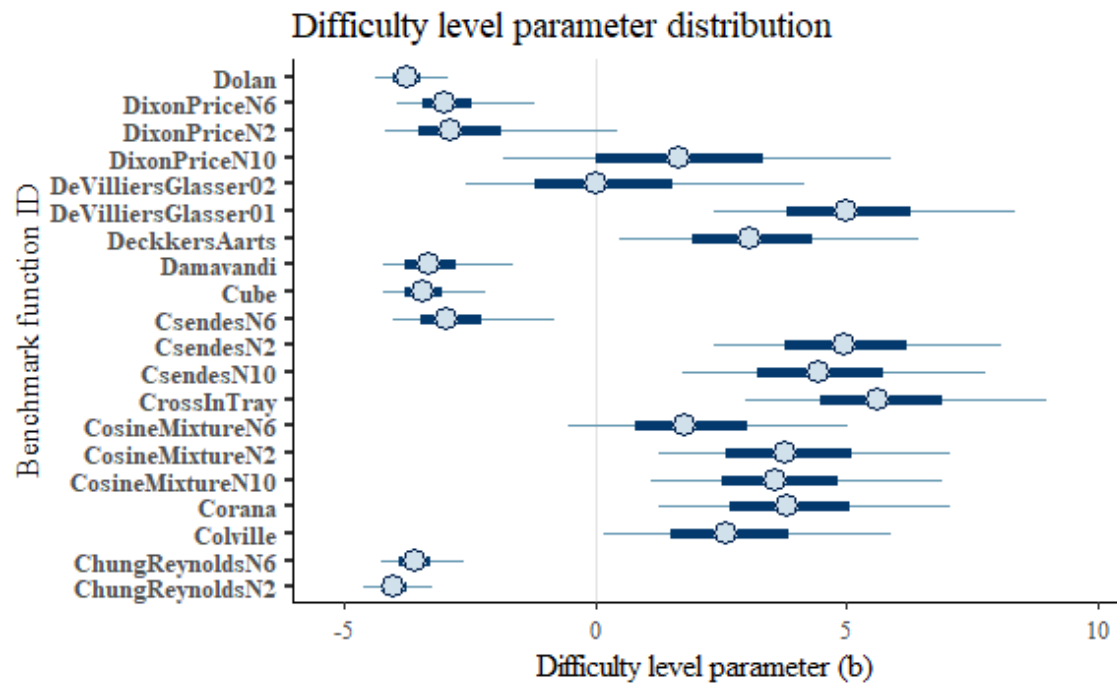


Figure E.3: Distribution of the difficulty parameters for the benchmarks

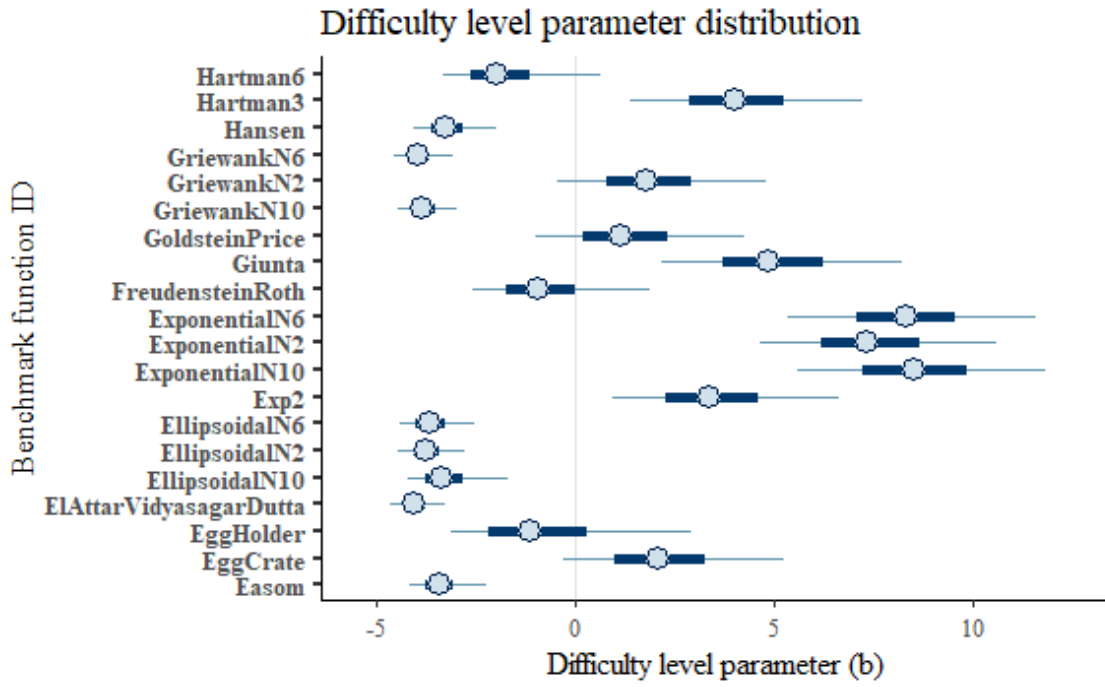


Figure E.4: Distribution of the difficulty parameters for the benchmarks

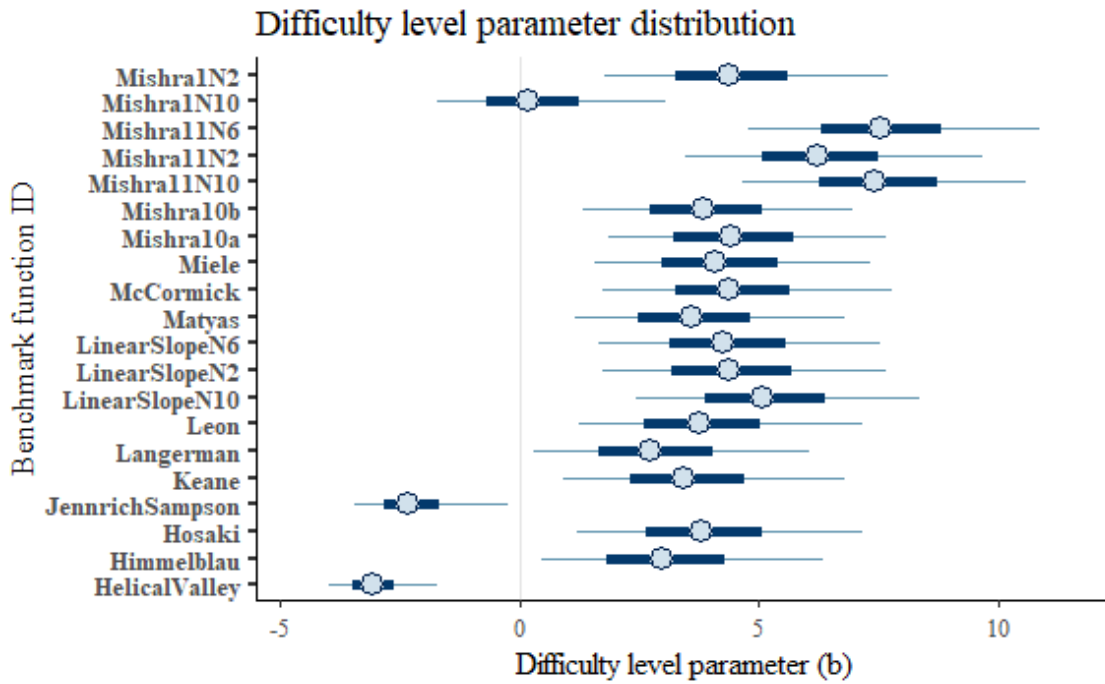


Figure E.5: Distribution of the difficulty parameters for the benchmarks

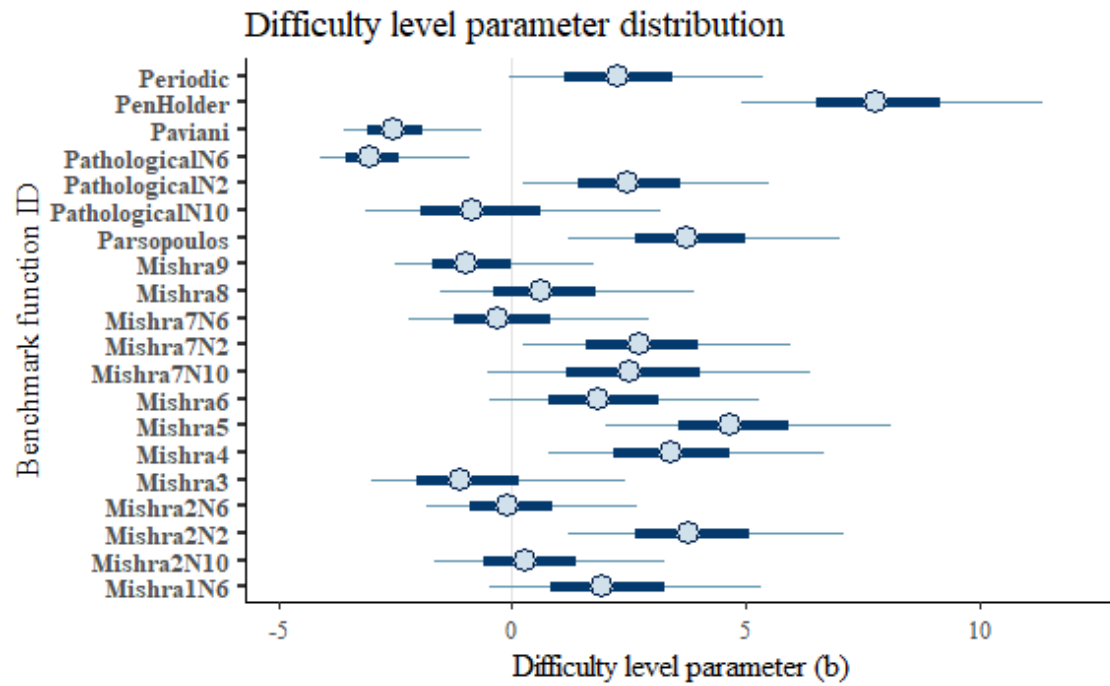


Figure E.6: Distribution of the difficulty parameters for the benchmarks

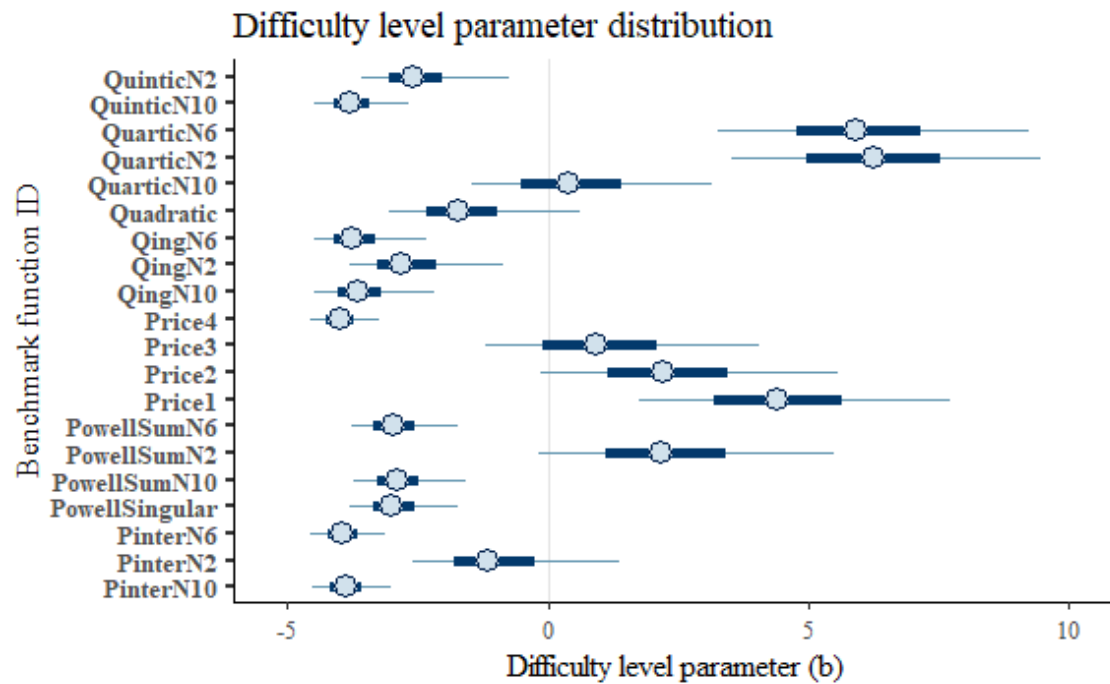


Figure E.7: Distribution of the difficulty parameters for the benchmarks

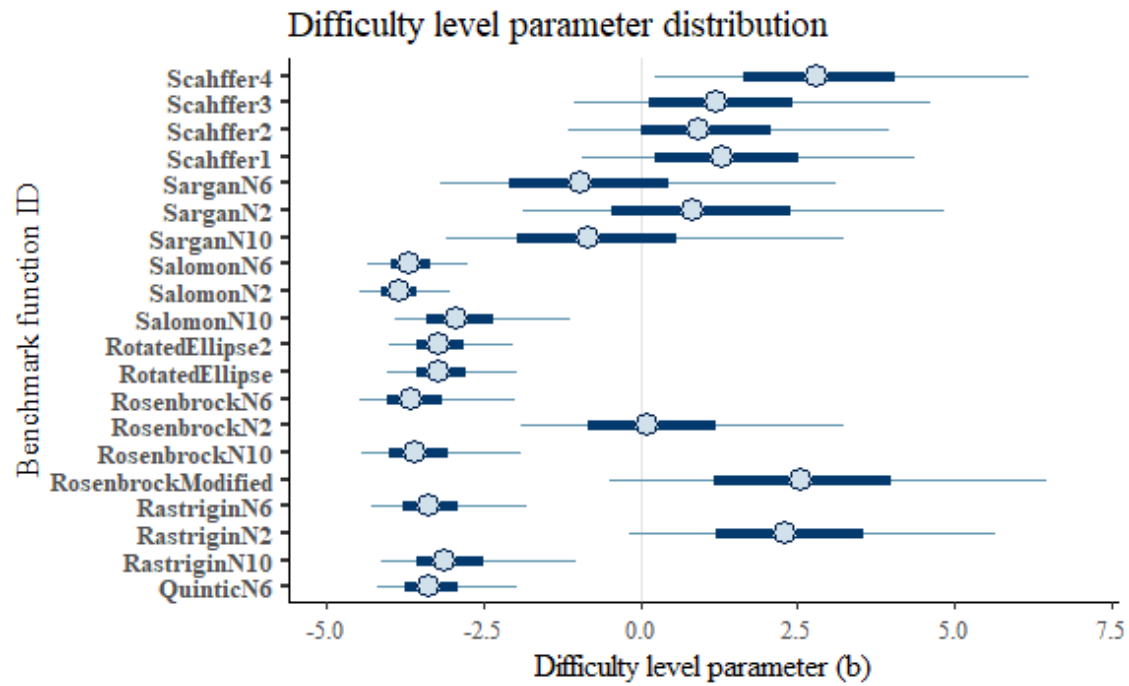


Figure E.8: Distribution of the difficulty parameters for the benchmarks

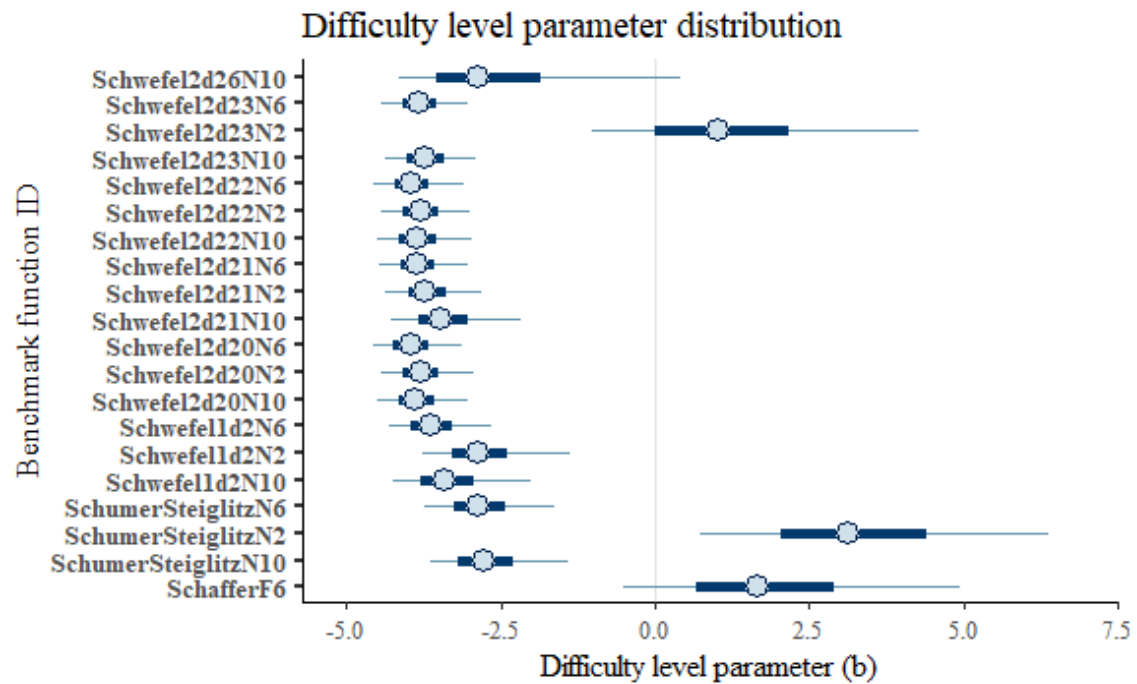


Figure E.9: Distribution of the difficulty parameters for the benchmarks

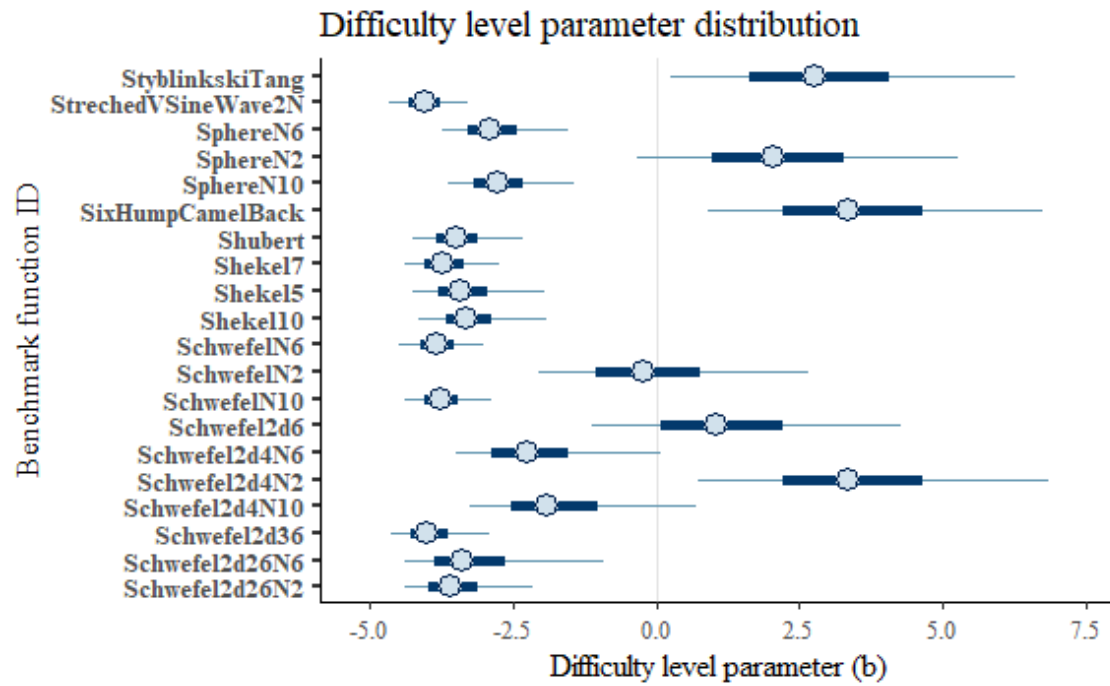


Figure E.10: Distribution of the difficulty parameters for the benchmarks

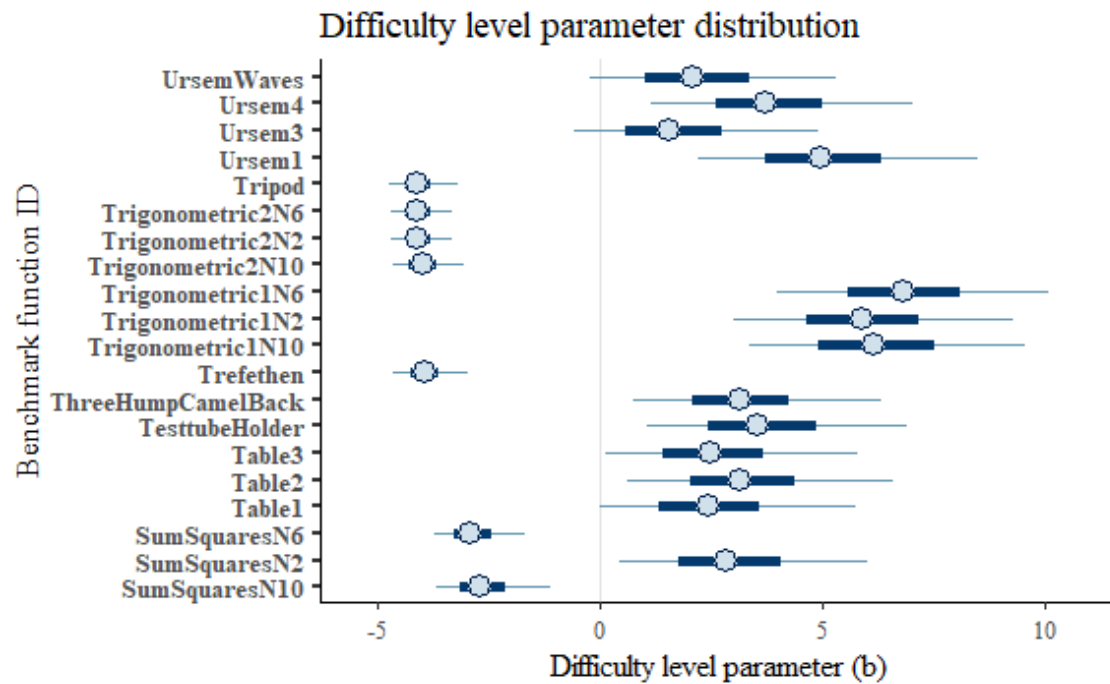


Figure E.11: Distribution of the difficulty parameters for the benchmarks

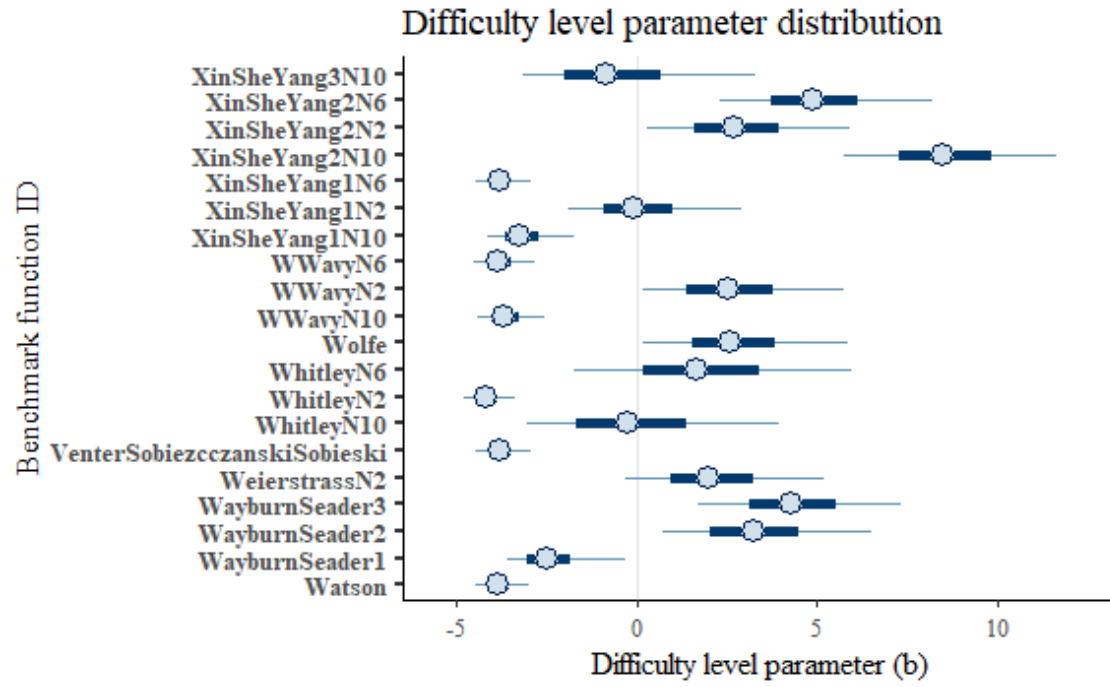


Figure E.12: Distribution of the difficulty parameters for the benchmarks

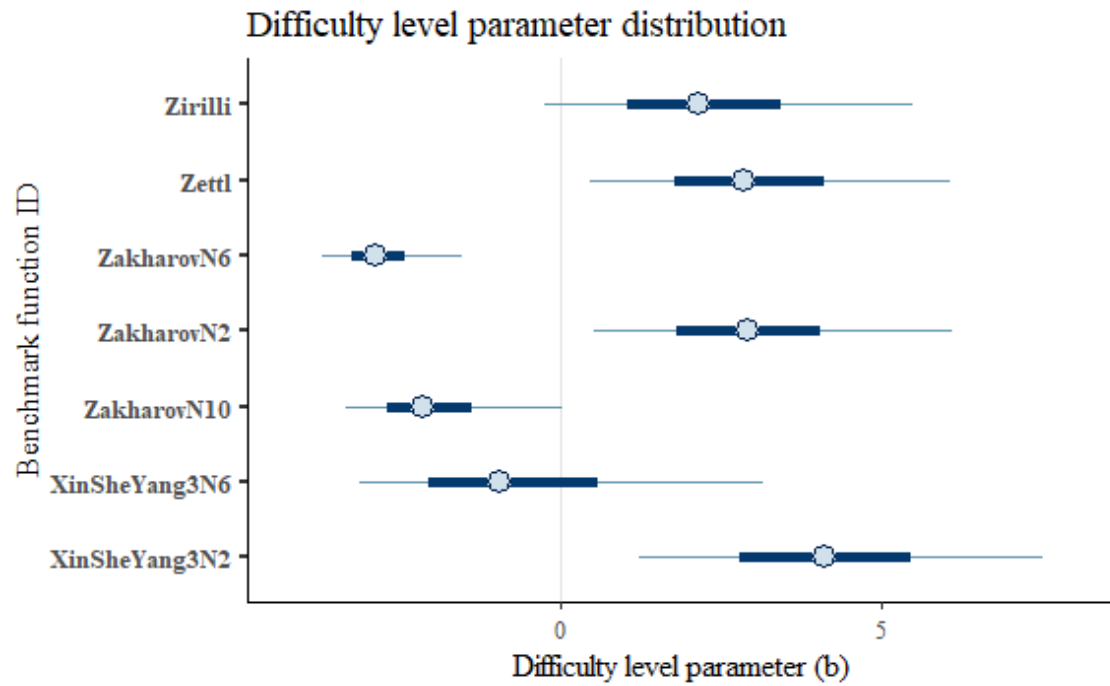


Figure E.13: Distribution of the difficulty parameters for the benchmarks

F

Appendix 6: Tables

Table F.1: Summary values of the discrimination and difficulty level parameters for the benchmarks

Benchmark ID	Median	CI 5%	CI 95%	R-hat
Discrimination value (a)				
Ackley1N10	6.2559600	3.8693940	9.0562580	1.0018221
Ackley1N2	4.7155550	3.2015875	6.5195515	1.0031595
Ackley1N6	6.1550800	3.8143105	9.0621190	1.0004973
Ackley2	4.9405600	3.4169535	6.7183505	0.9997429
Ackley3	4.9554800	3.2490040	7.0291530	1.0026405
Adjiman	1.0513900	0.7876584	1.4295005	1.0030619
Alpine1N10	7.0022700	4.7051540	9.7843535	1.0014663
Alpine1N2	1.1010400	0.8108959	1.5085015	0.9998836
Alpine1N6	5.4327400	3.7644255	7.4352635	1.0039429
Alpine2	3.8298850	2.3674755	5.5983845	1.0011397
AttractiveSectorN2	2.4923900	1.4625210	3.8947150	1.0014529
BartelsConn	6.3515200	4.4232890	8.6456550	1.0010761
Beale	0.9786810	0.7409370	1.3198180	1.0023815
BiggsEXP2	1.1288550	0.8129881	1.5777585	1.0005250
BiggsEXP3	1.1049250	0.8082143	1.5244025	1.0005665
BiggsEXP4	1.3299900	0.9427784	1.8770855	1.0009898
BiggsEXP5	1.2244050	0.8877958	1.7029360	1.0001425
BiggsEXP6	1.4249600	1.0183800	2.0364145	1.0004215
Bird	1.5654900	1.0539660	2.2828970	1.0008386
Bohachevsky1	5.8018200	3.9846175	7.9968910	1.0012746
Bohachevsky2	6.4053000	4.4139145	8.6515790	1.0009396
Bohachevsky3	5.0336800	3.4388700	6.9039620	1.0006164
Booth	1.2064250	0.8683593	1.6904735	1.0003059
BoxBettsQuadraticSum	1.0908700	0.8117755	1.4869900	1.0016432
BraninRCOS	1.0707200	0.7898040	1.4622590	1.0017916
BraninRCOS2	1.0490350	0.7767402	1.4243915	1.0033440
Brent	1.3500250	0.9510468	1.9604990	1.0021730
BrownN10	1.0075000	0.7755451	1.3112200	1.0013499
BrownN2	0.9677065	0.7289317	1.3259370	1.0009651
BrownN6	1.0247450	0.7906657	1.3473875	0.9999356
BucheRastriginN10	4.8315000	2.6565610	7.5719190	1.0006354

BucheRastriginN2	4.8164650	3.1019890	6.7855200	0.9995466
BucheRastriginN6	5.2635650	3.0227370	8.0659585	1.0018332
Bunkin2	1.1463050	0.8280116	1.6255185	1.0001661
Bunkin4	0.9210115	0.7025653	1.2209525	1.0003146
Bunkin6	2.1120000	1.2331155	3.8589110	0.9998983
ChenBird	3.6751700	1.8935030	7.0128765	1.0013593
ChenV	4.3300750	1.9625900	7.7329810	0.9997094
Chichinadze	4.0321150	2.4107055	5.9745485	1.0001253
ChungReynoldsN10	6.3419550	4.4464495	8.6824715	1.0003038
ChungReynoldsN2	1.5722800	1.0726265	2.2823275	1.0001473
ChungReynoldsN6	6.3457250	4.4889395	8.7391055	1.0012990
Colville	4.4150450	2.3329070	7.2198880	0.9999545
Corana	5.8269800	3.8917385	8.0884145	1.0000543
CosineMixtureN10	6.9065050	4.6873985	9.6654030	1.0001622
CosineMixtureN2	0.9744855	0.7365717	1.3132000	1.0024632
CosineMixtureN6	5.8657400	4.1136125	8.0276335	1.0000787
CrossInTray	1.0632150	0.7894426	1.4389640	1.0005322
CsendesN10	0.8030435	0.6459975	0.9965991	1.0006769
CsendesN2	0.7684485	0.6073478	0.9877043	1.0005811
CsendesN6	0.7781060	0.6263573	0.9710270	1.0006095
Cube	1.5886850	1.0551400	2.3106195	1.0014134
Damavandi	2.2219150	1.3017090	4.1923310	1.0003363
DeckkersAarts	3.6769400	1.8841970	7.0362870	1.0001687
DeVilliersGlasser01	3.7652250	1.9481015	7.1522790	1.0006709
DeVilliersGlasser02	3.7724100	1.9732265	7.0816200	1.0002629
DixonPriceN10	4.8858100	2.3309835	8.5692830	1.0011993
DixonPriceN2	1.0958250	0.8023120	1.5041230	1.0028291
DixonPriceN6	5.7056300	3.4031035	8.6539580	0.9998185
Dolan	1.0568600	0.8034309	1.4366790	0.9995896
Easom	4.7792150	3.1262955	6.7582030	1.0013139
EggCrate	1.2870800	0.9104012	1.8292965	1.0004711
EggHolder	2.5964700	1.3776150	4.6903730	1.0016848
ElAttarVidyasagarDutta	6.8585050	4.5925410	9.6220035	1.0009678
EllipsoidalN10	5.5877450	3.1999925	8.4597820	1.0007113
EllipsoidalN2	5.6942300	3.7425830	8.1163190	1.0002168
EllipsoidalN6	6.1890600	3.8597785	9.0221385	1.0032486
Exp2	1.1028250	0.8140511	1.5029315	1.0016643
ExponentialN10	0.8013355	0.6474288	1.0053435	1.0016766
ExponentialN2	0.7495965	0.6021502	0.9422220	1.0003364
ExponentialN6	0.7766925	0.6303851	0.9833230	1.0012389
FreudensteinRoth	2.2251450	1.3941440	3.3546700	1.0021779
Giunta	0.9398790	0.7130352	1.2449060	1.0015220
GoldsteinPrice	1.4854700	1.0257320	2.1448035	1.0017454
GriewankN10	6.8916450	4.6777510	9.6453990	1.0009250
GriewankN2	1.3422600	0.9462487	1.8945870	1.0001525
GriewankN6	6.9918450	4.7094690	9.8293090	1.0013238

Hansen	4.4320950	2.8977695	6.2995420	1.0000132
Hartman3	1.0675100	0.7980684	1.4621525	1.0019452
Hartman6	3.2672950	1.9275435	5.0345405	1.0007843
HelicalValley	4.2290150	2.7453805	6.0392910	0.9995877
Himmelblau	1.1509050	0.8304455	1.6177575	1.0009418
Hosaki	1.0445550	0.7744565	1.4340310	1.0017933
JennrichSampson	3.1347650	1.9219310	4.5761740	0.9999837
Keane	1.0956700	0.8060226	1.5058270	1.0008564
Langerman	1.1984500	0.8621298	1.6902675	1.0005538
Leon	1.0468100	0.7788909	1.4323970	1.0008657
LinearSlopeN10	1.0718650	0.8237401	1.4212930	0.9996885
LinearSlopeN2	0.9880080	0.7478451	1.3210280	1.0014109
LinearSlopeN6	1.1067300	0.8300255	1.4862610	1.0006990
Matyas	1.0582250	0.7866963	1.4378810	1.0016544
McCormick	0.9945600	0.7427400	1.3436505	1.0020902
Miele	1.0848300	0.8172925	1.4564710	1.0003815
Mishra10a	0.9704550	0.7346704	1.2987810	1.0046947
Mishra10b	1.0288700	0.7743836	1.3894175	1.0010237
Mishra11N10	0.8663950	0.7013314	1.0934850	1.0013067
Mishra11N2	0.8196755	0.6374425	1.0669940	1.0005121
Mishra11N6	0.8196940	0.6576297	1.0398195	1.0051814
Mishra1N10	2.0342950	1.3674760	2.9624970	1.0013288
Mishra1N2	0.9857395	0.7379430	1.3238965	0.9999120
Mishra1N6	1.4760950	1.0214005	2.1560185	1.0040044
Mishra2N10	1.9973550	1.3399370	2.9451305	1.0019231
Mishra2N2	1.0483150	0.7759285	1.4334805	1.0000746
Mishra2N6	2.0353900	1.3672490	2.9452370	1.0007167
Mishra3	2.4928750	1.3873265	4.2185870	0.9996129
Mishra4	1.1325400	0.8271356	1.5838020	1.0009513
Mishra5	0.9613830	0.7227413	1.2961115	0.9999639
Mishra6	1.3344550	0.9203105	1.9424625	0.9996405
Mishra7N10	1.7894100	1.2034625	2.8359705	1.0002544
Mishra7N2	1.1929850	0.8627734	1.6879415	1.0009563
Mishra7N6	2.1794600	1.3583690	3.3330370	1.0008275
Mishra8	1.6194300	1.0670815	2.4289635	0.9999410
Mishra9	2.3070150	1.4562155	3.3740365	1.0005510
Parsopoulos	1.0540500	0.7805754	1.4301990	1.0021058
PathologicalN10	2.8332950	1.5373960	5.2708570	1.0007593
PathologicalN2	1.2138750	0.8876326	1.6713575	1.0004069
PathologicalN6	4.8151550	2.5872975	7.5849120	1.0039778
Paviani	3.9136650	2.4303765	5.7807885	1.0000688
PenHolder	0.7321345	0.5786008	0.9275007	1.0026286
Periodic	1.2555450	0.8992555	1.7712860	0.9997690
PinterN10	7.0058150	4.6863240	9.6446970	1.0018423
PinterN2	2.2415600	1.4432600	3.2189400	1.0003958
PinterN6	7.0025850	4.7997040	9.6678720	1.0017750

PowellSingular	4.0970850	2.7682825	5.6106920	1.0005702
PowellSumN10	4.3364750	2.9590330	5.9408550	1.0005513
PowellSumN2	1.2565750	0.8921026	1.7859630	1.0009999
PowellSumN6	4.2117650	2.8925925	5.7907030	1.0003974
Price1	0.9827175	0.7418503	1.3276375	1.0001969
Price2	1.2632250	0.8887554	1.7885235	1.0005715
Price3	1.5317150	1.0446210	2.2525130	1.0027331
Price4	6.3388450	4.4319150	8.7101740	1.0009971
QingN10	6.5717250	3.7975335	10.0667650	1.0001283
QingN2	3.7661450	2.2185920	5.6265690	1.0012560
QingN6	6.5962000	3.7896125	9.9629130	0.9997511
Quadratic	2.6230850	1.6477770	3.8280635	1.0005142
QuarticN10	1.9204850	1.3133685	2.7510090	1.0000358
QuarticN2	0.8183690	0.6451744	1.0597230	1.0009032
QuarticN6	0.9336825	0.7283356	1.2138915	0.9997882
QuinticN10	6.8599650	4.3226130	9.9302105	1.0010339
QuinticN2	3.3782900	2.1170080	4.8724235	1.0031890
QuinticN6	5.1912250	3.2378230	7.5526435	1.0007647
RastriginN10	5.0521250	2.7932900	7.9053715	1.0007142
RastriginN2	1.2541700	0.8918968	1.8238065	1.0007647
RastriginN6	5.4165350	3.1729760	8.2728350	1.0002475
RosenbrockModified	1.5110000	1.0267990	2.4046490	0.9998318
RosenbrockN10	6.4045000	3.4983315	9.9702460	1.0003920
RosenbrockN2	1.7896450	1.1580530	2.7016065	1.0008319
RosenbrockN6	6.3406200	3.4695855	9.6831660	1.0014615
RotatedEllipse	4.2060150	2.7775030	5.9100230	1.0020278
RotatedEllipse2	4.2114150	2.8472760	5.8973990	0.9997327
SalomonN10	4.5606550	2.7495390	6.8067510	0.9999262
SalomonN2	5.7990100	4.0041000	7.8200585	1.0008714
SalomonN6	5.8789850	4.0402275	8.1008415	0.9996949
SarganN10	2.8388300	1.5453115	5.2596490	1.0015354
SarganN2	1.8035300	1.1172385	3.0515085	1.0049491
SarganN6	2.7965450	1.5071090	5.1547485	0.9998848
Scahffer1	1.4392850	1.0025695	2.0692660	1.0006266
Scahffer2	1.5123800	1.0387570	2.1846900	1.0003167
Scahffer3	1.4816650	0.9951760	2.1996280	1.0009724
Scahffer4	1.2016250	0.8585134	1.7157270	1.0007871
SchafferF6	1.3556850	0.9402004	1.9340400	1.0002497
SchumerSteiglitzN10	4.1003350	2.7952245	5.5883850	1.0014787
SchumerSteiglitzN2	1.1117650	0.8178093	1.5336405	1.0006920
SchumerSteiglitzN6	4.0914800	2.7906750	5.6046475	1.0007756
Schwefel1d2N10	5.6287100	3.4689655	8.3179000	1.0025743
Schwefel1d2N2	3.7037450	2.4107815	5.2287155	1.0004384
Schwefel1d2N6	5.8219000	3.9281445	8.0846510	1.0000011
Schwefel2d20N10	7.0255050	4.7730690	9.6958660	0.9997848
Schwefel2d20N2	5.6950350	3.9576670	7.7995240	1.0007236

Schwefel2d20N6	7.0016900	4.7345525	9.7760075	1.0055833
Schwefel2d21N10	5.7497100	3.5851325	8.5935855	0.9998421
Schwefel2d21N2	5.3927500	3.7337080	7.3353920	1.0019915
Schwefel2d21N6	6.5159000	4.5233845	8.9316750	1.0027973
Schwefel2d22N10	6.9206850	4.6501265	9.6375865	1.0008690
Schwefel2d22N2	5.6649750	3.9708225	7.7597410	1.0029778
Schwefel2d22N6	7.0075000	4.6755340	9.6122900	1.0040274
Schwefel2d23N10	6.3240350	4.3959510	8.6266970	1.0008021
Schwefel2d23N2	1.4994350	1.0119145	2.1743450	1.0005268
Schwefel2d23N6	6.3508850	4.4716660	8.6417820	1.0020068
Schwefel2d26N10	4.9687150	2.2389510	8.7724205	0.9995981
Schwefel2d26N2	5.3189000	3.1932160	8.1104980	1.0004645
Schwefel2d26N6	5.6384600	2.6984910	9.3799965	1.0010226
Schwefel2d36	6.6313100	4.1175080	9.6250285	1.0009613
Schwefel2d4N10	3.3080700	1.9490100	5.0281675	1.0005620
Schwefel2d4N2	1.1062350	0.7980306	1.5503380	1.0001933
Schwefel2d4N6	3.4689500	2.0644275	5.2460595	1.0017316
Schwefel2d6	1.5095600	1.0248170	2.2061415	1.0009708
SchwefelN10	6.3323900	4.4577185	8.6445065	1.0013619
SchwefelN2	1.8510850	1.2135945	2.7357305	1.0008212
SchwefelN6	6.3413750	4.4333565	8.6923130	0.9999400
Shekel10	4.8302450	3.0303465	6.9585035	1.0000413
Shekel5	5.0921400	3.1483755	7.5978135	1.0005967
Shekel7	5.8726450	3.9146635	8.2762015	0.9994835
Shubert	4.8025500	3.1823000	6.7977865	1.0004128
SixHumpCamelBack	1.1025500	0.8060859	1.5163420	1.0023900
SphereN10	4.0940050	2.8041885	5.5500505	1.0016951
SphereN2	1.2877400	0.9088784	1.8314415	1.0017256
SphereN6	4.0905550	2.7470355	5.6476400	1.0018514
StretchedVSineWave2N	6.5564300	4.5574915	9.1268065	1.0030107
StyblinskiTang	1.1795850	0.8437663	1.6667950	1.0017499
SumSquaresN10	4.0455150	2.6045665	5.7857515	1.0014176
SumSquaresN2	1.1537600	0.8423612	1.5986165	1.0013955
SumSquaresN6	4.0988800	2.8422795	5.6362845	1.0007795
Table1	1.2475200	0.8870920	1.7775145	0.9998706
Table2	1.1410050	0.8219420	1.5983755	1.0001461
Table3	1.2368100	0.8825430	1.7387745	1.0006630
TesttubeHolder	1.0823150	0.7998026	1.4769080	1.0012787
ThreeHumpCamelBack	1.1156100	0.8235215	1.5240050	1.0013656
Trefethen	6.5227200	4.0821490	9.5418695	1.0018539
Trigonometric1N10	0.9616445	0.7460355	1.2514590	1.0027296
Trigonometric1N2	0.8473070	0.6547589	1.1169405	1.0010410
Trigonometric1N6	0.8690815	0.6906126	1.1177695	1.0041034
Trigonometric2N10	7.6924650	4.8800660	11.0568800	1.0005485
Trigonometric2N2	6.8680050	4.6705750	9.5033525	1.0001986
Trigonometric2N6	7.8680400	5.1920205	11.0806400	1.0003244

Tripod	7.2020100	4.5464755	10.4219250	0.9998404
Ursem1	0.9309770	0.7028338	1.2587350	1.0029920
Ursem3	1.3852650	0.9559128	1.9474545	1.0000319
Ursem4	1.0615250	0.7884965	1.4467710	1.0002919
UrsemWaves	1.3019750	0.9171508	1.8532210	1.0014583
Watson	5.6220850	3.9369070	7.6963690	0.9999648
WayburnSeader1	3.7404100	2.2290050	5.5236010	1.0005572
WayburnSeader2	1.1296050	0.8242394	1.5681210	1.0013457
WayburnSeader3	0.9937575	0.7623490	1.3336360	1.0025614
WeierstrassN2	1.3100050	0.9268324	1.8624535	1.0012287
VenterSobieczcczanskiSobieski	5.5917100	3.9172350	7.6672400	0.9995379
WhitleyN10	2.8574050	1.6179880	5.7610290	1.0003320
WhitleyN2	7.2994650	4.8171090	10.3021250	1.0017540
WhitleyN6	3.8117500	1.9540900	7.1080435	1.0009979
Wolfe	1.2428100	0.8908062	1.7466310	0.9997556
WWavyN10	6.3048850	3.9667100	9.1249940	1.0052470
WWavyN2	1.2108900	0.8713817	1.6812405	1.0018034
WWavyN6	6.5636650	4.2639595	9.1925985	1.0022136
XinSheYang1N10	5.0787250	3.1458140	7.5298775	1.0013791
XinSheYang1N2	1.8118200	1.1909785	2.6786420	1.0011166
XinSheYang1N6	6.3343200	4.2380810	8.6355415	1.0034029
XinSheYang2N10	0.8021770	0.6557513	0.9908122	1.0013303
XinSheYang2N2	1.1817750	0.8522660	1.6446830	1.0007432
XinSheYang2N6	1.0298200	0.7871749	1.3507085	1.0028174
XinSheYang3N10	2.8162700	1.5183695	5.3563020	1.0004028
XinSheYang3N2	1.0932200	0.8109650	1.5406335	0.9995762
XinSheYang3N6	2.7917650	1.4946325	5.2019470	0.9999838
ZakharovN10	3.4879750	2.1675070	5.1720800	1.0028192
ZakharovN2	1.1441450	0.8360222	1.5822045	1.0012123
ZakharovN6	4.1065000	2.7430145	5.6092880	1.0004321
Zettl	1.1490700	0.8400539	1.5946905	0.9999796
Zirilli	1.2929050	0.9074066	1.8630400	1.0019879
Difficulty level (b)				
Ackley1N10	-3.6277750	-4.3623950	-2.3852275	1.0015657
Ackley1N2	-3.4583050	-4.1621635	-2.3622030	1.0026866
Ackley1N6	-3.7075900	-4.4286220	-2.4712595	1.0006779
Ackley2	-3.5707950	-4.2296140	-2.5611715	0.9998586
Ackley3	-3.5248050	-4.2784360	-2.4100950	1.0012085
Adjiman	3.7689500	1.2497920	6.9904035	1.0026442
Alpine1N10	-3.8937150	-4.5102635	-3.0140085	1.0017343
Alpine1N2	3.2265600	0.7814915	6.4853495	0.9995957
Alpine1N6	-3.5349250	-4.2033805	-2.5721930	1.0020502
Alpine2	-2.8108250	-3.7759635	-1.0290770	1.0009113
AttractiveSectorN2	-1.5820750	-3.0888640	1.3772555	1.0017538
BartelsConn	-4.0096500	-4.5879475	-3.2289550	1.0014304
Beale	4.4533200	1.8064060	7.7909980	1.0028593

BiggsEXP2	3.1664100	0.6745871	6.6295665	1.0014547
BiggsEXP3	3.7189950	1.1313690	7.1051300	1.0009396
BiggsEXP4	2.4143950	0.0247827	5.8042785	1.0003475
BiggsEXP5	3.2396500	0.7070561	6.5752660	0.9998767
BiggsEXP6	2.1722450	-0.2340701	5.3188950	1.0009408
Bird	0.8601830	-1.2250065	4.0626605	1.0008165
Bohachevsky1	-3.8606400	-4.4912995	-2.9951425	1.0011280
Bohachevsky2	-4.0095400	-4.5998915	-3.2439695	1.0011131
Bohachevsky3	-3.5978350	-4.2610625	-2.6156805	1.0014991
Booth	2.6094450	0.1812199	5.8840020	0.9999164
BoxBettsQuadraticSum	3.8133250	1.2594245	7.0860180	1.0010268
BraninRCOS	3.5894050	1.1234500	6.9220320	1.0010030
BraninRCOS2	3.7737050	1.2789245	7.0809870	1.0038023
Brent	1.7946500	-0.5115528	5.0491105	1.0020098
BrownN10	5.6340550	2.9874930	8.9894060	1.0016959
BrownN2	4.4432350	1.7405635	7.7694220	1.0014981
BrownN6	4.9611050	2.3828180	8.0777655	1.0001271
BucheRastriginN10	-2.9706500	-4.0223490	-0.8098627	1.0009786
BucheRastriginN2	-3.4463150	-4.1921400	-2.1727380	1.0006812
BucheRastriginN6	-3.3223300	-4.2247125	-1.6324235	1.0012720
Bunkin2	3.0573200	0.4941841	6.4292665	0.9997791
Bunkin4	4.9949550	2.3606380	8.3419025	1.0004397
Bunkin6	0.0227620	-2.5704025	4.1619400	0.9996487
ChenBird	1.6484350	-1.8329545	5.8731600	1.0003579
ChenV	-2.8865900	-4.1675765	0.4403063	0.9999845
Chichinadze	-3.0010900	-3.9336160	-1.1845440	1.0006079
ChungReynoldsN10	-3.7515850	-4.3690220	-2.9083980	1.0018487
ChungReynoldsN2	0.6891605	-1.3115230	3.7555425	1.0006815
ChungReynoldsN6	-3.8401150	-4.4444220	-2.9922045	1.0002388
Colville	-2.9380250	-4.0692630	-0.5481169	1.0004176
Corana	-3.7070700	-4.3814950	-2.7160310	1.0007602
CosineMixtureN10	-3.8678850	-4.5007580	-2.9764895	1.0034505
CosineMixtureN2	4.3653350	1.7513170	7.6560820	1.0018554
CosineMixtureN6	-3.6773600	-4.3424180	-2.7931695	1.0004243
CrossInTray	3.6558900	1.1779930	6.9332925	1.0015160
CsendesN10	8.4503650	5.6388640	11.8491550	1.0004265
CsendesN2	7.0233900	4.1428765	10.4000750	1.0008670
CsendesN6	8.2562500	5.4666005	11.6405450	1.0006396
Cube	0.7304720	-1.3602240	3.9206105	1.0007220
Damavandi	0.6136410	-2.4246975	4.9346285	0.9997166
DeckkersAarts	1.5500650	-1.9554050	5.9854870	1.0006395
DeVilliersGlasser01	1.6292150	-1.7609675	5.9875785	1.0007712
DeVilliersGlasser02	1.5582750	-1.6417670	5.9506240	1.0017544
DixonPriceN10	-2.8557400	-4.1058650	0.2584619	1.0018308
DixonPriceN2	3.3837850	0.9080347	6.7788180	1.0032134
DixonPriceN6	-3.5085300	-4.3203590	-2.0461700	0.9998617

Dolan	4.6740200	1.9072045	7.9734250	1.0000407
Easom	-3.4497900	-4.1992010	-2.2393000	1.0003555
EggCrate	2.0365250	-0.3210878	5.2330975	1.0012626
EggHolder	-1.1740650	-3.1560930	2.9193635	1.0021599
ElAttarVidyasagarDutta	-4.0853700	-4.6956955	-3.2723505	1.0009787
EllipsoidalN10	-3.3735950	-4.2137590	-1.6858240	1.0011968
EllipsoidalN2	-3.7742800	-4.4685785	-2.7742385	1.0004686
EllipsoidalN6	-3.6931450	-4.4223175	-2.5233385	1.0035210
Exp2	3.3432750	0.9018564	6.6202470	1.0022006
ExponentialN10	8.5024350	5.5769750	11.7935700	1.0018058
ExponentialN2	7.3228150	4.6545910	10.5803600	1.0005192
ExponentialN6	8.2777900	5.3447980	11.5519550	1.0018998
FreudensteinRoth	-0.9723545	-2.6090920	1.8494720	1.0026873
Giunta	4.8349600	2.1771320	8.1893270	1.0020082
GoldsteinPrice	1.1249450	-0.9870123	4.2393650	1.0017819
GriewankN10	-3.8668400	-4.4947850	-3.0111680	1.0002759
GriewankN2	1.7601950	-0.4608704	4.8060855	1.0003172
GriewankN6	-3.9642050	-4.5814630	-3.1007005	1.0017530
Hansen	-3.2950800	-4.0743735	-1.9984515	1.0007032
Hartman3	3.9694750	1.3528970	7.2224490	1.0024459
Hartman6	-2.0093450	-3.3394725	0.6446408	1.0002112
HelicalValley	-3.0793750	-3.9575680	-1.7132715	0.9995597
Himmelblau	2.9901850	0.4675440	6.3507355	1.0003839
Hosaki	3.7799400	1.2252200	7.1652195	1.0018440
JennrichSampson	-2.3440450	-3.4398625	-0.2266343	1.0006240
Keane	3.4205350	0.9138593	6.7862965	1.0008474
Langerman	2.7374550	0.3107510	6.0559180	1.0002018
Leon	3.7742250	1.2271225	7.1501635	1.0009804
LinearSlopeN10	5.0626550	2.4225810	8.3561755	0.9996477
LinearSlopeN2	4.3590000	1.7391970	7.6562480	1.0022940
LinearSlopeN6	4.2381050	1.6728305	7.5443890	1.0004218
Matyas	3.5954100	1.1442915	6.8017160	1.0024840
McCormick	4.3710900	1.7278055	7.7805905	1.0031244
Miele	4.0871800	1.5669305	7.3321105	1.0005985
Mishra10a	4.4173000	1.8508480	7.6773035	1.0027686
Mishra10b	3.8418150	1.3176745	6.9730000	1.0011307
Mishra11N10	7.4336700	4.6746965	10.5617500	1.0013225
Mishra11N2	6.2248800	3.4864095	9.6638345	1.0006988
Mishra11N6	7.5526750	4.7736075	10.8560400	1.0054943
Mishra1N10	0.1651635	-1.6970650	3.0571890	1.0008056
Mishra1N2	4.3900600	1.7970960	7.7115405	1.0001048
Mishra1N6	1.9431300	-0.4744529	5.3424460	1.0036190
Mishra2N10	0.2815795	-1.6683125	3.2686900	1.0041369
Mishra2N2	3.7866000	1.2235215	7.1067700	0.9997577
Mishra2N6	-0.0897478	-1.8189300	2.7001110	1.0002149
Mishra3	-1.1035800	-2.9951760	2.4444030	0.9998506

Mishra4	3.4063000	0.7679615	6.7003235	1.0011406
Mishra5	4.6532750	2.0056790	8.1130435	1.0006899
Mishra6	1.8588450	-0.4805098	5.2780820	0.9998099
Mishra7N10	2.5332750	-0.5136623	6.3972250	1.0012098
Mishra7N2	2.7049500	0.2494388	5.9691710	1.0005868
Mishra7N6	-0.3233530	-2.2074485	2.9265300	1.0008168
Mishra8	0.6220735	-1.5257385	3.8955725	1.0000708
Mishra9	-1.0008050	-2.4989410	1.7691920	1.0008537
Parsopoulos	3.7499200	1.2188785	7.0250630	1.0020421
PathologicalN10	-0.8406665	-3.1107665	3.2093270	1.0002854
PathologicalN2	2.4734050	0.2258771	5.5001195	1.0013115
PathologicalN6	-3.0490600	-4.0924575	-0.8955173	1.0015707
Paviani	-2.5438100	-3.6046420	-0.6316878	1.0009903
PenHolder	7.7711000	4.9203125	11.3604750	1.0026386
Periodic	2.2447900	-0.0479479	5.3980715	0.9996591
PinterN10	-3.8875100	-4.5127210	-3.0109260	1.0015404
PinterN2	-1.1604500	-2.5991070	1.3618235	1.0000632
PinterN6	-3.9602500	-4.5749230	-3.1298165	1.0019213
PowellSingular	-3.0041650	-3.7996050	-1.7094280	1.0005540
PowellSumN10	-2.9091000	-3.7440405	-1.5819790	1.0012464
PowellSumN2	2.1862100	-0.1694761	5.5110815	1.0004236
PowellSumN6	-2.9642200	-3.7754825	-1.7330110	1.0007085
Price1	4.3846150	1.7633140	7.7391205	1.0004353
Price2	2.1912000	-0.1186187	5.5889560	1.0008193
Price3	0.9268785	-1.2144450	4.0564150	1.0037913
Price4	-4.0028150	-4.5724020	-3.2377525	1.0006602
QingN10	-3.6703300	-4.4726565	-2.1780695	1.0009871
QingN2	-2.8100800	-3.8219330	-0.8467380	1.0013319
QingN6	-3.7503350	-4.5015255	-2.3423925	1.0004460
Quadratic	-1.7157800	-3.0312605	0.6069797	1.0017112
QuarticN10	0.3796440	-1.4596770	3.1480145	0.9997688
QuarticN2	6.2414600	3.5334825	9.4896895	1.0013008
QuarticN6	5.9194700	3.2752390	9.2501460	1.0004652
QuinticN10	-3.8069400	-4.4877865	-2.6803210	1.0016455
QuinticN2	-2.5901300	-3.5894925	-0.7517053	1.0020636
QuinticN6	-3.3700650	-4.1772930	-1.9647425	1.0007116
RastriginN10	-3.1156500	-4.1108450	-1.0254090	1.0021547
RastriginN2	2.2945450	-0.1638884	5.6497585	1.0015140
RastriginN6	-3.3804000	-4.2663055	-1.8207545	1.0001810
RosenbrockModified	2.5544550	-0.4822589	6.4588855	0.9997108
RosenbrockN10	-3.5859900	-4.4330625	-1.9086795	1.0023106
RosenbrockN2	0.0933755	-1.8850385	3.2216290	1.0006722
RosenbrockN6	-3.6476100	-4.4538300	-1.9993475	1.0011420
RotatedEllipse	-3.2192150	-4.0165775	-1.9701625	1.0015411
RotatedEllipse2	-3.2106250	-4.0017205	-2.0323120	0.9996703
SalomonN10	-2.9235650	-3.8996720	-1.1134535	0.9996843

SalomonN2	-3.8526750	-4.4602240	-3.0306700	1.0015248
SalomonN6	-3.6709550	-4.3364075	-2.7380745	1.0015242
SarganN10	-0.8463235	-3.0762655	3.2361400	1.0024069
SarganN2	0.8264030	-1.8685540	4.8428395	1.0053108
SarganN6	-0.9553085	-3.1764155	3.1227600	1.0002377
Scahffer1	1.2977150	-0.9141648	4.3687840	1.0005565
Scahffer2	0.9285580	-1.1360545	3.9467475	1.0006106
Scahffer3	1.2139900	-1.0573090	4.6270855	1.0005580
Scahffer4	2.7839950	0.2443601	6.1655460	1.0008137
SchafferF6	1.6592800	-0.5189149	4.9347595	1.0005012
SchumerSteiglitzN10	-2.7684650	-3.6304880	-1.4229775	1.0012848
SchumerSteiglitzN2	3.1229950	0.7166843	6.3656570	1.0003950
SchumerSteiglitzN6	-2.8892300	-3.7322020	-1.6370325	0.9998312
Schwefel1d2N10	-3.4236650	-4.2568050	-2.0215835	1.0021204
Schwefel1d2N2	-2.8783500	-3.7756520	-1.3744305	1.0008239
Schwefel1d2N6	-3.6542450	-4.3208040	-2.6560875	1.0009123
Schwefel2d20N10	-3.8932300	-4.5147110	-3.0440715	1.0004775
Schwefel2d20N2	-3.8152850	-4.4361895	-2.9358735	1.0004400
Schwefel2d20N6	-3.9659200	-4.5742365	-3.1253060	1.0023603
Schwefel2d21N10	-3.4849950	-4.2869120	-2.1688180	1.0005729
Schwefel2d21N2	-3.7234800	-4.3751025	-2.8189545	1.0025926
Schwefel2d21N6	-3.8604100	-4.4735350	-3.0501530	1.0017970
Schwefel2d22N10	-3.8595400	-4.5126225	-2.9869900	1.0010510
Schwefel2d22N2	-3.8176450	-4.4366645	-2.9899990	1.0028834
Schwefel2d22N6	-3.9665850	-4.5803195	-3.1008645	1.0024314
Schwefel2d23N10	-3.7448550	-4.3879290	-2.9052975	1.0006555
Schwefel2d23N2	1.0053850	-1.0330825	4.2719140	0.9999153
Schwefel2d23N6	-3.8287850	-4.4427555	-3.0312445	1.0013369
Schwefel2d26N10	-2.8748200	-4.1661375	0.4272453	0.9999800
Schwefel2d26N2	-3.5956100	-4.3764970	-2.1322265	0.9995161
Schwefel2d26N6	-3.3626700	-4.3677050	-0.9244707	1.0004991
Schwefel2d36	-3.9744700	-4.6147340	-2.8949510	1.0014899
Schwefel2d4N10	-1.9149950	-3.2296320	0.7061617	1.0023469
Schwefel2d4N2	3.3408200	0.7356389	6.8435010	1.0003148
Schwefel2d4N6	-2.2517800	-3.4771665	0.0774009	1.0041209
Schwefel2d6	1.0469450	-1.1195145	4.2536635	1.0012298
SchwefelN10	-3.7648800	-4.3779450	-2.8760595	1.0015958
SchwefelN2	-0.2370720	-2.0278795	2.6415805	1.0007791
SchwefelN6	-3.8171100	-4.4558910	-3.0060820	1.0002365
Shekel10	-3.3034600	-4.1238385	-1.9145610	0.9999434
Shekel5	-3.4063050	-4.2378330	-1.9324680	1.0000936
Shekel7	-3.7130700	-4.3694015	-2.7247930	1.0008754
Shubert	-3.4647650	-4.2339070	-2.3192630	0.9997792
SixHumpCamelBack	3.3419000	0.9075510	6.7132880	1.0032270
SphereN10	-2.7734650	-3.6286465	-1.4063510	1.0005422
SphereN2	2.0379700	-0.3242754	5.2533805	1.0014259

SphereN6	-2.9057650	-3.7308920	-1.5349255	1.0011882
StretchedVSineWave2N	-4.0386050	-4.6465300	-3.2861850	1.0016642
StyblinskiTang	2.7690800	0.2525298	6.2420890	1.0018946
SumSquaresN10	-2.6872750	-3.6492665	-1.1026935	1.0025429
SumSquaresN2	2.8416850	0.4455271	6.0278370	1.0010911
SumSquaresN6	-2.9079950	-3.7261630	-1.6647480	1.0006220
Table1	2.4193700	0.0168489	5.7359985	0.9999313
Table2	3.1379550	0.6098554	6.6026040	1.0002655
Table3	2.4709350	0.1118382	5.7687045	0.9999987
TesttubeHolder	3.5556400	1.0738370	6.8751395	1.0011444
ThreeHumpCamelBack	3.1408250	0.7668906	6.3076485	1.0007569
Trefethen	-3.9488200	-4.6204550	-2.9420115	1.0040579
Trigonometric1N10	6.1370700	3.3464585	9.5289995	1.0030787
Trigonometric1N2	5.8601450	3.0267480	9.2971805	1.0011339
Trigonometric1N6	6.8148900	3.9941360	10.0905550	1.0047042
Trigonometric2N10	-3.9807100	-4.6286590	-3.0423675	1.0000970
Trigonometric2N2	-4.0911100	-4.6686020	-3.3141830	1.0006784
Trigonometric2N6	-4.1060700	-4.7051220	-3.3056195	1.0014867
Tripod	-4.1079600	-4.7346030	-3.1960955	1.0009720
Ursem1	4.9413950	2.1955875	8.4644795	1.0038894
Ursem3	1.5587900	-0.5696218	4.8874710	0.9998760
Ursem4	3.7175200	1.1675350	7.0155715	1.0005738
UrsemWaves	2.0894600	-0.2292360	5.3087025	1.0011891
Watson	-3.8091650	-4.4191085	-2.9410325	1.0003699
WayburnSeader1	-2.4753650	-3.5833545	-0.3106643	1.0007651
WayburnSeader2	3.2098900	0.7191442	6.5032215	1.0004888
WayburnSeader3	4.2973900	1.7102965	7.3288760	1.0030215
WeierstrassN2	2.0008500	-0.3158997	5.1997320	1.0027254
VenterSobieskiSobieski	-3.7944400	-4.4306930	-2.9355360	1.0002356
WhitleyN10	-0.2712425	-3.0076515	3.9383365	1.0002063
WhitleyN2	-4.1429900	-4.7498810	-3.3374850	1.0013813
WhitleyN6	1.6319600	-1.7196510	5.9720115	1.0010817
Wolfe	2.5879800	0.1715528	5.8737700	0.9995668
WWavyN10	-3.6586800	-4.3864650	-2.5113545	1.0021806
WWavyN2	2.5121300	0.1640546	5.7334250	1.0020069
WWavyN6	-3.8176100	-4.4734090	-2.8011830	1.0027115
XinSheYang1N10	-3.2160050	-4.0912070	-1.6910535	1.0010632
XinSheYang1N2	-0.0596483	-1.8839490	2.8907505	1.0024776
XinSheYang1N6	-3.8003450	-4.4438765	-2.8997965	1.0020150
XinSheYang2N10	8.4600800	5.7621000	11.6131300	1.0011374
XinSheYang2N2	2.6789750	0.3079866	5.9031965	1.0007781
XinSheYang2N6	4.8715950	2.3123975	8.1979900	1.0027469
XinSheYang3N10	-0.8241220	-3.1066915	3.2932130	0.9997833
XinSheYang3N2	4.1096400	1.2162630	7.5377440	0.9993408
XinSheYang3N6	-0.9534960	-3.1568575	3.1500685	1.0000266
ZakharovN10	-2.1667650	-3.3540090	0.0182611	1.0027835

ZakharovN2	2.9209000	0.5054720	6.1046025	1.0012090
ZakharovN6	-2.9166500	-3.7342885	-1.5516115	1.0008493
Zettl	2.8489800	0.4412244	6.0755485	1.0007313
Zirilli	2.1507200	-0.2446074	5.5171165	1.0014999

Table F.2: Summary values of the discrimination and difficulty level parameters for the benchmarks of research question one part A

Benchmark ID	Median	CI 5%	CI 95%	R-hat
Discrimination value (a)				
Alpine1N10	7.0022700	4.7051540	9.7843535	1.0014663
BartelsConn	6.3515200	4.4232890	8.6456550	1.0010761
Bohachevsky1	5.8018200	3.9846175	7.9968910	1.0012746
Bohachevsky2	6.4053000	4.4139145	8.6515790	1.0009396
CosineMixtureN10	6.9065050	4.6873985	9.6654030	1.0001622
ElAttarVidyasagarDutta	6.8585050	4.5925410	9.6220035	1.0009678
EllipsoidalN2	5.6942300	3.7425830	8.1163190	1.0002168
GriewankN10	6.8916450	4.6777510	9.6453990	1.0009250
GriewankN6	6.9918450	4.7094690	9.8293090	1.0013238
PinterN10	7.0058150	4.6863240	9.6446970	1.0018423
PinterN6	7.0025850	4.7997040	9.6678720	1.0017750
Price4	6.3388450	4.4319150	8.7101740	1.0009971
QingN10	6.5717250	3.7975335	10.0667650	1.0001283
QingN6	6.5962000	3.7896125	9.9629130	0.9997511
QuinticN10	6.8599650	4.3226130	9.9302105	1.0010339
SalomonN2	5.7990100	4.0041000	7.8200585	1.0008714
Schwefel2d20N10	7.0255050	4.7730690	9.6958660	0.9997848
Schwefel2d20N6	7.0016900	4.7345525	9.7760075	1.0055833
Schwefel2d21N6	6.5159000	4.5233845	8.9316750	1.0027973
Schwefel2d22N10	6.9206850	4.6501265	9.6375865	1.0008690
Schwefel2d22N6	7.0075000	4.6755340	9.6122900	1.0040274
Schwefel2d36	-3.9744700	-4.6147340	-2.8949510	1.0014899
StretchedVSineWave2N	6.5564300	4.5574915	9.1268065	1.0030107
Trefethen	6.5227200	4.0821490	9.5418695	1.0018539
Trigonometric2N10	7.6924650	4.8800660	11.0568800	1.0005485
Trigonometric2N2	6.8680050	4.6705750	9.5033525	1.0001986
Trigonometric2N6	7.8680400	5.1920205	11.0806400	1.0003244
Tripod	7.2020100	4.5464755	10.4219250	0.9998404
WhitleyN2	7.2994650	4.8171090	10.3021250	1.0017540
WWavyN6	6.5636650	4.2639595	9.1925985	1.0022136
Difficulty level (b)				
Alpine1N10	-3.8937150	-4.5102635	-3.0140085	1.0017343
BartelsConn	-4.0096500	-4.5879475	-3.2289550	1.0014304
Bohachevsky1	-3.8606400	-4.4912995	-2.9951425	1.0011280
Bohachevsky2	-4.0095400	-4.5998915	-3.2439695	1.0011131

CosineMixtureN10	-3.8678850	-4.5007580	-2.9764895	1.0034505
ElAttarVidyasagarDutta	-4.0853700	-4.6956955	-3.2723505	1.0009787
EllipsoidalN2	-3.7742800	-4.4685785	-2.7742385	1.0004686
GriewankN2	1.7601950	-0.4608704	4.8060855	1.0003172
GriewankN6	-3.9642050	-4.5814630	-3.1007005	1.0017530
PinterN10	-3.8875100	-4.5127210	-3.0109260	1.0015404
PinterN6	-3.9602500	-4.5749230	-3.1298165	1.0019213
Price4	-4.0028150	-4.5724020	-3.2377525	1.0006602
QingN10	-3.6703300	-4.4726565	-2.1780695	1.0009871
QingN6	-3.7503350	-4.5015255	-2.3423925	1.0004460
QuinticN10	-3.8069400	-4.4877865	-2.6803210	1.0016455
SalomonN2	-3.8526750	-4.4602240	-3.0306700	1.0015248
Schwefel2d20N10	-3.8932300	-4.5147110	-3.0440715	1.0004775
Schwefel2d20N6	-3.9659200	-4.5742365	-3.1253060	1.0023603
Schwefel2d21N6	-3.8604100	-4.4735350	-3.0501530	1.0017970
Schwefel2d22N10	-3.8595400	-4.5126225	-2.9869900	1.0010510
Schwefel2d22N6	-3.9665850	-4.5803195	-3.1008645	1.0024314
Schwefel2d36	-3.9744700	-4.6147340	-2.8949510	1.0014899
StretchedVSineWave2N	-4.0386050	-4.6465300	-3.2861850	1.0016642
Trefethen	-3.9488200	-4.6204550	-2.9420115	1.0040579
Trigonometric2N10	-3.9807100	-4.6286590	-3.0423675	1.0000970
Trigonometric2N2	-4.0911100	-4.6686020	-3.3141830	1.0006784
Trigonometric2N6	-4.1060700	-4.7051220	-3.3056195	1.0014867
Tripod	-4.1079600	-4.7346030	-3.1960955	1.0009720
WhitleyN2	-4.1429900	-4.7498810	-3.3374850	1.0013813
WWavyN6	-3.8176100	-4.4734090	-2.8011830	1.0027115

Table F.3: Summary values of the discrimination and difficulty level parameters for the benchmarksof research question one part B

Benchmark ID	Median	CI 5%	CI 95%	R-hat
Discrimination value (a)				
AttractiveSectorN2	2.4923900	1.4625210	3.8947150	1.0014529
Bunkin6	2.1120000	1.2331155	3.8589110	0.9998983
Corana	5.8269800	3.8917385	8.0884145	1.0000543
DixonPriceN10	4.8858100	2.3309835	8.5692830	1.0011993
EggHolder	2.5964700	1.3776150	4.6903730	1.0016848
FreudensteinRoth	2.2251450	1.3941440	3.3546700	1.0021779
Hartman6	3.2672950	1.9275435	5.0345405	1.0007843
JennrichSampson	3.1347650	1.9219310	4.5761740	0.9999837
Keane	1.0956700	0.8060226	1.5058270	1.0008564
Langerman	1.1984500	0.8621298	1.6902675	1.0005538
LinearSlopeN10	1.0718650	0.8237401	1.4212930	0.9996885
LinearSlopeN6	1.1067300	0.8300255	1.4862610	1.0006990
Mishra10a	0.9704550	0.7346704	1.2987810	1.0046947

Mishra10b	1.0288700	0.7743836	1.3894175	1.0010237
Mishra11N10	0.8663950	0.7013314	1.0934850	1.0013067
Mishra1N10	2.0342950	1.3674760	2.9624970	1.0013288
Mishra1N6	1.4760950	1.0214005	2.1560185	1.0040044
Mishra2N10	1.9973550	1.3399370	2.9451305	1.0019231
Mishra2N6	2.0353900	1.3672490	2.9452370	1.0007167
Mishra3	2.4928750	1.3873265	4.2185870	0.9996129
Mishra7N6	2.1794600	1.3583690	3.3330370	1.0008275
Mishra9	2.3070150	1.4562155	3.3740365	1.0005510
QingN2	3.7661450	2.2185920	5.6265690	1.0012560
QuinticN6	5.1912250	3.2378230	7.5526435	1.0007647
SarganN10	2.8388300	1.5453115	5.2596490	1.0015354
SarganN6	2.7965450	1.5071090	5.1547485	0.9998848
Schwefel2d4N10	3.3080700	1.9490100	5.0281675	1.0005620
XinSheYang3N10	2.8162700	1.5183695	5.3563020	1.0004028
XinSheYang3N6	2.7917650	1.4946325	5.2019470	0.9999838
ZakharovN10	3.4879750	2.1675070	5.1720800	1.0028192
Difficulty level (b)				
AttractiveSectorN2	-1.5820750	-3.0888640	1.3772555	1.0017538
Bunkin6	0.0227620	-2.5704025	4.1619400	0.9996487
Corana	-3.7070700	-4.3814950	-2.7160310	1.0007602
DixonPriceN10	-2.8557400	-4.1058650	0.2584619	1.0018308
EggHolder	-1.1740650	-3.1560930	2.9193635	1.0021599
FreudensteinRoth	-0.9723545	-2.6090920	1.8494720	1.0026873
Hartman6	-2.0093450	-3.3394725	0.6446408	1.0002112
JennrichSampson	-2.3440450	-3.4398625	-0.2266343	1.0006240
Keane	3.4205350	0.9138593	6.7862965	1.0008474
Langerman	2.7374550	0.3107510	6.0559180	1.0002018
LinearSlopeN10	5.0626550	2.4225810	8.3561755	0.9996477
LinearSlopeN6	4.2381050	1.6728305	7.5443890	1.0004218
Mishra10a	4.4173000	1.8508480	7.6773035	1.0027686
Mishra10b	3.8418150	1.3176745	6.9730000	1.0011307
Mishra11N10	7.4336700	4.6746965	10.5617500	1.0013225
Mishra1N10	0.1651635	-1.6970650	3.0571890	1.0008056
Mishra1N6	1.9431300	-0.4744529	5.3424460	1.0036190
Mishra2N10	0.2815795	-1.6683125	3.2686900	1.0041369
Mishra2N6	-0.0897478	-1.8189300	2.7001110	1.0002149
Mishra3	-1.1035800	-2.9951760	2.4444030	0.9998506
Mishra7N6	-0.3233530	-2.2074485	2.9265300	1.0008168
Mishra9	-1.0008050	-2.4989410	1.7691920	1.0008537
QingN2	-2.8100800	-3.8219330	-0.8467380	1.0013319
QuinticN6	-3.3700650	-4.1772930	-1.9647425	1.0007116
SarganN10	-0.8463235	-3.0762655	3.2361400	1.0024069
SarganN6	-0.9553085	-3.1764155	3.1227600	1.0002377
Schwefel2d4N10	-1.9149950	-3.2296320	0.7061617	1.0023469
XinSheYang3N10	-0.8241220	-3.1066915	3.2932130	0.9997833

XinSheYang3N6	-0.9534960	-3.1568575	3.1500685	1.0000266
ZakharovN10	-2.1667650	-3.3540090	0.0182611	1.0027835

Table F.4: Summary values of the ability level of the algorithms

Algorithm	Median	CI 5%	CI 95%	R-hat
ABC	-5.554585	-5.948459	-5.164627	1.003999
Bat	-5.851360	-6.250458	-5.450210	1.004240
CuckooSearch	-6.098155	-6.500318	-5.695034	1.004339
DifferentialEvolution	-5.568640	-5.962815	-5.180652	1.003974
FireflyAlgorithm	-5.579595	-5.977728	-5.191306	1.004018
GeneticAlgorithm	-6.009775	-6.416409	-5.609196	1.004181
GWO	-5.642815	-6.037952	-5.251250	1.004066
NelderMead	-7.566780	-8.033384	-7.115275	1.004145
PSO	-5.648905	-6.044206	-5.254094	1.004000
RandomSearch	-6.020590	-6.426706	-5.618530	1.003953
SimulatedAnnealing	-6.145175	-6.555773	-5.743309	1.004240

Table F.5: Summary values of the results from the model from research question 2

Variable	mean	2.5%	97.5%	Rhat
a_alg[1]	-2.1714644	-2.7847606	-1.5303593	1.0002514
a_alg[2]	-0.7049285	-1.2976799	-0.0766559	1.0002734
a_alg[3]	-2.2932463	-2.8985939	-1.6547448	1.0002149
a_alg[4]	-4.2399781	-4.9385511	-3.5242094	1.0002004
a_alg[5]	-4.2402706	-4.9386117	-3.5194395	1.0001854
a_alg[6]	-4.2405967	-4.9374984	-3.5227482	1.0001466
a_alg[7]	-4.2414984	-4.9402407	-3.5172438	1.0001751
a_alg[8]	-2.6559204	-3.2867834	-1.9953685	1.0002139
a_alg[9]	0.3994076	-0.1782622	1.0182152	1.0002476
a_alg[10]	-2.9816481	-3.5998814	-2.3309262	1.0002266
a_alg[11]	-2.9643242	-3.5867573	-2.3148403	1.0002500
a_alg[12]	-2.3462999	-2.9483514	-1.7111823	1.0002119
a_alg[13]	-4.2401543	-4.9427546	-3.5215045	1.0001642
b_dim[1]	-0.0439025	-0.0612245	-0.0271424	0.9999722
b_dim[2]	-0.1313819	-0.1531415	-0.1109425	0.9999719
b_dim[3]	-0.0073741	-0.0201771	0.0049901	0.9999925
b_dim[4]	-0.0006533	-0.0221313	0.0202357	1.0000056
b_dim[5]	-0.0006579	-0.0223356	0.0203223	0.9999773
b_dim[6]	-0.0006258	-0.0221659	0.0204503	0.9999952
b_dim[7]	-0.0006175	-0.0225109	0.0204366	0.9999888
b_dim[8]	-0.0555736	-0.0766976	-0.0351067	0.9999734

b_dim[9]	-0.0925823	-0.1052351	-0.0804066	1.0000003
b_dim[10]	0.0302382	0.0188763	0.0413524	0.9999728
b_dim[11]	0.0297074	0.0183993	0.0408598	0.9999843
b_dim[12]	0.0188694	0.0087895	0.0289008	0.9999772
b_dim[13]	-0.0006572	-0.0226800	0.0204143	0.9999924
a_bm_norm[1]	0.2701895	-0.1649162	0.7091463	1.0002620
a_bm_norm[2]	-0.3625723	-0.8051825	0.0699652	1.0002373
a_bm_norm[3]	0.2937729	-0.1421272	0.7366962	1.0002406
a_bm_norm[4]	0.0925376	-0.3342260	0.5210693	1.0002659
a_bm_norm[5]	3.5691692	2.4188363	4.7915445	1.0001850
a_bm_norm[6]	-0.2450768	-0.6794540	0.1833016	1.0002549
a_bm_norm[7]	0.0743163	-0.3532788	0.5031219	1.0002418
a_bm_norm[8]	-0.3349834	-0.7747562	0.0950999	1.0001881
a_bm_norm[9]	0.6951840	0.2170949	1.1823282	1.0002493
a_bm_norm[10]	-0.3486431	-0.7949278	0.0845198	1.0001957
a_bm_norm[11]	-0.3910359	-0.8418620	0.0443095	1.0001875
a_bm_norm[12]	-0.5604362	-1.0317118	-0.1109872	1.0002432
a_bm_norm[13]	-0.9511850	-1.4949711	-0.4447014	1.0001703
a_bm_norm[14]	-0.9513650	-1.4960162	-0.4445405	1.0001544
a_bm_norm[15]	-0.0949843	-0.5233277	0.3303401	1.0002525
a_bm_norm[16]	-0.9010468	-1.4361863	-0.4071398	1.0001852
a_bm_norm[17]	-0.9257615	-1.4611040	-0.4242712	1.0002157
a_bm_norm[18]	-0.9514426	-1.4915199	-0.4481845	1.0001602
a_bm_norm[19]	0.6898056	0.2113899	1.1779708	1.0002753
a_bm_norm[20]	-0.1395732	-0.5697820	0.2868290	1.0002816
a_bm_norm[21]	0.6191629	0.1498431	1.0964535	1.0002317
a_bm_norm[22]	0.5573738	0.0980023	1.0253401	1.0002387
a_bm_norm[23]	-0.9508240	-1.4928490	-0.4486487	1.0001469
a_bm_norm[24]	-0.8539806	-1.3738174	-0.3664801	1.0001860
s	1.4454275	1.0740539	1.9843713	1.0001677
lp_	-5705.2831708	-5718.0452580	-5694.3561975	1.0000506

G

Appendix 7: RQ2 trace plots

The trace plots for each of the benchmark functions are presented here. The number represents each benchmark function and their specific names can be found in Appendix A, in section A.3.

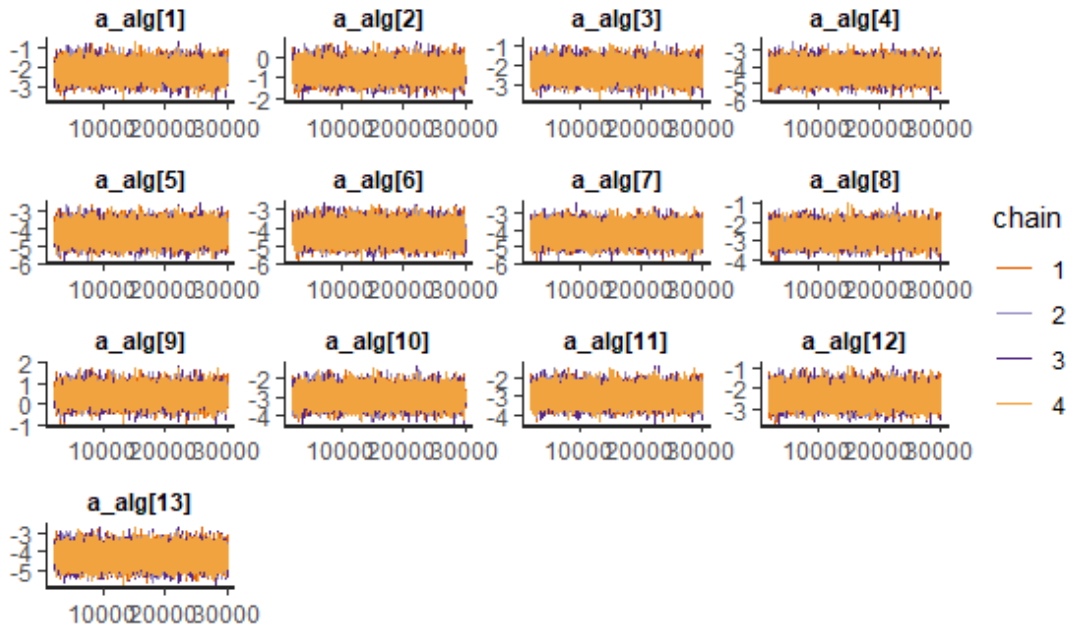


Figure G.1: Trace plot showing the chain convergence for the algorithms

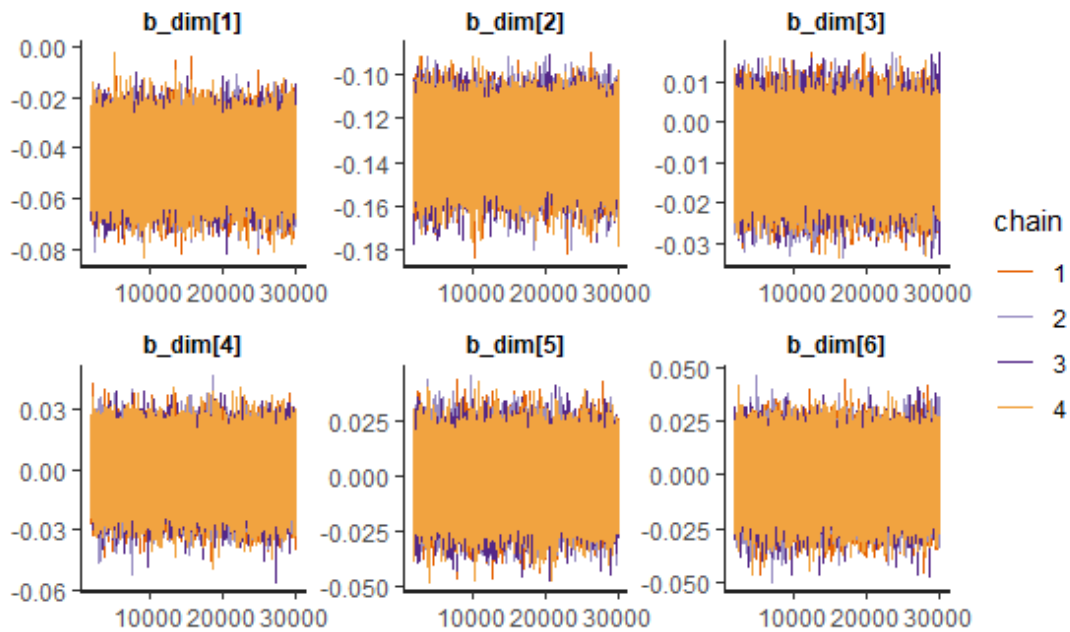


Figure G.2: Trace plot showing the chain convergence for the dimensions

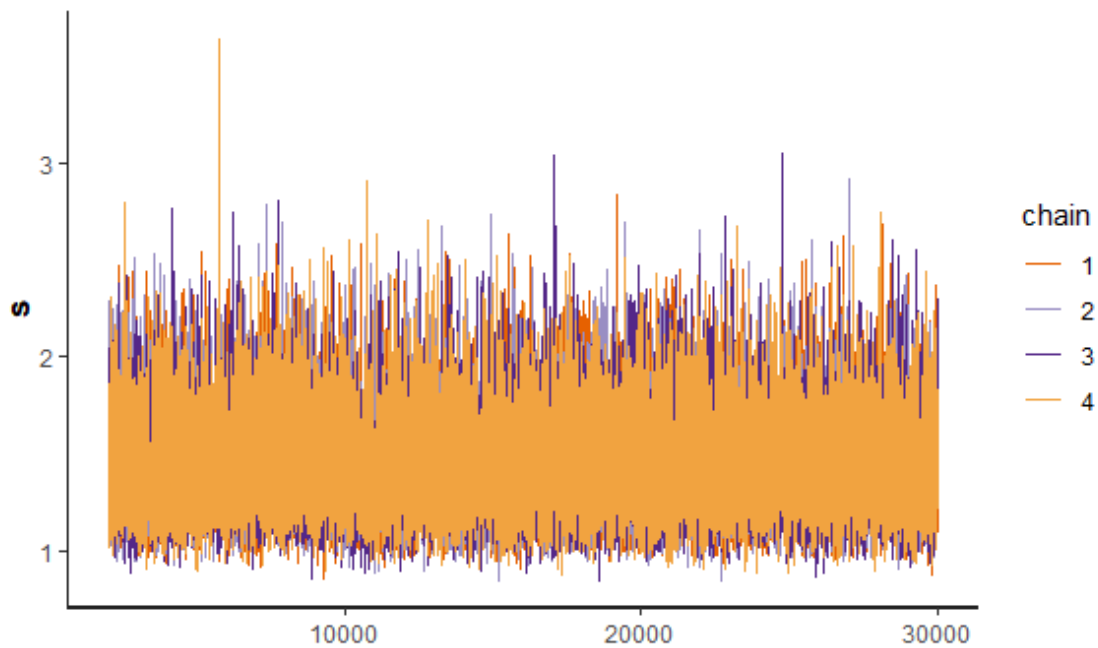


Figure G.3: Trace plot showing the chain convergence for s

H

Appendix 8: Clustering of benchmark functions

Table H.1: Clustering of the benchmark functions

Easy	Medium	Hard	Very Hard
Ackley1N10	AttractiveSectorN2	Adjiman	CsendesN10
Ackley1N2	BiggsEXP4	Alpine1N2	CsendesN6
Ackley1N6	BiggsEXP6	Beale	ExponentialN10
Ackley2	Bird	BiggsEXP2	ExponentialN6
Ackley3	Brent	BiggsEXP3	Mishra11N6
Alpine1N10	Bunkin6	BiggsEXP5	PenHolder
Alpine1N6	ChenBird	Booth	XinSheYang2N10
Alpine2	ChungReynoldsN2	BoxBettsQuadraticSum	
BartelsConn	Cube	BraninRCOS	
Bohachevsky1	Damavandi	BraninRCOS2	
Bohachevsky2	DeckkersAarts	BrownN10	
Bohachevsky3		BrownN2	
BucheRastriginN10	DeVilliersGlasser01	BrownN6	
BucheRastriginN2	DeVilliersGlasser02	Bunkin2	
BucheRastriginN6	EggCrate	Bunkin4	
ChenV	EggHolder	CosineMixtureN2	
Chichidze	FreudensteinRoth	CrossInTray	
ChungReynoldsN10	GoldsteinPrice	CsendesN2	
ChungReynoldsN6	GriewankN2	DixonPriceN2	
Colville	Hartman6	Dolan	
Cora	JennrichSampson	Exp2	
CosineMixtureN10	Mishra1N10	ExponentialN2	
CosineMixtureN6	Mishra1N6	Giunta	
DixonPriceN10	Mishra2N10	Hartman3	
DixonPriceN6	Mishra2N6	Himmelblau	
Easom	Mishra3	Hosaki	
ElAttarVidyasagarDutta	Mishra6	Keane	
EllipsoidalN10	Mishra7N6	Langerman	
EllipsoidalN2	Mishra8	Leon	

H. Appendix 8: Clustering of benchmark functions

EllipsoidalN6	Mishra9	LinearSlopeN10
GriewankN10	PathologicalN10	LinearSlopeN2
GriewankN6	PathologicalN2	LinearSlopeN6
Hansen	Periodic	Matyas
HelicalValley	PinterN2	McCormick
PathologicalN6	PowellSumN2	Miele
Paviani	Price2	Mishra10a
PinterN10	Price3	Mishra10b
PinterN6	Quadratic	Mishra11N10
PowellSingular	QuarticN10	Mishra11N2
PowellSumN10	RastriginN2	Mishra1N2
PowellSumN6	RosenbrockN2	Mishra2N2
Price4	SarganN10	Mishra4
QingN10	SarganN2	Mishra5
QingN2	SarganN6	Mishra7N10
QingN6	Scahffer1	Mishra7N2
QuinticN10	Scahffer2	Parsopoulos
QuinticN2	Scahffer3	Price1
QuinticN6	SchafferF6	QuarticN2
RastriginN10	Schwefel2d23N2	QuarticN6
RastriginN6	Schwefel2d4N10	RosenbrockModified
RosenbrockN10	Schwefel2d4N6	Scahffer4
RosenbrockN6	Schwefel2d6	SchumerSteiglitzN2
RotatedEllipse	SchwefelN2	Schwefel2d4N2
RotatedEllipse2	SphereN2	SixHumpCamelBack
SalomonN10	Table1	StyblinskiTang
SalomonN2	Table3	SumSquaresN2
SalomonN6	Ursem3	Table2
SchumerSteiglitzN10	UrsemWaves	TesttubeHolder
SchumerSteiglitzN6	WayburnSeader1	ThreeHumpCamelBack
Schwefel1d2N10	WeierstrassN2	Trigonometric1N10
Schwefel1d2N2	WhitleyN10	Trigonometric1N2
Schwefel1d2N6	WhitleyN6	Trigonometric1N6
Schwefel2d20N10	XinSheYang1N2	Ursem1
Schwefel2d20N2	XinSheYang3N10	Ursem4
Schwefel2d20N6	XinSheYang3N6	WayburnSeader2
Schwefel2d21N10	ZakharovN10	WayburnSeader3
Schwefel2d21N2	Zirilli	Wolfe
Schwefel2d21N6		WWavyN2
Schwefel2d22N10		XinSheYang2N2
Schwefel2d22N2		XinSheYang2N6
Schwefel2d22N6		XinSheYang3N2
Schwefel2d23N10		ZakharovN2
Schwefel2d23N6		Zettl

Schwefel2d26N10
Schwefel2d26N2
Schwefel2d26N6
Schwefel2d36
SchwefelN10
SchwefelN6
Shekel10
Shekel5
Shekel7
Shubert
SphereN10
SphereN6
StretchedVSineWave2N
SumSquaresN10
SumSquaresN6
Trefethen
Trigonometric2N10
Trigonometric2N2
Trigonometric2N6
Tripod
Watson
VenterSobieszcanskiSobieski
WhitleyN2
WWavyN10
WWavyN6
XinSheYang1N10
XinSheYang1N6
ZakharovN6
