GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Uploaddatum: 31.07.2020
Uploadzeit: 15:32

Dies ist ein von FlexNow automatisch beim Upload generiertes Deckblatt. Es dient dazu, die Arbeit automatisiert der Prüfungsakte zuordnen zu können.

This is a machine generated frontpage added by FlexNow. Its purpose is to link your upload to your examination file.

Matrikelnummer: 11404043

ISSN 1612-6793

Master's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Deployment of Sensitivity Analysis on Plant Models in GroIMP Utilizing R

Lukas Gürtler

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

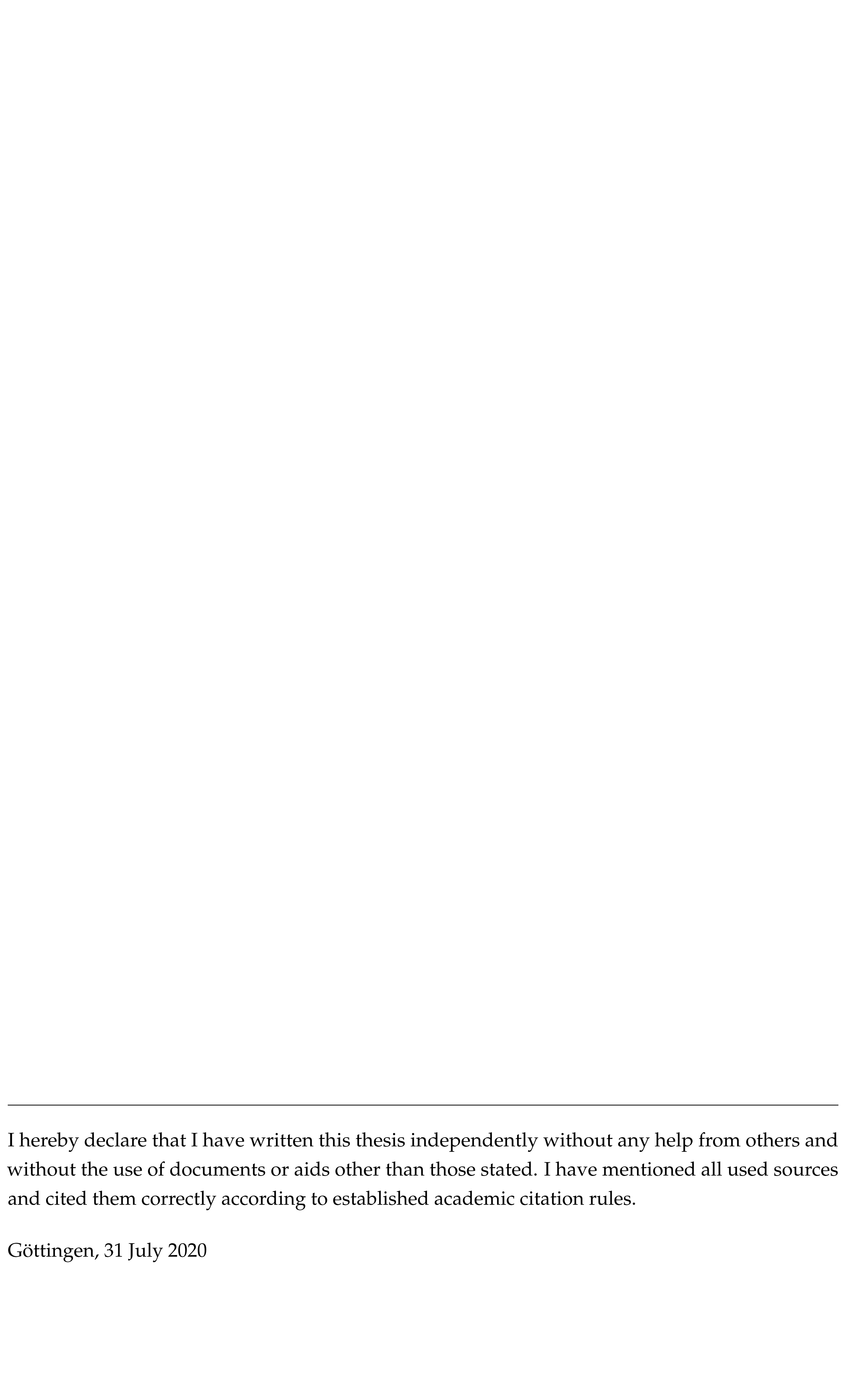
31 July 2020

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Winfried Kurth
Second Supervisor: Prof. Dr. Stephan Waack
Advisor: Aleksi Tavkhelidze
Advisor: Ernesto Rubio



I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 31 July 2020

Abstract

Sensitivity analysis is a very important domain of computer plant models. This thesis deals with the development and test of a plugin for the plant-modeling software GroIMP. The plugin supplements GroIMP with the functionality of systematic sensitivity analysis of plant models by utilizing the statistical computing software R. It is described how to connect the R language to the Java-based software GroIMP. Moreover, the complete plugin structure is elaborated. Seven sensitivity analysis approaches are implemented (local sensitivity analysis, Morris's elementary effects screening, main and interaction effects on extreme values, partial (rank) correlation coefficients, standardized (rank) regression coefficients, Sobol's method, extended Fourier amplitude sensitivity test). The mathematical base of each method is explained. The plugin is tested with a beech tree and an assimilate production model considering various input parameters and outputs. Finally the runtime, the memory consumption, the information quality, the runtime efficiency and the memory efficiency are analyzed.

Contents

1 Introduction **11**

2 Implementation Details **15**
2.1 Connecting R to Java 15
2.1.1 Class "RConnection" 16
2.1.1.1 Fields and Properties 16
2.1.1.2 Functions 17
2.1.1.3 Constructor 19
2.2 Simulation Identification 19
2.2.1 Interface "Simulation" 19
2.2.1.1 Functions 19
2.3 Passing Model Parameters 20
2.3.1 Class "NumberRef" 20
2.3.1.1 Fields and Properties 20
2.3.1.2 Functions 21
2.3.1.3 Constructors 21
2.4 Preparing a Plant Model for Sensitivity Analysis 22
2.4.1 Preparation Steps 22
2.4.2 Plant Model Preparation Example 24
2.5 Sensitivity Analysis Methods 27
2.5.1 Call Arguments 27
2.5.2 Class "Sensitivity" 28
2.5.2.1 Functions 28
2.5.2.2 Using R for Sensitivity Analysis 29
2.6 Plugin Structure 31

3 Mathematical Foundation **32**
3.1 Local Sensitivity Analysis 32
3.2 Morris's Elementary Effects Screening 34

3.3 Main And Interaction Effects On Extreme Values 37
3.4 Partial (Rank) Correlation Coefficients (PCC/PRCC) and Standardized (Rank) Regression Coefficients (SRC/SRRC) 40
3.5 Sobol's Method 45
3.6 Extended Fourier Amplitude Sensitivity Test (EFAST) 47

4 Sensitivity Analysis of Plant Models 50
4.1 Beech Tree 50
4.1.1 Local Sensitivity Analysis 53
4.1.2 Morris's Elementary Effects Screening 54
4.1.3 Main And Interaction Effects On Extreme Values 56
4.1.4 Partial (Rank) Correlation Coefficients 61
4.1.5 Standardized (Rank) Regression Coefficients 64
4.1.6 Sobol's Method 67
4.1.7 Extended Fourier Amplitude Sensitivity Test 70
4.2 Assimilate Production Model 72
4.2.1 Local Sensitivity Analysis 73
4.2.2 Morris's Elementary Effects Screening 74
4.2.3 Main And Interaction Effects On Extreme Values 76
4.2.4 Partial (Rank) Correlation Coefficients 79
4.2.5 Standardized (Rank) Regression Coefficients 81
4.2.6 Sobol's Method 83
4.2.7 Extended Fourier Amplitude Sensitivity Test 85

5 Runtime and Memory Consumption 86

6 Summary and Outlook 99

Bibliography 106

A Source Code 110
A.1 RConnection.java 110
A.2 Simulation.java 120
A.3 NumberRef.java 121
A.4 Sensitivity.java 123

List of Figures

2.1 Illustration of the implementation of the simulation interface 22
2.2 Illustration of the transformation of a parameter 23
2.3 Illustration of the usage modification of a parameter 23
2.4 Binary tree model 24
2.5 Prepared binary tree model 26
2.6 "Sensitivity" plugin structure 31

4.1 Results of Morris's elementary effects screening of the beech tree's height 54
4.2 Results of Morris's elementary effects screening of the beech tree's carbon production 54
4.3 Results for main effects on extreme values of the beech tree's height 56
4.4 Results for interaction effects on extreme values of the beech tree's height 57
4.5 Results for main effects on extreme values of the beech tree's carbon production . . 58
4.6 Results for interaction effects on extreme values of the beech tree's carbon production 59
4.7 Results for the partial correlation coefficients of the beech tree's height 61
4.8 Results for the partial rank correlation coefficients of the beech tree's height 62
4.9 Results for the partial correlation coefficients of the beech tree's carbon production 62
4.10 Results for the partial rank correlation coefficients of the beech tree's carbon production 63
4.11 Results for the standardized regression coefficients of the beech tree's height 64
4.12 Results for the standardized rank regression coefficients of the beech tree's height . 65
4.13 Results for the standardized regression coefficients of the beech tree's carbon production 65
4.14 Results for the standardized rank regression coefficients of the beech tree's carbon production 66
4.15 Results of Sobol's method for the beech tree's height 67
4.16 Results of Sobol's method for the beech tree's carbon production 68
4.17 Results for the extended Fourier amplitude sensitivity test of the beech tree's height 70
4.18 Results for the extended Fourier amplitude sensitivity test of the beech tree's carbon production 70

4.19 Results of Morris's elementary effects screening for the net photosynthetic rate of the assimilate production model . . . 74
4.20 Results for main effects on extreme values of the net photosynthetic rate of the assimilate production model . . . 76
4.21 Results for interaction effects on extreme values of the net photosynthetic rate of the assimilate production model . . . 77
4.22 Results for the partial correlation coefficients of the net photosynthetic rate of the assimilate production model . . . 79
4.23 Results for the partial rank correlation coefficients of the net photosynthetic rate of the assimilate production model . . . 80
4.24 Results for the standardized regression coefficients of the net photosynthetic rate of the assimilate production model . . . 81
4.25 Results for the standardized rank regression coefficients of the net photosynthetic rate of the assimilate production model . . . 82
4.26 Results of Sobol's method for the net photosynthetic rate of the assimilate production model . . . 83
4.27 Results for the extended Fourier amplitude sensitivity test of the net photosynthetic rate of the assimilate production model . . . 85

5.1 Deviation of the runtime for PCC with 4 parameters . . . 87
5.2 Comparison of the runtime for the sensitivity analysis methods . . . 96
5.3 Comparison of the memory consumption for the sensitivity analysis methods . . . 96

List of Tables

1.1 Overview of the implemented methods 14

4.1 Overview of the examined beech's input parameters 51
4.2 Results for the local sensitivity analysis of the beech tree 53
4.3 Overview of the examined assimilate production model input parameters 72
4.4 Results for the local sensitivity analysis of the net photosynthetic rate of the assimilate production model 73

5.1 Runtime and memory consumption measurements 88
5.2 Runtime and memory consumption of the local sensitivity analysis 89
5.3 Runtime and memory consumption of Morris's elementary effects screening 90
5.4 Runtime and memory consumption of the main and interaction effects on extreme values 91
5.5 Runtime and memory consumption of the partial (rank) correlation coefficients (PCC/PRCC) 92
5.6 Runtime and memory consumption of the standardized (rank) regression coefficients (SRC/SRRC) 93
5.7 Runtime and memory consumption of Sobol's method 94
5.8 Runtime and memory consumption of the extended Fourier amplitude sensitivity test (EFAST) 95
5.9 Information quality assessment of the sensitivity analysis methods 97

List of Abbreviations

SA	sensitivity analysis
LSA	local sensitivity analysis
GSA	global sensitivity analysis
OAT	once-at-time
GroIMP	growth grammar related interactive modeling platform
RGG	relational growth grammar
DoE	design of experiments
PCC	partial correlation coefficients
PRCC	partial rank correlation coefficients
SRC	standardized regression coefficients
SRRC	standardized rank regression coefficients
CC	Pearson correlation coefficient
FAST	Fourier amplitude sensitivity test
EFAST	extended Fourier amplitude sensitivity test
PPFD	photosynthetically active photon flux density
PAR	photosynthetically active radiation

Chapter 1

Introduction

The question of input to output relationship is the leading subject of simulation models in all scientific domains. A common mathematical instrument to accomplish this problem setting is the so-called process of *sensitivity analysis* (SA). It can be considered as the study of how model input parameters influence a particular interest quantity as the model or simulation output [1]. The amount of influence of input variables on a certain model output is thus called *parameter sensitivity* [2], which is a measure resulting from an SA method. Therefore SA estimates how sensitive designated outputs are to fluctuations within the input parameter space [3]. Another definition that can be found in literature [4] [5] is the "study of how uncertainty in the model output can be attributed to different sources of uncertainty in the model input". Because of that, the method of SA is also referred to by the term "What-if-analysis" [6]. SA is used in a wide variety of domains including economics, geography, biology, physics, human medicine, engineering and ecology. It is auxiliary to detect connections or correlations between model inputs, predictions and observations [7]. SA can deliver important information in order to reveal model complexity or for model understanding. This particularly is the case for plant models [8], because they are meant to represent complex biological processes with various organs (functional units) interacting. In order to get a rough understanding of the field of SA, questions of interest could be [9] [3]:

- What level of validity does the model possess?
- How much will the outcome change for little parameter modifications?
- Can parameters be omitted in order to simplify the model?
- What is the most influential parameter? Is there a linear or non-linear relationship?
- Do parameters interact?
- Are the initial assumptions valid?
- Can reasonable results be obtained for all parameter combinations?

Considering the questions above, SA has a far reaching scope of objectives. The following should give an overview [7] [10] [11] [12] of the most common use cases of SA: One main issue is the test of model robustness. This outcome focuses on how the model behaves when the input parameter setting is prone to high uncertainties. SA can help understanding the relationship between input and output. This is especially important in order to reveal parameter interactions. Moreover it can deliver a significant contribution for the model understanding by unfolding primarily hidden effects. Thus, model errors can also be derived when obtaining unexpected or empiricially wrong output. Another very important intent of SA is the ranking of parameters by their impact (sensitivity measure). The rank information for each parameter can be very useful for a later model simplification. However, that does not mean that always non-influential parameters are dropped. It can also lead to an omission of a highly sensitve parameter in order to decrease the output variance and thus increase the model robustness. In principle, SA can be applied for the purpose of validity and accuracy checking of the model. This is in particular the case when looking at the outcome of global SA methods (definition in next paragraph). Moreover one main objective of SA is given by obtaining the input to output relation in a functional manner. This can be beneficial in order to identify critical values, thresholds or break-even points [7] in the input space respecting the desired output measure. Hereby calibration of the initial parameter configuration can be accomplished [1].

For the subject of SA, many mathematical approaches that address the different objectives exist. In general, it is distinguished between local (LSA) and global SA (GSA) [4]. For nearly all SA methods it holds true that a set of parameter combinations is created and the simulation is executed whilst gathering the specific output value for each combination. Afterwards, the input to output value data mapping is used for the concrete method's further calculations. The difference between local and global SA is that in local SA the initial input parameter configuration is only examined within a percentage interval (for example 90% to 110%), whereas in global SA the input parameters are varied over the whole possible input space. A point from the input space is given by a vector of parameter realizations where each entry is within the range of the lower and upper bound (minimum and maximum value) for the respective parameter. In practice, the most often used SA methods belong to the so-called *OAT class* [13]. OAT stands for "once-at-time" and means that only one parameter is varied at a time step whilst all others are kept constant. The reason can be found in maintaining simplicity, since models potentially possess many parameters and the systematic variation of multiple parameters simultaneously can be challenging because of the exponential increase in the number of samples (curse of dimensionality). However, OAT approaches cannot give full insight into interaction effects [4].

Input parameters in SA that govern some output generating simulation generally have no fixed shape and thus can be, for example, any mathematical objects, like matrices, vectors or functions. However, in this work it is by definition the case that an arbitrary input or output parameter always refers to a single, one-dimensional, numeric-valued variable. Hence, each SA method will rely on the same parameter shape such that comparability is guaranteed.

When considering computer simulation models it is comprehensible to expect input to output relations within the source code. This knowledge then could be used by incorporating it in the SA. Nevertheless, such connections should be initially revealed by SA. This is particularly important for plant models as they often possess processes that are not fully understood. Because of that, it is determined that when referring to SA the input to output transition is considered as a black box transformation. Hence, no a priori knowledge about the transformation process is assumed.

An important application field where SA can deliver insightful results is the domain of plant modeling. GroIMP (Growth Grammar related Interactive Modeling Platform) is a 3D plant modeling software platform, which processes code written in the language XL. XL is a superset of Java and implements the so-called *relational growth grammars*. For more information see [14] [15]. The software R [16] is a very sophisticated statistical computing platform coming with its own programming language. It supplies various packages for nearly all mathematical problem settings and has thus become an indispensable analysis tool for many scientific domains.

The current paper deals with the deployment, implementation and testing of SA tools for GroIMP by the utilization of the R language. Since GroIMP is composed of plugins, where each one constitutes a part of the provided functionality, a plugin that extends GroIMP with SA functions has been developed. The topic of using R in order to conduct SA has been treated by the authors of [17]. In that paper SA was not applied to plant models, but to agent-based models that target to model real-world phenomena. Nevertheless, in the current paper the procedures and methods described in [17] have been used as leads for the deployment of SA in GroIMP. An overview of the SA methods that have been implemented and tested can be found in table 1.1.

Method	LSA	GSA	OAT
Local Sensitivity Analysis	X		X
Morris's Elementary Effects Screening		X	X
Main And Interaction Effects On Extreme Values		X	X
Partial (Rank) Correlation Coefficients		X	
Standardized (Rank) Regression Coefficients		X	
Sobol's Method		X	
Extended Fourier Amplitude Sensitivity Test		X	

Table 1.1: Overview of the implemented methods
LSA = local sensitivity analysis approach
GSA = global sensitivity analysis approach
OAT = once-at-time approach

The document structure reveals as follows:

- In **chapter 1** the domain of sensitivity analysis is examined. Besides, the motivation and topic of the current paper are described.
- In **chapter 2** details about the implementation are given. This includes how the process communication has been realized and moreover, which specific classes and interfaces are needed for the plugin. Furthermore, it is described what modifications are necessary in order to prepare a certain plant model - implemented in GroIMP - for SA.
- In **chapter 3** the mathematical foundation for each method (see table 1.1) is presented.
- In **chapter 4** the developed plugin is tested by applying each implemented method to two plant models, considering different outputs.
- In **chapter 5** the SA methods are compared, regarding runtime, memory consumption and the information quality.
- In **chapter 6** the results are summarized and an outlook for future developments is given.

Chapter 2

Implementation Details

In this chapter different aspects of the plugin implementation are described. At first the problems one is facing using the Java language for the outcome of SA are explained. Furthermore, the results for the task to connect the R language to Java are depicted. Additionally, the plugin structure is revealed including the description of the helper classes and interfaces necessary. Moreover it is elaborated how an arbitrary plant model must be modified in order to be able to perform an SA method. Since in GroIMP every plugin - which is a contribution to the functionality scope - is identified by its unique name, and because of the fact that there is no SA plugin contained at the moment, it was chosen to patently name the developed plugin "Sensitivity". In the following paragraphs only few source code parts are shown, but references to code lines. Thus, the complete source code can be found in the appendix (A.1, A.2, A.3, A.4).

2.1 Connecting R to Java

The fact that GroIMP is written in Java makes it platform independent. For the version 1.5 at least Java 7 is required in order to execute the software. For later versions GroIMP has also been adapted to Java 8. This is in particular advantageous because there are many proprietary and non-proprietary Java plugins available that intend to deploy the R language in the Java environment, and often those plugins only work for Java versions higher than 8. Those are for example Renjin, FastR, JRI, Rserve or rJava. However, the existing solutions come with many disadvantages that make the use of the plugins in association with GroIMP unattractive. It is for instance the case, that the plugin Renjin does not support all R packages available. This fact is especially unfavorable because packages usually possess dependencies. Hence, one missing package in Renjin can render a bunch of important packages useless. Moreover it holds true that new released R packages need to be translated by the Renjin developer team. Since often there is only a small user community for special packages, they will potentially be neglected for

incorporation. In the majority of cases the plugins that connect R to Java are not platform independent. Nevertheless, concerning the platform independence of GroIMP, this property should also apply to its plugins. For some only a Windows-based implementation is available. For others, in order to make a plugin work for a certain operating system, heavy user work for customization is needed, as often platform dependent plugin adaptation is mandatory. On account of these disadvantage it was decided not to use an existing Java plugin at all.

Another approach to consider - apart from using an existing plugin - is the connection of R and Java via a server-client application. An implementation of R then would run on a server. The concrete Java application would use an HTTP-connection in order to send and execute R commands and also to receive the response and to parse the output. However, considering this strategy, one faces huge unreliability issues. The plugin would heavily depend on a flawlessly working server. Furthermore, it would hinge on the availability of an internet connection. Because of these major disadvantages, this approach will not be taken into consideration since it cannot fulfill the reliability demands for GroIMP plugins that should also function offline.

Because of the described existing approaches and their drawbacks it was chosen to implement a class that encapsulates the connection to R. A prerequisite for this approach is an existing R installation including the necessary packages (for example the "sensitivity" package for SA). For every operating system an R implementation contains an executable that handles byte streams for process input and output. Thus, the data exchange is done by process communication techniques. Hence, executing an R command is carried out by writing to the process's standard input stream. In order to feed the output back to Java datatypes, parsing of the read lines from the standard output stream is necessary. In the following, the structure of the Java class "RConnection" is described. This will also include all declared fields as well as the public and private functions. Since a line by line walk through the code is not very insightful, the main objectives in the code will be sequentially processed.

2.1.1 Class "RConnection"

2.1.1.1 Fields and Properties

- **private BufferedWriter w** (line 57) — This is the global variable of the buffered writer for the R process's standard output stream.
- **private BufferedReader r** (line 58) — This is the global variable of the buffered reader for the R process's standard input stream. One should recall that input and output stream refer to the identical single stream in a process. As an example, a classical terminal console should be considered.

- **private boolean _success** (line 61) — This variable specifies if a connection could successfully be established.
- **public boolean success()** (line 63-65) — This is a getter for the variable _success.

2.1.1.2 Functions

- **private boolean isOSWindows()** (line 188-190) — Checks if operating system is Windows.
- **private boolean isOSLinux()** (line 195-197) — Checks if operating system is Linux.
- **private boolean isOSMac()** (line 202-204) — Checks if operating system is Mac.
- **private String get_R_executable()** (line 233-378) — This function returns the path of the R executable, which is depending of the current operating system (OS). When the function was not successful, and thus no R installation could be found, *null* is returned. In case that the OS is Windows, the R executable cannot be obtained by a system environment variable. Because of that, for the initial use of the sensitivity plugin the R installation folder must be determined by the user. When this is the case, a dialog pops up and the user is asked to navigate to that folder. If the search for the R executable was successful, the path will be stored in a setting file. In case it was not found, an error message will indicate this failure. For further plugin usage the saved path is recovered from the settings file. In case that the OS is Linux, the R executable is given by a system environment variable. However, if R is not installed and the executable does not exist, a dialog will pop up and indicate what exact steps need to be taken in order to install R using the terminal. Inside this dialog, buttons for copying the necessary commands to the clipboard - the user has to execute in order to install R - represent a convenience offer. If the OS Mac is detected, the R executable is assumed to be located at the default location. If that is not the case, an error message will indicate the missing R installation. Since R versions are only provided for the three mentioned OS, if none of them could be identified, an error message will tell the user that the OS is not supported.
- **private boolean init_R_connection(String rpath)** (line 81-109) — In this function a connection to an R process is established. The argument *rpath* must be the result of the function *get_R_executable()*. Initially, a process that redirects its output stream must be created and started. Since the stream writers and readers are globally defined in order to be used in later functions, they are obtained and assigned. For the case that no error exception occurs, *true* is returned, otherwise the error message is printed on the debug and GroIMP console and *false* is returned.
- **public String[] eval(String expression)** (line 117-153) — This function represents the communication interface to R. It is used to state or evaluate any arbitrary R expression. The statement that is meant to be executed is specified by the argument *expression*. The return value is the response that comes from the R process. Since it is often the case that a

multiple-lined output is returned, it was chosen to store each output line in a separate entry in a string array. That is why the function returns a string array that can be used for later processing, for example for the purpose of number or matrix parsing. Due to the fact that reading from a stream is potentially a thread-blocking command, it is very important that the number of reads does not exceed the number of lines that are in the present buffer. Hence, in order to detect the end of any output, a special token is used for the output end identification. Because of that, the expression that should be executed is written to the process's standard input followed by the special token. In addition, it is very important to reconsider the fact that the standard output and input stream of a process refer to the same single stream. On account of this fact, every line that is written to the process's input stream simultaneously appears in the output stream. Since the return value of the function *eval()* should not contain the expression itself, it is skipped by reading from the stream exactly the number of times as there are line breaks in the expression. This guarantees the behavior desired. Afterwards, every output line is read from the stream and stored until the special end token is detected. Finally, the resulting array that contains the command response from R can be returned. In case of any error occurrence, a message is printed on the debug and GroIMP console and *null* is returned.

- **private boolean check_R_Packages()** (line 385-435) — For the plugin it is mandatory that not only an R installation is present, but also specific R packages. Such packages are "sensitivity" [18], "FrF2" [19], "tgp" [20], "MASS" [21], "gridExtra" [22], "latex2exp" [23], "lattice" [24] and "directlabels" [25]. Therefore, the function *check_R_packages()* determines whether all required packages are installed. For that outcome with the help of the *eval()* function it is examined if a package is missing. If no missing package was detected, they will be loaded via the *library* command from R, and success for the package check is indicated by the return of the value *true*. Additionally, in case of positive check the R helper function *flat* is defined. This R function is relevant to every SA function as a parsing helper that sequentially outputs every desired output matrix entry. This is in particular needed to obtain the parameter combinations. For detected missing packages, an error message informs the user about the packages that need to be installed in order to work with the SA plugin. Hence, *false* is returned.
- **public void X11()** (line 158-172) — In order to open a plot window, R offers different functions for the different operating systems. The wrapper function *X11()* calls the appropriate one, depending on the current system.
- **public void waitForClose()** (line 177-183) — For the purpose of keeping plot windows, created with the command *X11()* open and reactive, the function *waitForClose()* needs to be used. When the window is closed by the user, the R process is automatically terminated and the connection ends.

2.1.1.3 Constructor

- **public RConnection()** (line 70-74) — The constructor of the class "RConnection" calls the functions *get_R_executable()*, *init_R_connection()* and *check_R_packages()*. Only if all three calls have been successful, a valid connection to an R process can be established. Because of that, the value of the variable *_success*, which indicates the connection state, is set accordingly. When an instance of the type *RConnection* is created, the evaluation function *eval()* represents the primarily used function in order to execute R commands and to receive their response.

2.2 Simulation Identification

SA requires that the same simulation is repeated again and again. However, each simulation run is performed with a different parameter combination. In GroIMP the simulation process can incorporate many functions as well as derivation rules. Nevertheless, it is the case for GroIMP that a certain, identifiable function triggers the start of a simulation run. Since the developed SA methods are considered to be generic and not fixed to a concrete model, there is a need to be able to hand over a function as a variable argument. An SA function then would be called with an argument that represents the dedicated simulation. Unfortunately, Java does not offer the possibility to pass functions directly as object references. The solution that remains is the use of a simulation interface. It is very important to recall that in GroIMP every RGG-file itself represents a Java class. The file's name is simultaneously the Java class name. Its superclass is the RGG-class. The solution to the problem described ahead is that the class in the RGG-file needs to implement a certain interface which contains the dummy functions needed to run the simulation and to obtain the simulation output. Thus, the interface "Simulation" was developed. The key point is, that when calling an SA function an instance of "Simulation" must be passed over. When an SA is done in the class itself that implements the "Simulation" interface, it is only needed that the instance is passed to the SA method by the *this* statement.

2.2.1 Interface "Simulation"

2.2.1.1 Functions

- **public void run()** (line 29) — This is the dummy function in order to start the simulation. It needs to be be implemented from the class where the simulation is located. It is crucial that in this function before or after the simulation terminates or is started again, it is reset to its initial state. This guarantees that always comparable simulation runs constitute the base of SA.

- **public double getOutput()** (line 31) — This dummy function represents the output measurement that is performed after every simulation run. The outputs are thus gathered in every SA method by the call of this function. It was determined that the output type is *double*, which coincides with real-valued numbers and covers all other integer-based number datatypes. Performing SA, the user is obligated to write the output generating function himself, since there are arbitrary readings as potential values of sensitivity interest. Nevertheless for complex readings, for example structure-aggregated values, GroIMP and XL possess the very advantageous tool of graph queries in order to obtain them.
- **public String outputName()** (line 33) — This function is meant to represent the name of the output that is generated by the function *getOutput()*. In particular, the function is necessary for the correct labeling of the plot resulting from an SA method. Since it is internally used to assign data vectors, no spaces in this string are allowed.

2.3 Passing Model Parameters

SA is the process of systematically varying the model parameters and observing the output. That means when calling an SA method, the model parameters whose sensitivity is meant to be examined respecting a certain output, must be handed over as arguments. However, it is the case that the model parameters are always given by primitive datatypes, which are by default passed by value. Java does not offer any other possibility to pass primitive datatypes as reference than the so-called process of *boxing* and *unboxing*. To hand over the parameters as references is crucial because they are supposed to be systematically varied. If passed by value, a modification in the code of an SA function would not have any impact on a global variable at all. In Java all primitive numerical value types have a reference-typed clone. This is for example the case for the type *double*, where the reference clone is *Double*. Nevertheless, it was decided to implement an own generic class for the boxing of number types, which delivers additional functionality compared to the existing Java approaches. This class is called "NumberRef".

2.3.1 Class "NumberRef"

2.3.1.1 Fields and Properties

- **private double value** (line 31) — This field represents the number value for the concrete class instance. The type *double* was chosen because it can hold any number for example double, float, int and long.
- **private String name** (line 32) — For SA it is important to uniquely identify every parameter. Furthermore, every parameter needs a name for later plotting purposes (labeling), since it

is desired that the parameter can possess another name than the variable itself. Due to that reason, the field *name* stores the parameter name.

- **public void set(int value)** (line 67-69) — This is the setter for the field *value* (*int* version).
- **public void set(long value)** (line 71-73) — This is the setter for the field *value* (*long* version).
- **public void set(float value)** (line 75-77) — This is the setter for the field *value* (*float* version).
- **public void set(double value)** (line 79-81) — This is the setter for the field *value* (*double* version).
- **public String name()** (line 83-85) — This is the getter for the field *name*.
- **public void setName(String name)** (line 87-89) — This is the setter for the field *name*.

2.3.1.2 Functions

- **public int getInt()** (line 51-53) — This is the getter for the field *value* type-casted to *int*.
- **public long getLong()** (line 55-57) — This is the getter for the field *value* type-casted to *long*.
- **public float getFlt()** (line 59-61) — This is the getter for the field *value* type-casted to *float*.
- **public double getDbl()** (line 63-65) — This is the getter for the field *value* type-casted to *double*.

2.3.1.3 Constructors

There is a constructor for every number type. It mandatory to always specify a value and the name of the model parameter.

- **public NumberRef(int value, String name)** (line 34-36) — The constructor creates a new instance of the class *NumberRef*, when the type is *int*.
- **public NumberRef(long value, String name)** (line 38-40) — The constructor creates a new instance of the class *NumberRef*, when the type is *long*.
- **public NumberRef(float value, String name)** (line 42-44) — The constructor creates a new instance of the class *NumberRef*, when the type is *float*.
- **public NumberRef(double value, String name)** (line 46-49) — The constructor creates a new instance of the class *NumberRef*, when the type is *double*.

2.4 Preparing a Plant Model for Sensitivity Analysis

In the previous two sections it was described how to overcome the Java drawbacks considering parameter and function passing. This is the reason why a plant model needs to be adapted in order to be able to perform an SA method. In the following, the steps that are necessary to prepare a plant model are described. Afterwards, as an example, the binary tree from the GroIMP example section is used to show the adaptations.

2.4.1 Preparation Steps

1. **Implementing the Simulation Interface —** Since every RGG-file constitutes a Java class, it might be the case that one cannot find the class definition statement within the file itself. Because of that, in order to tell the class to implement the interface the code has to be reorganized. Therefore, after all import statements have been made, the code has to be wrapped by the class name "Main", which is nevertheless overwritten by the file name of the RGG-file. Hence, the class is not called "Main" as the code might suggest, but this token is relevant for the XL compiler. In addition, to correctly implement the interface, the embraced functions need to be implemented according to the statements made in the previous section. The framework is given by figure 2.1.

```
[...]  //import statements

public class Main extends RGG implements Simulation {

   [...] //source code

   @Override
   public void run() {
      [...]
   }

   @Override
   public double getOutput() {
      [...]
   }

   @Override
   public String outputName() {
      [...]
   }
}
```

Figure 2.1: Illustration of the implementation of the simulation interface

2. **Replacing the model parameter value variables with a *NumberRef* version** — In order to be able to systematically vary the model parameters from within the code of any SA method, it is necessary to replace the value type model parameters with a wrapper version as an instance of the class *NumberRef*. The parameters that are designated for SA then can be passed over to the concrete SA method as a call argument. Since value-typed number variables have been replaced by reference-typed versions, the parameters cannot be directly used any longer in the simulation code, for example for the internal calculations. Because of that, on the instances of the new variables the appropriate getter and setter functions must be called on every single occurrence in the simulation code. The transformations are illustratively given by the code sample in the figures 2.2 and 2.3.

```
double model_parameter_1 = 5.78;
            ⇓
NumberRef model_parameter_1 = new NumberRef(5.78, "model_parameter_1");
```

Figure 2.2: Illustration of the transformation of a parameter

```
void simulation() {
   [...]
   double volume = 3.14 * Math.pow(r, 2) * model_parameter_1;
   model_parameter_1 += 1.0;
   [...]
}
            ⇓
void simulation() {
   [...]
   double volume = 3.14 * Math.pow(r, 2) * model_parameter_1.getDbl();
   model_parameter_1.set(model_parameter_1.getDbl() + 1.0);
   [...]
}
```

Figure 2.3: Illustration of the usage modification of a parameter

2.4.2 Plant Model Preparation Example

In this section it is exemplarily shown how to prepare a plant model for SA using the binary tree from the GroIMP example section. This is done by applying the steps described in the section before. In the transformed plant model the implemented output generating function *getOutput()* will measure the maximum extent of the plant organs in the xy-plane. This will be done with the help of a special XL rule type namely the so-called *execution rule* or *update rule*. Moreover, it is determined that a simulation run is given by applying the derivation rule *derive()* ten times. Since the concrete plugin's SA functions were not introduced yet, the SA function that represents a certain method is plainly given by the dummy function *SA_method()*, which is a deputy for all methods offered. In order to start the SA process, the function *sensitivity_analysis()* has to be called. The available SA functions as well as their call argument lists will be explained in the next chapter. However, the example makes clear how to correctly pass the arguments. In figure 2.4 the original code from the binary tree can be found. Figure 2.5 shows the code of the prepared model.

```
module A(float l, float w);
module B(float l, float w);

const float r1 = 0.9F;
const float r2 = 0.8F;
const float a1 = 35;
const float a2 = 35;
const float wr = 0.707F;

protected void init()
[
   Axiom ==> A(1, 0.1F);
]

public void derive()
[
   A(l,w) ==> F(l,w) [RL(a1) B(l*r1, w*wr)]
            RH(180) [RL(a2) B(l*r2, w*wr)];

   B(l,w) ==> F(l,w) [RU(-a1) AdjustLU B(l*r1, w*wr)]
                 [RU(a2) AdjustLU B(l*r2, w*wr)];
]
```

Figure 2.4: Binary tree model

```
import de.grogra.sensitivity.*;

public class Main extends RGG implements Simulation {

   module A(float l, float w);
   module B(float l, float w);

   NumberRef r1 = new NumberRef(0.9F,"r1");
   NumberRef r2 = new NumberRef(0.8F,"r2");
   NumberRef a1 = new NumberRef(35.0F,"a1");
   NumberRef a2 = new NumberRef(35.0F,"a2");
   NumberRef wr = new NumberRef(0.707F,"wr");

   protected void init()
   [
      Axiom ==> A(1, 0.1F);
   ]

   public void derive()
   [
      A(l,w) ==> F(l,w) [RL(a1.getFlt()) B(l*r1.getFlt(), w*wr.getFlt())]
                 RH(180) [RL(a2.getFlt()) B(l*r2.getFlt(), w*wr.getFlt())];

      B(l,w) ==> F(l,w) [RU(-a1.getFlt()) AdjustLU B(l*r1.getFlt(), w*wr.getFlt())
          ] [RU(a2.getFlt()) AdjustLU B(l*r2.getFlt(), w*wr.getFlt())];
   ]

   @Override
   public void run() {
      reset();
      for (apply (10)) derive();
   }

   @Override
   public double getOutput() {
      NumberRef output = new NumberRef(Double.MIN_VALUE, "");
      xy_extent_query(output);
      return output.getDbl();
   }

   @Override
   public String outputName() {
      return "XY-EXTENT";
   }

   private void xy_extent_query(NumberRef output) [
      x:Node ::> {
```

```
            Point3d p = location(x);
            output.set(Math.max(output.getDbl(), Math.sqrt(p.x*p.x + p.y*p.y)))
                ;
            }
    ]

    public void sensitivity_analysis() {

        /*
        Call arguments of the SA function:
        1 : array of the model parameters to examine
        2 : array of the parameters' lower bounds (minimum values)
        3 : array of the parameters' upper bounds (maximum values)
        4 : instance of the simulation to run (implementation of the "Simulation"
            interface)
        */

        Sensitivity.SA_method(  new NumberRef[] {r1,r2,a1,a2,wr},
                                new double[] {0.5,0.5,45.0,45.0,0.0},
                                new double[] {1.0,1.0,135.0,135.0,2.0},
                                this  );
    }
}
```

Figure 2.5: Prepared binary tree model

2.5 Sensitivity Analysis Methods

The class that contains all SA methods is called "Sensitivity". It furthermore comprises some helper functions which are nevertheless inaccessible for the user since they are declared as *private*. It was decided to make every single SA function static. As a consequence, the functions can be directly accessed without creating an instance of the class. To avoid confusion it was also decided to mark the "Sensitivity" class as *abstract*, so that it is already impossible to instantiate an object from it. It should be reconsidered that this class can only be used on plant models that have been prepared as prescribed according to the procedure described in the previous section. In the following, every function call argument will be elaborated. It should be noted that function signature is equal for every single function, except for the method *local_sensitivity_analysis(...)*. To avoid misunderstanding the scope of each argument is stated subsequently. After that, the functions - where every one is an implementation of an SA method from table 1.1 - contained in the class will be given. All implemented SA functions possess the return type *void*, hence nothing is returned. However, the output from each one is given by the graphical representation (tables or graphs) of the methods' outcomes (sensitivity measures).

2.5.1 Call Arguments

- **NumberRef[] parameters** (*scope: all*) — This array must contain all model parameters whose sensitivity is meant to be studied. In case of a high number of parameters, there is the strong endorsement that this array is only a subset of all available model parameters. It is recommend that the complete set of model parameters is only processed with local SA and with Morris's screening method. The reason is given by the fact that the other methods are much more complex compared to the two mentioned ones. Furthermore, local SA and Morris's screening are used to initially determine the potentially influential parameters that are worth to be examined further. This leads to a reduction in the number of examined parameters and thus a reduction of computational complexity.
- **double min** (*scope: local_sensitivity_analysis*) — This parameter specifies the relative minimum test value for the examination range. It must be recalled that local SA only examines a bounded percentage interval around the initial parameter value. The expected value range of *min* is given by $\{0.0 \leq min \leq 1.0\} \subseteq \mathbb{R}$.
- **double max** (*scope: local_sensitivity_analysis*) — This parameter specifies the relative maximum test value for the examination range. The expected value range of *max* is given by $\{1.0 \leq max < \infty\} \subseteq \mathbb{R}$.
- **double[] min_var** (*scope: all* $\setminus\{$*local_sensitivity_analysis*$\}$) — Global SA methods examine the complete definition ranges of model parameters. Every entry in the array *min_var* coincides

with the lower bound (minimum value) of each parameter that is meant to be examined and that is given by the specific entry in the array *parameters*. It must be stated that these bounds may deviate on account of the parameter selection and the desired output measurement.

- **double[] max_var** (*scope: all* $\setminus\{$*local_sensitivity_analysis*$\}$) — This array is defined equivalently to *min_var*, except that it is defined for the parameters' upper bounds (maximum values).
- **Simulation simulation** (*scope: all*) — This argument expects an instance of a class that has implemented the "Simulation" interface. The instance variable is then used to identify the simulation run function that will be executed for every parameter combination coming from the prevailing SA function. Furthermore, the interface implementation also is utilized to perform the output measurement after every simulation run as well as to obtain the output's name. It should again be mentioned that, when the SA method is called within the class that implements the "Simulation" interface, the argument that has to be passed is simply given by the identifier *this*.

2.5.2 Class "Sensitivity"

2.5.2.1 Functions

- **public static void local_sensitivity_analysis(NumberRef[] parameters, double min, double max, Simulation simulation)** (line 41-92) — This function performs the local SA method.
- **public static void morris_elementary_effects_screening(NumberRef[] parameters, double[] min_var, double[] max_var, Simulation simulation)** (line 103-155) — This function performs the method of Morris's elementary effects screening. Since all following SA functions possess the same signature as this function, it will thus be indicated by (...).
- **public static void main_and_interaction_effects_on_extreme_values(...)** (line 166-230) — This function calculates the main and interaction effects based on an approach by Kathrin Happe. The method is applied to the extreme values of the parameters, hence 2^n parameter combinations are tested, when n is the number of tested parameters.
- **public static void partial_correlation_coefficients(...)** (line 241-296) — This function calculates the partial correlation coefficients and also their rank version.
- **public static void standardized_regression_coefficients(...)** (line 307-357) — This function calculates the standardized regression coefficients. Additionally, these coefficients are also given in their rank version.
- **public static void sobol(...)** (line 367-420) — This function performs Sobol's method, which is attributed to the variance-based SA methods. This results in the calculation of the parameters' first-order and total-effect indices.

- **public static void extended_fourier_amplitude_sensitivity_test(...)** (line 431-480) — This function calculates first-order and total-effect indices of the parameters like Sobol's method. Nevertheless, the calculation base is different and although it is belonging to the variance-based SA methods, the comparability of these indices is only given within the domain of the concrete method output itself. The method's foundation is based on variance decomposition using Fourier series expansion.

2.5.2.2 Using R for Sensitivity Analysis

The Java functions described in the previous section, where each one implements an SA method, make use of the developed framework that was described in sections 2.1.1, 2.2.1 and 2.3.1. In the following, it is elaborated how Sobol's method was implemented to show in an exemplary way how to employ the R language whilst using the developed Java helper constructs. The fundamental train of thought also applies for all other SA functions. Because of that and due to the fact that handling the code of all SA methods is not very insightful it was decided to only process the code of Sobol's method.

The first task in order to be able to execute R commands is to establish a connection to an R process. Because of that, a new instance of the class "RConnection" is created (line 371). Nevertheless, it could be the case that the constructor was unable to successfully create a valid connection. Hence, in line 372-373 the result of the function *success()* on the created instance is checked. If *false* is returned, the connection is invalid and the *sobol* function will be exited by the *return* statement.

Sobol's method requires two sample sets of the model parameters that are meant to be examined. This is the reason why in line 377-379 two latin hypercube samples are created, which is done by the use of R commands. Thus, the *eval* function from the connection instance is used to hand over the appropriate commands and assignments to the R process. It should be noticed that the sample set objects for now only exist within the R environment in the context of the stack of the R process and have thus no mirrored representation in the Java code.

Since the package "sensitivity" from R also possesses the possibility to plot SA results, a correct naming of data objects is mandatory. Due to that fact, the column names of the created parameter sets must be set according to the parameters that have been handed over to the function (array of *NumberRef* variables). Hence, in line 382-385 the parameter names are assigned for the R data frame objects with the use of the *eval* function again.

In R it is most commonly the case that SA is done with the help of specialized SA objects. For Sobol's method this also holds true. Because of that, in line 390 a so-called *sobol* object is created whilst passing all necessary call arguments. It should be reconsidered that it was predetermined

that initial model assumptions (for example the conjecture of a linear relationship) are neglected for SA. That is why the *model* argument of the *sobol* object is set to *NULL*, thus model-freeness is guaranteed. The created *sobol* object contains the parameter combinations that were selected to be run with the simulation. Since the model parameters must then be varied in the Java environment for every simulation run, they have to be parsed from R's *sobol* object (line 393). The parsing is accomplished with the help of the self-defined R helper function *flat*, which outputs every matrix entry on a single new output line. That is why the result of the corresponding *eval* call is handed over to the Java function *parseMatrix*, which creates a 2D array with the parameter combinations.

Owning the test set of parameters, the simulation must be run for each parameter combination whilst gathering the corresponding output measurement. Hence, in line 396 a vector in R that collects the output readings is created. Afterwards, the simulation is run for every single parameter combination (line 397-403). For that outcome on every instance of the model parameters from the *NumberRef* array *parameters* the function *set* is called to globally manipulate the corresponding parameter value (line 399). Then, the simulation is run (line 401). After that, the output generating function is called and the result is passed over to R's target vector (line 402). What remains is to hand over the simulation results to the *sobol* object, so that the desired effect indices can be calculated. On that account, the *tell* command from R is used in line 406.

As stated before, the results can be plotted using the already existing plotting capability of the "sensitivity" package. For that outcome, initially a window has to be opened (line 409). The window will be the address where the later stated *plot* command is directed to for displaying charts. Hence, in order to plot the results of Sobol's method, the *plot* command is straightforwardly called with the *sobol* object as argument (line 410). In order to keep the plot window reactive the function *waitForClose* must be used afterwards (line 411).

Furthermore, it must be mentioned that the complete code is wrapped up by a try-catch-statement. Thus, in case of any error occurrence in line 416-418 the relevant error message is processed and then printed on the standard debug output and also on the GroIMP console in order to make clear which operation failed.

2.6 Plugin Structure

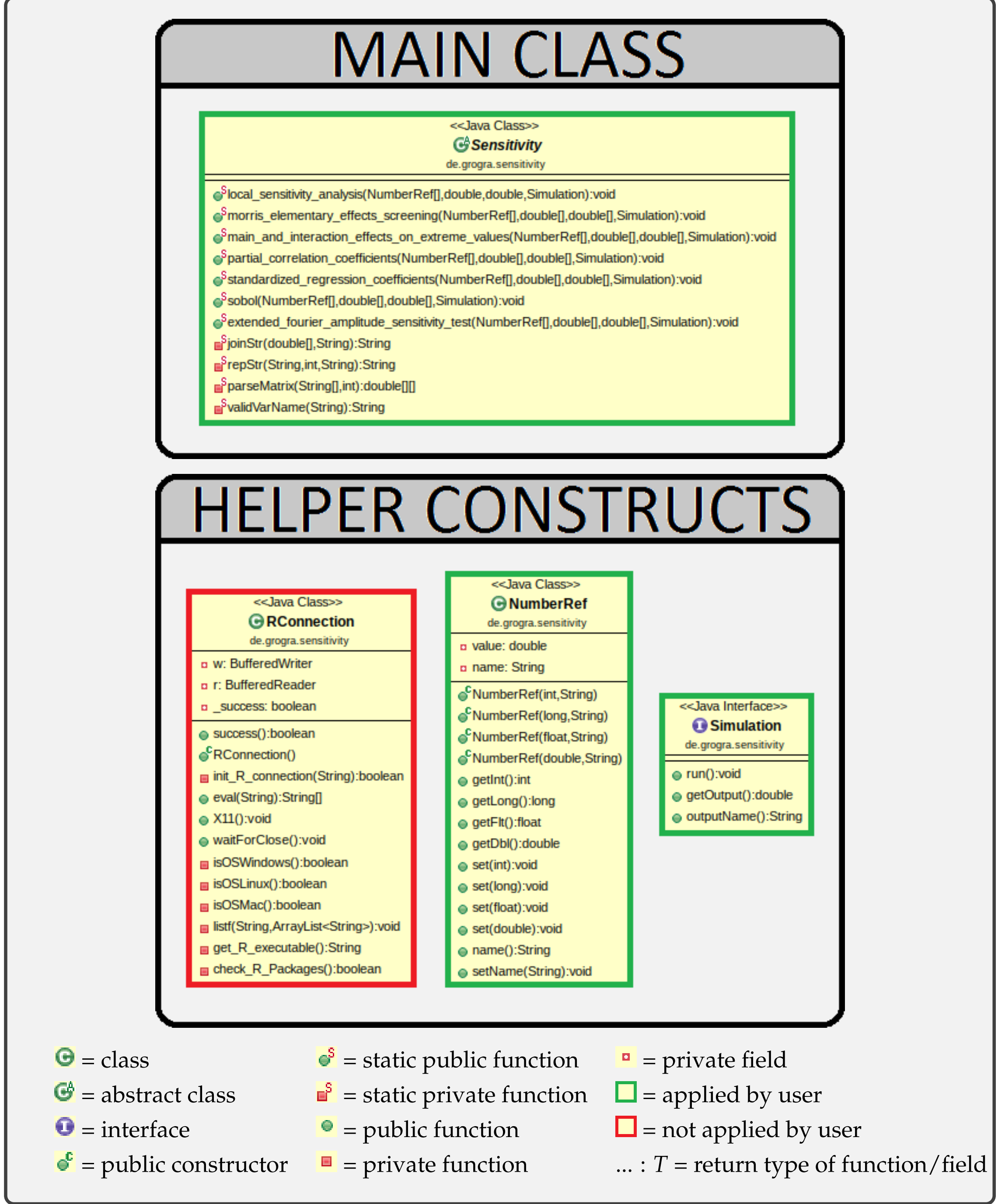


Figure 2.6: "Sensitivity" plugin structure

Chapter 3

Mathematical Foundation

In this chapter every SA approach listed in table 1.1 will be dealt with theoretically.

3.1 Local Sensitivity Analysis

Local SA is a procedure where the model parameters are examined only within a percentage range of the default value. Hence, the parameter space of an individual model parameter is only covered partially. Moreover, local SA belongs to the OAT-class (once-at-time) of SA methods. Thus, for every simulation run only one parameter is modified at a time and then output is obtained. It is important to recall that hence any local SA approach can only obtain limited information [17]. Thus it is primarily used to initially identify highly-sensitive parameters and to get a general overview on sensitivity behavior. However, it is crucial to understand that for a variant initial parameter configuration the sensitivity measures could be completely different. In general, there is no definite method one could refer to, as *the* local SA method. Instead, there are fundamental mathematical approaches that must always be considered when dealing with the task of quantifying the change in output value for parameter variations, for example finite difference approximation or Taylor series expansion [26]. Nevertheless, in this paper local SA is always referred to the approach described in [17]. The authors suggest calculating two normalized effect values for every parameter. The sensitivity values are given as a percentage value for the minimum and the maximum of the examination range. By that, the effect strength is comparable among the model parameters. Moreover, the sign of the sensitivity value indicates whether the output decreases or increases within the examination interval extrema.

Let

$$\mathbf{P} := (P_1, P_2, \ldots, P_n)^T \tag{3.1}$$

be the vector of the n parameters meant to be examined.

Let furthermore

$$m^- \in [0,1] \subseteq \mathbb{R} \tag{3.2}$$

and

$$m^+ \in [1,\infty) \subseteq \mathbb{R} \tag{3.3}$$

be the relative minimum respectively maximum test value of the examination range. Let the fixed initial parameter configuration be given by

$$p = (p_1, p_2, \ldots, p_n)^T . \tag{3.4}$$

Every entry of the vector p is a realization of a parameter at the corresponding index in the vector $\mathbf{P}$. Having defined m^-, m^+ and p it is now possible to define the minimum and maximum versions of the parameter vector:

$$p^- := \left(p_1 m^-, p_2 m^-, \ldots, p_n m^-\right)^T , \tag{3.5}$$

$$p^+ := \left(p_1 m^+, p_2 m^+, \ldots, p_n m^+\right)^T . \tag{3.6}$$

A single entry from p^- and p^+ is referenced by $p_i^- = p_i m^-$ and $p_i^+ = p_i m^+$.

The simulation function is given by $SIM(x)$, where it holds true that x is a fixed parameter configuration and thus a realization of $\mathbf{P}$. The result of the function is the desired uni-dimensional output measurement where the sensitivity is analyzed on, thus

$$SIM : \mathbb{R}^n \to \mathbb{R}. \tag{3.7}$$

$SIM(p) = SIM\left((p_1, p_2, \ldots, p_n)^T\right)$ is the base value of the simulation where p is the initial parameter configuration, thus it is the reference output measurement where the variation in output is based on relatively when one parameter is manipulated. The function that calculates the relative change in output (deviation) and is the representative sensitivity measure is called $dev(x)$, where x is a fixed parameter configuration. In the vector x, only one parameter is varied at a time. Thus there is exactly one i such that $(x_i \neq p_i)$ in $x = (x_1, x_2, \ldots, x_n)^T$. The function $dev(x)$ is defined as

$$\begin{aligned} dev(x) = dev\left((x_1, x_2, \ldots, x_n)^T\right) &:= \frac{SIM(x) - SIM(p)}{SIM(p)} * 100 \\ = \frac{SIM\left((x_1, x_2, \ldots, x_n)^T\right) - SIM\left((p_1, p_2, \ldots, p_n)^T\right)}{SIM\left((p_1, p_2, \ldots, p_n)^T\right)} &* 100. \end{aligned} \tag{3.8}$$

The result of the local SA is given by the so-called *effect matrix* which is defined as

$$\mathcal{M}_{\text{eff}}(p) = \mathcal{M}_{\text{eff}}\left((p_1, p_2, \dots, p_n)^T\right) := \begin{pmatrix} dev\left((p_1^-, p_2, \dots, p_n)^T\right) & dev\left((p_1^+, p_2, \dots, p_n)^T\right) \\ dev\left((p_1, p_2^-, \dots, p_n)^T\right) & dev\left((p_1, p_2^+, \dots, p_n)^T\right) \\ \vdots & \vdots \\ dev\left((p_1, p_2, \dots, p_n^-)^T\right) & dev\left((p_1, p_2, \dots, p_n^+)^T\right) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{SIM\left((p_1^-, p_2, \dots, p_n)^T\right) - SIM(p)}{SIM(p)} * 100 & \frac{SIM\left((p_1^+, p_2, \dots, p_n)^T\right) - SIM(p)}{SIM(p)} * 100 \\ \frac{SIM\left((p_1, p_2^-, \dots, p_n)^T\right) - SIM(p)}{SIM(p)} * 100 & \frac{SIM\left((p_1, p_2^+, \dots, p_n)^T\right) - SIM(p)}{SIM(p)} * 100 \\ \vdots & \vdots \\ \frac{SIM\left((p_1, p_2, \dots, p_n^-)^T\right) - SIM(p)}{SIM(p)} * 100 & \frac{SIM\left((p_1, p_2, \dots, p_n^+)^T\right) - SIM(p)}{SIM(p)} * 100 \end{pmatrix}. \tag{3.9}$$

In the matrix $\mathcal{M}_{\text{eff}}(p)$ each row belongs to a model parameter. The first and second entries are the relative deviations in output when the parameter is varied by being multiplied with m^- and m^+ respectively, whilst all other parameters are kept constant to their initial readings. The sign of each entry indicates whether the output is reduced or increased. The strength of the effect is then given by the absolute percentage value.

3.2 Morris's Elementary Effects Screening

The method of effects screening was developed by Max Morris. It is an assumption-free approach that is meant to explore models whose numbers of parameters are very high [17] and hence the application of a variance-based SA technique would be too computationally expensive [27]. Morris's elementary effects screening is considered to be efficient and fast. The method belongs to the OAT-class (once-at-time), hence only one parameter is varied whilst the others are kept constant. However, the parameter configurations that are run with the simulation are randomized and not fixed to default values [17]. The classification as OAT yields the common fact that the interacting of parameters cannot be examined by this approach. Since every single model parameter can obtain any value within its definition range, Morris's effects screening is also a global SA method.

For every parameter so-called *elementary effects* are calculated based on the mentioned random parameter configuration. Nevertheless, the effect values do not constitute the output of the screening, but a statistical analysis of them. Because of that, for each parameter two different measures emerge: the so-called *mu-value* μ which is the mean, and the sigma-value σ which is the standard deviation of the elementary effect readings. It is often the case that the sign of an effect changes for different parameter configurations. Thus it can occur that in the mean μ the effects are canceled out and a value close to zero is resulting. This could lead to the false conclusion that the parameter sensitivity is low and there is only little effect on the output. Because of that, it is reasonable to calculate the value μ^*, which uses the absolute values of the effect readings as the base for the mean.

In general a high/low value of μ indicates a likewise high/low impact of the input parameter on the output. The sigma-value σ on the other hand reveals the degree of non-linear behavior. Typically the result of Morris's method is a 2D-plot where μ is the x-axis and σ is the y-axis. Every model parameter $\mathbf{P}$ then is identified by its screening result with the point $(\mu^{\mathbf{P}}|\sigma^{\mathbf{P}})$ in the plot.

Let the output be denoted by $\mathbf{Y}$, let furthermore the Simulation be identified as the function f. The input factors are given by $X_1, X_2, \ldots, X_n$. Thus, the input to output relation is given by

$$\mathbf{Y} = f(X_1, X_2, \ldots, X_n). \tag{3.10}$$

It should be noted that the simulation could possess more input factors - than those in the set above which are chosen for examination. In that case, the missing parameters are considered as constants and hence do not appear in the variables list of the output-generating function.

Let the definition ranges of the input parameters be given by

$$X_1 \in \left[X_1^{min}, X_1^{max}\right],$$

$$X_2 \in \left[X_2^{min}, X_2^{max}\right],$$

$$\vdots$$

$$X_n \in \left[X_n^{min}, X_n^{max}\right].$$

Since in Morris's method the simulation is run on random parameter configurations, there is a need to introduce the input parameters as random variables. Let the random parameters be uniformly distributed by $\mathbf{X}_1 \sim \mathcal{U}\left(X_1^{min}, X_1^{max}\right)$, $\mathbf{X}_2 \sim \mathcal{U}\left(X_2^{min}, X_2^{max}\right)$, ... , $\mathbf{X}_n \sim \mathcal{U}\left(X_n^{min}, X_n^{max}\right)$. Let furthermore $\widetilde{x_i}$ be a realization of $\mathbf{X}_i$, where $i \in \{1, \ldots, n\}$.

For every input parameter, $r \in \mathbb{N}$ ($r \geq n$) so-called *elementary effects* are calculated. The base paper from Morris [27] recommends to choose the r-value directly proportional to the number of tested model parameters n. In order to obtain a variation value for the i-th parameter and the j-th calculation (of an elementary effect) the variable Δ is introduced. Let $(\widetilde{x_i})_j^\Delta$ be an arbitrary realization of $\mathbf{X}_i$ in the j-th calculation. Then

$$\Delta^j(\widetilde{x_i}) := |\widetilde{x_i} - (\widetilde{x_i})_j^\Delta|, \text{ where } i \in \{1, \dots, n\} \text{ and } j \in \{1, \dots, r\}. \tag{3.11}$$

Having defined the parameter realizations and Δ, it is now possible to define the j-th elementary effect of the i-th input parameter:

$$d_{ij} = d_i\left((\widetilde{x_1})_j, (\widetilde{x_2})_j, \dots, (\widetilde{x_n})_j\right) := \frac{f\left((\widetilde{x_1})_j, (\widetilde{x_2})_j, \dots, (\widetilde{x_{i-1}})_j, (\widetilde{x_i})_j + \Delta^j(\widetilde{x_i}), (\widetilde{x_{i+1}})_j, \dots, (\widetilde{x_n})_j\right)}{\Delta^j(\widetilde{x_i})} \tag{3.12}$$

It must be reconsidered that the parameter configuration is randomly drawn for each index j in the calculation of d_{ij}. The formula also makes the OAT affiliation clear, since in the simulation's parameter list only the i-th parameter is varied at a time by adding the random deviation value $\Delta^j(\widetilde{x_i})$. Having calculated the elementary effects for every parameter, the statistical moments which yield the sensitivity measures must be calculated as

$$\begin{aligned} \mu_i &:= \frac{1}{r}\sum_{j=1}^{r} d_{ij} = \frac{1}{r}\sum_{j=1}^{r} d_i\left((\widetilde{x_1})_j, (\widetilde{x_2})_j, \dots, (\widetilde{x_n})_j\right) \\ &= \frac{1}{r}\sum_{j=1}^{r} \frac{f\left((\widetilde{x_1})_j, (\widetilde{x_2})_j, \dots, (\widetilde{x_{i-1}})_j, (\widetilde{x_i})_j + \Delta^j(\widetilde{x_i}), (\widetilde{x_{i+1}})_j, \dots, (\widetilde{x_n})_j\right)}{\Delta^j(\widetilde{x_i})}, \end{aligned} \tag{3.13}$$

$$\begin{aligned} \mu_i^* &:= \frac{1}{r}\sum_{j=1}^{r} |d_{ij}| = \frac{1}{r}\sum_{j=1}^{r} \left|d_i\left((\widetilde{x_1})_j, (\widetilde{x_2})_j, \dots, (\widetilde{x_n})_j\right)\right| \\ &= \frac{1}{r}\sum_{j=1}^{r} \left|\frac{f\left((\widetilde{x_1})_j, (\widetilde{x_2})_j, \dots, (\widetilde{x_{i-1}})_j, (\widetilde{x_i})_j + \Delta^j(\widetilde{x_i}), (\widetilde{x_{i+1}})_j, \dots, (\widetilde{x_n})_j\right)}{\Delta^j(\widetilde{x_i})}\right|, \end{aligned} \tag{3.14}$$

$$\sigma_i := \sqrt{\frac{1}{r-1}\sum_{j=1}^{r} (d_{ij} - \mu_i)^2} \tag{3.15}$$

In the equation for σ_i it is very important to recall that it holds true that d_{ij} coincides with the instance originating in equation 3.13, hence it is no newly calculated effect value.

3.3 Main And Interaction Effects On Extreme Values

The approach to examine the main and interaction effects is based on the so-called *Design of Experiments* (DoE) methodology [17]. The DoE method tries to find the influential model parameters but also addresses the planning of simulation runs in order to obtain the desired information. The authors from [17] suggest using a full-factorial design on the extreme values of the input parameters. Hence it holds true that one parameter can only obtain two values, namely the minimum and the maximum. When the number of examined parameters in denoted by n, the simulation is then run with every of the 2^n possible parameter combinations. The result of the method is given by a linear regression on the averaged output readings for the simulation runs on the extrema.

By that, a main effect of an input parameter is given by the regression line on the summed up and averaged output measurements for the two cases where in the parameter configuration the value appears as minimum respectively maximum. Hence, the resulting line which is the result of the approach is drawn between the point of the mean output where the parameter is minimum and the point of the mean output where the parameter is maximum. All other parameters will obtain any of their two possible readings. This formalism makes also clear that for the calculation of any main effect the complete result set of the input to output mapping obtained by the 2^n simulation runs is needed. The difference from one parameter to another is thus the correct equipartition splitting of the result set based on the value occurrence (minimum/maximum).

In order to examine interaction effects it is necessary to vary at least two parameters at a time. In consideration of the problem of "curse of dimensionality" it was chosen to only analyze binary parameter relations. Therefore, for the arbitrary parameters X_i and X_j the process of obtaining the linear relation is firstly done by keeping X_j to its minimum value whilst X_i is varied from its minimum to maximum. The same procedure is then repeated while X_j is kept to its maximum reading. By that, the resulting lines reveal whether there is any interaction between the two parameters synergistically influencing the dedicated output quantity.

Let the parameters be denoted by $X_1, X_2, \dots, X_n$ and only obtain the values from the following sets:

$$X_1 \in \{X_1^{min}, X_1^{max}\},$$

$$X_2 \in \{X_2^{min}, X_2^{max}\},$$

$$\vdots$$

$$X_n \in \{X_n^{min}, X_n^{max}\}.$$

Let furthermore the output-generating simulation be given by

$$\mathbf{Y} = f(X_1, X_2, \dots, X_n). \tag{3.16}$$

In order to obtain the main effect of a parameter the simulation must be run with the parameter as minimum and then as maximum. Let $\mathbf{Y}_i^{min}$ and $\mathbf{Y}_i^{max}$ be defined as the sum of output readings where the i-th parameter occurs solely as minimum respectively maximum:

$$\mathbf{Y}_i^{min} := \sum_{* \begin{cases} \widetilde{X_1} \in \{X_1^{min}, X_1^{max}\}, \\ \widetilde{X_2} \in \{X_2^{min}, X_2^{max}\}, \\ \vdots \\ \widetilde{X_{i-1}} \in \{X_{i-1}^{min}, X_{i-1}^{max}\}, \\ \widetilde{X_{i+1}} \in \{X_{i+1}^{min}, X_{i+1}^{max}\}, \\ \vdots \\ \widetilde{X_n} \in \{X_n^{min}, X_n^{max}\} \end{cases}} f(\widetilde{X_1}, \widetilde{X_2}, \dots, X_i^{min}, \dots, \widetilde{X_n}). \tag{3.17}$$

$$\mathbf{Y}_i^{max} := \sum_{*} f(\widetilde{X_1}, \widetilde{X_2}, \dots, X_i^{max}, \dots, \widetilde{X_n}). \tag{3.18}$$

Since the number of simulation runs is 2^n and since a parameter can only obtain two different values, the parameter occurs half the number of simulations as minimum respectively maximum. Thus, the averaged output for a single input variable on the minimum and maximum is given by

$$\overline{\mathbf{Y}_i^{min}} = \frac{\mathbf{Y}_i^{min}}{2^{n-1}} \quad \text{and} \tag{3.19}$$

$$\overline{\mathbf{Y}_i^{max}} = \frac{\mathbf{Y}_i^{max}}{2^{n-1}}. \tag{3.20}$$

The linear regression for the main effect of the single parameter X_i is calculated as

$$g_i(x) = \frac{\overline{\mathbf{Y}_i^{max}} - \overline{\mathbf{Y}_i^{min}}}{X_i^{max} - X_i^{min}} \left(x - X_i^{min}\right) + \overline{\mathbf{Y}_i^{min}}, \text{ where } x \in \left[X_i^{min}, X_i^{max}\right]. \tag{3.21}$$

For the interaction effect it is necessary to define the summed up outputs for the setting of two variables being minimum or maximum. Hence

$$\mathbf{Y}^{i|min}_{j|min} := \sum_{** \begin{cases} \widetilde{X_1} \in \{X_1^{min}, X_1^{max}\}, \\ \widetilde{X_2} \in \{X_2^{min}, X_2^{max}\}, \\ \vdots \\ \widetilde{X_{i-1}} \in \{X_{i-1}^{min}, X_{i-1}^{max}\}, \\ \widetilde{X_{i+1}} \in \{X_{i+1}^{min}, X_{i+1}^{max}\}, \\ \vdots \\ \widetilde{X_{j-1}} \in \{X_{j-1}^{min}, X_{j-1}^{max}\}, \\ \widetilde{X_{j+1}} \in \{X_{j+1}^{min}, X_{j+1}^{max}\}, \\ \vdots \\ \widetilde{X_n} \in \{X_n^{min}, X_n^{max}\} \end{cases}} f(\widetilde{X_1}, \widetilde{X_2}, \ldots, X_i^{min}, \ldots, X_j^{min}, \ldots, \widetilde{X_n}), \tag{3.22}$$

$$\mathbf{Y}^{i|min}_{j|max} := \sum_{**} f(\widetilde{X_1}, \widetilde{X_2}, \ldots, X_i^{min}, \ldots, X_j^{max}, \ldots, \widetilde{X_n}), \tag{3.23}$$

$$\mathbf{Y}^{i|max}_{j|min} := \sum_{**} f(\widetilde{X_1}, \widetilde{X_2}, \ldots, X_i^{max}, \ldots, X_j^{min}, \ldots, \widetilde{X_n}), \tag{3.24}$$

$$\mathbf{Y}^{i|max}_{j|max} := \sum_{**} f(\widetilde{X_1}, \widetilde{X_2}, \ldots, X_i^{max}, \ldots, X_j^{max}, \ldots, \widetilde{X_n}). \tag{3.25}$$

With two constraints on the selection of parameters in the sum of outputs there are only $\frac{1}{4}$ addends of the number of simulation runs. Thus, the averaged outputs are given by

$$\overline{\mathbf{Y}^{i|min}_{j|min}} = \frac{\mathbf{Y}^{i|min}_{j|min}}{2^{n-2}}, \tag{3.26}$$

$$\overline{\mathbf{Y}^{i|min}_{j|max}} = \frac{\mathbf{Y}^{i|min}_{j|max}}{2^{n-2}}, \tag{3.27}$$

$$\overline{\mathbf{Y}^{i|max}_{j|min}} = \frac{\mathbf{Y}^{i|max}_{j|min}}{2^{n-2}} \quad \text{and} \tag{3.28}$$

$$\overline{\mathbf{Y}^{i|max}_{j|max}} = \frac{\mathbf{Y}^{i|max}_{j|max}}{2^{n-2}}. \tag{3.29}$$

The interaction effect between the parameters X_i and X_j is represented by two regression lines. The first line represents the behavior of parameter X_i which is varied from minimum to maximum whilst parameter X_j is kept to its minimum. The second line represents the behavior of parameter X_i when parameter X_j is kept to its maximum. This convention also illustrates that the interaction between X_i and X_j versus the interaction between X_j and X_i is a completely different concern and will result in two differing plots.

The regression is carried out as follows:

$$g_i^{j|min}(x) = \frac{\overline{\mathbf{Y}_{j|min}^{i|max}} - \overline{\mathbf{Y}_{j|min}^{i|min}}}{X_i^{max} - X_i^{min}} \left(x - X_i^{min}\right) + \overline{\mathbf{Y}_{j|min}^{i|min}}, \text{ where } x \in \left[X_i^{min}, X_i^{max}\right] \quad \text{and} \tag{3.30}$$

$$g_i^{j|max}(x) = \frac{\overline{\mathbf{Y}_{j|max}^{i|max}} - \overline{\mathbf{Y}_{j|max}^{i|min}}}{X_i^{max} - X_i^{min}} \left(x - X_i^{min}\right) + \overline{\mathbf{Y}_{j|max}^{i|min}}, \text{ where } x \in \left[X_i^{min}, X_i^{max}\right]. \tag{3.31}$$

3.4 Partial (Rank) Correlation Coefficients (PCC/PRCC) and Standardized (Rank) Regression Coefficients (SRC/SRRC)

The domain of calculating correlation measures studies the amount of linearity between variables in order to reveal a mutual influence. The term correlation is herein explicitly attributed to a linear functional dependency. Classical correlation techniques are only capable of examining binary parameter relations. Because of that, the approaches to calculate the so-called *partial correlation coefficients* (PCC) and *standardized regression coefficients* (SRC) are able to examine multiple input variables influencing a single output reading [17]. Thus, the PCC/SRC are also used as a tool for SA. For the case that a non-linear relationship between input and output is expected, the so-called *rank versions* of the partial correlation coefficients and standardized regression coefficients PRCC and SRRC are used to measure the effect strength. The data base here is not given by the real data but by the rank (index of an item in the sorted list) of every date. The coefficients then are calculated based on ranks rather than on the true values. However both methods rely on the Pearson correlation coefficient (CC) and on linear regression as their base for the further calculation of the coefficients [28].

The approaches of PCC/PRCC and SRC/SRRC make no statements about the planning of experiments. They are supposed to be applied when the data of input to output mapping is already available. That is why it is supposed that the data was generated by the means of drawing each parameter configuration randomly from a uniform distribution over the parameter space. Utilizing a uniform distribution is very important since the calculated sensitivity values heavily depend on the initial input parameter distribution.

PCC and SRC deliver similar sensitivity measures. However the methods differ slightly in their addressed objectives. The SRC approach tries to find the relative importance of the input parameters [28]. This is done by calculating the regression coefficients, which are the partial derivatives of the linear regression model considering the input variables. Nevertheless a relevant drawback is the unreliability issue originating from the order/magnitude of a single input reading (for example meter to millimeter) making a regression coefficient inappropriate as a relative

importance measure [28]. Because of that, the regression coefficients are standardized by being multiplied with the ratio of the standard deviations between the input and output variables [17]. This formulation makes the SRC a direct criterion for the relative parameter importance.

The PCC uses the sample correlation as a measure for the strength of the linear relationship between a single input and the output. In order to build the linear model the input parameter that has the strongest influence could be used as a starting point [28]. The PCC therefore represents a measure for the singular connection amongst two variables when the relation cannot be explained for the two ones regarding all remaining variables [28]. Hence the PCC indicates the goodness of adding a particular parameter to an existing linear model.

Let the n input parameters be given as random variables:

$$\begin{gathered} X_1 \sim \mathcal{U}\left(X_1^{min}, X_1^{max}\right), \\ X_2 \sim \mathcal{U}\left(X_2^{min}, X_2^{max}\right), \\ \vdots \\ X_n \sim \mathcal{U}\left(X_n^{min}, X_n^{max}\right). \end{gathered}$$

Furthermore let $(\widetilde{x_1})_i, (\widetilde{x_2})_i, \ldots, (\widetilde{x_n})_i$ denote the respective i-th realization of $X_1, X_2, \ldots, X_n$. The following elaborations will use the matrix notation according to Iman et al. [28]. The matrix M_{IO} maps the input configuration to the output readings. Let y_k be the k-th output generated by the simulation run with the k-th random parameter configuration. It is supposed that m parameter configurations were drawn. Hence,

$$M_{\text{IO}} := \left(\begin{array}{cccc|c} (\widetilde{x_1})_1 & (\widetilde{x_2})_1 & \ldots & (\widetilde{x_n})_1 & y_1 \\ (\widetilde{x_1})_2 & (\widetilde{x_2})_2 & \ldots & (\widetilde{x_n})_2 & y_2 \\ \vdots & \vdots & & \vdots & \vdots \\ (\widetilde{x_1})_m & (\widetilde{x_2})_m & \ldots & (\widetilde{x_n})_m & y_m \end{array}\right) \tag{3.32}$$

For the calculation of the PCC/SRC, obtaining the empirical Pearson correlation coefficient (CC) between parameters X_i and X_j is mandatory. Thus, let r_{ij} denote the CC by

$$r_{ij} := \frac{\sum_{l=1}^{m} \left((\widetilde{x}_i)_l - \overline{x_i}\right)\left((\widetilde{x}_j)_l - \overline{x_j}\right)}{\sqrt{\sum_{l=1}^{m} \left((\widetilde{x}_i)_l - \overline{x_i}\right)^2 \cdot \sum_{l=1}^{m} \left((\widetilde{x}_j)_l - \overline{x_j}\right)^2}}, \quad i, j \in \{1, \ldots, n\}, \tag{3.33}$$

where

$$\overline{x_i} := \frac{1}{m}\sum_{l=1}^{m}(\widetilde{x_i})_l. \tag{3.34}$$

The definition 3.33 makes also clear that $r_{ii} = 1$, and that $r_{ij} = r_{ji}$. Having defined M_{IO} and r_{ij} it is necessary to define the so-called *correlation matrix* [28] which holds all possible Pearson correlation coefficients including the values of r_{yi} which stand for the correlation of the output y with a certain input parameter X_i. The matrix is given by

$$C := \left(\begin{array}{ccccc|c} 1 & r_{12} & r_{13} & \dots & r_{1n} & r_{1y} \\ r_{21} & 1 & r_{23} & \dots & r_{2n} & r_{2y} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & \dots & 1 & r_{ny} \\ \hline r_{y1} & r_{y2} & \dots & \dots & r_{yn} & 1 \end{array}\right), \tag{3.35}$$

where

$$r_{yi} := \frac{\sum_{l=1}^{m}(y_l - \overline{y})\left((\widetilde{x_j})_l - \overline{x_j}\right)}{\sqrt{\sum_{l=1}^{m}(y_l - \overline{y})^2 \cdot \sum_{l=1}^{m}\left((\widetilde{x_j})_l - \overline{x_j}\right)^2}}, \quad i \in \{1, \dots, n\}, \tag{3.36}$$

and

$$\overline{y} := \frac{1}{m}\sum_{i=1}^{m} y_i. \tag{3.37}$$

The definition of r_{ij} makes the correlation matrix C symmetric. The dashed lines in matrix C indicate a partitioning that is in the following denoted as

$$C = \left(\begin{array}{ccccc|c} 1 & r_{12} & r_{13} & \dots & r_{1n} & r_{1y} \\ r_{21} & 1 & r_{23} & \dots & r_{2n} & r_{2y} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & \dots & 1 & r_{ny} \\ \hline r_{y1} & r_{y2} & \dots & \dots & r_{yn} & 1 \end{array}\right) = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & 1 \end{array}\right). \tag{3.38}$$

To derive the PCC/SRC values the inverse of C needs to be calculated. It is well known [28] that C^{-1} can be written as

$$C^{-1} = \begin{pmatrix} (C_{11} - C_{12}C_{21})^{-1} & -C_{11}^{-1}C_{12}\left(1 - C_{21}C_{11}^{-1}C_{12}\right)^{-1} \\ -\left(C_{11}^{-1}C_{12}\left(C_{21}C_{11}^{-1}C_{12}\right)^{-1}\right)^T & \left(1 - C_{21}C_{11}^{-1}C_{12}\right)^{-1} \end{pmatrix}. \tag{3.39}$$

Since PCC and SRC are regression based methods - hence the output values are regressed on the input values - the so-called *coefficient of determination*, which is the variance of the dependent parameter that is predictable from the independent parameters, is given by

$$R_y^2 = C_{21}C_{11}^{-1}C_{12}. \tag{3.40}$$

Let $B = C_{11}^{-1} C_{12}$, then C^{-1} can be written as

$$C^{-1} = \begin{pmatrix} (C_{11} - C_{12}C_{21})^{-1} & -B/\left(1 - R_y^2\right) \\ -B^T/\left(1 - R_y^2\right) & 1/\left(1 - R_y^2\right) \end{pmatrix}. \tag{3.41}$$

A closer look to the matrix B reveals that it coincides with the SRC, the desired regression solution. Thus

$$\begin{pmatrix} SRC_{X_1} \\ SRC_{X_2} \\ \dots \\ SRC_{X_n} \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ \dots \\ B_n \end{pmatrix} = C_{11}^{-1} C_{12}, \tag{3.42}$$

where B_i is the i-th entry in the vector B and $i \in \{1, \dots, n\}$.

In the submatrix $(C_{11} - C_{12}C_{21})^{-1}$ the diagonal entries contain the coefficients of determination of the single regression of a parameter X_i on the output Y and the remaining parameters by the relation $1/(1 - R^2)$. In particular, a single entry from the diagonal in the i-th column of the submatrix is given by $1/\left(1 - R_{X_i}^2\right)$. Thus C^{-1} expands to

$$C^{-1} = \left(\begin{array}{cccc|c} 1/\left(1 - R_{X_1}^2\right) & c_{12} & \dots & c_{1n} & -B_1/\left(1 - R_Y^2\right) \\ c_{21} & 1/\left(1 - R_{X_2}^2\right) & \dots & c_{2n} & -B_2/\left(1 - R_Y^2\right) \\ \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & 1/\left(1 - R_{X_n}^2\right) & -B_n/\left(1 - R_Y^2\right) \\ \hline -B_1/\left(1 - R_Y^2\right) & -B_2/\left(1 - R_Y^2\right) & \dots & -B_n/\left(1 - R_Y^2\right) & 1/\left(1 - R_Y^2\right) \end{array}\right). \tag{3.43}$$

The equation 3.43 makes it possible to directly obtain the PCC value for the input parameter X_i:

$$\begin{aligned} PCC_{X_i} &:= -c_{iy}/\left(c_{ii}c_{yy}\right)^{\frac{1}{2}} \\ &= \frac{B_i/\left(1 - R_Y^2\right)}{\sqrt{\left(1/\left(1 - R_{X_i}^2\right)\right)\left(1/\left(1 - R_Y^2\right)\right)}} \\ &= B_i \sqrt{\frac{1 - R_{X_i}^2}{1 - R_Y^2}},\ i \in \{1, \dots, n\}. \end{aligned} \tag{3.44}$$

In order to calculate the rank version of the partial correlation coefficients and the standardized regression coefficients, it is necessary to transform the input data according to their ranks. The original data matrix that holds the input to output mapping was given by definition 3.32. Each column in the matrix M_{IO} represents the values a parameter (input or output) can obtain. For the purpose of transforming every single entry in the matrix to its rank, the values need to be sorted.

Let the ascending sorting be denoted by the transformation:

$$\begin{pmatrix} (\widetilde{x_i})_1 \\ (\widetilde{x_i})_2 \\ \vdots \\ (\widetilde{x_i})_m \end{pmatrix} \longrightarrow \begin{pmatrix} (\widetilde{x_i})_{\boxed{1}} \\ (\widetilde{x_i})_{\boxed{2}} \\ \vdots \\ (\widetilde{x_i})_{\boxed{m}} \end{pmatrix}. \tag{3.45}$$

The rank of a single parameter value is given by the index in the sorted vector. Let the index be obtained by the function $Rank$:

$$Rank\left((\widetilde{x_i})_j\right) = k, \text{ when } \begin{pmatrix} \vdots \\ (\widetilde{x_i})_j \\ \vdots \end{pmatrix} \xrightarrow{\text{match}} \begin{pmatrix} \vdots \\ (\widetilde{x_i})_{\boxed{k}} \\ \vdots \end{pmatrix} \text{ and } (\widetilde{x_i})_j = (\widetilde{x_i})_{\boxed{k}}. \tag{3.46}$$

In the case that an element in vector $\left((\widetilde{x_i})_{\boxed{1}}\ (\widetilde{x_i})_{\boxed{2}}\ \dots\ (\widetilde{x_i})_{\boxed{m}}\right)^T$ occurs multiple times, for example $(\widetilde{x_i})_{\boxed{k_1}} = (\widetilde{x_i})_{\boxed{k_2}} = \ldots = (\widetilde{x_i})_{\boxed{k_p}}$, then it is defined that the Rank is calculated as

$$\begin{aligned} &Rank\left((\widetilde{x_i})_{j_1}\right) = Rank\left((\widetilde{x_i})_{j_2}\right) = \ldots = Rank\left((\widetilde{x_i})_{j_p}\right) = \frac{k_1 + k_2 + \ldots + k_p}{p}, \\ &\text{where } (\widetilde{x_i})_{j_1} = (\widetilde{x_i})_{j_2} = \ldots = (\widetilde{x_i})_{j_p} = (\widetilde{x_i})_{\boxed{k_1}} = (\widetilde{x_i})_{\boxed{k_2}} = \ldots = (\widetilde{x_i})_{\boxed{k_p}}. \end{aligned} \tag{3.47}$$

The rank version of the matrix M_{IO} is then given by

$$(M_{\text{IO}})^{Rank} := \left(\begin{array}{cccc|c} Rank\left((\widetilde{x_1})_1\right) & Rank\left((\widetilde{x_2})_1\right) & \dots & Rank\left((\widetilde{x_n})_1\right) & Rank\left(y_1\right) \\ Rank\left((\widetilde{x_1})_2\right) & Rank\left((\widetilde{x_2})_2\right) & \dots & Rank\left((\widetilde{x_n})_2\right) & Rank\left(y_2\right) \\ \vdots & \vdots & & \vdots & \vdots \\ Rank\left((\widetilde{x_1})_m\right) & Rank\left((\widetilde{x_2})_m\right) & \dots & Rank\left((\widetilde{x_n})_m\right) & Rank\left(y_m\right) \end{array}\right). \tag{3.48}$$

The further procession to calculate the partial rank correlation coefficients (PRCC) and the standardized rank regression coefficients is equivalent to the calculation of the PCC/SRC values, however the base in order to yield the matrix C is given by the matrix $(M_{\text{IO}})^{Rank}$.

3.5 Sobol's Method

The approach of Sobol represents a member of the so-called *variance decomposition SA methods* [17]. The main idea is to decompose the output's variance such that it is constructed of variance contributions from the input factors. The total variance is attributed not only to the sum of variances of the single input parameters themselves but also to interaction effects between them [29]. The sensitivity measure for a certain parameter is called *main effect* and coincides with the variance contribution in the total output variance formula. Thus, the variance can be directly used in SA as a relative parameter importance measure [29]. This measure is also called *first-order Sobol index* or *first-order sensitivity index* [8]. The higher-order indices denote the interaction effects. Herein all variances of the subsets from the parameter space contribute to the total output variance. Hence, when the number of input parameters is n, Sobol's method calculates the first-order up to the n-th-order effect indices.

Let the input to output (scalar) mapping be given by

$$\mathbf{Y} = f\left(X_1, X_2, \ldots, X_n\right). \tag{3.49}$$

The input parameters $X_1, X_2, \ldots, X_n$ are considered to be random variables and being identically independently distributed by the (not-explicitly defined) density $p(x)$. Thus, $X_1 \sim p(x), X_2 \sim p(x), \ldots, X_n \sim p(x)$. In the following, the notation will stick to the formulation using random variables. Nevertheless, when considering concrete input parameter configurations (for example drawn from a uniform distribution) the simulation is run with, the expectation operator $\mathbb{E}$ should be replaced by the sample mean, and the variance operator $\mathbb{V}$ by the sample variance.

The base of Sobol is the decomposition of the target function $\mathbf{Y}$ in terms of increasing dimensionality [8]:

$$\begin{aligned}\mathbf{Y} = f\left(X_1, X_2, \ldots, X_n\right) = f_0 &+ \sum_{i=1}^{n} f_i(X_i) \\ &+ \sum_{1 \leq i < l \leq n} f_{il}\left(X_i, X_l\right) \\ &+ \ldots + f_{1,2,\ldots,n}\left(X_1, \ldots, X_n\right).\end{aligned} \tag{3.50}$$

Since it is supposed that all parameters are independent, the output variance is given by

$$\mathbb{V}\left[\mathbf{Y}\right] = \mathbb{V}\left[f\left(X_1, X_2, \ldots, X_n\right)\right] = \sum_{i=1}^{n} V_i + \sum_{1 \leq i < l \leq n} V_{il} + \ldots + V_{1,2,\ldots,n}, \tag{3.51}$$

where

$$\begin{aligned}
V_i &= \mathbb{V}\left[f_i\right] = \mathbb{V}_{X_i}\left[\mathbb{E}_{X_{-i}}\left[\mathbf{Y}|X_i\right]\right] \\
V_{il} &= \mathbb{V}\left[f_{il}\right] = \mathbb{V}_{X_{il}}\left[\mathbb{E}_{X_{-il}}\left[\mathbf{Y}|X_i, X_l\right]\right] - V_i - V_l \\
V_{ilk} &= \mathbb{V}\left[f_{ilk}\right] = \mathbb{V}_{X_{ilk}}\left[\mathbb{E}_{X_{-ilk}}\left[\mathbf{Y}|X_i, X_l, X_k\right]\right] - V_{il} - V_{ik} - V_{lk} - V_i - V_l - V_k \\
&\vdots \\
V_{1,2,\ldots,n} &= \mathbb{V}\left[f_{1,2,\ldots,n}\right] = \mathbb{V}\left[\mathbf{Y}\right] - \sum_{i=1}^{n} V_i - \sum_{1\leq i<l\leq n} V_{il} - \ldots - \sum_{1\leq i_1<\ldots<i_{n-1}\leq n} V_{i_1,\ldots,i_{n-1}}.
\end{aligned} \tag{3.52}$$

Let the effect indices be denoted by $S_i, S_{il}, \ldots, S_{1,2,\ldots,n}$. The first-order effect indices are defined as

$$S_i := \frac{\mathbb{V}_{X_i}\left[\mathbb{E}_{X_{-i}}\left[\mathbf{Y}|X_i\right]\right]}{\mathbb{V}\left[\mathbf{Y}\right]} = \frac{\mathbb{V}_i\left[\mathbb{E}_{-i}\left[\mathbf{Y}|X_i\right]\right]}{\mathbb{V}\left[\mathbf{Y}\right]} = \frac{V_i}{\mathbb{V}\left[\mathbf{Y}\right]}, \tag{3.53}$$

where $\mathbb{V}_{X_i} = \mathbb{V}_i$ denotes the parameter-specific variance, hence the variance over all values the random variable X_i can obtain. In the notation for the expectation value $\mathbb{E}_{X_{-i}} = \mathbb{E}_{-i}$, the $-i$ denotes all parameters except X_i.

The higher-order effect indices are further given by

$$\begin{aligned}
S_{il} &:= \frac{V_{il}}{\mathbb{V}\left[\mathbf{Y}\right]}, \\
S_{ilk} &:= \frac{V_{ilk}}{\mathbb{V}\left[\mathbf{Y}\right]}, \\
&\vdots \\
S_{i_1,\ldots,i_s} &:= \frac{V_{i_1,\ldots,i_s}}{\mathbb{V}\left[\mathbf{Y}\right]}, \\
&\vdots \\
S_{1,\ldots,n} &:= \frac{V_{1,\ldots,n}}{\mathbb{V}\left[\mathbf{Y}\right]}.
\end{aligned} \tag{3.54}$$

The equation 3.52 and the definitions 3.53 and 3.54 yield that the effect indices sum up to 1. Thus

$$1 = \sum_{i=1}^{n} S_i + \sum_{1\leq i<l\leq n} S_{il} + \ldots + S_{1,2,\ldots,n}. \tag{3.55}$$

The authors from [30] also suggest calculating the so-called *total effect index*. This measure can be used as a parameter screening approach [8]. It is defined as the sum over all variance contributions

a parameter possesses in the total output variance. Hence

$$\begin{aligned} ST_i := & \sum_{\substack{j \,|\, \text{index } i \text{ contained in} \\ \text{index combination } j}} S_j \\ &= 1 - \frac{\mathbb{V}_{X_{-i}}\left[\mathbb{E}_{X_i}\left[\mathbf{Y}|X_{-i}\right]\right]}{\mathbb{V}\left[\mathbf{Y}\right]} \\ &= 1 - \frac{\mathbb{V}_{X_{-i}}\left[\mathbb{E}_{X_i}\left[\mathbf{Y}|X_1, X_2, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n\right]\right]}{\mathbb{V}\left[\mathbf{Y}\right]} \\ &= 1 - S_{-i} \end{aligned} \tag{3.56}$$

The formula 3.56 reveals that an essential figure is the difference $|ST_i - S_i|$ between the total effect index and the main effect index. It indicates the amount of interaction effects [8] [31].

3.6 Extended Fourier Amplitude Sensitivity Test (EFAST)

The extended Fourier amplitude sensitivity test (EFAST) produces the same sensitivity measures as Sobol's method [17]: first-order and total effect indices. Calculated indices from Sobol and EFAST converge to the same readings when the number of simulation samples goes to infinity. However, EFAST is more efficient because for the same accuracy EFAST requires substantially less samples than Sobol's method [32]. Like Sobol, EFAST also belongs to the variance decomposition approaches. Here, Fourier series expansion is used to obtain the sensitivity indices which indicate the influence of input parameters to a specific output. EFAST is an extension of the classical FAST approach. The extension was established by Saltelli [32] who suggests to also calculate the total effect indices which sum up all variance contributions of a single parameter (including interactions) appearing in the variance output formula. The following elaborations use the notation by Saltelli [32].

Let the input to output relationship be given by the function f:

$$\mathbf{Y} = f\left(X_1, X_2, \ldots, X_n\right). \tag{3.57}$$

For the sake of simplicity it is assumed that all input parameters can only take values in $[0, 1] \subseteq \mathbb{R}$. Hence the n-dimensional input space is given by

$$K^n = \left\{\mathbf{X} = \left(X_1, \ldots, X_n\right) | 0 \leq X_i \leq 1, i = 1, \ldots, n\right\} \tag{3.58}$$

For the exploration of the output's variance the mono-dimensional Fourier decomposition needs to systematically explore the input space K^n. Thus, the decomposition is done along the set of

parametric exploration curves [32] defined by

$$X_i(s) = \frac{1}{2} + \frac{1}{\pi} \arcsin\left(\sin \omega_i s\right) \forall i = 1, \ldots, n, \tag{3.59}$$

where $s \in (-\infty, +\infty)$ is the variation variable where when s changes all parameters simultaneously are varied, systematically exploring K^n. The values $\{\omega_i\}, i = 1, \ldots, n$ constitute the set of angular frequencies. Each frequency is selected beforehand individually for every single parameter. Thus, each X_i oscillates with the frequency ω_i resulting in oscillations in the output reading which then can be associated with the input oscillation frequencies [32]. The selection of frequencies is slightly complicated. It depends on the number of parameters as well as the sufficient number of samples. Since only the main idea is meant to be presented here, for more information about determining the sample size and selecting the frequencies it is recommend to sight the base paper of EFAST by Saltelli [32].

The sensitivity measure that results is coinciding with the strength of the amplitude an input parameter evokes in the output curve. An element of the set of oscillation frequencies requires not to be linearly combinable of the other ones. The sufficient condition to ensure this property is given by

$$\sum_{i=1}^{n} r_i \omega_i \neq 0, -\infty < r_i < +\infty. \tag{3.60}$$

Let the output that is generated by the s-th variation of the input parameters be denoted by

$$\mathbf{Y}_s = f(s) := f\left(X_1(s), X_2(s), \ldots, X_n(s)\right) \tag{3.61}$$

Fourier series expansion of $f(s)$ yields

$$f(s) = \sum_{j=-\infty}^{+\infty} \left(A_j \cos js + B_j \sin js\right), \tag{3.62}$$

where A_j and B_j represent the Fourier coefficients. They are defined as

$$A_j := \frac{1}{2\pi} \int_{-\pi}^{+\pi} f(s) \cos js \, \mathrm{ds} \tag{3.63}$$

and

$$B_j := \frac{1}{2\pi} \int_{-\pi}^{+\pi} f(s) \sin js \, \mathrm{ds} \tag{3.64}$$

where $j \in \mathbb{Z}$. The Fourier series' spectrum is by definition:

$$\Lambda_j = A_j^2 + B_j^2, j \in \mathbb{Z}. \tag{3.65}$$

Obtaining the spectrum for the fundamental frequency ω_i and its multiples $p\omega_i$ yields the partial variance coming from the concrete input parameter X_i. Let it be denoted by $\hat{D}_i$. Hence

$$\hat{D}_i = \sum_{p\in\mathbb{Z},p\neq 0} \Lambda_{p\omega_i} = 2\sum_{p=1}^{\infty} \Lambda_{p\omega_i}. \tag{3.66}$$

The sum over all $\Lambda_j, j \in \mathbb{Z}\backslash\{0\}$ coincides with the total variance. Let this quantity be denoted by

$$\hat{D} := \sum_{j\in\mathbb{Z},j\neq 0} \Lambda_j = 2\sum_{j=1}^{\infty} \Lambda_j. \tag{3.67}$$

Then, a parameter's main effect is given by the ratio of its variance and the total variance. Thus

$$\hat{S}_i^{FAST} = \frac{\hat{D}_i}{\hat{D}}. \tag{3.68}$$

Saltelli [32] suggests then calculating the total effect index for the parameters using the following approach: For the i-th parameter the frequencies are chosen such that ω_i is assigned to X_i and $\omega_{(i')}$ to all other parameters. When the spectrum for the frequencies ω_i and $\omega_{(i')}$ is evaluated, the Variance $\hat{D}_{(-i)}$ can be obtained, where the index $(-i)$ stands for all parameters except the i-th one. The total effect index is then obtained by

$$\hat{S}T_i^{FAST} = \frac{\hat{D} - \hat{D}_{(-i)}}{\hat{D}} = 1 - \frac{\hat{D}_{(-i)}}{\hat{D}}. \tag{3.69}$$

Chapter 4

Sensitivity Analysis of Plant Models

In this chapter the sensitivity analysis of two plant models is carried out using the developed plugin.

4.1 Beech Tree

The beech tree is a simplified plant model. It contains the modeling of the primary and secondary growth (length and width) of young beeches (Fagus sylvatica). A light model that represents a light source and calculates the absorbed light of the leaves including shadowing is incorporated. Hence, an implementation of the photosynthesis process can also be found. Furthermore, the transport of biomass within the plant organs (leaves, shoot nodes, stem) is comprised. The model was developed by Ole Kniemeyer and an early version was published in his dissertation [33]. As known from chapter 2, a plant model must be prepared for SA. However, it was chosen to omit a description of the preparation since the basic concept was thoroughly described in chapter 2. The original source code as well as the modified version can be found on the supplementary CD. The main objective is to test the developed sensitivity plugin and to examine the beech tree, which may yield some insightful results. Nevertheless, due to the amount of data that is processed by the SA functions and the number of results they produce, only a selection of parameters can be analyzed. Therefore, seven input parameters were chosen regarding two outputs, namely the height of the tree and the total carbon production. Since the beech model's source code in many places contains randomness (Gaussian distribution), for a fixed setting the output is generated by averaging over 100 simulation runs with identical input parameter configuration. The SA of the tree is carried out for one and the same fixed point in its lifetime: after ten years of growth. In table 4.1 the input parameters are described including their initial readings and variation ranges. For local sensitivity analysis it was determined that the initial parameter value is varied with a deviation of $\pm 20\%$.

parameter name	description	initial value	variation range
branching_angle	The angle between the leaf and the branch it is attached to is controlled by the parameter *branching_angle*. As a side effect, the parameter also indirectly governs the angle between a light ray and the leaf surface.	$64°$	$35 \ldots 85°$
createShort_ vitality_threshold	The emergence of new shoot parts from a bud is influenced by the parameter *createShort_vitality_threshold*. If the vitality of the bud is below the threshold, only a short branch will be created. Otherwise - depending on the vitality - a certain number of newly created internodes constitute the newly generated shoot.	2.2	$0.5 \ldots 5.0$
leaf_area	The area a newly created leaf possesses is directly controlled by the parameter *leaf_area*. In the model it is supposed that a new leaf instantly arises without any growth occurring to create it.	$0.002m^2$	$0.0005 \ldots 0.04m^2$
VIT_MAX	The parameter *VIT_MAX* is the maximal vitality value a bud can obtain. It is a dimensionless reading maintained and assigned to each bud depending on its position in the order hierarchy. The vitality is in a functional dependency to the number of branches that are created.	8	$2 \ldots 20$

Table 4.1: Overview of the examined beech's input parameters

parameter name	description	initial value	variation range
light_days	The number of days per year with full light is given by the parameter *light_days*. That means, if there are ten hours of light per day, the reading will be: light_days = $\frac{10}{24} \times 365 = 152$.	152	$1 \dots 365$
PPFD_FACTOR	The parameter *PPFD_FACTOR* represents the variation factor for the so-called *photosynthetically active photon flux density* - which is the intensity value for the photosynthetically relevant light and also the amount of light available. In order to vary the amount of available light the variation of the photosynthetically active photon flux density is done by simply multiplying the value with the variation factor inducing a linear change.	1.0	$0.22 \dots 2.17$
EFFICIENCY	The ability of the tree to metabolize the available light is affected by the *EFFICIENCY* parameter. That means a plant can always only use a fraction of the incoming energy. This value is given by the parameter.	0.07	$0.005 \dots 0.12$

Table 4.1: Overview of the examined beech's input parameters (cont'd)

4.1.1 Local Sensitivity Analysis

parameter value	tree height	total carbon production
branching_angle_min	8.53	-19.85
branching_angle_max	-9.66	6.43
createShort_vitality_threshold_min	-1.87	-1.41
createShort_vitality_threshold_max	-2.93	-4.59
leaf_area_min	-13.57	-28.92
leaf_area_max	2.21	28.30
VIT_MAX_min	-22.47	-40.97
VIT_MAX_max	14.66	62.94
light_days_min	-12.40	-47.63
light_days_max	1.68	65.85
PPFD_FACTOR_min	-16.13	-48.89
PPFD_FACTOR_max	-0.31	68.20
EFFICIENCY_min	-15.51	-50.15
EFFICIENCY_max	1.83	76.55

Table 4.2: Results for the local sensitivity analysis of the beech tree

In table 4.2 the results of the local SA can be found. It should be recalled that the subscripts *min* and *max* stand for the particular extreme value of a parameter in the examination range. The values in table 4.2 indicate the percentage change of the output value relatively to the output reading for the initial parameter configuration when the respective parameter appears as minimum or maximum. The first thing noticeable is the fact that the output deviation of the tree height or the total carbon production does not exceed approximately 23% respectively 77%. However, since LSA belongs to the OAT-class of SA methods, it is too early to consider this as the deviation's upper bound because LSA does not examine interaction effects. Another interesting observation is that the tree height is less sensitive than the total carbon production. The variances of the carbon readings are noticeably higher. For the tree height the vitality parameter has the highest influence compared to the other parameters. The lowest and thus non-essential sensitivity values can be observed for the branching angle and the vitality threshold. Nevertheless, the highest reading in the column of the tree height resides in the range of the 4th lowest reading of the carbon production sensitivity values. Considering the parameter influence, it has to be stated that the leaf area, the vitality, the light days and the two photosynthesis parameters possess the most important influence in the LSA. For the total carbon production the efficiency parameter must be attributed to be the most influential one. The results prove that LSA can just enable an initial parameter screening and thus results in a rough overview of the most likely expected behavior neglecting interaction effects.

4.1.2 Morris's Elementary Effects Screening

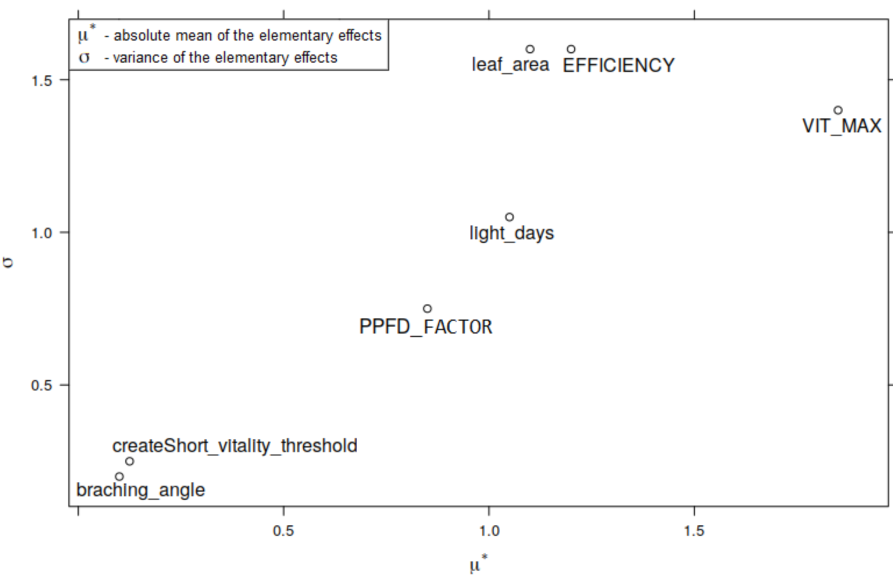


Figure 4.1: Results of Morris's elementary effects screening of the beech tree's height

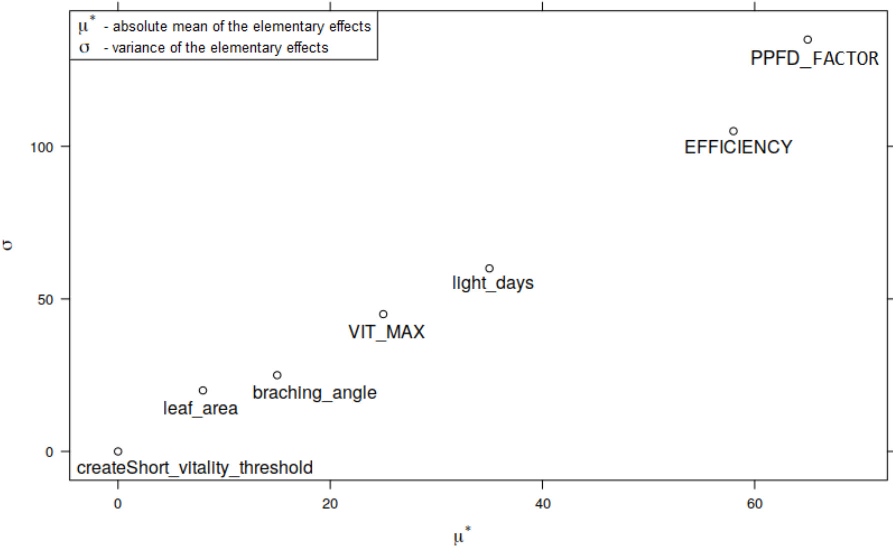


Figure 4.2: Results of Morris's elementary effects screening of the beech tree's carbon production

In figure 4.1 and figure 4.2 the results of Morris's elementary effects screening can be found. At a first look it clearly can be seen that one result of the local SA is also detectable in the Morris plots: the effects of the input parameters on the total carbon production reside much higher compared to the effects on the tree height. It should be noted that in the plots the modified mu value μ^* is shown due to the reason mentioned in chapter 3, that means to avoid canceling out of calculation terms because of their signs. Using the original definition of mu could mistakenly lead to the conclusion that a parameter has just a minor effect, which comes from not recognizing highly non-linear effects (values cancel out to near zero). In the plot it holds true that the further right a parameter is, the more influence can be attributed to it considering the respective output.

For the tree height the result of the local SA can be confirmed besides some minor differences. The vitality and efficiency parameters also belong to the most influential ones regarding tree height. The branching angles as well as the vitality threshold values seem to have little to none influence and thus very little sensitivity. Morris's methods can reveal the degree of non-linearity by the sigma value. However, it must be reconsidered that all values of mu and sigma are no absolute values. They are only suitable for comparison purposes within the set of the examined parameters. Concerning non-linear behavior the leaf area, the efficiency and the vitality have the highest variance readings indicating a more complex parameter association with the height output.

The carbon production mainly is influenced by the efficiency and the available light (PPFD_FACTOR). The plot yields the interesting fact that the efficiency essentially affects the tree height as well as the carbon production. Nevertheless, one can expect that the higher the tree grows the more carbon is needed to create the required biomass. Moreover, the plot makes clear that the amount of available light is also crucial for the production of carbon. The vitality threshold does not govern the carbon production. That is why the value is near zero. The variance of the efficiency parameter and the PPFD_FACTOR is high. Hence, it can be expected that there is a rather complex link between carbon production and these parameters, which is not uncommon considering the process of photosynthesis. At the end it must be mentioned that only elementary effects are revealed by the screening. It should only be used for the ranking of parameters and for obtaining a first impression of the parameter behavior. Aside from that, interaction effects remain untouched. Because of that, the overall sensitivity can differ heavily when interactions of parameters take place.

4.1.3 Main And Interaction Effects On Extreme Values

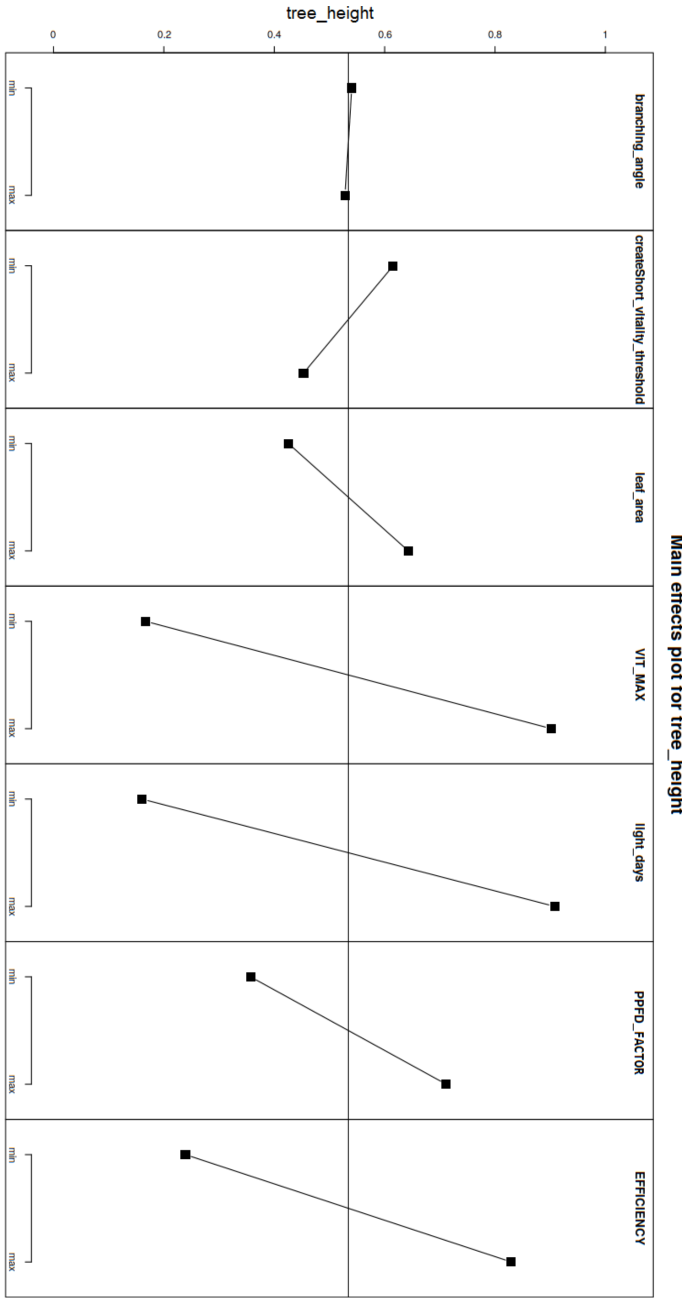


Figure 4.3: Results for main effects on extreme values of the beech tree's height

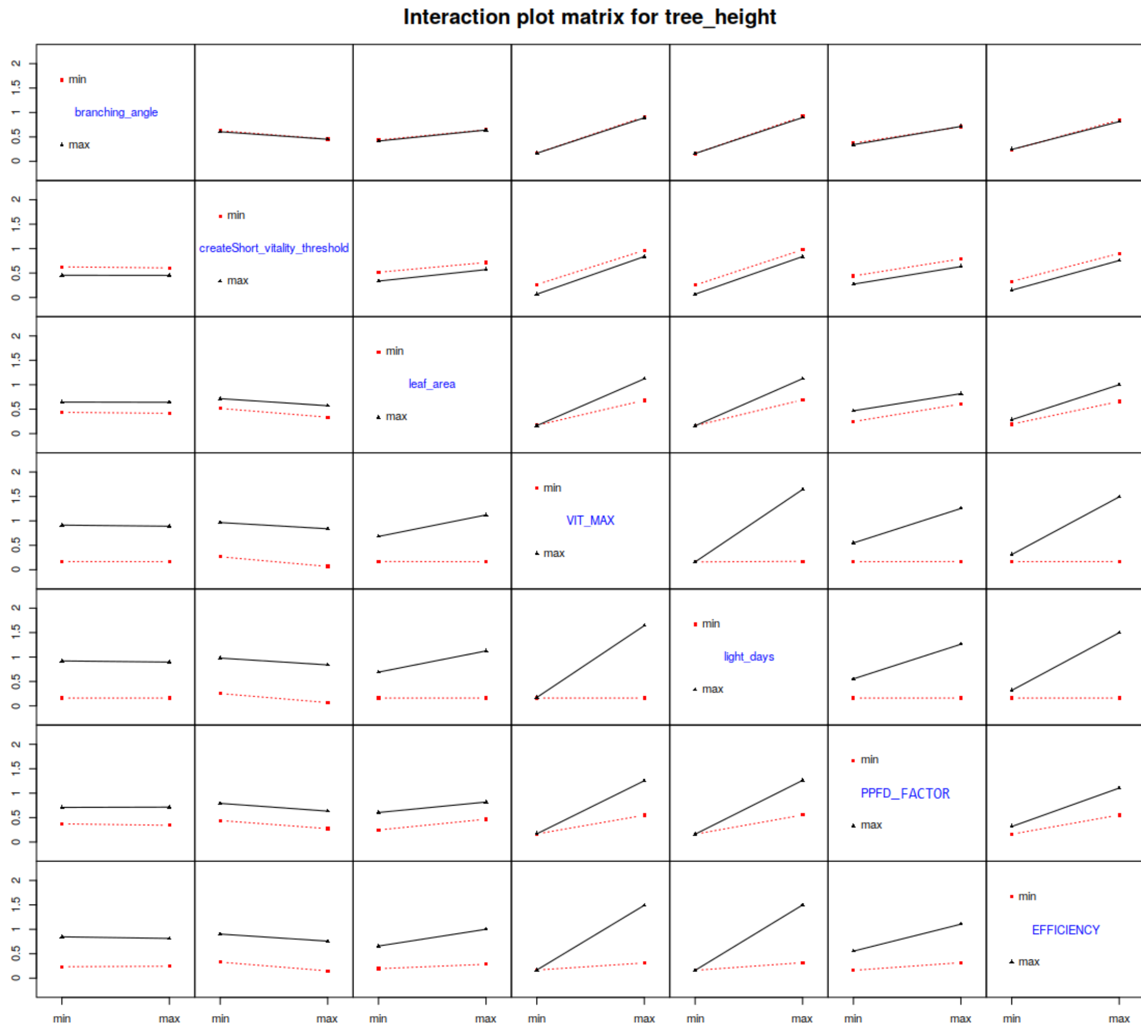


Figure 4.4: Results for interaction effects on extreme values of the beech tree's height

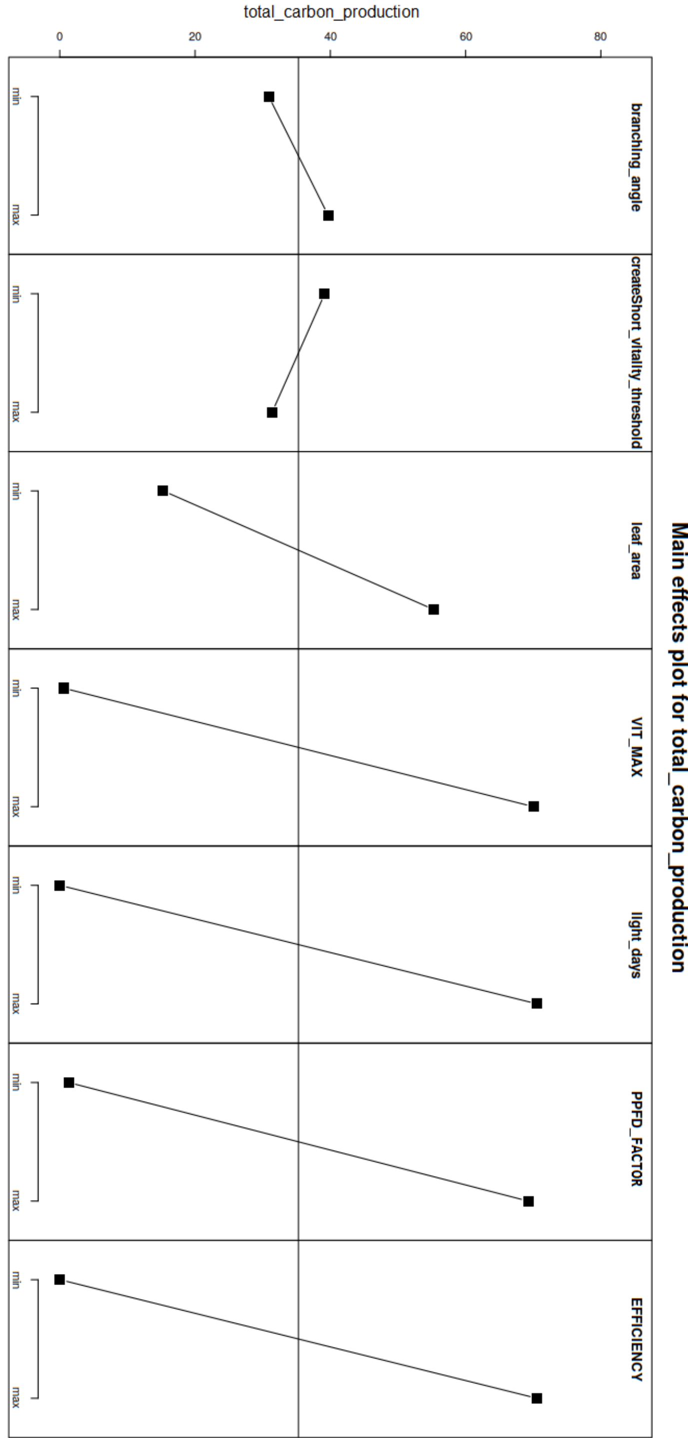


Figure 4.5: Results for main effects on extreme values of the beech tree's carbon production

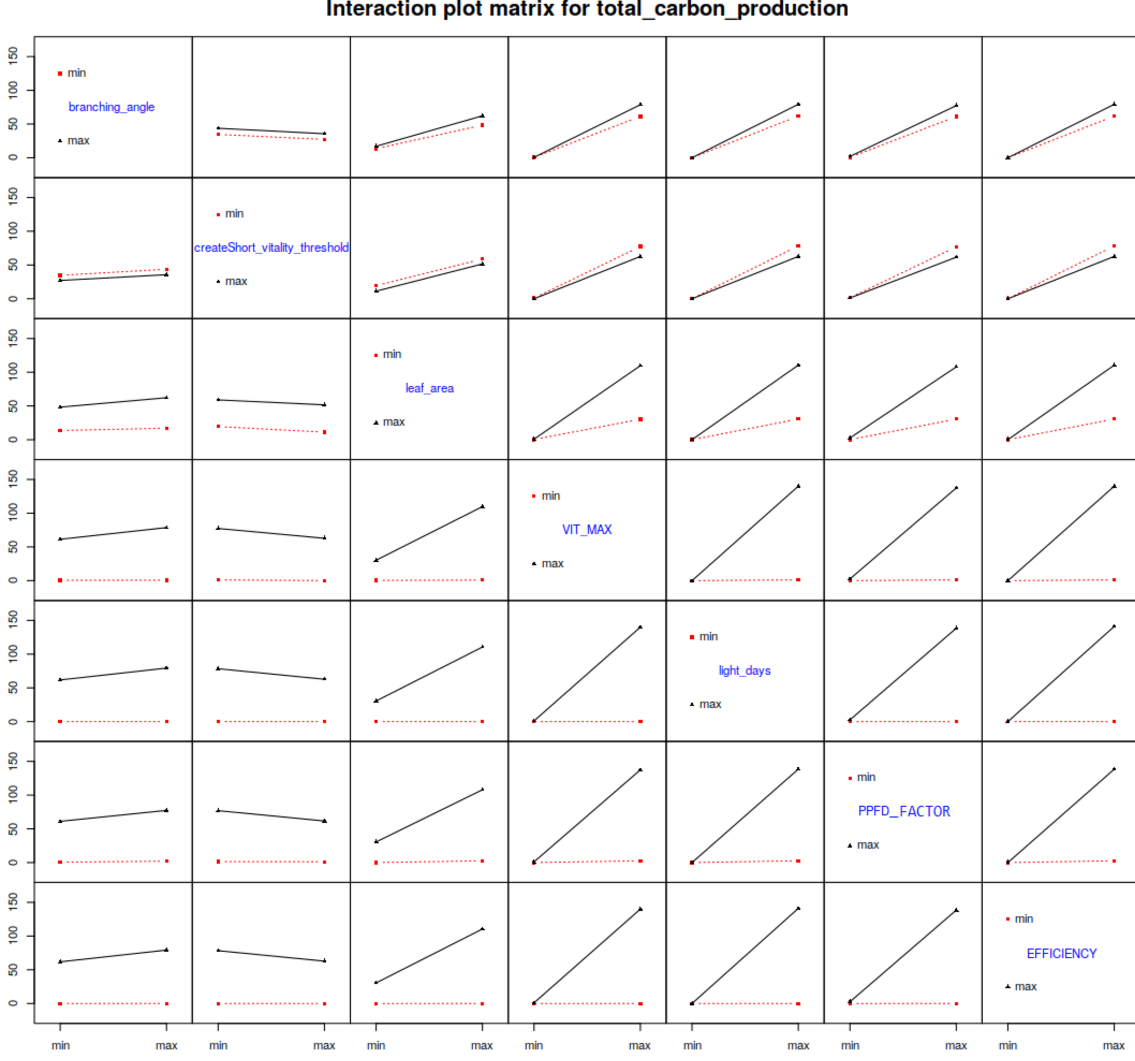


Figure 4.6: Results for interaction effects on extreme values of the beech tree's carbon production

The figures 4.3 and 4.5 show the main effect plots for the tree height and the total carbon production. Figures 4.4 and 4.6 show the interaction effects plot. It must be reconsidered that the examination was done with the extreme values of the examination range. Thus, this method can deliver sufficient sensitivity data to obtain large interaction effects. The results confirm the findings of the local SA and the effects screening. The branching angle and the vitality threshold have minor effects on both output readings. The vitality threshold has a negative correlation. Hence, with higher values for the threshold the output readings decline. For the other parameters a positive correlation can always be found except the curve for the tree height against the

branching angle, which is approximately flat. Again a strong relationship between the tree height and the total carbon production with the vitality and the efficiency can be recognized. For the carbon production the sensitivity of the light days is shown to be more important than expected from the Morris plots. All in all, the main effect plots deliver only a few new insight compared to the already applied SA methods. More interesting results are yielded by the interaction plots.

In the interaction plots each row can be attributed to a special setting of the blue colored parameter. The corresponding entry in the column is the plot when the parameter from this column is varied from minimum to maximum. A red square indicates that the row's parameter is at its minimum. A black triangle coincides with the maximum. For the tree height the branching angle does not interact with any other parameter. That is why in the plot the curves are identical for both extreme values of the angle. The vitality threshold interacts with each other parameter. For its minimum reading the effect of the other parameters is greater. For the maximum the opposite is the case. Additionally, the slopes of the curves are identical, but the curves are shifted up or down. The leaf area has a slight influence on the tree height. All variation curves for the maximal leaf area are above the curves with minimal leaf area. The curves' slope is increased for the vitality, the light days and the efficiency. Thus, in connection with a bigger leaf area and an increase of the mentioned parameters the tree height also increases further. The vitality parameter was known from the local SA to be greatly influential on the tree height. The interaction plot supports this. This can be seen in the jumps the curves experience when comparing minimum to maximum lines. Synergistic effects can be observed between the vitality and the light days, the the input of photosynthesis (PPFD_FACTOR) and the efficiency. When the vitality is at its maximum, the tree height drastically increases for rising values of these three parameters. The branching angle, the vitality threshold and the leaf area do not interact with the available light (PPFD_FACTOR) and the efficiency. The curves' starting points are just higher, but this is attributed solely to the respective parameter. In the plot it clearly can be seen that the number of light days, the input of photosynthesis (PPFD_FACTOR), the vitality and the efficiency synergistically influence each other leading to a strong height growth of the beech. Furthermore, when these parameters reside in their minimum reading, just minor growth takes place. This is indicated by the flatness of the red curves that coincide with the parameter setting where the respective ones are minimum.

The carbon production is affected marginally by the branching angle parameter. This can be clearly seen in the plot as the slope of the curves is just slightly higher with the maximum compared to the minimum setting. The vitality threshold shows the opposite behavior. Here, for the minimum value of the threshold the curves' slopes are a bit higher. However, compared to the other effects the results of the first two parameters are negligible and the findings of the local SA and Morris's screening are confirmed. In the examination of the carbon production the same result was revealed as from the tree height. The leaf area, the vitality parameter, the light days, the available light (PPFD_FACTOR) and the vitality heavily interact. This finding is considerably

visible in the interaction plots. Whilst the value for the carbon production remains near zero when each of the mentioned parameters is at its minimum, the curve drastically rises when it is at its maximum. This is the case for every single binary parameter interaction considering the most sensitive parameters.

4.1.4 Partial (Rank) Correlation Coefficients

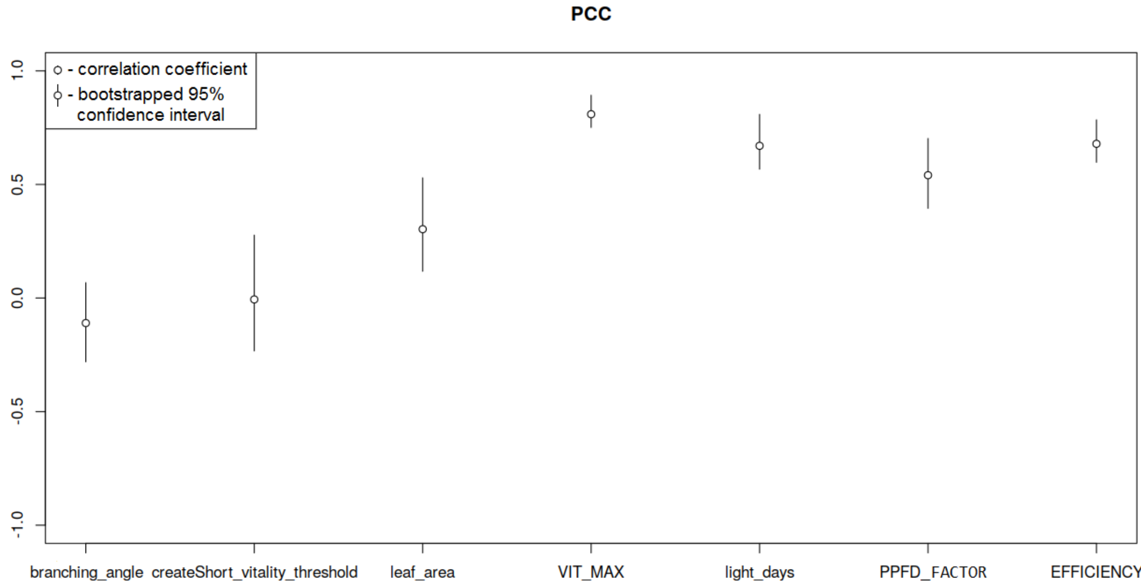


Figure 4.7: Results for the partial correlation coefficients of the beech tree's height

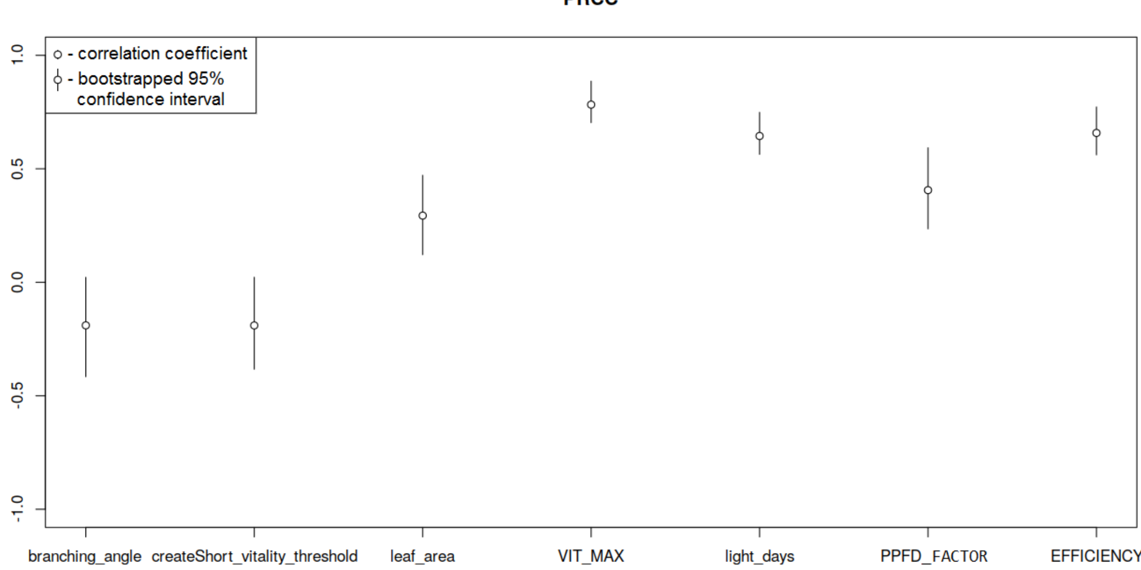


Figure 4.8: Results for the partial rank correlation coefficients of the beech tree's height

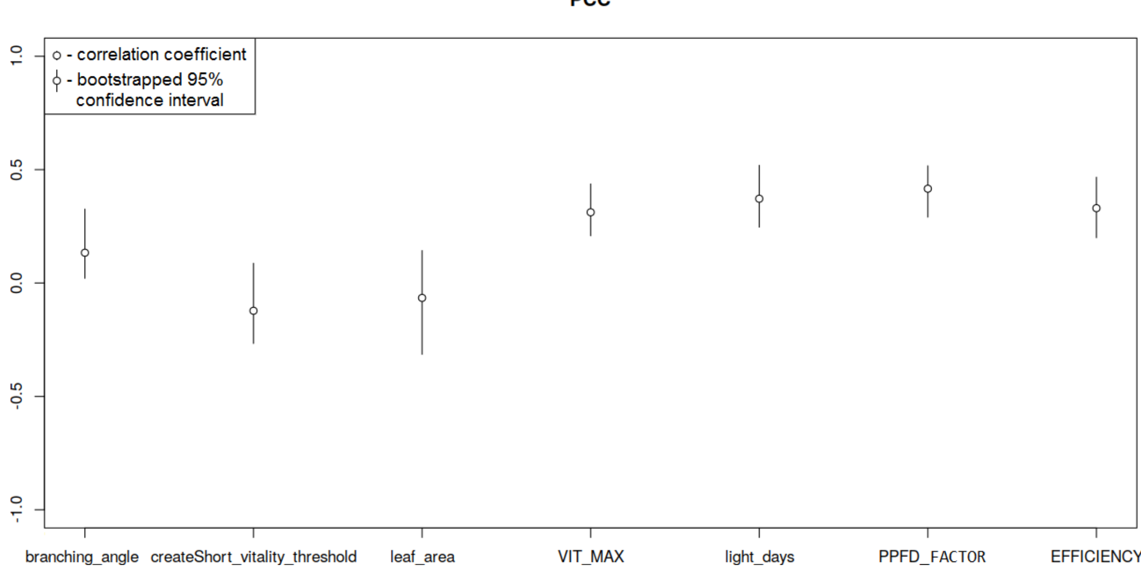


Figure 4.9: Results for the partial correlation coefficients of the beech tree's carbon production

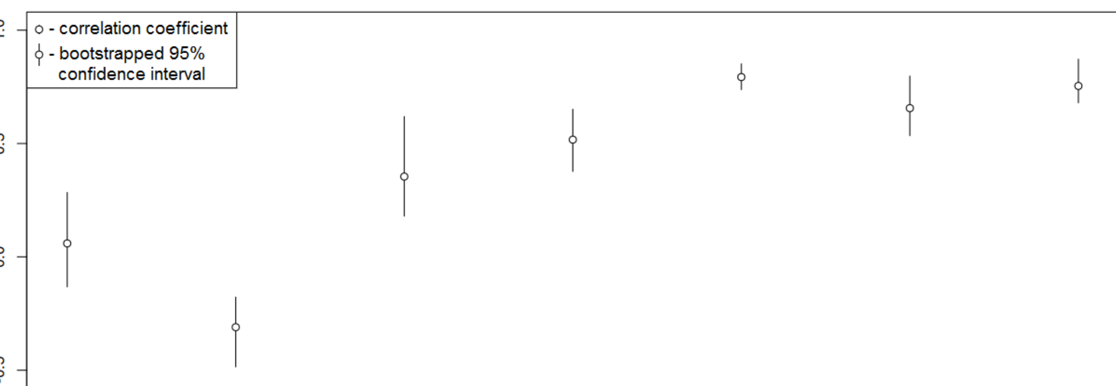


Figure 4.10: Results for the partial rank correlation coefficients of the beech tree's carbon production

The figures 4.7, 4.9, 4.8 and 4.10 show the partial correlation coefficients and their rank versions for the examination of the beech's height and carbon production. In the plots to each parameter the respective correlation value is assigned. The vertical lines around every point coincide with the predicted uncertainty of the sensitivity measure. A value in the plot indicates whether there is a link between the input and output parameters and the degree of linearity between them. A value of 1.0 would yield that there is a direct linear dependency between the input and the output. A value near 0.0 signifies that there is virtually no influence and thus low or no sensitivity.

Concerning the results for the tree height, the first thing noticeable is that the rank version shows approximately the same readings as the non-rank version. However, this does not show that there is no non-linear behavior present. It reveals the monotonicity of the parameters. That means, increasing an input parameter leads to an increase in output. Nevertheless, the amount of effect increase (curve's derivative) may differ noticeably depending on the kind of functional - however monotonic - dependency. The insignificance of the branching angle parameter as well as the vitality threshold is confirmed repeatedly. The uncertainty of the first three parameters is considerably higher compared to the remaining ones. This indicates that depending on the concrete parameter configuration the value's contribution to the output could exceed the initial expectation. The lowest uncertainty and further the highest influence and linearity is attributed to the vitality parameter. This confirms the findings of the SA methods applied before. The remaining parameters also show relatively high readings in the range of $0.6 \dots 0.75$ which yields an important influence. However, the readings are clearly below 1.0 which exhibits that the connection is not totally linear, revealing presumably a more complex process.

In the plots for the total carbon production there is a noticeable difference between the rank and the non-rank version. The identified influential parameters possess higher correlation readings in the rank plots than in the non-rank plots. This yields that the carbon production is more sensitive and obviously experiences higher fluctuations. The non-rank plots assign a very low influence to the first three parameters. However, the leaf-area in the rank-plot is considered to be much more important. This is a very insightful finding and it shows that the leaf area is a complex biological entity. The statement is also supported by the fact that the leaf area in connection with the carbon production possesses the highest uncertainty reading, which can clearly be seen in both plots. In the rank-plot the negative correlation value of the vitality threshold matches the findings from the main and interaction effect plots. The correlation technique furthermore reveals that the beech's carbon production is again perceptibly influenced by the vitality, the number of light days, the available light (PPFD_FACTOR) and the efficiency. Compared to the output parameter of the tree height, the carbon production is more complex. Especially the rank plot for the carbon production yields that the mutual parameter influence is a non-linear functional dependency. In summary, it can be said that the findings on balance support the results that were obtained by the already applied SA methods.

4.1.5 Standardized (Rank) Regression Coefficients

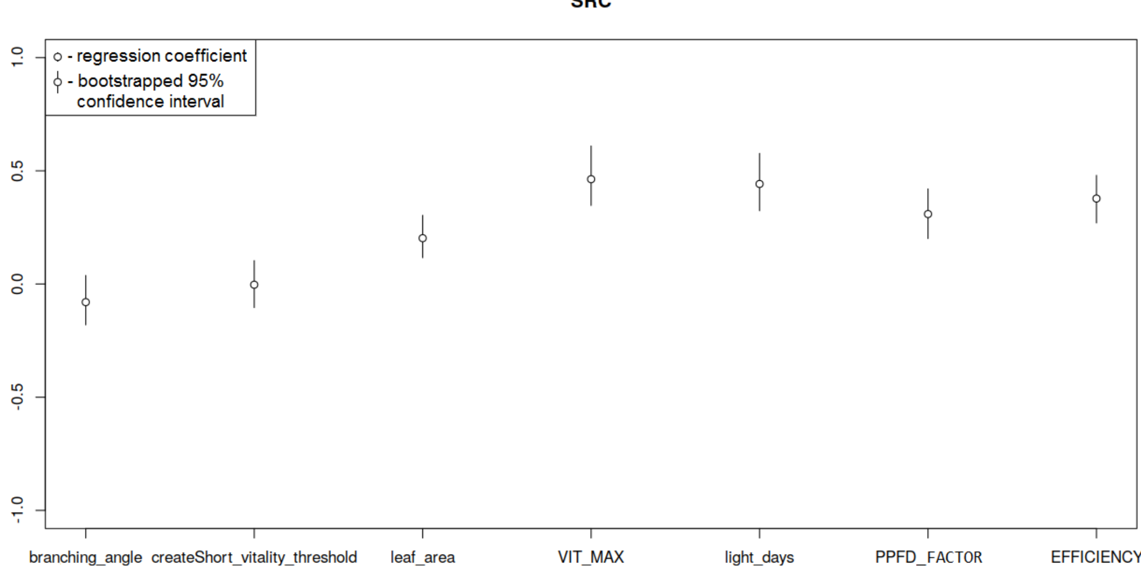


Figure 4.11: Results for the standardized regression coefficients of the beech tree's height

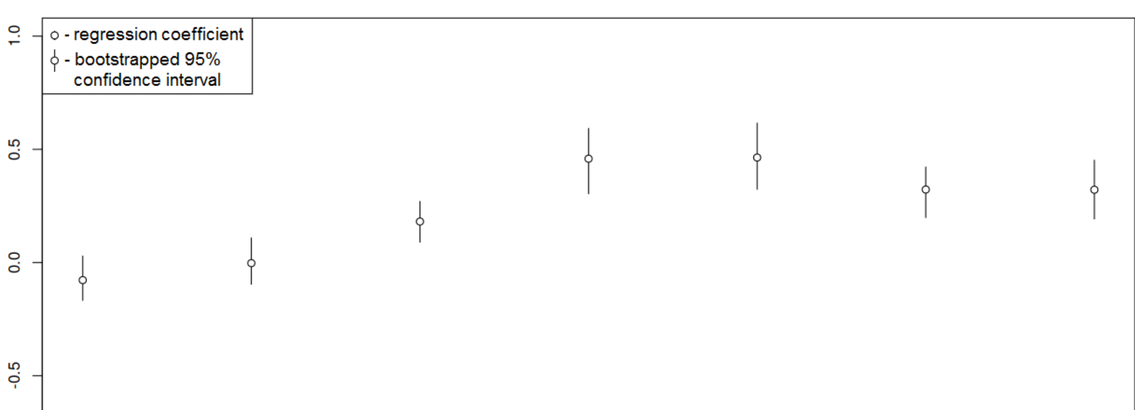


Figure 4.12: Results for the standardized rank regression coefficients of the beech tree's height

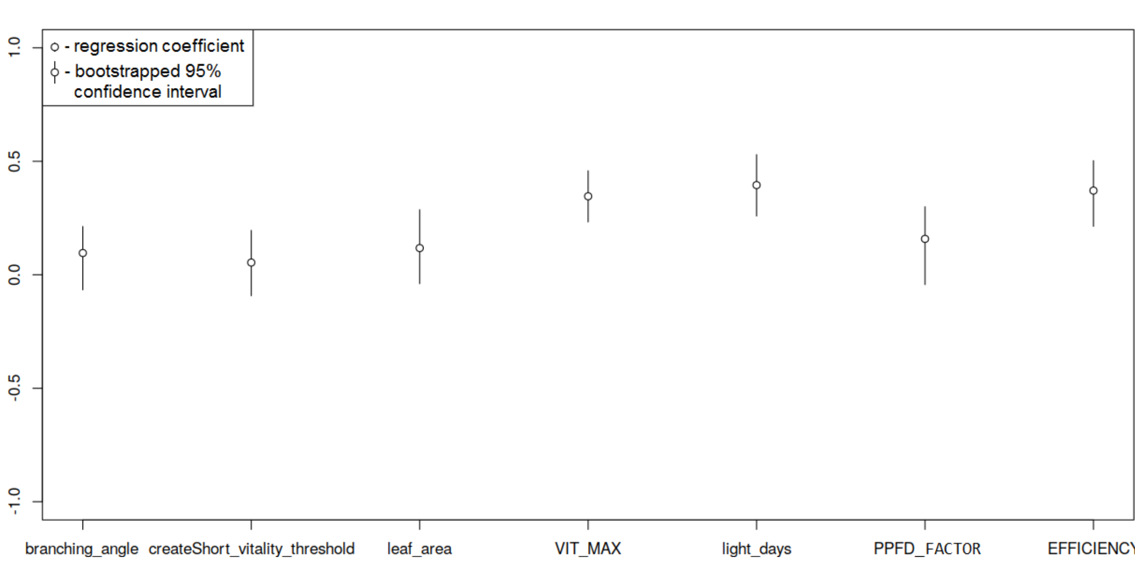


Figure 4.13: Results for the standardized regression coefficients of the beech tree's carbon production

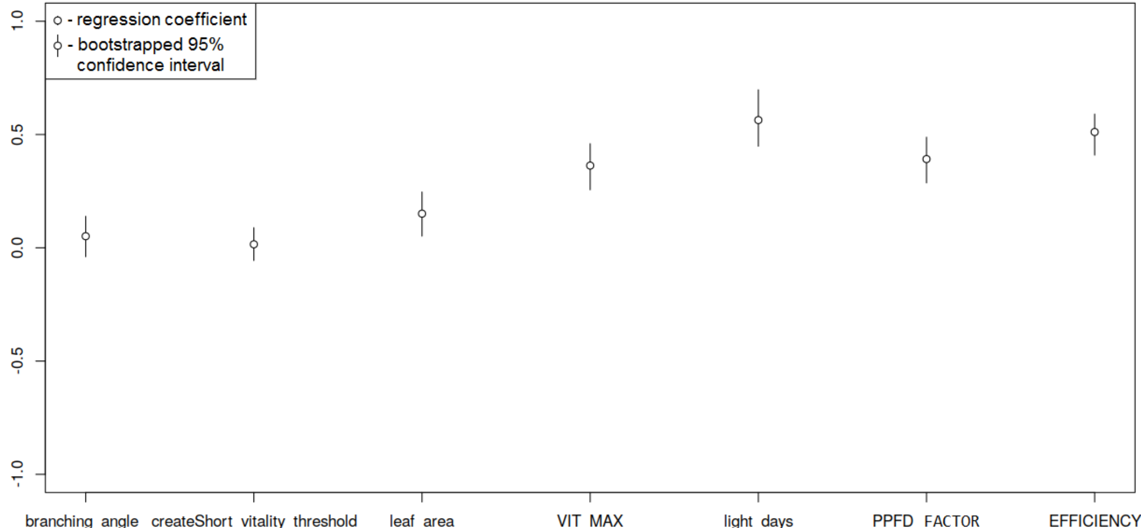


Figure 4.14: Results for the standardized rank regression coefficients of the beech tree's carbon production

The figures 4.11, 4.13, 4.12 and 4.14 represent the standardized regression coefficients and their rank versions for the examination of the beech's height and the total carbon production. The regression coefficients must be seen as importance measures for the different input parameters concerning a single output.

It can be said for the tree height that when sorting the input parameter by their SRC reading, the same order results when sorting the PCC. This holds true for both, the non-rank as well as the rank plots. However, for the influential parameters the values reside in a lower range compared to the PCC/PRCC. Concerning the importance, the last four parameters are considered to be approximately equally important. Again, the lowest sensitivity and thus importance is assigned to the branching angle and the vitality threshold.

For the carbon production the regression coefficients indicate a little less importance for the available light (PPFD_FACTOR). In general, the deviation within the regression coefficients is lower compared to the correlation coefficients. This holds true for the rank as well as the non-rank version. Another interesting fact is that the uncertainty is for all parameters approximately identical and resides in a relatively low range. Like for the tree height the SRC for the carbon production assigns to the input of photosynthesis (PPFD_FACTOR) a lower sensitivity value compared to the PCC. Nevertheless, the SRRC were not able to detect the negative influence of the

vitality threshold. In summary, the conclusion that can be drawn does not question the overall parameter behavior, which is again confirmed: The last four parameters are the most important ones.

4.1.6 Sobol's Method

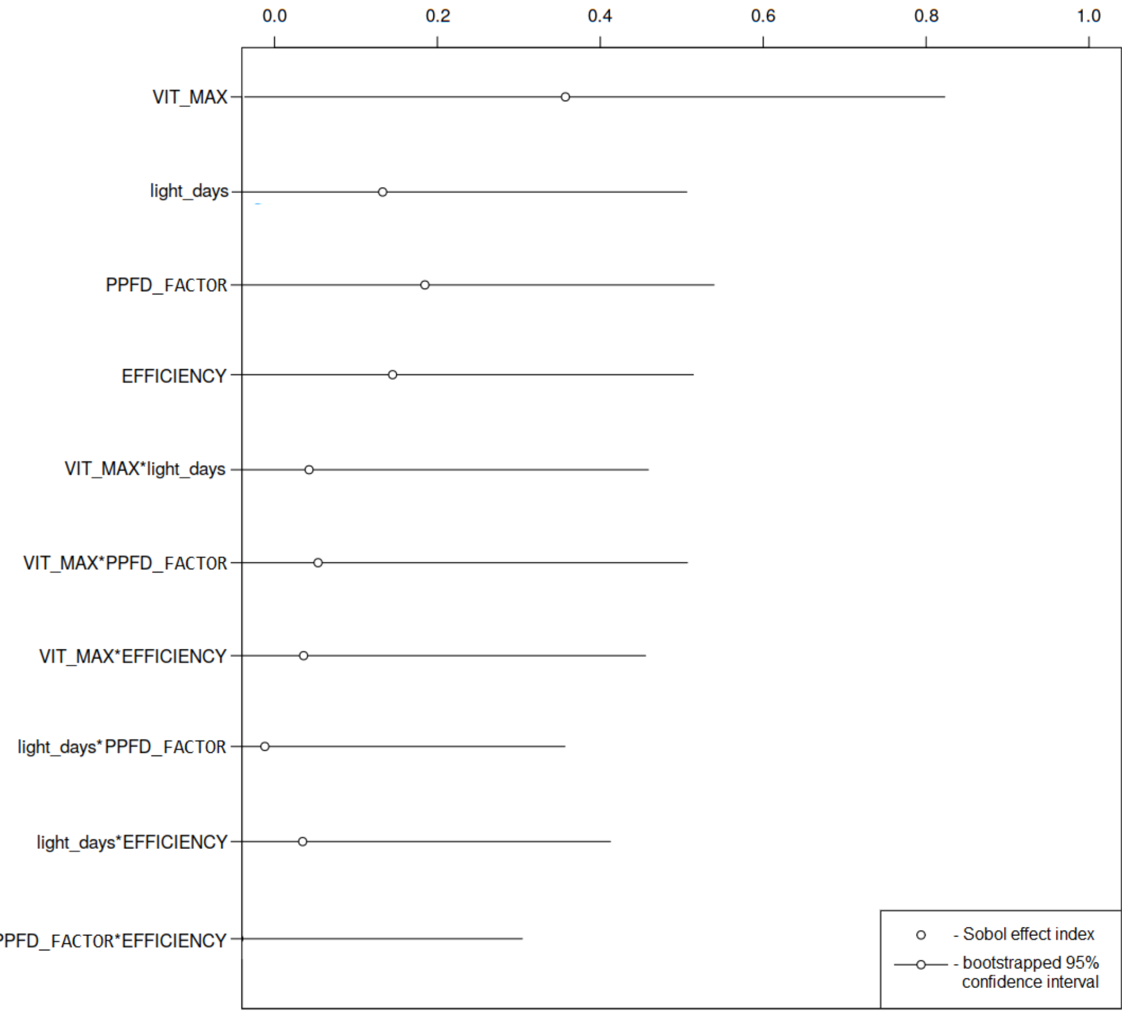


Figure 4.15: Results of Sobol's method for the beech tree's height

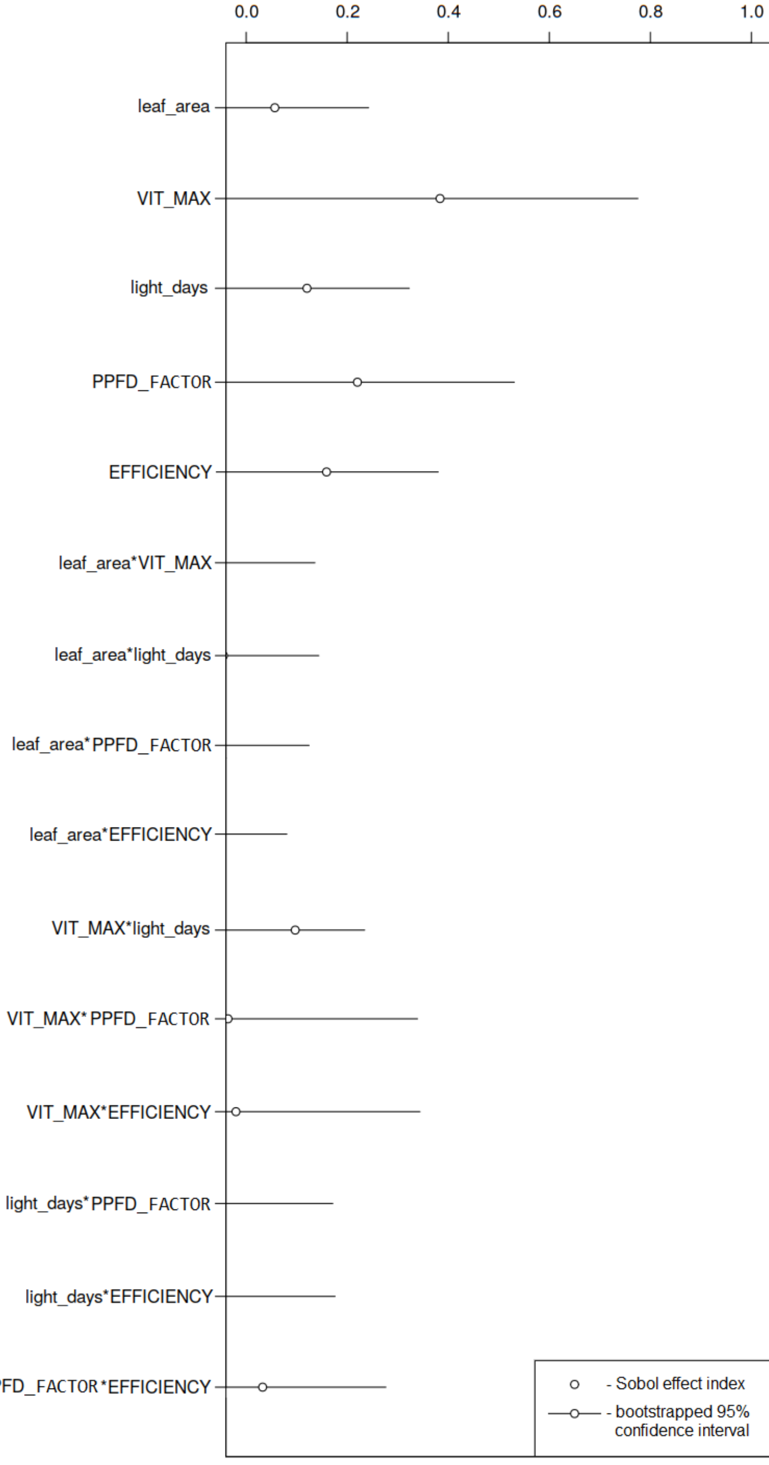


Figure 4.16: Results of Sobol's method for the beech tree's carbon production

In figures 4.15 and 4.16 the plots of the calculated Sobol effect indices of the tree height and the carbon production can be found. Due to the enormous amount of samples required by Sobol's method, the highly-ranged runtime and the exponential number of interaction effects, it was chosen to use the screening results from the previously applied SA methods to omit parameters by their sensitivity namely to solely examine the most sensitive parameters regarding the specific output. Because of that, the last four parameters from the table 4.1 will be examined.

A cirle in a plot coincides with the value of the effect index. The line right or left of a circle is the attributed index's uncertainty (95% confidence interval). In both plots (height and carbon) it can clearly be seen that the uncertainty of the single main and interaction effects is on the high side. However, this was exactly what was expected since the previously carried out SA have revealed the interactions that take place. By that, in particular an individual main effect index must by the laws of mathematics possess a high uncertainty. For the tree height again the vitality parameter is the most influential one. The number of light days, the available light (PPFD_FACTOR) and the efficiency reside in the range of approximately half the sensitivity index reading of the vitality. Furthermore the uncertainty of these last three parameters also is nearly identical. It should again be recalled that the sensitivity indices (main and interaction) sum up to one. Hence, a direct parameter ranking and comparing is possible. In Sobol, interaction effects are binary relations. Hence, the interacting of two parameters is expressed in a single interaction effect index. For the tree height, interactions takes place though at a low level. The number of light days and the available light (PPFD_FACTOR) as well as the efficiency and the input of photosynthesis hardly interact. Nevertheless, the uncertainties of the interaction effect indices remain high.

For the carbon production the leaf area parameter was also examined, since a non-zero influence can be expected when looking at the results of the already applied SA methods. Nevertheless, the plot shows that the leaf area possesses the lowest influence on the carbon production compared to all other examined parameters. Concerning the parameter with the highest sensitivity, the PCC method was not able to clearly identify it since the readings' deviations for the last four parameters were not very high. Sobol's indices assign the vitality parameter the most important contribution to the amount of total carbon production. After the vitality, the sensitivity ranking is followed by the available light (PPFD_FACTOR) and the efficiency. It is interesting to see that the uncertainty of the main and interaction effects is distributed much more irregularly compared to the tree height results. There is in particular a lower uncertainty for the interaction indices of parameters that do not interact. An interaction of medium strength can be recognized between the light days and the vitality and between the efficiency and the available light. The other calculated interaction effect indices only possess an irrelevant value. This is very insightful because in the interaction plot 4.6 on the extreme values a strong interaction was yielded. However, this finding reveals that the interacting is highly non-linear. Additionally, it shows that the number of samples may need to be increased in order to avoid wrongly assumed sensitivity that in reality comes from the output deviation that is a byproduct of the model randomness.

4.1.7 Extended Fourier Amplitude Sensitivity Test

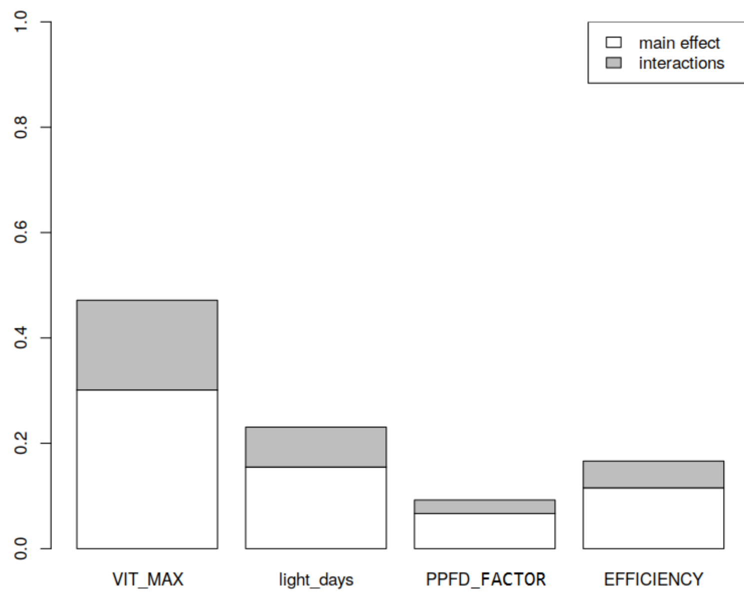


Figure 4.17: Results for the extended Fourier amplitude sensitivity test of the beech tree's height

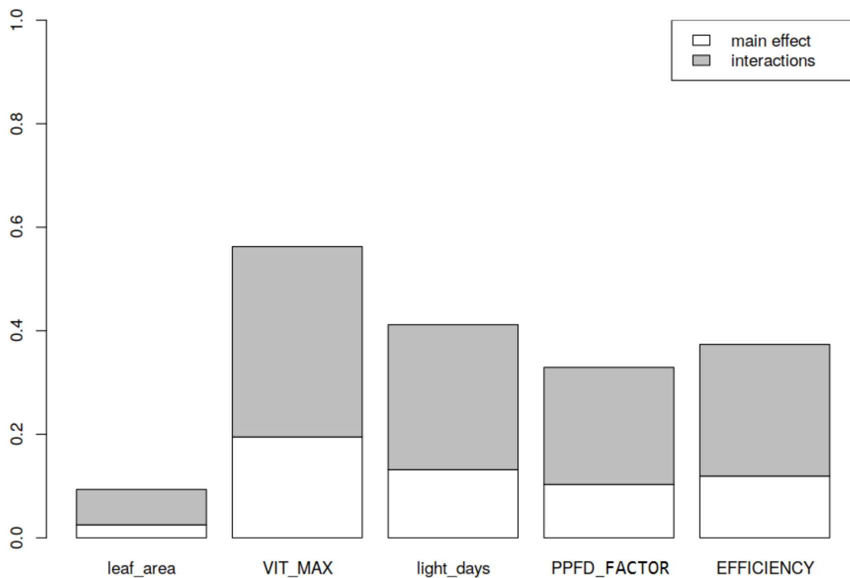


Figure 4.18: Results for the extended Fourier amplitude sensitivity test of the beech tree's carbon production

In figures 4.17 and 4.18 the results for the extended Fourier amplitude sensitivity test (EFAST) can be found. Sobol and EFAST calculate the same sensitivity indices. In the limit, when the number of samples goes to infinity, the difference between a Sobol effect index and an EFAST effect index converges to zero. In the plots the height of a white bar coincides with the main effect of a parameter. The grey bar on top of every white bar is the summed up interaction effects where the particular parameter is involved. Hence, the total parameter effect is the sum of the main effects and the interaction effects. However, concerning the effect readings, EFAST does not show the exactly same sensitivity values compared to Sobol. There are mainly two reasons for that. First of all, EFAST indices are calculated with newly generated input parameter configurations (samples). Thus, the data which is the base of calculations is completely different. By that, the mentioned appearance of non-linear behavior also leads to fluctuations in output that can only be flattened with a sufficiently high number of samples. Second of all, it is due to the variation that comes from the model randomness itself. The variance could in particular be decreased by modifying the code of the beech through decreasing but not fully setting to zero the variance of all random variables. Another way could be given by increasing the number of model runs greatly, for example by a factor of 10, namely from currently 100 to 1000.

For the tree height the vitality parameter is again considered as most influential. Contrary to Sobol, the input of photosynthesis (PPFD_FACTOR) possesses the lowest sensitivity reading. The light days and the efficiency approximately show the same values. These are the only differences compared to Sobol due to the mentioned reasons. The height of the grey bars can be obtained by the Sobol plot by adding up every interaction value where the particular parameter appears. When doing so, EFAST confirms quite well the existing findings. Nevertheless, in the EFAST plot it is not evident where the sensitivity contribution (from the interactions) comes from, since it disappears in the sum.

The carbon production is most sensitive on the vitality parameter. The lowest influence is given by the leaf area. These findings are consistent with the results of Sobol's method. The other parameters differ slightly. While the light days and the efficiency are considered to be equally important, the input of photosynthesis is less sensitive for EFAST than for Sobol. However, the confidence interval that appears in the Sobol plots yields that EFAST results are valid, since they are within the expected range. The same holds true for the interaction effects, which are higher for EFAST than for Sobol. Only the flattening of randomness and the increase in the number samples can yield the desired convergence of the sensitivity indices.

4.2 Assimilate Production Model

The assimilate production model is a plant model that was developed by Gerhard Buck-Sorlin and Michael Henke. Its main focus is not situated in a structural context, but on the sophisticated modeling of radiation and photosynthesis. The target function for the SA is the so-called *net photosynthetic rate*. The speed at which a plant converts radiant energy into carbon products is called *rate of photosynthesis*. However, plants loose matter through respiration and the loss of leaves or other plant parts. Thus, the *net photosynthetic rate* considers the final carbon allocation rate including gains and losses. For the examination five parameters were chosen for SA (see table 4.3). Contrary to the beech model, the assimilate production model does not contain any randomness. Therefore, the output can be directly used without any processing in between like multiple function calls or averaging. As a side effect, the time consumption to carry out the different methods is essentially lower compared to the beech. Since the output is generated by the function that constitutes the simulation and because the model parameters are passed as variables of that function, the adaptation of the model code in order to perform SA only consists of replacing each input variable by a *NumberRef* version. The code of the model and the code for the SA can be found on the supplementary CD. In order to increase the quality of information, the local sensitivity analysis was done twice with different deviation values ($\pm 20\%$ and $\pm 50\%$).

parameter name	description	initial value	variation range
TEMPERATURE	The temperature of the ambient air is given by the parameter *TEMPERATURE*.	$25°C$	$10 \ldots 50°C$
PAR	The amount of photosynthetically active radiation is given by the parameter *PAR*. This value indicates how much radiation energy per area unit the light source produces.	$600 \frac{W}{m^2}$	$100 \ldots 1500 \frac{W}{m^2}$
AGE	The age of the tree (in days) is set by the parameter *AGE*.	$100d$	$50 \ldots 180d$
CO2	The parameter *CO2* holds the amount of carbon dioxide in the plant's ambient air.	$400 \frac{\mu mol}{mol}$	$50 \ldots 1500 \frac{\mu mol}{mol}$
RELATIVE_ HUMIDITY	The water content in the air is given by the parameter *RELATIVE_HUMIDITY*.	75%	$10 \ldots 100\%$

Table 4.3: Overview of the examined assimilate production model input parameters

4.2.1 Local Sensitivity Analysis

parameter value	net photosynthetic rate ($\pm 20\%$ deviation)	net photosynthetic rate ($\pm 50\%$ deviation)
TEMPERATURE_min	-14.61	-37.92
TEMPERATURE_max	-16.12	-43.19
PAR_min	-4.04	-15.23
PAR_max	2.88	3.46
AGE_min	18.18	48.98
AGE_max	-16.02	-37.28
CO2_min	-12.86	-45.76
CO2_max	5.89	12.27
RELATIVE_HUMIDITY_min	-1.97	-5.41
RELATIVE_HUMIDITY_max	1.96	3.25 ($+33\%$ deviation)

Table 4.4: Results for the local sensitivity analysis of the net photosynthetic rate of the assimilate production model

The table 4.4 shows the results of the local sensitivity analysis for $\pm 20\%$ and $\pm 50\%$ deviation. It must be noted that the relative humidity cannot exceed 100%. Hence, for the maximum value the deviation is 33% instead of 50%. For the first test, the overall difference in output (absolute value) does not exceed approximately 18% and for the second test 49%. Relatively low sensitivity was found in both experiments for the light intensity (PAR) and the humidity. The air temperature, the CO2 value as well as the age possess high sensitivity values. It is interesting to see that for both extreme values of the temperature (colder and hotter), the net photosynthetic rate is noticeably decreased. Hence, the assimilate production model very precisely reproduces the biological behavior that plants do not grow well (do photosynthesis) when it is too cold or too hot.

Concerning the highest sensitivity, the age parameter is the most influential one. One can conclude that the plant in a younger state performs more photosynthesis than in an older state, meaning that the net rate is slowly decreasing over time. It is also compliant with biology that the photosynthesis is heavily reduced when the amount of carbon dioxide is too low. This can clearly be seen as the net photosynthetic rate is decreased by approximately 46% when the CO2 concentration of the air is cut in half. Furthermore the data clearly shows that a plant can only use a fraction of the incoming light. When looking at the output's behavior regarding the PAR value, a reduction of the initial reading by 50% leads to a decrease of the net photosynthetic rate by approximately 15%, whilst an increase of the same amount only leads to a minor increase of approximately 3.5%.

All in all, it is again shown that a local sensitivity analysis - which is the most simple approach of all considered SA methods - can reveal insightful model behavior. At the moment, the plausible readings that are compliant with biology can support the validity of the assimilate production model. However, it should be noted again that interaction effects are not obtained yet. Nevertheless, the sensitivity values for the different parameters make clear that especially from the temperature, the age and the CO2 concentration the most essential influence can be expected.

4.2.2 Morris's Elementary Effects Screening

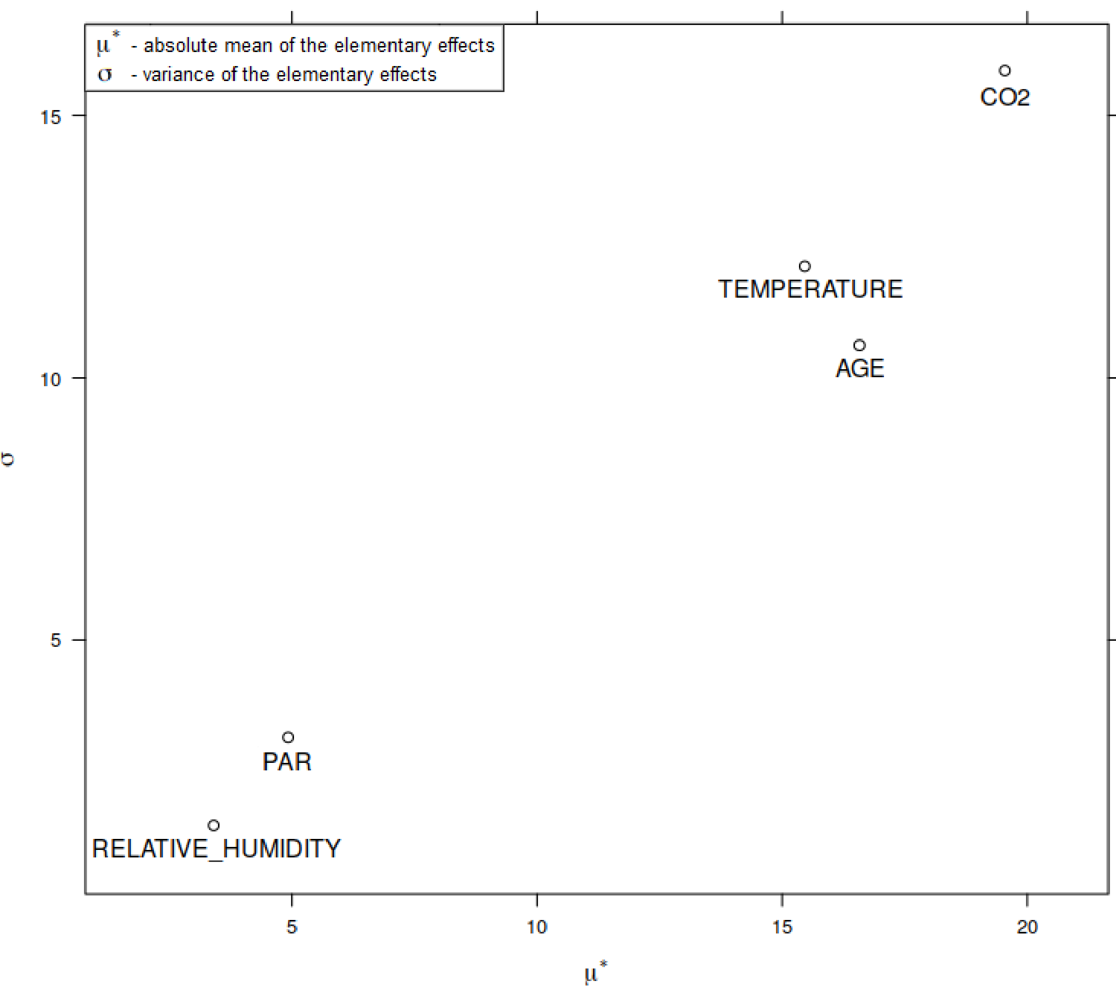


Figure 4.19: Results of Morris's elementary effects screening for the net photosynthetic rate of the assimilate production model

In the figure 4.19 the elementary effect readings can be found for each test parameter. A first look initially reveals that the parameters are divided into two groups of different sensitivity: low and high. The relative humidity and the light strength (PAR) belong to the low sensitivity parameters regarding the net photosynthetic rate. It is interesting to see that the effect values very precisely confirm the findings made with the local SA. However, the Morris plot is a statistical analysis of the elementary effects. To every parameter two values are assigned: the effects' mean and standard deviation. The mean delivers the information about the effect strength and the deviation about possible non-linear effects as well as interaction effects. Nevertheless, it should be recalled that the Morris method belongs to the OAT-class of SA methods, hence interactions are not directly examined. The standard deviation can just give a hint what further SA approaches can expect regarding the amount of interaction effects, when using this value as a parameter ranking indicator.

The plot shows that the effect of the light parameter (PAR) possesses slightly more fluctuations compared to the humidity. This could indicate that the light parameter (PAR) is more likely to interact with other parameters or to produce the output's non-linearity. The group of parameters with a high sensitivity consists of the age, the CO2 concentration and the temperature. The most influential parameter for the net photosynthetic rate is the CO2 concentration. This is a difference to the results of the local sensitivity analysis where the age parameter possessed the highest output deviation. The Morris plot admittedly shows that the temperature has a higher standard deviation than the age, but it also reveals a disadvantage of the Morris method: the direction of an effect cannot be seen in contrast to the local SA. This is especially important for the temperature parameter, since the local SA yielded that low and high temperatures were greatly reducing the net photosynthetic rate. In conclusion, the results from the local SA are on balance confirmed.

4.2.3 Main And Interaction Effects On Extreme Values

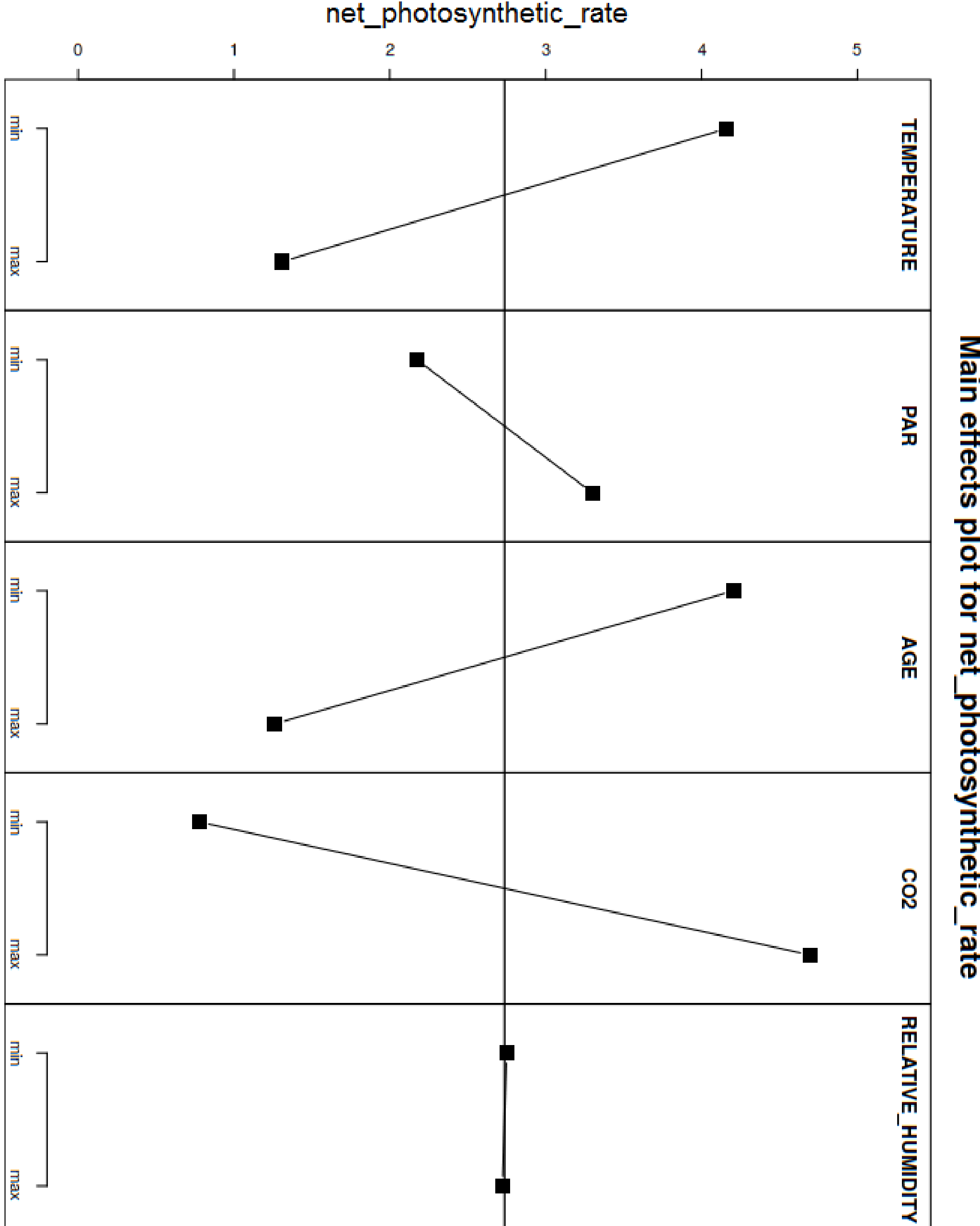


Figure 4.20: Results for main effects on extreme values of the net photosynthetic rate of the assimilate production model

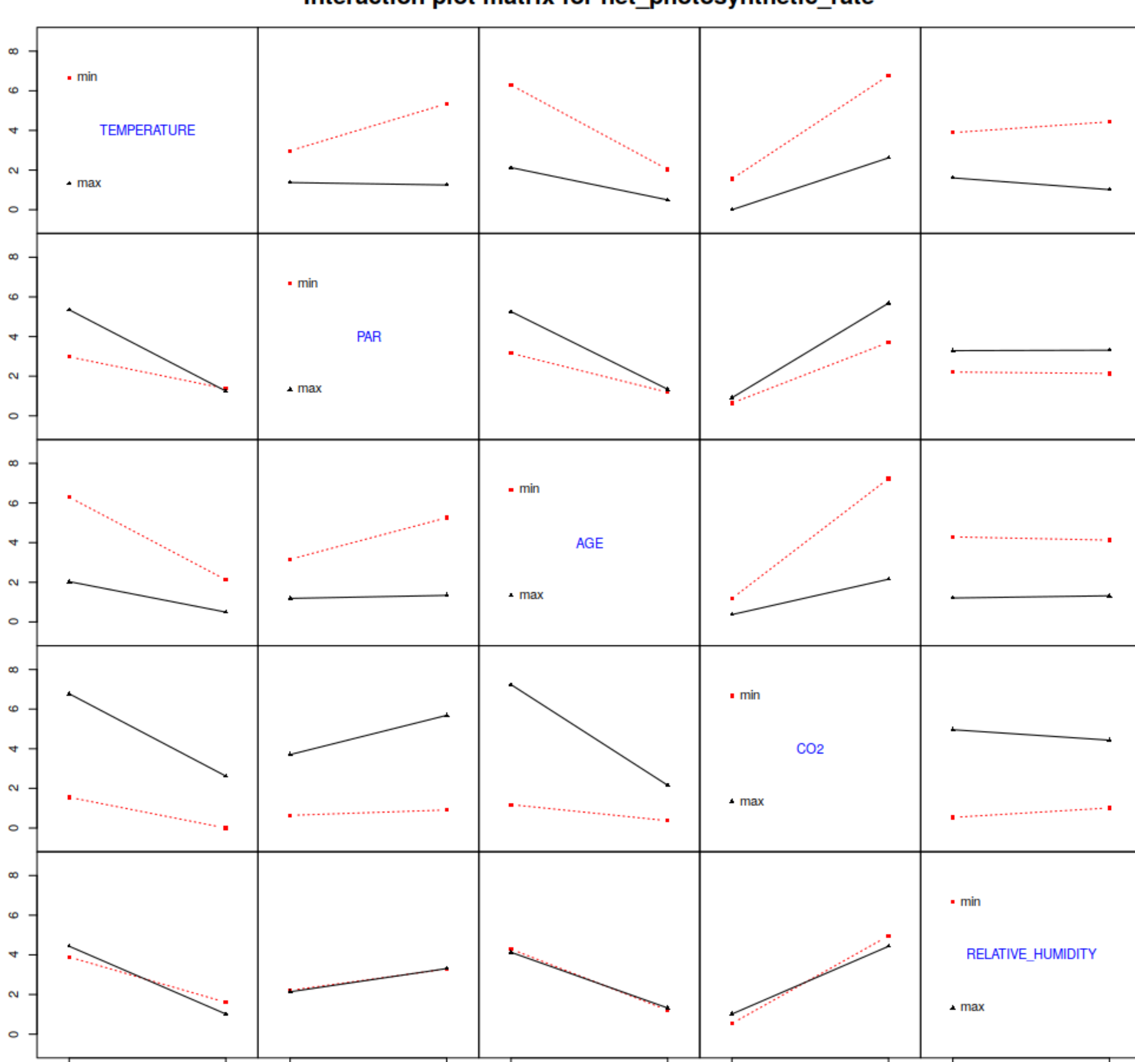


Figure 4.21: Results for interaction effects on extreme values of the net photosynthetic rate of the assimilate production model

The main and interaction effects plots on the extreme values of the input parameters can be found in figures 4.20 and 4.21. In the main effect plot it clearly can be seen that the results of the local SA and Morris's screening can be recognized. The curve of the light parameter (PAR) and the humidity have the lowest slope and height. The humidity parameter curve is very flat compared to the others. The only positive influence on the net photosynthetic rate is yielded by the CO2 concentration and the light parameter (PAR). Concerning the strength of the effects, the plot

confirms the findings from the Morris plots very well: the CO2 concentration is most influential on the net photosynthetic rate followed by the age and temperature. Additionally, the main effect plot reveals the signs of the effects that were before only obtained through a combination of the local SA and Morris's elementary effects.

The interaction plot can be considered as very insightful. The first thing noticable is the minor influence of the relative humidity. There is no interaction with the light parameter (PAR) or the age. However, the humidity slightly interacts with the temperature and the CO2 concentration. When the humidity is at its minimum the decrease of the net photosynthetic rate along the temperature is flatter, however it starts from a lower point. For the maximum CO2 concentration it holds true that when the humidity is at the minimum the maximum net photosynthetic rate is higher, whereas when the CO2 concentration is minimal a low humidity is disadvantageous for the net photosynthetic rate.

The light parameter (PAR) interacts with all other parameters except the humidity. For all curves it is visible that for the maximal light intensity the curve for the net photosynthetic rate is well above the minimum curve. Nevertheless, the effect is decreasing when PAR interacts with the temperature and the age parameter. A positive synergistic effect can be recognized with the CO2 concentration. When both readings are at their maximum the net photosynthetic rate is considerably increased.

An essential interaction can be seen for the temperature parameter. It interacts with all other input parameters. It is insightful to see that when the temperature is too high, the net photosynthetic rate is considerably reduced compared to the lower temperature. A positive effect on the net photosynthetic rate can be seen for the interaction of the lower temperature with the light parameter (PAR) and for the CO2 concentration.

Considering the amount of influence, the age parameter is as sensitive as the temperature. Furthermore, the behavior of both parameters is overall comparable. It interacts with all other parameters, except the humidity. A young tree does more photosynthesis than an old one. This can clearly be read off the interaction plots. This dependency is most affected when looking at the interaction of the age and the CO2 concentration. For the parameter configuration where a high CO2 concentration is found, the net photosynthetic rate of a young tree is at its maximum.

The most influential parameter is the CO2 concentration. Here, the biggest gap between the minimum and maximum curves can be found. A positive interaction where both parameters contribute to the net photosynthetic rate can be recognized for the light intensity parameter (PAR). Summarizing, it must be said that the interaction plots are very good revealing initially hidden model behavior. Furthermore, the results of the already applied SA methods are greatly supported.

Compared to the beech model, the results of the assimilate production model are up to now consistent along the different methods and thus unambiguous. The main reason for that finding is given by the stochasticity of the beech model in contrast to the predetermination of the assimilate production model behavior.

4.2.4 Partial (Rank) Correlation Coefficients

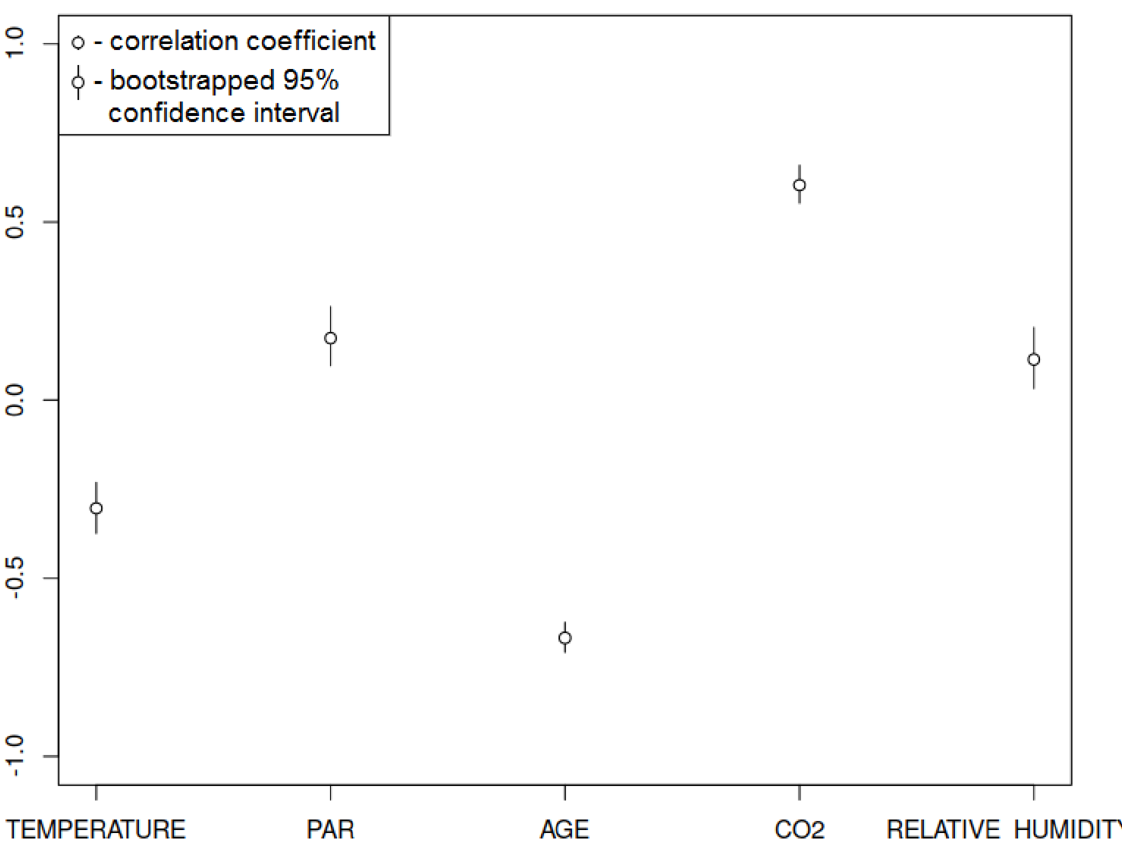


Figure 4.22: Results for the partial correlation coefficients of the net photosynthetic rate of the assimilate production model

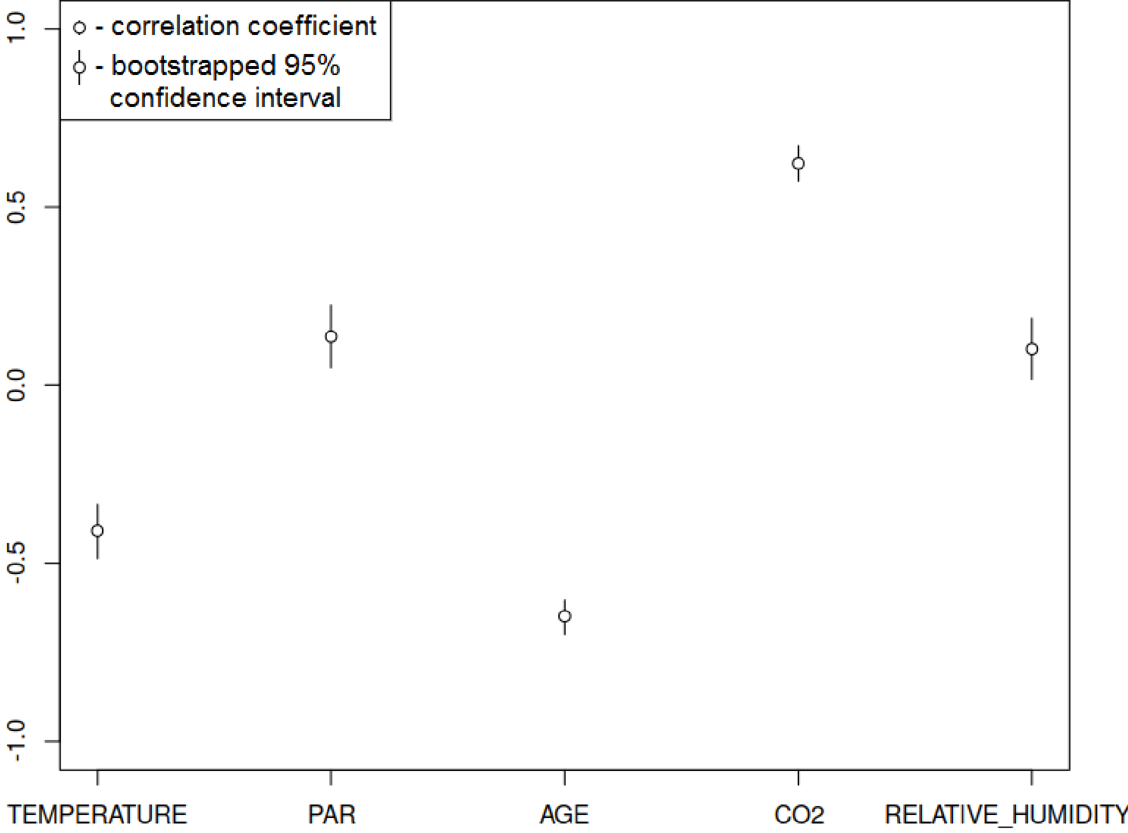


Figure 4.23: Results for the partial rank correlation coefficients of the net photosynthetic rate of the assimilate production model

The figures 4.22 and 4.23 are the plots that show the graphical representation of the calculated partial correlation coefficients and their rank versions. The first insight is given by the fact that the PCC and PRCC plots are approximately identical. This proves that the influence of the input parameters is consistent by the means of monotonic behavior. In contrast to the beech tree, two PCC and PRCC values of the assimilate production model input parameters possess negative signs. This holds true for the temperature as well as the age parameter. It is insightful to recognize than the negative influence on the net photosynthetic rate of these two parameters is compliant with the sensitivity readings of all other applied SA methods. Especially the main and interaction plots revealed the decrease of the net photosynthetic rate with rising temperature or age. Another main observation is given by the circumstance that the confidence interval (uncertainty) for each input parameter is very narrow and resides much lower compared to the beech's parameters. This is due

to the fact that the assimilate production model does not contain any randomness. Furthermore the low uncertainty indicates that there is a sufficiently large sample base for the sensitivity calculations. Concerning the most important positive influence on the net photosynthetic rate, the CO2 concentration parameter can be considered as the most important one. A relatively low reading and thus low importance is again assigned to the humidity. The relatively high but negative dependency between the age or temperature and the net photosynthetic rate - hence decreasing photosynthesis with higher age or temperatures - can again be derived from the negative PCC and PRCC readings. The intensity of the light (PAR) has only mediocre influence on the net photosynthetic rate. However, the fact that the absolute values of all correlation measures are well below 1.0 indicates that there is a non-linear relationship between the input and output parameters. Finally it must be said that the PCC and PRCC methods support the findings of the model behavior examined with the local SA, Morris's method and the main and interaction effects.

4.2.5 Standardized (Rank) Regression Coefficients

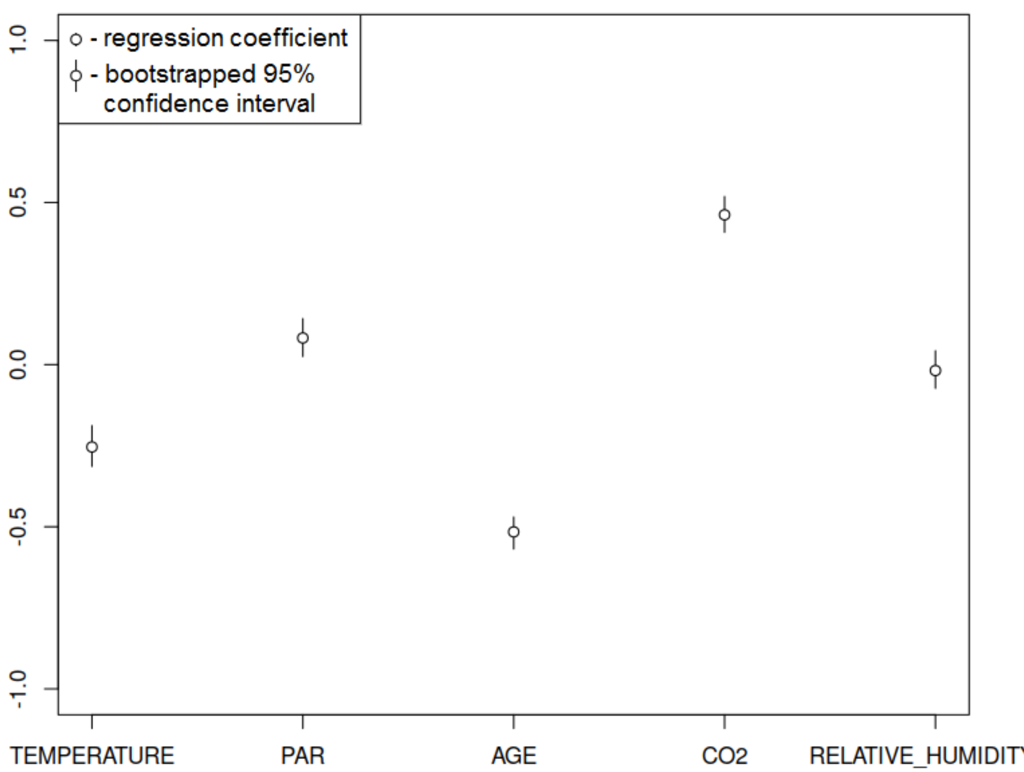


Figure 4.24: Results for the standardized regression coefficients of the net photosynthetic rate of the assimilate production model

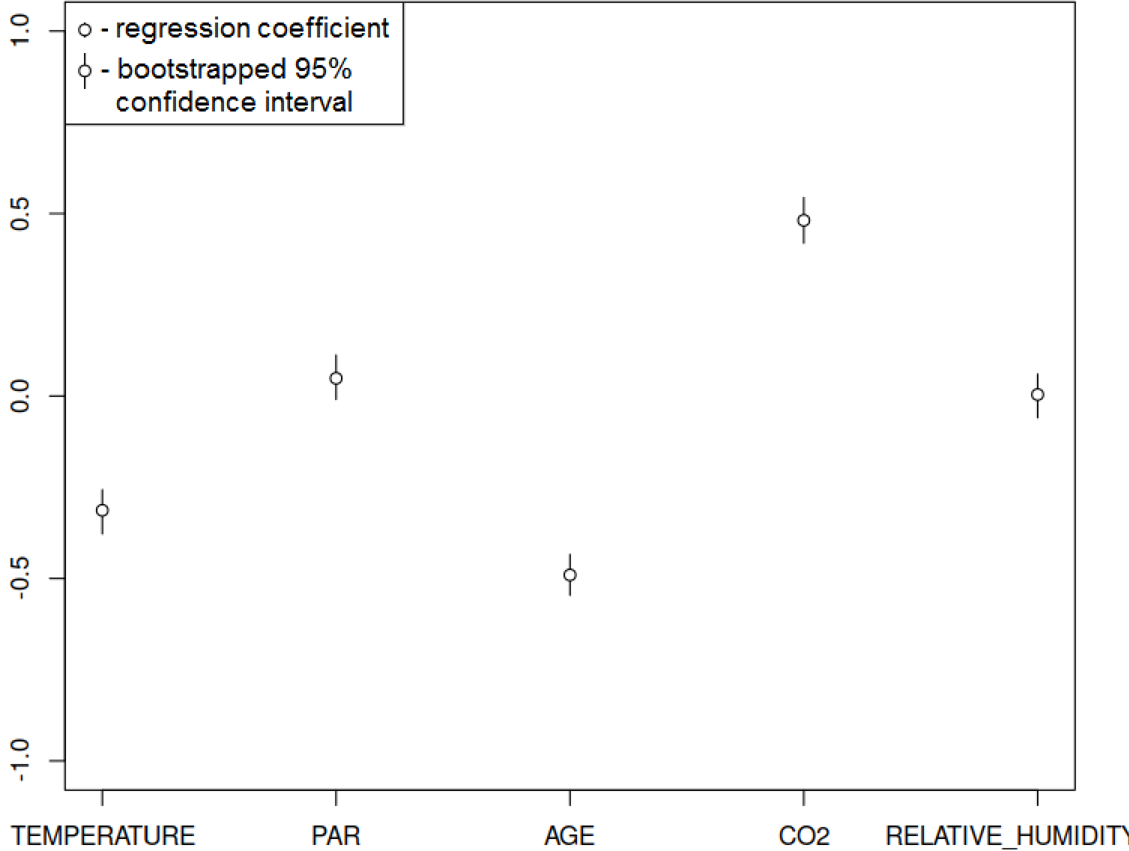


Figure 4.25: Results for the standardized rank regression coefficients of the net photosynthetic rate of the assimilate production model

The plots of the standardized regression coefficients and the rank version for the assimilate production model can be found in figures 4.24 and 4.25. Like for the PCC and PRCC plots, the plots for SRC and SRRC approximately show the same sensitivity values for each input parameter. Only a slight difference can be seen in the rank version where the temperature and the age parameter possess non-essentially higher sensitivity values. However, the difference is so small that the ranking of the parameters remains unchanged. The calculated coefficients do not reveal any new finding. They are quite consistent with the PCC and PRCC readings. Nevertheless, it should be recalled that both SA methods address different objectives. The SRC/SRRC are used as relative importance measures (result of a regression) and the PCC/PRCC as correlation measures. It is insightful to experience that here the importance approximately coincides with the correlation, which underlines the model behavior consistency along all applied SA methods.

4.2.6 Sobol's Method

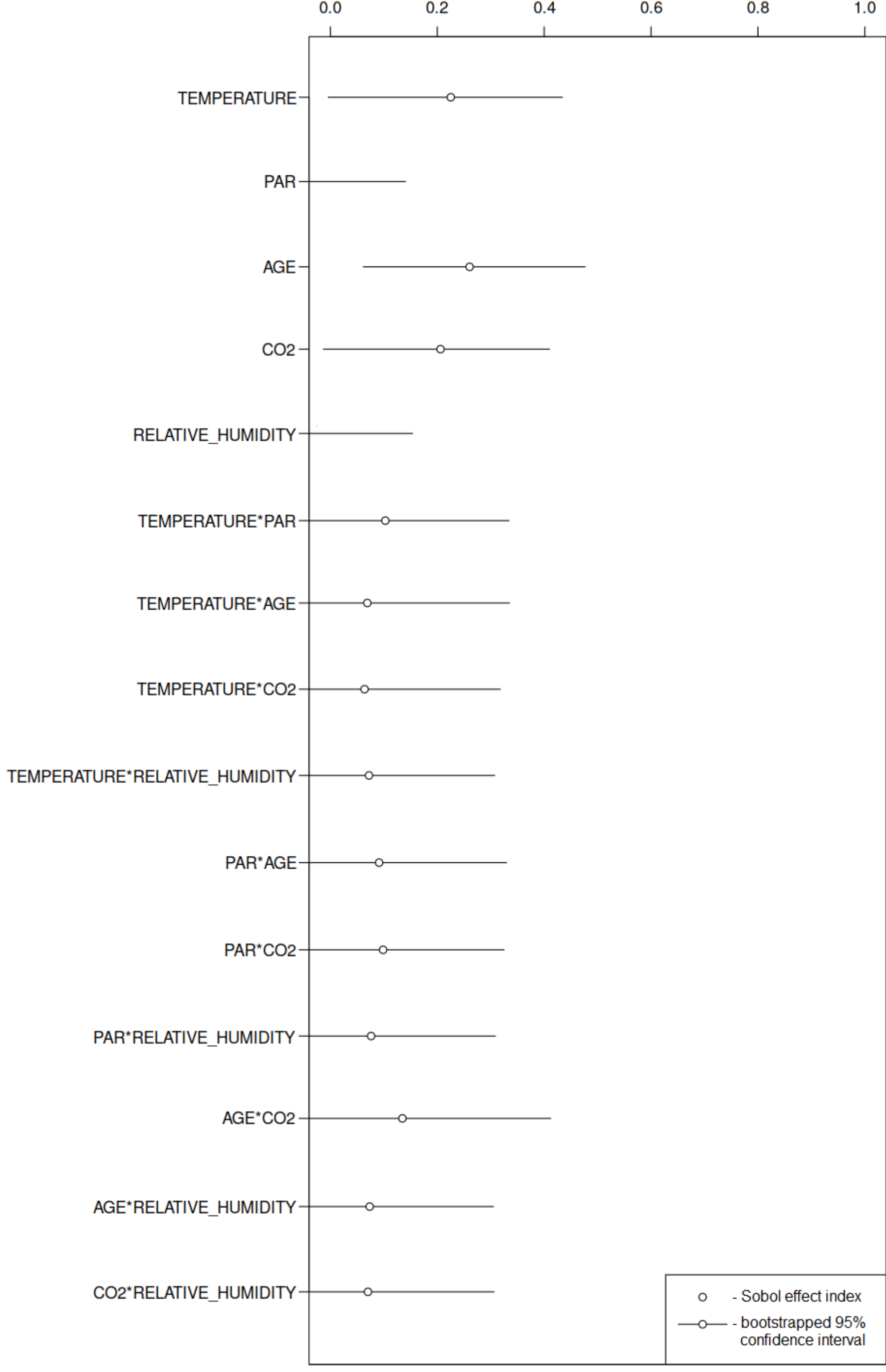


Figure 4.26: Results of Sobol's method for the net photosynthetic rate of the assimilate production model

A visual representation of the main and interaction effect indices that are obtained by Sobol's method can be found in figure 4.26. The first thing observable is the fact that the uncertainty is evenly distributed and resides relatively high. This is a big contrast to the PCC/PRCC and SRC/SRRC values where the uncertainty was very low. Nevertheless, when looking at the main effect indices, a clear division between the influential parameters (temperature, CO2 concentration, age) and the non-influential ones (PAR, relative humidity) can be recognized. However, the ranking of input parameters is slightly different to the consistent findings from all other applied SA methods. The age parameter is the most important one regarding the net photosynthetic rate, then followed by the temperature and the CO2 concentration. Notwithstanding, it holds true that the differences between the main effect indices of the important input parameters are low and do not exceed approximately 0.05. Furthermore, the 95%-confidence intervals (uncertainty) of these parameters make clear that also the ranking revealed by the other SA methods is possible with a high probability.

Concerning the interaction effect indices, Sobol's method shows slight interactions for all binary parameter relations. However, the uncertainty is very high. In order to reveal a clear model behavior considering interactions would require indices less prone to error. This can only be achieved with a higher number of samples. Nevertheless, when looking at the actual Sobol interaction indices, the relative humidity possesses the least influence on the net photosynthetic rate. All interaction readings are well below 0.1, which underscores the minor significance of this parameter. In total, the CO2 concentration interacts the most. However, the difference to the other parameters, for example the temperature, is not very high.

All in all, the interpretation of the interaction indices is difficult due to the mentioned uncertainty and the relatively even index distribution. Leastwise, Sobol's method was able to clearly separate the input parameters by their main effect. Anyway, this finding is compliant to the results from the applied SA methods. The drawback of Sobol by the means of requiring a tremendous amount of data in order to produce reasonable measures, is supplied evidence.

4.2.7 Extended Fourier Amplitude Sensitivity Test

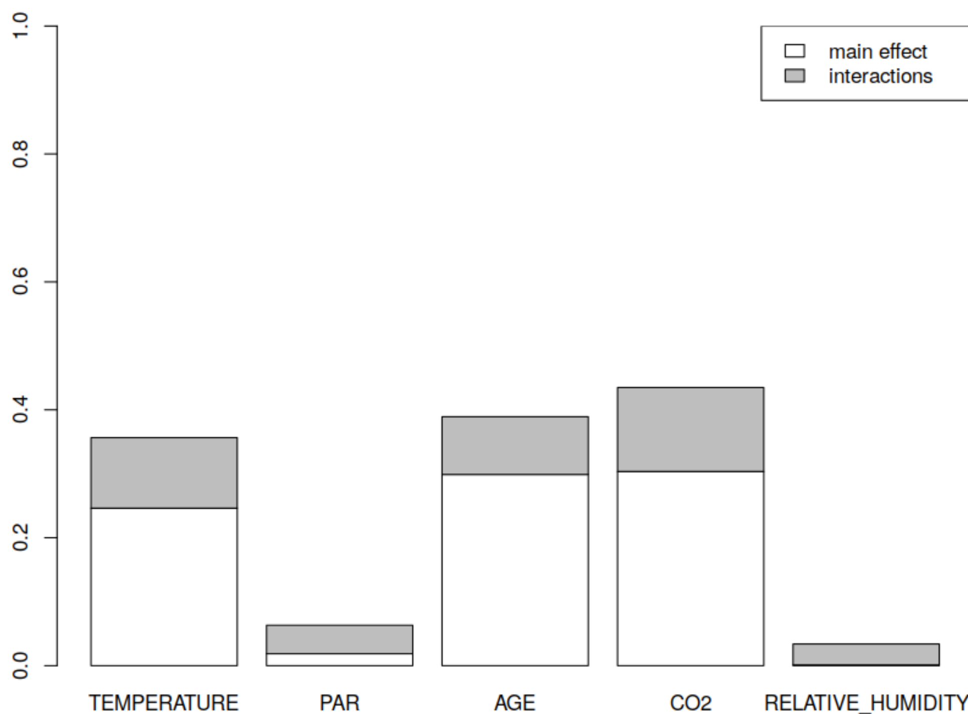


Figure 4.27: Results for the extended Fourier amplitude sensitivity test of the net photosynthetic rate of the assimilate production model

In figure 4.27 the results of the extended Fourier amplitude sensitivity test can be found. The low influence on the net photosynthetic rate can be again seen for the light intensity (PAR) and the relative humidity. The main effect on the photosynthesis of the humidity is considered so low that the white bar representing the sensitivity index can only be seen when zoomed in. However, the interaction of the humidity is non-zero. This confirms the findings of Sobol's method. When looking at the ranking of the most important input parameters, the results from the first five SA methods are greatly confirmed. The CO2 concentration is the most influential one, followed by the age and the temperature. The EFAST results are more accurate and compliant with all other SA methods, except Sobol's approach. This is especially true due to the reason that EFAST effect indices possess a higher convergence rate than Sobol for the same number of samples. Interaction effects occur for each input parameter, which was expected as Sobol has shown non-zero effects for all parameters. However for the most influential parameters (CO2 concentration, age, temperature) the amount of interaction effect does not exceed approximately 30% of the particular main effect. Recapitulating the results, it is evident that the calculated effect indices support the present outcomes of the applied SA methods. With the exception of Sobol's method, where the results where slightly deviating, the consistency of the findings emerges.

Chapter 5

Runtime and Memory Consumption

In this chapter the runtime and memory consumption of the "Sensitivity" plugin is examined. Since for the plugin user the only available functions are the implemented sensitivity analysis functions, each of them will be tested separately. The main insight is yielded by the dependency of the runtime and memory consumption on the number of input parameters. The tests were carried out using the binary tree model from the transformation example of chapter 2. The test code can be found on the supplementary CD. Due to the fact that in Java the full CPU time of a thread cannot be easily yielded with out-of-the-box methods, the runtime of an SA method is obtained by the function *nanoTime()*. Before a method is tested, *nanoTime()* is called and the return value is stored. When the SA method terminates, the value of *nanoTime()* is again stored. The difference of the two values is the runtime in nanoseconds of the SA function. However, it cannot be guaranteed that no context switch occurs within the CPU scheduler. This results in unreliable and non-reproducible runtime readings. In order to compensate this drawback, the runtime measurements are repeated 100 times (tradeoff between runtime and variance). Then the minimum value of the measurements is taken, because it can be assumed that for the minimal runtime the least number of scheduler interruptions occurs and thus the most accurate measurement is achieved. Figure 5.1 shows the fluctuations of the runtime along the test runs for the partial correlation coefficients where the number of input parameters is four. It can clearly be seen that the values are within a small corridor of approximately $0.7s$ width. However, along the simulations the runtime several times approaches its minimum which will be used as the measured runtime reading. The test machine for all measurements is a *Dell Vostro 5590 / CPU: Intel i7-10510U / RAM: 16GB*.

The examination of the memory consumption requires more attention. The memory usage can be divided into a Java part and an R part. For the Java part it is not trivial to measure the memory consumption because the memory management and garbage collection are internal mechanisms of Java and hence largely inaccessible for the programmer. However, the Java part is a fixed code layout and is seen only as a frame for the R calculations. Since it always possesses a constant memory usage and all the sensitivity calculations are done in the R process, the Java part can be

neglected. In other words, measuring the memory consumption of the Java part cannot give insight to the memory behavior of the SA methods, which is the examination target. Furthermore, the visual representation of the sensitivity plots is also carried out using R. Because of these facts, it was decided to only examine the memory occupancy of the R process. In order to measure the memory consumption of the SA methods, R offers the function *mem_used()* from the package "pryr" [34] that outputs the total occupied system memory. Unfortunately, R also incorporates inaccessible garbage collection. To avoid false measurements and in order to obtain a reasonable memory value, the internal function of the "Sensitivity" plugin *eval()* was temporarily altered in a way that in every execution step of the R command sequence the used memory was outputted to the command line. The output lines then were used to obtain the maximal memory consumption reading. This reading then is subtracted from the initial memory consumption of the R system. This procedure guarantees that the true value for each SA function is gathered. In table 5.1 the runtime and memory consumption measurements can be found. Additionally, the findings including a visual representation via bar plots can be seen in tables 5.2 - 5.8. In order to make the differences in memory allocation visible, these bars do not start at 0. Figures 5.2 and 5.3 show in a comparative manner the runtime and memory consumption when the number of parameters is 6.

Having obtained the runtime and memory consumption, these entities are related to the quality of information for each SA method. In other words, for the different functions a cost-benefit analysis is conducted. For this task, the paper [17] provides a rough categorization of the different SA approaches. However, for this chapter it was chosen to develop steadfast criteria in order to individually assign assessment readings. The sum of points each method achieves represents a dimension-less indicator for the quality of information. The runtime and memory consumption can be related to the information quality resulting in an efficiency indicator. Thus, the SA methods can be compared considering that property. The table 5.9 shows the corresponding results.

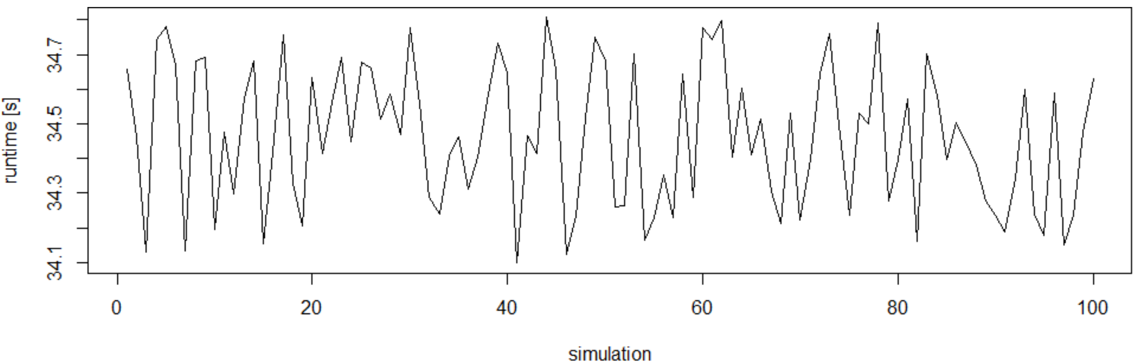
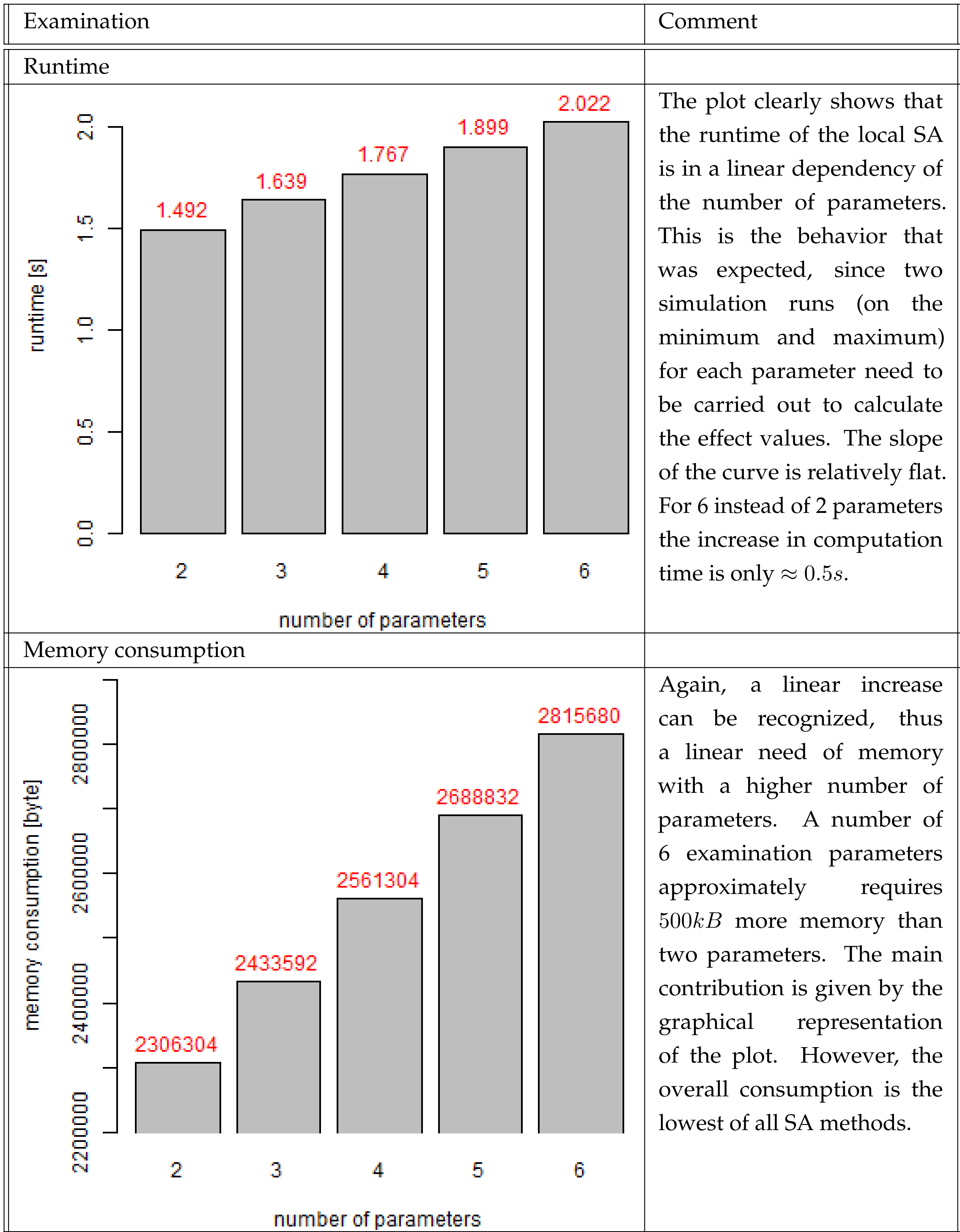


Figure 5.1: Deviation of the runtime for PCC with 4 parameters

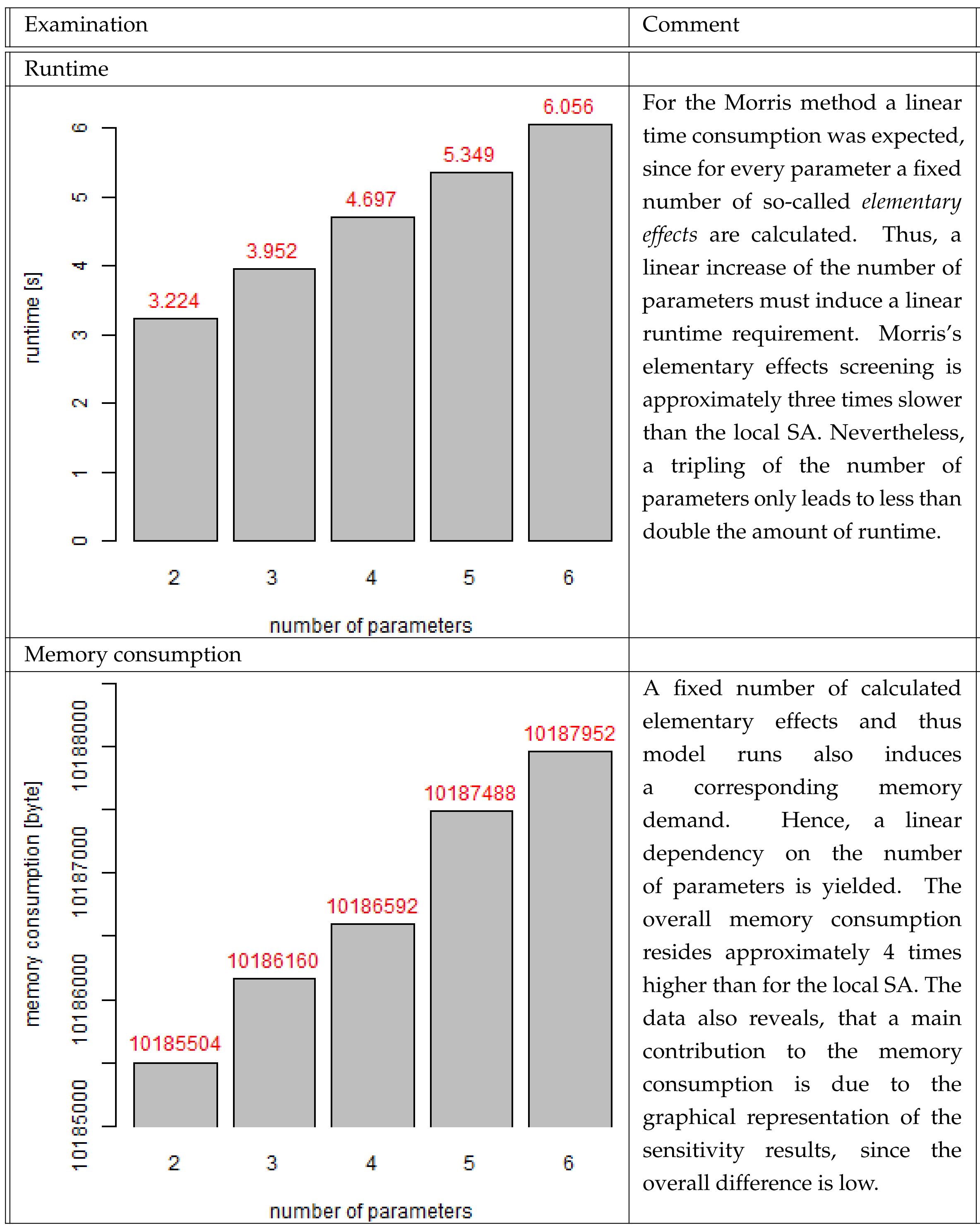
sensitivity analysis method \ number of parameters	2	3	4	5	6
Local Sensitivity Analysis					
runtime [s]	1.492	1.639	1.767	1.899	2.022
memory consumption [byte]	2306304	2433592	2561304	2688832	2815680
Morris's Elementary Effects Screening					
runtime [s]	3.224	3.952	4.697	5.349	6.056
memory consumption [byte]	10185504	10186160	10186592	10187488	10187952
Main And Interaction Effects On Extreme Values					
runtime [s]	1.579	1.842	2.340	3.234	5.033
memory consumption [byte]	15843920	15897544	15951536	16020432	16116016
Partial (Rank) Correlation Coefficients					
runtime [s]	14.090	22.929	34.096	48.028	64.502
memory consumption [byte]	6820560	6834584	6850144	6867888	6880776
Standardized (Rank) Regression Coefficients					
runtime [s]	12.157	17.747	23.432	29.198	35.093
memory consumption [byte]	6575504	6589528	6605088	6622832	6635720
Sobol's Method					
runtime [s]	42.335	109.163	228.155	416.185	697.454
memory consumption [byte]	6308512	6371704	6503392	6741792	7138648
Extended Fourier Amplitude Sensitivity Test					
runtime [s]	21.897	47.795	83.504	130.036	186.899
memory consumption [byte]	5693984	5716056	5753552	5812248	5897200

Table 5.1: Runtime and memory consumption measurements



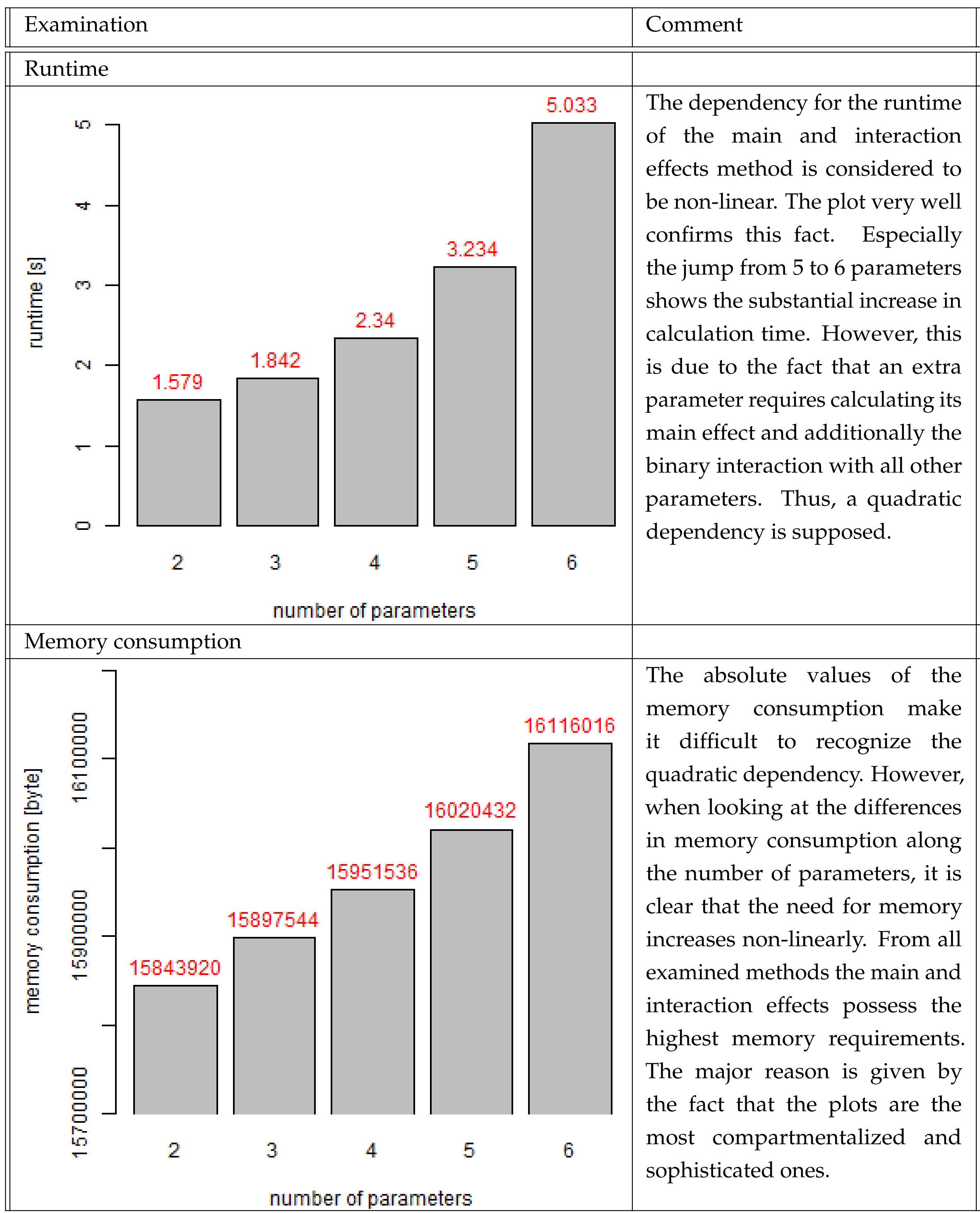
Examination	Comment
Runtime	
	The plot clearly shows that the runtime of the local SA is in a linear dependency of the number of parameters. This is the behavior that was expected, since two simulation runs (on the minimum and maximum) for each parameter need to be carried out to calculate the effect values. The slope of the curve is relatively flat. For 6 instead of 2 parameters the increase in computation time is only $\approx 0.5s$.
Memory consumption	
	Again, a linear increase can be recognized, thus a linear need of memory with a higher number of parameters. A number of 6 examination parameters approximately requires $500kB$ more memory than two parameters. The main contribution is given by the graphical representation of the plot. However, the overall consumption is the lowest of all SA methods.

Table 5.2: Runtime and memory consumption of the local sensitivity analysis



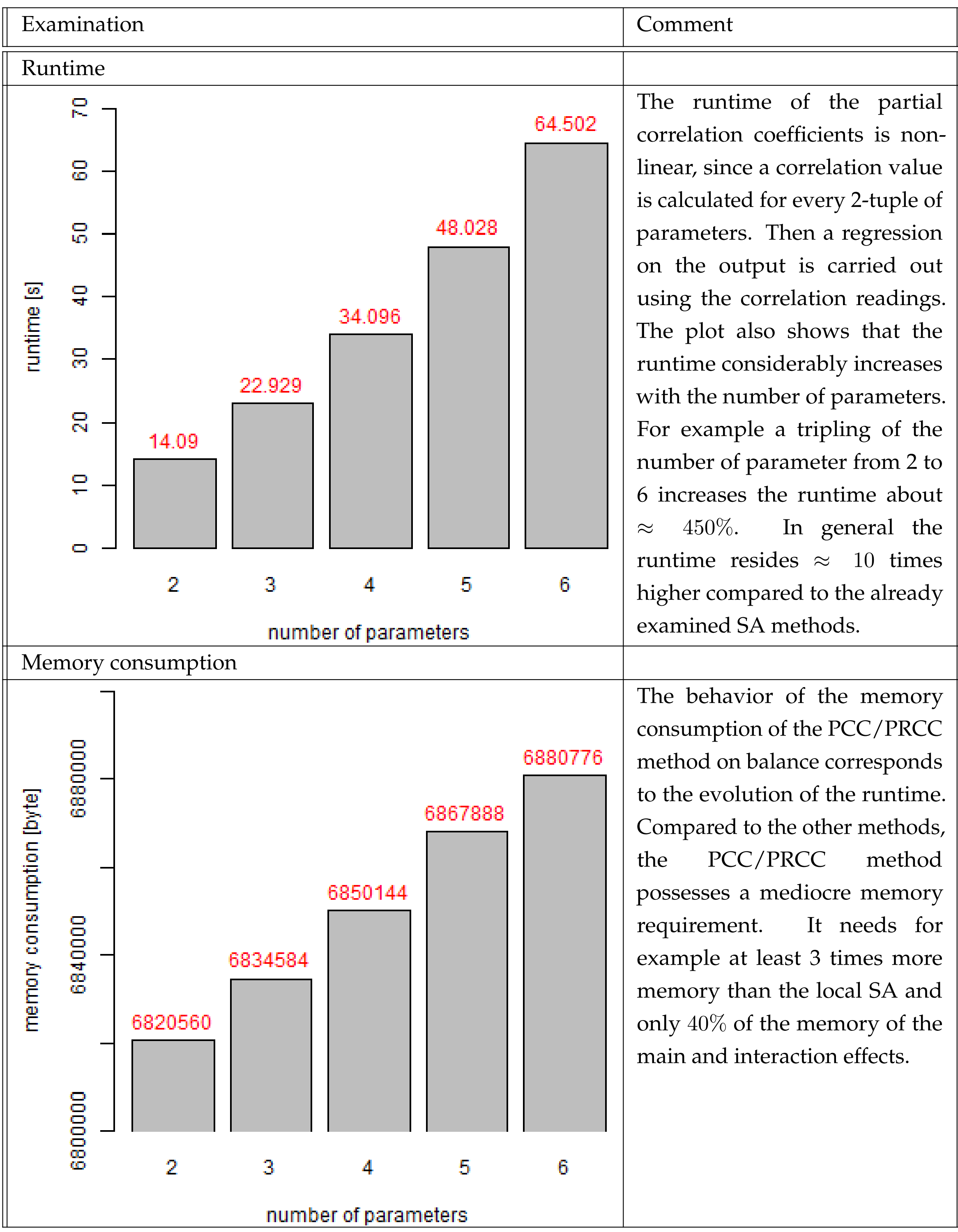
Examination	Comment
Runtime	
	For the Morris method a linear time consumption was expected, since for every parameter a fixed number of so-called *elementary effects* are calculated. Thus, a linear increase of the number of parameters must induce a linear runtime requirement. Morris's elementary effects screening is approximately three times slower than the local SA. Nevertheless, a tripling of the number of parameters only leads to less than double the amount of runtime.
Memory consumption	
	A fixed number of calculated elementary effects and thus model runs also induces a corresponding memory demand. Hence, a linear dependency on the number of parameters is yielded. The overall memory consumption resides approximately 4 times higher than for the local SA. The data also reveals, that a main contribution to the memory consumption is due to the graphical representation of the sensitivity results, since the overall difference is low.

Table 5.3: Runtime and memory consumption of Morris's elementary effects screening



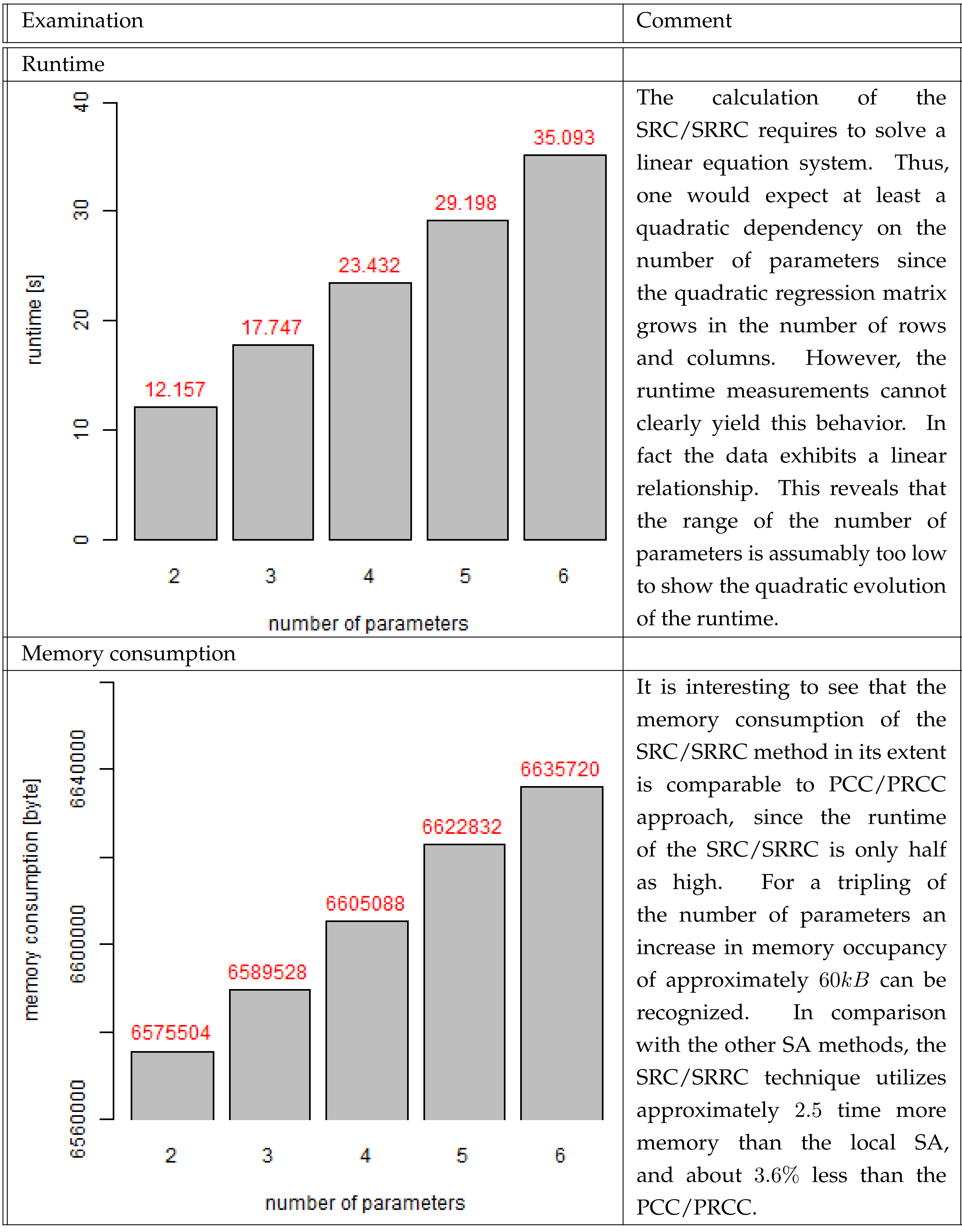
Examination	Comment
Runtime	
	The dependency for the runtime of the main and interaction effects method is considered to be non-linear. The plot very well confirms this fact. Especially the jump from 5 to 6 parameters shows the substantial increase in calculation time. However, this is due to the fact that an extra parameter requires calculating its main effect and additionally the binary interaction with all other parameters. Thus, a quadratic dependency is supposed.
Memory consumption	
	The absolute values of the memory consumption make it difficult to recognize the quadratic dependency. However, when looking at the differences in memory consumption along the number of parameters, it is clear that the need for memory increases non-linearly. From all examined methods the main and interaction effects possess the highest memory requirements. The major reason is given by the fact that the plots are the most compartmentalized and sophisticated ones.

Table 5.4: Runtime and memory consumption of the main and interaction effects on extreme values



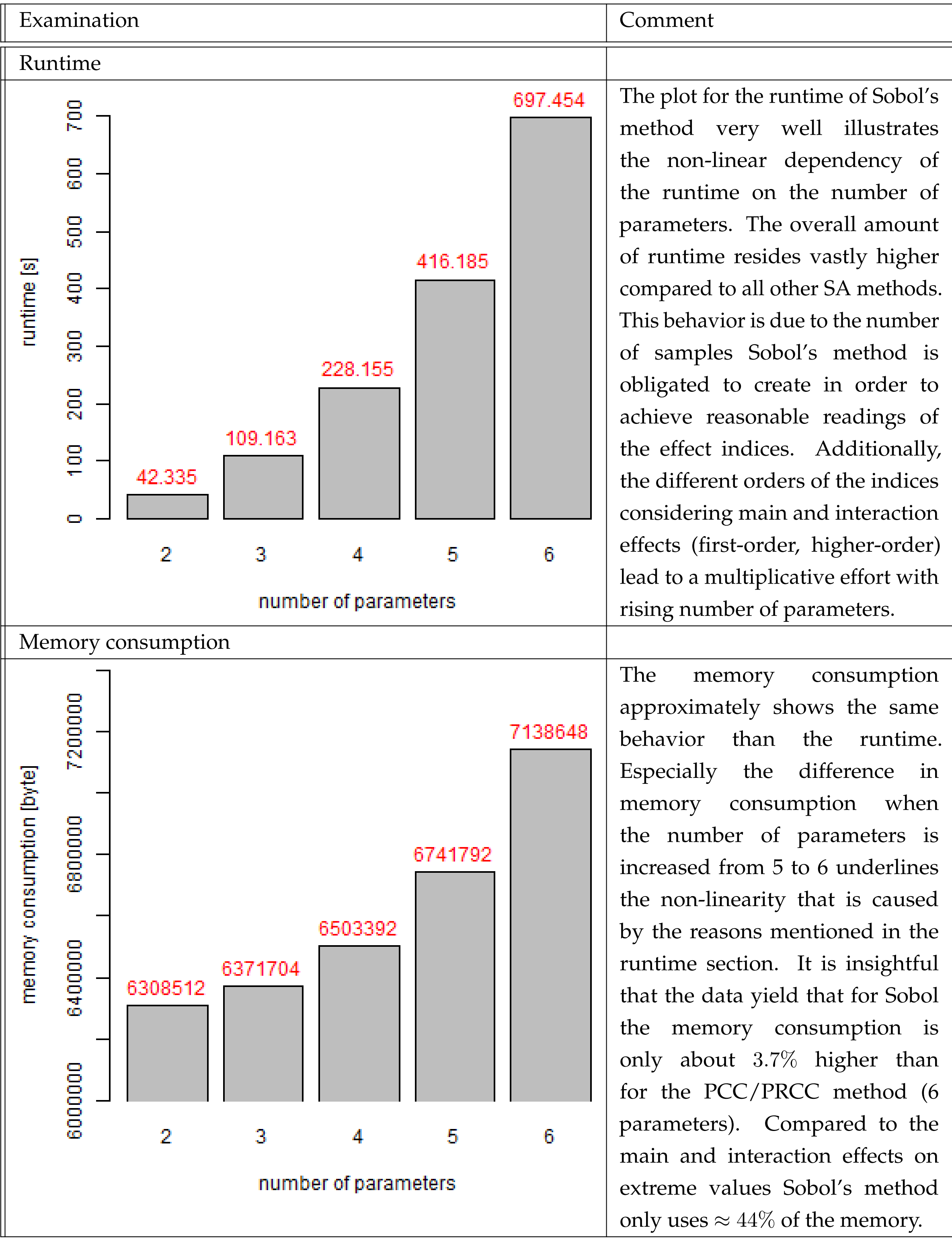
Examination	Comment
Runtime	
	The runtime of the partial correlation coefficients is non-linear, since a correlation value is calculated for every 2-tuple of parameters. Then a regression on the output is carried out using the correlation readings. The plot also shows that the runtime considerably increases with the number of parameters. For example a tripling of the number of parameter from 2 to 6 increases the runtime about $\approx 450\%$. In general the runtime resides ≈ 10 times higher compared to the already examined SA methods.
Memory consumption	
	The behavior of the memory consumption of the PCC/PRCC method on balance corresponds to the evolution of the runtime. Compared to the other methods, the PCC/PRCC method possesses a mediocre memory requirement. It needs for example at least 3 times more memory than the local SA and only 40% of the memory of the main and interaction effects.

Table 5.5: Runtime and memory consumption of the partial (rank) correlation coefficients (PCC/PRCC)



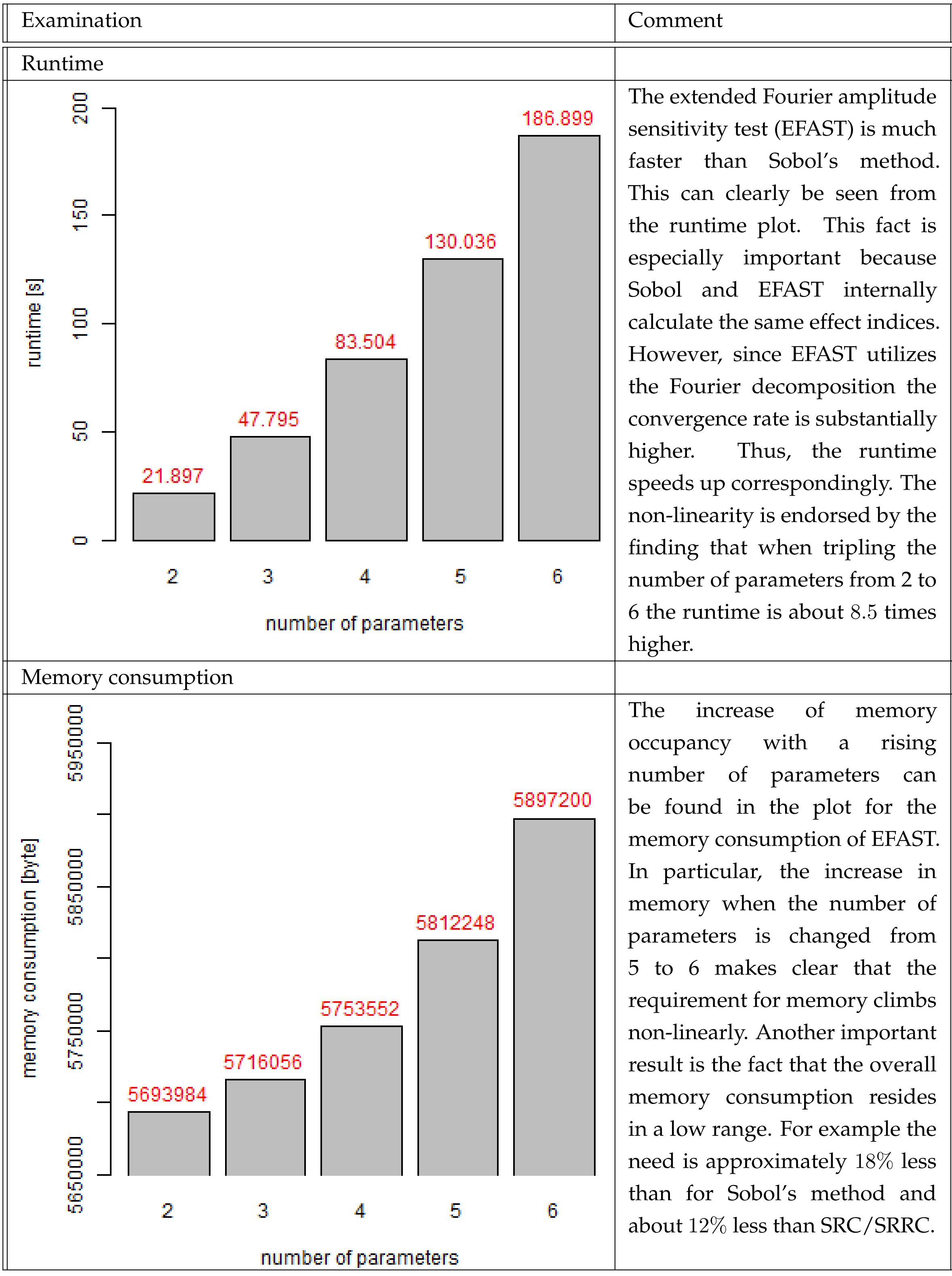
Examination	Comment
Runtime	
	The calculation of the SRC/SRRC requires to solve a linear equation system. Thus, one would expect at least a quadratic dependency on the number of parameters since the quadratic regression matrix grows in the number of rows and columns. However, the runtime measurements cannot clearly yield this behavior. In fact the data exhibits a linear relationship. This reveals that the range of the number of parameters is assumably too low to show the quadratic evolution of the runtime.
Memory consumption	
	It is interesting to see that the memory consumption of the SRC/SRRC method in its extent is comparable to PCC/PRCC approach, since the runtime of the SRC/SRRC is only half as high. For a tripling of the number of parameters an increase in memory occupancy of approximately $60kB$ can be recognized. In comparison with the other SA methods, the SRC/SRRC technique utilizes approximately 2.5 time more memory than the local SA, and about 3.6% less than the PCC/PRCC.

Table 5.6: Runtime and memory consumption of the standardized (rank) regression coefficients (SRC/SRRC)



Examination	Comment
Runtime	
	The plot for the runtime of Sobol's method very well illustrates the non-linear dependency of the runtime on the number of parameters. The overall amount of runtime resides vastly higher compared to all other SA methods. This behavior is due to the number of samples Sobol's method is obligated to create in order to achieve reasonable readings of the effect indices. Additionally, the different orders of the indices considering main and interaction effects (first-order, higher-order) lead to a multiplicative effort with rising number of parameters.
Memory consumption	
	The memory consumption approximately shows the same behavior than the runtime. Especially the difference in memory consumption when the number of parameters is increased from 5 to 6 underlines the non-linearity that is caused by the reasons mentioned in the runtime section. It is insightful that the data yield that for Sobol the memory consumption is only about 3.7% higher than for the PCC/PRCC method (6 parameters). Compared to the main and interaction effects on extreme values Sobol's method only uses $\approx$ 44% of the memory.

Table 5.7: Runtime and memory consumption of Sobol's method



Examination	Comment
Runtime	
(runtime plot)	The extended Fourier amplitude sensitivity test (EFAST) is much faster than Sobol's method. This can clearly be seen from the runtime plot. This fact is especially important because Sobol and EFAST internally calculate the same effect indices. However, since EFAST utilizes the Fourier decomposition the convergence rate is substantially higher. Thus, the runtime speeds up correspondingly. The non-linearity is endorsed by the finding that when tripling the number of parameters from 2 to 6 the runtime is about 8.5 times higher.
Memory consumption	
(memory consumption plot)	The increase of memory occupancy with a rising number of parameters can be found in the plot for the memory consumption of EFAST. In particular, the increase in memory when the number of parameters is changed from 5 to 6 makes clear that the requirement for memory climbs non-linearly. Another important result is the fact that the overall memory consumption resides in a low range. For example the need is approximately 18% less than for Sobol's method and about 12% less than SRC/SRRC.

Table 5.8: Runtime and memory consumption of the extended Fourier amplitude sensitivity test (EFAST)

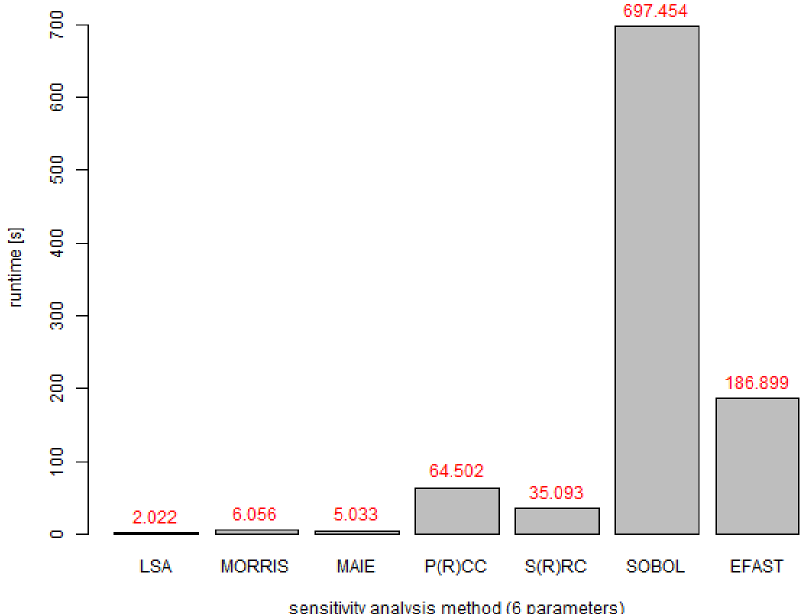


Figure 5.2: Comparison of the runtime for the sensitivity analysis methods

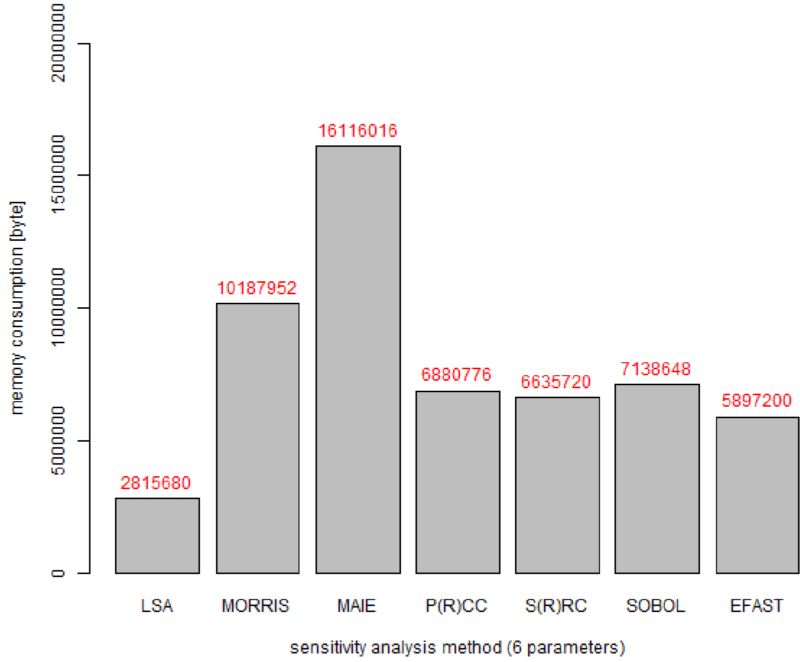


Figure 5.3: Comparison of the memory consumption for the sensitivity analysis methods

Criterion	LSA	MORRIS	MAIE	P(R)CC	S(R)RC	SOBOL	EFAST
Main effect	1	2	2	2	2	3	3
Interaction effects	0	0	2	0	0	3	2
Effect direction	2	0	3	2	2	0	0
Effect height relative	2	2	0	3	3	3	3
Effect height absolute	0	0	3	0	0	0	0
Input space coverage	1	2	2	3	3	3	3
Reading robustness	0	1	1	2	2	3	3
Uncertainty evaluation	0	0	0	3	3	3	0
Sum (=information)	6	7	13	15	15	18	14
Runtime efficiency ($= \frac{\text{information} \times \text{s}}{\text{runtime}}$)	2.97	1.16	2.58	0.23	0.43	0.026	0.075
Memory efficiency ($= \frac{10^6 \times \text{information} \times \text{byte}}{\text{memory consumption}}$)	2.13	0.69	0.81	2.18	2.26	2.52	2.37

Table 5.9: Information quality assessment of the sensitivity analysis methods

The table 5.9 shows the assessment of the different SA methods considering information quality. For every criterion an assessment value from $0 \ldots 3$ indicates the goodness of the particular SA method. A value of 0 accords to the fact that no information is delivered on that criterion. For values from $1 \ldots 3$ the depth of the information is assessed, where 3 coincides with the best possible reading. The dimension-less information value then is related to the runtime and memory consumption by the division formula given in the cell entry. This yields a dimension-less efficiency indicator for the runtime and the memory consumption.

The table clearly shows that Sobol's method reveals the deepest information on parameter sensitivity. Also the PCC/PRCC and SRC/SRRC approaches give good insight in parameter behavior. As expected, the local SA and Morris's method can be seen as methods that give a hint which parameters should be examined further, but cannot deliver deep information on parameter sensitivity. The main and interaction effects on extreme values are advantageous for estimating the direction of a parameter effect and its absolute effect height (model output on minimum and maximum input parameters). All other methods only output readings that are suitable for parameter ranking or comparison, since they deliver effect indices that are standardised on 1 or percentage deviation values like the local SA, but no nominal output evaluation. Despite the fact that Sobol and EFAST calculate the same effect indices, the assessment considers the user

representation of the calculations. Hence, the differences in graphical illustration for example yield that for EFAST the interaction effect contribution cannot be attributed to a single input parameter, because all interaction indices are summed up. Because of that EFAST possesses a lower assessment reading than Sobol. When looking at the informative entity of the sensitivity measure validity, an uncertainty range is only provided for Sobol's effect indices and the PCC/PRCC and SRC/SRRC. Since many plant models contain randomness, the output often deviates slightly. Thus, sensitivity values possess a certain degree of robustness, with the manner that calculations are stabilized by carrying out repeated experiments. The local SA and the main and interaction effects only conduct examinations on fixed input parameter configurations. Thus, in case of output fluctuations the obtained values are not reliable unless the user averages the output over simulation runs with an identical parameter setting, which was for example done for the beech tree.

Considering the runtime efficiency, the methods that deliver the highest amount of sensitivity information are the slowest ones. This group consists of the PCC/PRCC, SRC/SRRC, Sobol's method and EFAST. All efficiency values are well below 0.5. In particular Sobol is largely time-consuming. Thus, the efficiency is the lowest of all SA approaches. Also EFAST, which is more than 3 times faster than Sobol, possesses a relatively low runtime efficiency reading. A balanced ratio of runtime versus information is yielded by PCC/PRCC and SRC/SRRC. The assessment rating of these methods is high while requiring an acceptable runtime effort. The most runtime efficient approaches are the local SA and the main and interaction effects on extreme values. That is why it is preferable to initially screen the input parameters with these methods to omit non-sensitive parameters before ever applying more profound techniques like Sobol. This is especially important when the number of parameters is very high, since the deeper methods have non-linear runtimes. Because of that, the runtime efficiency will drop correspondingly with an increasing number of parameters. A mediocre runtime efficiency was found for Morris's elementary effects screening. Nevertheless, in contrast to Sobol or EFAST the runtime of Morris's method is reasonable and the yielded parameter screening is very useful for the further analysis proceeding.

The memory efficiency for most methods shows the opposite to the runtime efficiency. In particular the PCC/PRCC, SRC/SRRC, Sobol's method and EFAST possess high efficiency reading of approximately the same magnitude. This yields that the methods that have high runtimes on the other side very economically utilize the memory. The local SA also is relatively low on memory consumption, since only a fixed and low number of samples needs to be calculated and stored. Thus, the efficiency reading is high. The most insightful result is given by the fact that Morris's method and the main and interaction effects - that belonged on balance to the runtime efficient methods - are inefficient in terms of memory consumption. However, this is mainly caused by the plots that incorporate different R packages which vary in their memory utilization.

Chapter 6

Summary and Outlook

This master thesis deals with the development and implementation of a plugin for the growth grammar related interactive modeling platform GroIMP that deploys sensitivity analysis of plant models through the utilization of the statistical computing software R. In chapter 1 an introduction to the field of sensitivity analysis is given. Additionally, it is described why there is a need for the sensitivity analysis capability of plant models within GroIMP. In chapter 2 implementation details of the developed plugin are provided. In particular, it is shown how to connect R to GroIMP. Furthermore, it is presented what the single entities of the plugin framework are and how a plant model must be prepared in order to apply SA methods to it. In chapter 3 for all implemented SA approaches the mathematical foundation is exposed. This includes how the concrete calculation of the sensitivity measures - that are outcomes of the SA methods - is conducted (effect indices, correlation/regression values, deviation readings). In chapter 4 the plugin is tested with two existing plant models, where respectively a selection of the input parameter space is examined by every single SA function deployed. On the one hand a beech tree is explored considering two outputs, namely the total carbon production and the tree height. On the other hand an assimilate production model is analyzed regarding the net photosynthetic rate. In chapter 5 the runtime and the memory consumption of the SA methods are examined. Additionally, an assessment of quality of information is done for all SA methods. These findings then are related to one another resulting in individual efficiency evaluations.

Sensitivity analysis of simulation models examines input parameters in terms of their influence on a dedicated output. Hence, for computer plant models it is of particular interest to study the model behavior, since the complexity of biological processes is high. Thus, in order to understand the connection between input and output, to check the model validity or to rank parameters by their importance, advanced sensitivity analysis techniques are desirable especially for plant models within plant modeling software. The Java-based 3D plant modeling platform GroIMP in its version 1.6 does not incorporate any automatically conducted systematic sensitivity analysis capability. Since the statical computing platform R possesses various sensitivity analysis functions

that output the desired sensitivity measures, when sufficient simulation data is provided, it is very beneficial to apply the sensitivity analysis of R to the plant models of GroIMP, which was the motivation for this thesis. The task was done by implementing a GroIMP plugin - since all functionalities of GroIMP are arranged via plugins - named "Sensitivity". The paper reference [17] gives a hint which SA approaches within R are useful. Thus, the plugin possesses seven user-accessible SA functions which are implementations of the following SA methods: local sensitivity analysis, Morris's elementary effects screening, main and interaction effects on extreme values, partial (rank) correlation coefficients, standardized (rank) regression coefficients, Sobol's method and extended Fourier amplitude sensitivity test. In order to ensure user-friendliness and comparability, it is predetermined for the developed plugin that the fraction of the input space, whose sensitivity is meant to be examined, is an array of arbitrary length and must contain only one-dimensional and numeric-valued model parameters. Furthermore, the SA methods require that the output is a single, one-dimensional, numeric-valued reading.

GroIMP is based on Java. Hence, the "Sensitivity" plugin is also written in Java. Since it was decided to use R in order to conduct SA analysis, it is mandatory to connect both computing languages in order to execute commands or to transfer simulation data. However, there is no out-of-the-box approach within Java programs to make the R language interpretable, due to the complete difference in paradigms and programming framework. Nevertheless, there are Java plugins available that deploy the R language in the Java environment. But it was found that those do not suit the platform-independence of GroIMP, since they come along with a lot of disadvantages like Java-version requirements, incomplete R deployment (missing R packages) or OS-problems. Because of that, the connection of R and Java was carried out by developing a communication framework based on process communication (reading and writing of process streams). The internal and thus user-inaccessible class "RConnection" treats all tasks that are necessary in order to establish a connection to R, to execute R commands and to preprocess their return values. This also includes platform-different ways of proceeding, for example in obtaining the R executable or opening plot windows. It is also tested whether all R packages, that are a prerequisite for performing SA, are installed. If R is not installed or a package is missing, a message will indicate which steps need to be taken by the plugin user. Since in GroIMP a plant simulation most commonly consists of multiple functional parts and the naming of the corresponding functions or program routines can be arbitrary, a concrete identification of the simulation routine, the output measurement and the output name is crucially important. That is why the developed interface "Simulation" contains dummy functions that must be correctly overwritten by the plugin user. By that, the SA functions are able to systematically vary the input parameters (according to the configuration required by the SA procedure), run the simulation process and gather the dedicated output. At the beginning it was predetermined that a model parameter is a one-dimensional, numeric-valued variable. In Java the natural choice for such parameters would be primitive datatypes like *int*, *long*, *double* or *float*. However, if in a call of an SA

function the input parameters were passed by value, which is in Java the case for primitive datatypes, a systematic varying of the parameters within the function would be impossible. Because of that, to create reference type input parameters, the developed wrapper class "NumberRef" can hold any value of a number datatype and contains getter and setter for all of them. Thus, the process of boxing and unboxing in order to pass value-type variables as references, is done by the use of the tailored class "NumberRef". The need of the "Simulation" interface and the "NumberRef" class makes clear that the code of a plant model must be prepared before being able to carry out SA. The interface needs to be implemented by the simulation-containing class in GroIMP. The three dummies must be overwritten accordingly. Additionally, all model parameters that are meant to be examined must be replaced by a "NumberRef" version. An SA function then can be called by passing the appropriate arguments including an array of parameters and an instance of the main class itself (usually through the keyword *this* when the function is called within the main class). All seven SA functions are contained in the class "Sensitivity". They are marked as static, so the functions can be directly used without the user needing to instantiate an object of the class. Summarizing it can be said that using the plugin is very easy and only requires two steps: preparing the plant model as described and then calling the desired SA function. When the SA is finished a plot window opens up and a graphical representation of the results is displayed.

Local SA measures how much (in percent) the output changes when a single parameter is varied around its initial reading. It delivers a rough impression of the model behavior, but does not examine interaction effects. It can be considered as the most simple SA approach. Morris's elementary effects screening is a method that tries to rank the input parameters by their expected effect strength. For that outcome for every parameter a fixed number of so-called *elementary effects* is calculated. After that a statistical analysis of the effect readings delivers the desired parameter sensitivity measures. The approach to calculate the main and interaction effects on extreme values examines the height of the model output when the input parameters are set to their extreme values. The main effects are obtained by setting a single parameter to its minimum or maximum. This can reveal the overall direction of an effect and also the most influential parameters. In order to see whether two parameters interact by the means of synergetically influencing the output, all combinations of the parameter setting (extreme values) for the two parameters are tested. The results are well interpretable in the corresponding interaction plot. The partial (rank) correlation coefficient approach calculates the amount of influence between input and output using correlation techniques. This results in sensitivity coefficients that determine the amount of linearity between the input and output. The rank versions of these coefficients address the case when a non-linear relationship is expected in order to obtain correct influence measures. The standardized (rank) regression coefficients regresses the output on the input values. The calculated regression coefficients hence are a direct importance measure of the input parameters. The rank version again considers the possible non-linear parameter behavior. Sobol's method,

which belongs to the variance-decomposition methods, can be considered as the deepest SA approach of all implemented ones. It calculates so-called *effect indices* of different order. A first-order effect is a parameter's main effect. Higher-order indices represent interaction effects. Sobol requires a very high number of samples in order to achieve less-error-prone indices. The extended Fourier amplitude sensitivity test (EFAST) internally calculates the same effect indices as Sobol. However, the output variance decomposition is carried out using Fourier decomposition. This leads to an increased convergence rate compared to Sobol. Thus, a lower number of samples is required to achieve the same accuracy. Nevertheless, in the graphical representation of the EFAST results the higher-order effect contributions are summed up; making - unlike Sobol - a distinct effect contribution assignment difficult.

The plugin was tested with two plant models, a beech tree and an assimilate production model. The beech is a plant model with a simulated light source, primary and secondary growth, leaves and a photosynthesis implementation. For the SA different input parameters were examined regarding the influence on the tree height and the total carbon production. Through its contained random behavior, the outputs had to be averaged over 100 simulation runs. The outcomes of the SA methods slightly differed. But in the end, for the tree height it was found out that the vitality, the number of light days and the photosynthetic efficiency were most influential. A variation of the branching angle or the vitality threshold had a negative effect on the tree height. These findings were consistent along every SA method. The interaction plot furthermore revealed that especially the parameters that are in association with photosynthesis (light days, PPFD_FACTOR, efficiency) heavily interact by the means of boosting the slope of the output curve when two parameters were maximal. It can be expected that the total carbon production is mainly affected by the parameters that influence the photosynthesis. This is exactly what was found. The local SA already showed that the deviation in output is very high when these parameters (leaf area, vitality, light days, PPFD_FACTOR, efficiency) were slightly varied. High sensitivity on the five parameters was also indicated by nearly all other methods. However, the results of Morris's screening were slightly different and could not completely reveal the model behavior that was consistently found by all other methods. Concerning interactions, also the photosynthesis-connected parameters greatly interacted. The setting of the maximum value always led to a massive increase in output. This is the case for the total carbon production as well as the tree height. For PCC/PRCC and SRC/SRRC it held true that for both outputs the approaches possessed a relatively narrow uncertainty range. This indicates that the number of samples was sufficiently high. However, this cannot be in total said for Sobol's method. On the one hand uncertainties had been expected when looking at the amount of interaction effects. But the effect index instabilities mainly come from a too low number of simulation samples. This reveals a major drawback of Sobol's method. The examination of the assimilate production model was very insightful, since this model possesses a vastly superior implementation of the photosynthetic processes compared to the beech. Additionally, this model does not contain any

randomness. Thus, the gathered output values are more stable leading to less uncertain sensitivity measures. The SA of the model was carried out considering the influence of five input parameters (CO2 concentration, temperature, age, photosynthetically active radiation, relative humidity) on the net photosynthetic rate. All in all, the findings for the assimilate production model along the SA methods were very consistent, which shows that the reliability of the sensitivity measures also depends on the model behavior itself (determination, non-randomness). The relative humidity parameter possesses a very low sensitivity and the influence is negligible compared to the other parameters. The temperature and the age have a negative influence of the photosynthesis. The most important parameter regarding the net photosynthetic rate is the CO2 concentration, followed by the amount of photosynthetically active radiation (PAR). Interaction effects take place with the exception for the relative humidity where the influence in connection with other parameters was very low. For the CO2 concentration the biggest gap between the minimum and maximum curves of the interactions can be recognized, underlining its sensitivity on the net photosynthetic rate. For PCC/PRCC and SRC/SRRC the uncertainty is very low and the calculated coefficients perfectly comply with the other results. However, the Sobol indices come with high uncertainties, but the overall separation between important and non-important parameters was confirmed.

The runtime and memory consumption among all SA methods is vastly different. It was tested for a varying number of parameters. As expected the local SA had a linear and fast runtime ($\approx 2s$, 6 parameters). The memory consumption is the lowest of all methods. Morris's elementary effects screening and the main and interaction effects on extreme values were three times slower than the local SA ($\approx 6s$, $\approx 5s$, 6 parameters). Additionally, compared to all other approaches the memory consumption was found out to be approximately $50\% - 100\%$ higher, yielding that these two methods were the most memory-consuming ones. The runtime of the PCC/PRCC and the SRC/SRRC was 6-10 times higher in comparison to Morris's method ($\approx 65s$, $\approx 35s$, 6 parameters). The memory consumption here was in the mediocre range and resided approximately 2.5 times greater than the one of the local SA. Non-linear runtime had been expected for most of the methods. In particular the test data of Sobol and EFAST showed a non-linear increase in runtime with rising number of parameters. Moreover, Sobol's method was the slowest of all SA approaches ($\approx 700s$, 6 parameters). This is exactly what had been expected, since Sobol processes the highest amount of simulation data. EFAST was essentially faster than Sobol ($\approx 190s$, 6 parameters), but compared to the other methods it was also greatly time consuming. The memory consumption of Sobol's method and EFAST resided in the mediocre range and was approximately comparable to PCC/PRCC and SRC/SRRC. Regarding the quality of information, Sobol's method delivers the deepest insight in parameter sensitivity followed by PCC/PRCC, SRC/SRRC and EFAST. The interaction plot of the method of main and interaction effects on extreme values is also very insightful since the visual representation can yield a good impression on the overall parameter behavior for interactions and additionally the corresponding absolute output height. The local SA

and Morris's screening must be considered as methods that help to get a first impression on parameter sensitivity and also help carrying out a preselection of important parameters that should be further examined, since the deeper methods are computationally expensive and the effort is commonly in a non-linear functional dependence of the number of parameters. When the runtime or the memory consumption of an SA method is related to its information quality one can obtain a runtime or memory efficiency reading. The local SA, Morris's screening and the main and interaction effects on extreme values belong to the group of methods that are runtime efficient. Induced by the high runtime Sobol's method possesses a very low runtime efficiency reading. Moreover, PCC/PRCC, SRC/SRRC and EFAST show low values. Regarding memory efficiency, for the runtime-inefficient methods the opposite is the case. They posses a very high memory efficiency. Due to their effort in displaying graphical results, Morris's method and the main and interaction effects on extreme values receive low memory efficiency values. The local SA approach is the only method that is simultaneously runtime and memory efficient.

Considering the developed plugin one can say that the seven chosen and implemented SA approaches are a good cross-selection of available SA methods, since the addressed objective and revealed information are eclectic. However, the plugin could be supplemented by additional functionality for example further SA approaches like the sequential bifurcation screening method [35] or Csiszar F-divergence sensitivity indices [36]. In addition, a method that automatically plots the functional dependence of the input and output could be helpful in order to examine the model output evolution regarding extreme values or break-even points. Furthermore, since R possesses sophisticated data analysis capability, methods that conduct statistical tests or maintain value distribution charts could be extremely helpful in order to be able to deeply analyze plant models within GroIMP.

The examination of the beech tree has revealed that for randomized plant models the output is slightly deviating for one and the same parameter configuration. However, especially for methods like local SA, Morris's screening or main and interaction effects on extreme values the output fluctuations result in unreliable and incomparable sensitivity measures. These readings make it very difficult to attribute the sensitivity height to the parameter itself or to the model randomness, leading to incorrect interpretations. Because of that, in the examination of the beech the output had to be manually averaged over 100 simulation runs in order to decrease the variance and in this way to stabilize the sensitivity values. Thus, it would be advantageous for the plugin if for the different SA methods call parameters could be set in case of randomness-containing models, in order to tell that the model contains randomness and on how many simulations the output shall be processed. It is also imaginable that the output is not solely averaged by calculating the mean, but also every mathematical procedure like the variance, median, minimum, maximum or the application of any arbitrary function could be enabled.

The tests of Sobol's method have shown in both examinations a specific and often high degree of

uncertainty of the effect indices. This can be attributed to the number of samples that are used in order to calculate the indices. In the current plugin implementation the number of samples is chosen programmatically depending on the number of examined parameters. However, the plots show that it would be beneficial to give the user more control on that particular quantity. Thus, the uncertainty could be reduced depending on the requirements given by the concrete plant model. Furthermore, the number of samples could be dynamically adapted by the index calculation itself. Hence, when the error drops below an threshold - that is set by the user - the calculation can be stopped. As a consequence, the runtime could also be improved when a lower number of samples yields the same sensitivity accuracy.

In order to be able to carry out sensitivity analysis with the developed plugin, the user must prepare the code by implementing the described interface and by replacing value-typed model variables by reference-typed ones. This procedure is due to the structural architecture of Java/XL and GroIMP. Nevertheless, since XL is a superset of Java, special compiler tokens could be introduced for the purpose of marking variables and functions. By adding special prefixes or suffixes to the identifiers of functions or variables, the model parameters that are meant to be examined and the simulation function could be easily marked, diminishing the effort of the plugin user to prepare a plant model down to the minimum.

In the plugin the output generating function that is needed for SA has to be defined by the user. Structural plant information in GroIMP is commonly obtained by the use of XL graph queries, since the plant and its parts are internally represented by a graph. This is how the tree height was yielded in the examination of the beech model. However, since structural information of plants is very often subject of SA, it would be favorable if regularly used graph queries, that obtain for example the tree height, the extent in the xy-plane, the total volume or the surface area, would be available as a convenience offer within the plugin.

The graphical representation of the outcome of the SA methods was done with R. Hence, the plot window that opens up after a function terminates only exists in context of the R process. Furthermore, the plots are created with the standard R packages for plots and charts. This indicates that a further processing of the plots or the usage of packages with more sophisticated plot options is not possible in the current plugin. The author of this thesis has developed - as part of a research internship [37] - a charting plugin for GroIMP based on the R package "ggplot2". This plugin is called "Rchart". It makes the far-reaching capabilities of the "ggplot2" package in GroIMP available and allows the user to create advanced plots and to inspect or export the data. However, since "Rchart" was still in a developing state and in order to ensure the complete executability, it was chosen to develop the "Sensitivity" plugin independently without any mutual dependence. Nevertheless, it is of advantage to combine both plugins in order to enhance the plot quality of the SA plots, to make them more user-customizable and to ease the result extraction.

Bibliography

[1] B. Iooss and P. Lemaître, "A Review on Global Sensitivity Analysis Methods," in *Simulation-Optimization of Complex Systems. Operations Research / Computer Science Interfaces Series*, G. Dellino and C. Meloni, Eds. Springer, Boston, MA, 2015, vol. 59, pp. 101–122.

[2] S. Jørgensen and B. Fath, "2 - Concepts of Modelling," in *Fundamentals of Ecological Modelling*, ser. Developments in Environmental Modelling, S. Jørgensen and B. Fath, Eds. Elsevier, 2011, vol. 23, pp. 19–93.

[3] J. Salciccioli, Y. Crutain, M. Komorowski, and D. Marshall, "Sensitivity Analysis and Model Validation," in *Secondary Analysis of Electronic Health Records*. Springer, Cham, 2016, pp. 263–271.

[4] A. Saltelli, *Global Sensitivity Analysis: The Primer*. John Wiley, 2008.

[5] J. Landsberg and P. Sands, "Chapter 8: Modelling Tree Growth: Concepts and Review," in *Physiological Ecology of Forest Production: Principles, Processes and Models*, J. Landsberg and P. Sands, Eds. Academic Press, 2011, pp. 221–240.

[6] V. Bagad, *Management Information Systems*. Technical Publications, 2009.

[7] D. Pannell, "Sensitivity Analysis of Normative Economic Models: Theoretical Framework and Practical Strategies," *Agricultural Economics*, vol. 16, no. 2, pp. 139–152, 1997.

[8] Q. Wu and P.-H. Cournède, "A Comprehensive Methodology of Global Sensitivity Analysis for Complex Mechanistic Models with an Application to Plant Growth," *Ecological Complexity*, vol. 20, pp. 219–232, 2014.

[9] L. Thabane, L. Mbuagbaw, S. Zhang, Z. Samaan, M. Marcucci, C. Ye, M. Thabane, L. Giangregorio, B. Dennis, D. Kosa, V. Debono, R. Dillenburg, V. Fruci, M. Bawor, S. Lee, G. Wells, and C. Goldsmith, "A Tutorial on Sensitivity Analyses in Clinical Trials: The What, Why, When and How," *BMC Medical Research Methodology*, vol. 13, 2013.

[10] S. Balaman, "Chapter 5 - Uncertainty Issues in Biomass-Based Production Chains," in *Decision-Making for Biomass-Based Production Chains*, S. Balaman, Ed. Academic Press, 2019, pp. 113–142.

[11] F. Civan, "Model-Assisted Analysis and Interpretation of Laboratory and Field Tests," in *Reservoir Formation Damage: Fundamentals, Modeling, Assessment, and Mitigation*, F. Civan, Ed. Gulf Publishing Company, 2000, p. 561.

[12] H. Gupta and S. Razavi, "Chapter 20 - Challenges and Future Outlook of Sensitivity Analysis," in *Sensitivity Analysis in Earth Observation Modelling*, G. Petropoulos and P. Srivastava, Eds. Elsevier, 2017, pp. 397–415.

[13] A. Saltelli and P. Annoni, "How to Avoid a Perfunctory Sensitivity Analysis," *Environmental Modelling & Software*, vol. 25, no. 12, pp. 1508–1517, 2010.

[14] M. Henke, "GroIMP v1.4.2 Introduction and Overview," 2013, [Online]. Available: http://www.uni-forst.gwdg.de/~wkurth/ssc13/GroIMPIntroduction.pdf. Accessed: 04|17|20.

[15] O. Kniemeyer, R. Hemmerling, and W. Kurth, "GroIMP," [Online]. Available: http://www.grogra.de/. Accessed: 04|17|20.

[16] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2020, [Online]. Available: https://www.R-project.org/. Accessed: 05|11|20.

[17] J. Thiele, W. Kurth, and V. Grimm, "Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and 'R'," *Journal of Artificial Societies and Social Simulation*, vol. 17, no. 3, 2014.

[18] B. Iooss, A. Janon, G. Pujol, with contributions from Baptiste Broto, K. Boumhaout, S. D. Veiga, T. Delage, R. E. Amri, J. Fruth, L. Gilquin, J. Guillaume, L. Le Gratiet, P. Lemaitre, A. Marrel, A. Meynaoui, B. L. Nelson, F. Monari, R. Oomen, O. Rakovec, B. Ramos, O. Roustant, E. Song, J. Staum, R. Sueur, T. Touati, and F. Weber, *sensitivity: Global Sensitivity Analysis of Model Outputs*, 2020, R package version 1.18.0. [Online]. Available: https://CRAN.R-project.org/package=sensitivity. Accessed: 06|24|20.

[19] U. Grömping, "R Package FrF2 for Creating and Analyzing Fractional Factorial 2-Level Designs," *Journal of Statistical Software*, vol. 56, no. 1, pp. 1–56, 2014, [Online]. Available: http://www.jstatsoft.org/v56/i01/. Accessed: 06|24|20.

[20] R. B. Gramacy, "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models," *Journal of Statistical Software*, vol. 19, no. 9, pp. 1–46, 2007, [Online]. Available: http://www.jstatsoft.org/v19/i09/. Accessed: 06|24|20.

[21] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. Springer, New York, 2002, [Online]. Available: http://www.stats.ox.ac.uk/pub/MASS4. Accessed: 06|24|20.

[22] B. Auguie, *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2017, R package version 2.3. [Online]. Available: https://CRAN.R-project.org/package=gridExtra. Accessed: 06|24|20.

[23] S. Meschiari, *latex2exp: Use LaTeX Expressions in Plots*, 2015, R package version 0.4.0. [Online]. Available: https://CRAN.R-project.org/package=latex2exp. Accessed: 06|24|20.

[24] D. Sarkar, *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008, [Online]. Available: http://lmdvr.r-forge.r-project.org. Accessed: 06|24|20.

[25] T. D. Hocking, *directlabels: Direct Labels for Multicolor Plots*, 2020, R package version 2020.1.31. [Online]. Available: https://CRAN.R-project.org/package=directlabels. Accessed: 06|24|20.

[26] T. Turányi, "Lecture 1-2 Local Sensitivity Analysis," 2016, [Online]. Available: http://garfield.chem.elte.hu/COST_Training_School_2016/overheads/Turanyi_1-2_Local_Sensitivity_Analysis.pdf. Accessed: 05|07|20.

[27] M. Morris, "Factorial Sampling Plans for Preliminary Computational Experiments," *Technometrics*, vol. 33, no. 2, pp. 161–174, 1991.

[28] R. Iman, M. Shortencarier, and J. Johnson, “FORTRAN 77 Program and User's Guide for the Calculation of Partial Correlation and Standardized Regression Coefficients,” *United States Government Publications*, 1985.

[29] I. Sobol, “Sensitivity Estimates for Nonlinear Mathematical Models,” *Mathematical Modelling and Computational Experiments*, vol. 1, no. 4, pp. 407–414, 1993.

[30] T. Homma and A. Saltelli, “Importance Measures in Global Sensitivity Analysis of Nonlinear Models,” *Reliability Engineering & System Safety*, vol. 52, no. 1, pp. 1–17, 1996.

[31] P.-H. Cournède, Y. Chen, Q. Wu, C. Baey, and B. Bayol, “Development and Evaluation of Plant Growth Models: Methodology and Implementation in the PYGMALION Platform,” *Mathematical Modelling of Natural Phenomena*, vol. 8, 2013.

[32] A. Saltelli, S. Tarantola, and K. Chan, “A Quantitative Model-Independent Method for Global Sensitivity Analysis of Model Output,” *Technometrics*, vol. 41, no. 1, pp. 39–56, 1999.

[33] O. Kniemeyer, “Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling,” dissertation, Brandenburgische Technische Universität Cottbus, 2008.

[34] H. Wickham, *pryr: Tools for Computing on the Language*, 2018, R package version 0.1.4. [Online]. Available: https://CRAN.R-project.org/package=pryr, Accessed: 07|12|20.

[35] B. Bettonvil and J. Kleijnen, “Searching for Important Factors in Simulation Models with Many Factors: Sequential Bifurcation,” *European Journal of Operational Research*, vol. 96, no. 1, pp. 180–194, 1997.

[36] E. Borgonovo, “A new Uncertainty Importance Measure,” *Reliability Engineering & System Safety*, vol. 92, no. 6, pp. 771–784, 2007.

[37] L. Gürtler, “Internship Report: Connecting R to GroIMP - Rchart — A Chart Plugin for GroIMP based on R/ggplot2,” *not published*, 2020.

Appendix A

Source Code

A.1 RConnection.java

```
1  /*
2   * Copyright (C) 2020 GroIMP Developer Team
3   *
4   * This program is free software; you can redistribute it and/or
5   * modify it under the terms of the GNU General Public License
6   * as published by the Free Software Foundation; either version 3
7   * of the License, or any later version.
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
17 02111-1307, USA.
18  */
19
20 package de.grogra.sensitivity;
21
22 import java.awt.BorderLayout;
23 import java.awt.Component;
24 import java.awt.Dialog;
25 import java.awt.GridLayout;
26 import java.awt.Toolkit;
27 import java.awt.datatransfer.StringSelection;
28 import java.awt.event.ActionEvent;
29 import java.awt.event.ActionListener;
30 import java.io.BufferedReader;
```

```
31 import java.io.BufferedWriter;
32 import java.io.File;
33 import java.io.FileWriter;
34 import java.io.IOException;
35 import java.io.InputStreamReader;
36 import java.io.OutputStreamWriter;
37 import java.nio.file.Files;
38 import java.util.ArrayList;
39 import javax.swing.JButton;
40 import javax.swing.JDialog;
41 import javax.swing.JFileChooser;
42 import javax.swing.JLabel;
43 import javax.swing.JOptionPane;
44 import javax.swing.JPanel;
45 import javax.swing.border.EmptyBorder;
46 import de.grogra.rgg.Library;
47
48 /**
49  * Class for process communication with R
50  *
51  * @author Lukas Guertler
52  * @version 1.0
53  */
54 public class RConnection {
55
56    // stream writer and reader
57    private BufferedWriter w;
58    private BufferedReader r;
59
60    // connection successfully established?
61    private boolean _success = false;
62
63    public boolean success() {
64       return _success;
65    }
66
67    /**
68     * constructor, init a new connection
69     */
70    public RConnection() {
71
72       String rpath = get_R_executable();
73       _success = (rpath != null && init_R_connection(rpath) && check_R_packages());
74    }
75
76    /**
77     * start a connection to R for given executable
78     *
79     * @param rpath path to R executable file
```

```
80      */
81     private boolean init_R_connection(String rpath) {
82        try {
83
84           // open R process
85           ProcessBuilder RPB = new ProcessBuilder();
86           RPB.command(rpath, "--no-save");
87           RPB.redirectErrorStream(true);
88           Process R = RPB.start();
89
90           // get reader and writer for stdin and stdout
91           BufferedWriter w = new BufferedWriter(new OutputStreamWriter(R.getOutputStream
                 ()));
92           BufferedReader r = new BufferedReader(new InputStreamReader(R.getInputStream())
                 );
93
94           // set writer and reader
95           this.w = w;
96           this.r = r;
97
98           // success
99           return true;
100
101       } catch (Exception e) {
102
103          // print error message
104          String errorMsg = "init_R_connection: " + e;
105          System.out.println(errorMsg);
106          Library.println(errorMsg);
107          return false;
108       }
109    }
110
111    /**
112     * evaluate an expression
113     *
114     * @param expression expression
115     * @return String[] response
116     */
117    public String[] eval(String expression) {
118       try {
119          System.out.println(expression);
120
121          // special token to detect end of output
122          String token = "AAAA";
123
124          // send expression
125          w.write(String.format("%s\n", expression));
126          w.write(String.format("'%s'\n", token));
```

```
127            w.flush();
128
129            // skip own expression, recall: write to out-->in
130            long lines = 1 + expression.chars().filter(ch -> ch == '\n').count();
131            for (int i = 0; i < lines; i++)
132               r.readLine();
133
134            // read output until end token is reached
135            String tmp;
136            String res = "";
137            while (!(tmp = r.readLine()).endsWith(String.format("'%s'", token))) {
138               res += "\n" + tmp;
139            }
140            r.readLine();
141
142            // return response
143            return res.substring(res.length() > 0 ? 1 : 0).split("\n");
144
145         } catch (Exception e) {
146
147            // print error message
148            String errorMsg = "eval: " + e;
149            System.out.println(errorMsg);
150            Library.println(errorMsg);
151            return null;
152         }
153      }
154
155      /**
156       * open plot window
157       */
158      public void X11() {
159         try {
160            if (isOSWindows()) {
161               w.write("windows()\n");
162               w.flush();
163            } else if (isOSLinux()) {
164               w.write("x11()\n");
165               w.flush();
166            } else if (isOSMac()) {
167               w.write("quartz()\n");
168               w.flush();
169            }
170         } catch (Exception e) {
171         }
172      }
173
174      /**
175       * wrapper for function that waits for closing of plot window
```

```
176      */
177     public void waitForClose() {
178        try {
179           w.write("waitForClose()\n");
180           w.flush();
181        } catch (Exception e) {
182        }
183     }
184
185     /**
186      * check if OS is Windows
187      */
188     private boolean isOSWindows() {
189        return System.getProperty("os.name").toLowerCase().contains("windows");
190     }
191
192     /**
193      * check if OS is Linux
194      */
195     private boolean isOSLinux() {
196        return System.getProperty("os.name").toLowerCase().contains("nux");
197     }
198
199     /**
200      * check if OS is Mac
201      */
202     private boolean isOSMac() {
203        return System.getProperty("os.name").toLowerCase().contains("mac");
204     }
205
206     /**
207      * list all files (including subdirectories)
208      *
209      * @param directoryName directory
210      * @param files         ArrayList of files
211      */
212     private void listf(String directoryName, ArrayList<String> files) {
213        File directory = new File(directoryName);
214
215        // Get all files from a directory.
216        File[] fList = directory.listFiles();
217        if (fList != null) {
218           for (File file : fList) {
219              if (file.isFile()) {
220                 files.add(file.toString());
221              } else if (file.isDirectory()) {
222                 listf(file.getAbsolutePath(), files);
223              }
224           }
```

```
225         }
226     }
227
228     /**
229      * get the R executable path
230      *
231      * @return path, returns null in case of missing R installation
232      */
233     private String get_R_executable() {
234
235         if (isOSWindows()) {
236             try {
237                 File settingsDir = new File("plugins\\Sensitivity\\settings");
238
239                 if (!settingsDir.exists())
240                     settingsDir.mkdirs();
241
242                 File settingsFile = new File("plugins\\Sensitivity\\settings\\RPATH");
243
244                 if (settingsFile.exists()) {
245                     String rpath = new String(Files.readAllBytes(settingsFile.toPath()));
246
247                     if (new File(rpath).getAbsoluteFile().exists()) {
248                         return rpath;
249                     }
250                 }
251
252                 JFileChooser folderPicker = new JFileChooser();
253                 folderPicker.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
254                 folderPicker.setDialogTitle("Please select the R installation folder!");
255
256                 if (folderPicker.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
257
258                     String dir = folderPicker.getSelectedFile().toString();
259                     ArrayList<String> files = new ArrayList<String>();
260                     listf(dir, files);
261                     String toSearch = "r.exe";
262                     boolean found = false;
263                     String ScriptEngine = null;
264
265                     for (String file : files) {
266                         if (file.toLowerCase().endsWith(toSearch)) {
267                             ScriptEngine = file;
268                             BufferedWriter writer = new BufferedWriter(new FileWriter(
                                    settingsFile));
269                             writer.write(file);
270                             writer.close();
271                             found = true;
272                             break;
```

```
273                 }
274             }
275
276             if (!found) {
277                JOptionPane.showMessageDialog(null, "R installation corrupted!", "
                       Error",
278                      JOptionPane.INFORMATION_MESSAGE);
279             } else {
280                return ScriptEngine;
281             }
282          }
283
284       } catch (Exception e) {
285          System.out.println(e);
286       }
287
288    } else if (isOSLinux()) {
289       try {
290          new ProcessBuilder("R").start();
291          return "R";
292       } catch (IOException e) {
293
294          JDialog diag = new JDialog();
295          diag.setTitle("R installation required");
296          diag.setLayout(new BorderLayout());
297          EmptyBorder eb = new EmptyBorder(5, 10, 5, 10);
298
299          JLabel l1 = new JLabel("R could not be found! Please perform the following
                 steps:");
300          JLabel l2 = new JLabel("     1. Open terminal");
301          JLabel l3 = new JLabel("     2. Type in 'sudo apt-get install r-base' and
                 press Enter");
302          JLabel l4 = new JLabel("     3. Type in 'sudo apt-get install r-recommended'
                  and press Enter");
303          JLabel l5 = new JLabel("     4. Type in: 'sudo R' and press Enter");
304          JLabel l6 = new JLabel("     5. Type in: 'install.packages(c('sensitivity','
                 FrF2','tgp','MASS',");
305          JLabel l7 = new JLabel("          'gridExtra','latex2exp','lattice','
                 directlabels'))' and press Enter");
306          JLabel l8 = new JLabel("     6. Close terminal");
307
308          l1.setBorder(eb);
309          l2.setBorder(eb);
310          l3.setBorder(eb);
311          l4.setBorder(eb);
312          l5.setBorder(eb);
313          l6.setBorder(eb);
314          l7.setBorder(eb);
315          l8.setBorder(eb);
```

```
316
317            JButton b1 = new JButton("copy to clipboard");
318            b1.setName("sudo apt-get install r-base");
319            JButton b2 = new JButton("copy to clipboard");
320            b2.setName("sudo apt-get install r-recommended");
321            JButton b3 = new JButton("copy to clipboard");
322            b3.setName("sudo R");
323            JButton b4 = new JButton("copy to clipboard");
324            b4.setName(
325                  "install.packages(c('sensitivity','FrF2','tgp','MASS','gridExtra','
                      latex2exp','lattice','directlabels'))");
326
327            ActionListener al = new ActionListener() {
328               public void actionPerformed(ActionEvent e) {
329                  Component component = (Component) e.getSource();
330                  StringSelection stringSelection = new StringSelection(component.
                      getName());
331                  Toolkit.getDefaultToolkit().getSystemClipboard().setContents(
                      stringSelection, null);
332               }
333            };
334
335            b1.addActionListener(al);
336            b2.addActionListener(al);
337            b3.addActionListener(al);
338            b4.addActionListener(al);
339
340            JPanel jp1 = new JPanel(new GridLayout(8, 1));
341            JPanel jp2 = new JPanel(new GridLayout(8, 1));
342
343            jp1.add(l1);
344            jp1.add(l2);
345            jp1.add(l3);
346            jp1.add(l4);
347            jp1.add(l5);
348            jp1.add(l6);
349            jp1.add(l7);
350            jp1.add(l8);
351            jp2.add(new JLabel(""));
352            jp2.add(new JLabel(""));
353            jp2.add(b1);
354            jp2.add(b2);
355            jp2.add(b3);
356            jp2.add(b4);
357            diag.add(jp1, BorderLayout.WEST);
358            diag.add(jp2, BorderLayout.EAST);
359            diag.pack();
360            diag.setModalityType(Dialog.ModalityType.APPLICATION_MODAL);
361            diag.setLocationRelativeTo(null);
```

```
362             diag.setVisible(true);
363          }
364
365       } else if (isOSMac()) {
366          String rpath = "/usr/bin/R";
367          try {
368             new ProcessBuilder(rpath).start();
369             return rpath;
370          } catch (IOException e) {
371             JOptionPane.showMessageDialog(null, "R installation required!", "Error",
372                   JOptionPane.INFORMATION_MESSAGE);
373          }
374       } else {
375          JOptionPane.showMessageDialog(null, "OS not supported!", "", JOptionPane.
              INFORMATION_MESSAGE);
376       }
377       return null;
378    }
379
380    /**
381     * check R installation for necessary packages
382     *
383     * @return true if all packages are installed
384     */
385    private boolean check_R_packages() {
386       try {
387
388          // load necessary R packages
389          eval("Sys.setenv(LANGUAGE = 'en')");
390          eval("packages <- c('sensitivity','FrF2','tgp','MASS','gridExtra','latex2exp','
              lattice','directlabels')");
391          String res = eval("print(setdiff(packages, rownames(installed.packages())))")
              [0];
392
393          if (res.contains("character(0)")) {
394
395             // load necessary R packages
396             eval("library(sensitivity)");
397             eval("library(FrF2)");
398             eval("library(tgp)");
399             eval("library(MASS)");
400             eval("library(gridExtra)");
401             eval("library(latex2exp)");
402             eval("library(lattice)");
403             eval("library(directlabels)");
404
405             // definition of function needed for later matrix parsing
406             eval("flat <- function(var) {\n" + "tmp = unname(var)\n" + "for (i in 1:dim(
                 tmp)[1]) {\n"
```

```
407                    + "for (j in 1:dim(tmp)[2]) {\n" + "print(tmp[i,j])\n" + "}\n" + "}\n"
                           + "}");
408
409              // definition of function that waits for closing of plot window
410              eval("waitForClose <- function() {\n" + "while (length(dev.list())!=0) {\n"
                     + "Sys.sleep(1)\n" + "}\n"
411                    + "q()\n" + "}");
412
413              return true;
414
415           } else {
416              String[] missingPackages = res.split(" ");
417
418              String outMsg = "";
419
420              for (String p : missingPackages) {
421
422                 if (p.startsWith("\""))
423                    outMsg += String.format("\n    > %s", p);
424              }
425
426              JOptionPane.showMessageDialog(null, "Please install the following R packages
                     :" + outMsg, "",
427                    JOptionPane.INFORMATION_MESSAGE);
428              return false;
429           }
430
431        } catch (Exception e) {
432           System.out.println(e);
433           return false;
434        }
435     }
436  }
```

A.2 Simulation.java

```
1  /*
2    * Copyright (C) 2020 GroIMP Developer Team
3    *
4    * This program is free software; you can redistribute it and/or
5    * modify it under the terms of the GNU General Public License
6    * as published by the Free Software Foundation; either version 3
7    * of the License, or any later version.
8    *
9    * This program is distributed in the hope that it will be useful,
10   * but WITHOUT ANY WARRANTY; without even the implied warranty of
11   * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12   * GNU General Public License for more details.
13   *
14   * You should have received a copy of the GNU General Public License
15   * along with this program; if not, write to the Free Software
16   * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
17 02111-1307, USA.
18   */
19
20 package de.grogra.sensitivity;
21
22 /**
23  * Interface to implement a simulation
24  *
25  * @author Lukas Guertler
26  * @version 1.0
27  */
28 public interface Simulation {
29    public void run();
30
31    public double getOutput();
32
33    public String outputName();
34 }
```

A.3 NumberRef.java

```
1  /*
2   * Copyright (C) 2020 GroIMP Developer Team
3   *
4   * This program is free software; you can redistribute it and/or
5   * modify it under the terms of the GNU General Public License
6   * as published by the Free Software Foundation; either version 3
7   * of the License, or any later version.
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
17 02111-1307, USA.
18  */
19
20 package de.grogra.sensitivity;
21
22 import java.io.Serializable;
23
24 /**
25  * Class for boxing and unboxing of number types
26  *
27  * @author Lukas Guertler
28  * @version 1.0
29  */
30 public class NumberRef implements Serializable {
31    private double value;
32    private String name;
33
34    public NumberRef(int value, String name) {
35       this((double) value, name);
36    }
37
38    public NumberRef(long value, String name) {
39       this((double) value, name);
40    }
41
42    public NumberRef(float value, String name) {
43       this((double) value, name);
44    }
45
46    public NumberRef(double value, String name) {
```

```
47        this.value = value;
48        this.name = name;
49     }
50
51     public int getInt() {
52        return (int) value;
53     }
54
55     public long getLong() {
56        return (long) value;
57     }
58
59     public float getFlt() {
60        return (float) value;
61     }
62
63     public double getDbl() {
64        return (double) value;
65     }
66
67     public void set(int value) {
68        this.value = (double) value;
69     }
70
71     public void set(long value) {
72        this.value = (double) value;
73     }
74
75     public void set(float value) {
76        this.value = (double) value;
77     }
78
79     public void set(double value) {
80        this.value = value;
81     }
82
83     public String name() {
84        return name;
85     }
86
87     public void setName(String name) {
88        this.name = name;
89     }
90  }
```

A.4 Sensitivity.java

```
1  /*
2   * Copyright (C) 2020 GroIMP Developer Team
3   *
4   * This program is free software; you can redistribute it and/or
5   * modify it under the terms of the GNU General Public License
6   * as published by the Free Software Foundation; either version 3
7   * of the License, or any later version.
8   *
9   * This program is distributed in the hope that it will be useful,
10  * but WITHOUT ANY WARRANTY; without even the implied warranty of
11  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12  * GNU General Public License for more details.
13  *
14  * You should have received a copy of the GNU General Public License
15  * along with this program; if not, write to the Free Software
16  * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
17 02111-1307, USA.
18  */
19
20 package de.grogra.sensitivity;
21
22 import de.grogra.rgg.Library;
23
24 /**
25  * Sensitivity analysis plugin for GroIMP using R
26  *
27  * @author Lukas Guertler
28  * @version 1.0
29  */
30 public abstract class Sensitivity {
31
32    /**
33     * perform local sensitivity analysis
34     *
35     * @param parameters array of model parameters
36     * @param min        relative minimum test value for examination range, e.g. 0.9
37     * @param max        relative maximum test value for examination range, e.g. 1.1
38     * @param simulation simulation to run
39     */
40    public static void local_sensitivity_analysis(NumberRef[] parameters, double min,
           double max,
41          Simulation simulation) {
42
43       // open new R connection
44       RConnection rc = new RConnection();
45       if (!rc.success())
```

```
46            return;
47
48         // create and fill array for output measurements for tested parameter
49         // combinations
50         simulation.run();
51         double baseValue = simulation.getOutput();
52         double[] levels = new double[] { min, max };
53         double measurements[][] = new double[parameters.length][levels.length];
54
55         for (int i = 0; i < parameters.length; i++) {
56
57            double orgVal = parameters[i].getDbl();
58
59            // run simulation with OAT approach
60            for (int j = 0; j < levels.length; j++) {
61               parameters[i].set(levels[j] * orgVal);
62               simulation.run();
63               measurements[i][j] = simulation.getOutput();
64               parameters[i].set(orgVal);
65            }
66         }
67
68         // calculate individual output deviation
69         int z = 1;
70         rc.eval("df <- data.frame(c(0))");
71         for (int i = 0; i < parameters.length; i++) {
72            double minOutput = measurements[i][0];
73            double maxOutput = measurements[i][1];
74
75            double sens_min = (minOutput - baseValue) / baseValue * 100.0;
76            double sens_max = (maxOutput - baseValue) / baseValue * 100.0;
77
78            rc.eval(String.format("df[%s,1] <- %s", z, sens_min));
79            rc.eval(String.format("rownames(df)[%s] <- '%s_min'", z, parameters[i].name()))
                 ;
80            z++;
81            rc.eval(String.format("df[%s,1] <- %s", z, sens_max));
82            rc.eval(String.format("rownames(df)[%s] <- '%s_max'", z, parameters[i].name()))
                 ;
83            z++;
84         }
85
86         // plot result
87         rc.eval(String.format("colnames(df)[1] <- '%s'", validVarName(simulation.
               outputName())));
88         rc.X11();
89         rc.eval("grid.table(df)");
90         rc.waitForClose();
91
```

```
92    }
93
94    /**
95     * perform Morris's elementary effects screening
96     *
97     * @param parameters array of model parameters
98     * @param min_var    minimum test value for each parameter
99     * @param max_var    maximum test value for each parameter
100    * @param simulation simulation to run
101    */
102   public static void morris_elementary_effects_screening(NumberRef[] parameters, double
          [] min_var, double[] max_var,
103         Simulation simulation) {
104      try {
105
106         // open new R connection
107         RConnection rc = new RConnection();
108         if (!rc.success())
109            return;
110
111         // create morris object
112         rc.eval(String.format(
113               "mo <- morris(model = NULL, factors = %s, r = 10," + "design = list(type
                      = 'oat', levels = 10),"
114                  + "binf = c(%s)," + "bsup = c(%s), scale=TRUE)",
115               parameters.length, joinStr(min_var, ","), joinStr(max_var, ",")));
116
117         // read the parameter combinations from the morris object
118         double[][] param_combs = parseMatrix(rc.eval("flat(mo$X)"), parameters.length);
119
120         // run simulations for parameter combinations and store output
121         rc.eval("res = 0");
122         for (int i = 0; i < param_combs.length; i++) {
123            for (int j = 0; j < parameters.length; j++)
124               parameters[j].set(param_combs[i][j]);
125
126            simulation.run();
127            rc.eval(String.format("res[%s] <- %s", i + 1, simulation.getOutput()));
128         }
129
130         // hand over the simulation results to the morris object
131         rc.eval("tell(mo, res)");
132
133         // calculate desired sensitivity measures
134         rc.eval("mu <- as.matrix(apply(mo$ee, 2, mean))");
135         rc.eval("mu.star <- as.matrix(apply(mo$ee, 2, function(x) mean(abs(x))))");
136         rc.eval("sigma <- as.matrix(apply(mo$ee, 2, function(x) sd(x, na.rm=T)))");
137
138         // plot results
```

```
139         rc.eval("labels <- '0'");
140         for (int i = 0; i < parameters.length; i++)
141            rc.eval(String.format("labels[%s] <- '%s'", i + 1, validVarName(parameters[i
                   ].name())));
142         rc.X11();
143         rc.eval(String.format(
144               "direct.label(xyplot(sigma~mu.star,data=data.frame(mu.star,sigma),group=
                      labels,col='black',xlab = TeX('$\\\\mu^{*}$'),ylab = TeX('$\\\\sigma$
                      '),main='Morris\\'s elementary effects screening for %s'))",
145               validVarName(simulation.outputName())));
146         rc.waitForClose();
147
148      } catch (Exception e) {
149
150         // print error message
151         String errorMsg = "morris_elementary_effects_screening: " + e;
152         System.out.println(errorMsg);
153         Library.println(errorMsg);
154      }
155   }
156
157   /**
158    * calculate main and interaction effects on extreme values
159    *
160    * @param parameters array of model parameters
161    * @param min_var    minimum test value for each parameter
162    * @param max_var    maximum test value for each parameter
163    * @param simulation simulation to run
164    */
165   public static void main_and_interaction_effects_on_extreme_values(NumberRef[]
          parameters, double[] min_var,
166         double[] max_var, Simulation simulation) {
167      try {
168
169         // open new R connection
170         RConnection rc = new RConnection();
171         if (!rc.success())
172            return;
173
174         // make label with parameter names
175         String labels = "";
176         for (int i = 0; i < parameters.length; i++)
177            labels += String.format(",'%s'", parameters[i].name());
178
179         // create full factorial design
180         rc.eval(String.format("tmp <- FrF2(nruns=2^%s, nfactors=%s, randomize=FALSE," +
                 "default.levels = c(0,1))",
181               parameters.length, parameters.length));
182
```

```
183            rc.eval(String.format(
184                  "ff <- FrF2(nruns=2^%s, nfactors=%s, randomize=FALSE,"
185                        + "factor.names=c(%s),default.levels = c('min','max'))",
186                  parameters.length, parameters.length, labels.substring(1)));
187
188            String[] vars_str = rc.eval("flat(as.matrix(tmp))");
189
190            // number of samples
191            int samples = vars_str.length / parameters.length;
192
193            // parse parameter combinations
194            double[][] param_combs = new double[samples][parameters.length];
195            int k = 0;
196            for (int i = 0; i < samples; i++) {
197               for (int j = 0; j < parameters.length; j++) {
198                  double val = Double.parseDouble(vars_str[k++].substring(4).replace("\"",
                         ""));
199                  param_combs[i][j] = val == 0.0 ? min_var[j] : max_var[j];
200               }
201            }
202
203            // run simulations for parameter combinations and store output
204            rc.eval("res = 0");
205            for (int i = 0; i < samples; i++) {
206               for (int j = 0; j < parameters.length; j++)
207                  parameters[j].set(param_combs[i][j]);
208
209               simulation.run();
210               rc.eval(String.format("res[%s] <- %s", i + 1, simulation.getOutput()));
211            }
212
213            // plot results
214            String outName = validVarName(simulation.outputName());
215            rc.eval(String.format("%s <- res", outName));
216            rc.eval(String.format("ffr <- add.response(ff, response=%s)", outName));
217            rc.X11();
218            rc.eval("MEPlot(ffr)");
219            rc.X11();
220            rc.eval("IAPlot(ffr)");
221            rc.waitForClose();
222
223         } catch (Exception e) {
224
225            // print error message
226            String errorMsg = "main_and_interaction_effects_on_extreme_values: " + e;
227            System.out.println(errorMsg);
228            Library.println(errorMsg);
229         }
230      }
```

```
231
232     /**
233      * calculate partial (including rank) correlation coefficients
234      *
235      * @param parameters array of model parameters
236      * @param min_var    minimum test value for each parameter
237      * @param max_var    maximum test value for each parameter
238      * @param simulation simulation to run
239      */
240     public static void partial_correlation_coefficients(NumberRef[] parameters, double[]
            min_var, double[] max_var,
241           Simulation simulation) {
242        try {
243
244           // open new R connection
245           RConnection rc = new RConnection();
246           if (!rc.success())
247              return;
248
249           // create latin hypercube sample
250           rc.eval(String.format("param.sets <- lhs(n=%s00, rect=matrix(c(%s,%s), %s))",
                parameters.length,
251                 joinStr(min_var, ","), joinStr(max_var, ","), parameters.length));
252
253           // parse parameter combinations
254           double[][] param_combs = parseMatrix(rc.eval("flat(param.sets)"), parameters.
                length);
255
256           // run simulations for parameter combinations and store output
257           rc.eval("res = 0");
258           for (int i = 0; i < param_combs.length; i++) {
259              for (int j = 0; j < parameters.length; j++)
260                 parameters[j].set(param_combs[i][j]);
261
262              simulation.run();
263              rc.eval(String.format("res[%s] <- %s", i + 1, simulation.getOutput()));
264           }
265
266           // label columns
267           rc.eval("param.sets <- as.data.frame(param.sets)");
268           for (int i = 0; i < parameters.length; i++)
269              rc.eval(String.format("colnames(param.sets)[%s] <- '%s'", i + 1, parameters[
                   i].name()));
270
271           // calculate pcc and prcc
272           rc.eval(String.format("pcc.result <- pcc(X=param.sets, y=res, nboot = %s00,
                rank = FALSE)",
273                 parameters.length));
274           rc.eval(String.format("prcc.result <- pcc(X=param.sets, y=res, nboot = %s00,
```

```
              rank = TRUE)",
275               parameters.length));
276
277         // plot results
278         rc.X11();
279         rc.eval("plot(pcc.result)");
280         rc.X11();
281         rc.eval("plot(prcc.result)");
282         rc.eval("par(col.main='white')");
283         for (int i = 0; i < 10; i++)
284            rc.eval("title('PRRC')");
285         rc.eval("par(col.main='black')");
286         rc.eval("title('PRCC')");
287         rc.waitForClose();
288
289      } catch (Exception e) {
290
291         // print error message
292         String errorMsg = "partial_correlation_coefficients: " + e;
293         System.out.println(errorMsg);
294         Library.println(errorMsg);
295      }
296   }
297
298   /**
299    * calculate standardized (including rank) regression coefficients
300    *
301    * @param parameters array of model parameters
302    * @param min_var    minimum test value for each parameter
303    * @param max_var    maximum test value for each parameter
304    * @param simulation simulation to run
305    */
306   public static void standardized_regression_coefficients(NumberRef[] parameters,
          double[] min_var, double[] max_var,
307         Simulation simulation) {
308      try {
309
310         // open new R connection
311         RConnection rc = new RConnection();
312         if (!rc.success())
313            return;
314
315         // create latin hypercube sample
316         rc.eval(String.format("param.sets <- lhs(n=%s00, rect=matrix(c(%s,%s), %s))",
              parameters.length,
317               joinStr(min_var, ","), joinStr(max_var, ","), parameters.length));
318
319         // parse parameter combinations
320         double[][] param_combs = parseMatrix(rc.eval("flat(param.sets)"), parameters.
```

```
               length);
321
322            // run simulations for parameter combinations and store output
323            rc.eval("res = 0");
324            for (int i = 0; i < param_combs.length; i++) {
325               for (int j = 0; j < parameters.length; j++)
326                  parameters[j].set(param_combs[i][j]);
327
328               simulation.run();
329               rc.eval(String.format("res[%s] <- %s", i + 1, simulation.getOutput()));
330            }
331
332            // label columns
333            rc.eval("param.sets <- as.data.frame(param.sets)");
334            for (int i = 0; i < parameters.length; i++)
335               rc.eval(String.format("colnames(param.sets)[%s] <- '%s'", i + 1, parameters[
                    i].name()));
336
337            // calculate src and arrc
338            rc.eval(String.format("src.result <- src(X=param.sets, y=res, nboot = %s00,
                rank = FALSE)",
339                  parameters.length));
340            rc.eval(String.format("srrc.result <- src(X=param.sets, y=res, nboot = %s00,
                rank = TRUE)",
341                  parameters.length));
342
343            // plot results
344            rc.X11();
345            rc.eval("plot(src.result)");
346            rc.X11();
347            rc.eval("plot(srrc.result)");
348            rc.waitForClose();
349
350         } catch (Exception e) {
351
352            // print error message
353            String errorMsg = "standardized_regression_coefficients: " + e;
354            System.out.println(errorMsg);
355            Library.println(errorMsg);
356         }
357      }
358
359      /**
360       * perform Sobol's method
361       *
362       * @param parameters array of model parameters
363       * @param min_var    minimum test value for each parameter
364       * @param max_var    maximum test value for each parameter
365       * @param simulation simulation to run
```

```
366      */
367     public static void sobol(NumberRef[] parameters, double[] min_var, double[] max_var,
            Simulation simulation) { //

368        try {
369
370           // open new R connection
371           RConnection rc = new RConnection();
372           if (!rc.success())
373              return;
374
375           // create two latin hypercube sample
376           rc.eval(String.format("input.set.1 <- as.data.frame(lhs(n=%s00, rect=matrix(c(%
                  s,%s), %s)))",
377                 parameters.length, joinStr(min_var, ","), joinStr(max_var, ","),
                        parameters.length)); //

378           rc.eval(String.format("input.set.2 <- as.data.frame(lhs(n=%s00, rect=matrix(c(%
                  s,%s), %s)))",
379                 parameters.length, joinStr(min_var, ","), joinStr(max_var, ","),
                        parameters.length)); //

380
381           // label columns
382           for (int i = 0; i < parameters.length; i++) {
383              rc.eval(String.format("colnames(input.set.1)[%s] <- '%s'", i + 1, parameters
                     [i].name()));
384              rc.eval(String.format("colnames(input.set.2)[%s] <- '%s'", i + 1, parameters
                     [i].name()));
385           }
386
387           // create sobol object
388           rc.eval(String.format(
389                 "so <- sobol(model = NULL, X1 = input.set.1, X2 = input.set.2, order = 2,
                         nboot = %s00)",
390                 parameters.length));
391
392           // parse parameter combinations
393           double[][] param_combs = parseMatrix(rc.eval("flat(as.matrix(so$X))"),
                  parameters.length); //

394
395           // run simulations for parameter combinations and store output
396           rc.eval("res = 0");
397           for (int i = 0; i < param_combs.length; i++) {
398              for (int j = 0; j < parameters.length; j++)
399                 parameters[j].set(param_combs[i][j]);
400
401              simulation.run();
```

```
402            rc.eval(String.format("res[%s] <- %s", i + 1, simulation.getOutput()));
403         }
404
405         // hand over the simulation results to the sobol object
406         rc.eval("tell(so, res)");
407
408         // plot results
409         rc.X11();
410         rc.eval("plot(so)");
411         rc.waitForClose();
412
413      } catch (Exception e) {
414
415         // print error message
416         String errorMsg = "sobol: " + e;
417         System.out.println(errorMsg);
418         Library.println(errorMsg);
419      }
420   }
421
422   /**
423    * perform extended fourier amplitude sensitivity test
424    *
425    * @param parameters array of model parameters
426    * @param min_var    minimum test value for each parameter
427    * @param max_var    maximum test value for each parameter
428    * @param simulation simulation to run
429    */
430   public static void extended_fourier_amplitude_sensitivity_test(NumberRef[] parameters
          , double[] min_var,
431         double[] max_var, Simulation simulation) {
432      try {
433
434         // open new R connection
435         RConnection rc = new RConnection();
436         if (!rc.success())
437            return;
438
439         // argument for EFAST
440         String tmp = "";
441         for (int i = 0; i < parameters.length; i++)
442            tmp += String.format(",list(min=%s,max=%s)", min_var[i], max_var[i]);
443
444         // create EFAST object
445         rc.eval(String.format("f99 <- fast99(model = NULL, factors = %s, n=%s00, q = c
              (%s), q.arg = list(%s))",
446              parameters.length, parameters.length, repStr("'qunif'", parameters.length
                 , ","), tmp.substring(1)));
447
```

```
448            // parse parameter combinations
449            double[][] param_combs = parseMatrix(rc.eval("flat(as.matrix(f99$X))"),
                   parameters.length);
450
451            // run simulations for parameter combinations and store output
452            rc.eval("res = 0");
453            for (int i = 0; i < param_combs.length; i++) {
454               for (int j = 0; j < parameters.length; j++)
455                  parameters[j].set(param_combs[i][j]);
456
457               simulation.run();
458               rc.eval(String.format("res[%s] <- %s", i + 1, simulation.getOutput()));
459            }
460
461            // label columns
462            for (int i = 0; i < parameters.length; i++)
463               rc.eval(String.format("colnames(f99$X)[%s] <- '%s'", i + 1, parameters[i].
                      name()));
464
465            // hand over the simulation results to the EFAST object
466            rc.eval("tell(f99, res)");
467
468            // plot results
469            rc.X11();
470            rc.eval("plot(f99)");
471            rc.waitForClose();
472
473         } catch (Exception e) {
474
475            // print error message
476            String errorMsg = "extended_fourier_amplitude_sensitivity_test: " + e;
477            System.out.println(errorMsg);
478            Library.println(errorMsg);
479         }
480      }
481
482      /**
483       * convert double array to string
484       *
485       * @param arr       double array
486       * @param delimiter delimiter
487       * @return String result
488       */
489      private static String joinStr(double[] arr, String delimiter) {
490         StringBuilder sb = new StringBuilder();
491         for (double dbl : arr) {
492            sb.append(delimiter);
493            sb.append(dbl);
494         }
```

```
495         return sb.substring(1);
496     }
497
498     /**
499      * get repeated string version
500      *
501      * @param str       string to repeat
502      * @param count     number of repetitions
503      * @param delimiter delimiter
504      * @return String result
505      */
506     private static String repStr(String str, int count, String delimiter) {
507         StringBuilder sb = new StringBuilder();
508         for (int i = 0; i < count; i++) {
509             sb.append(delimiter);
510             sb.append(str);
511         }
512         return sb.substring(1);
513     }
514
515     /**
516      * parse double matrix from string array
517      *
518      * @param strarr   string array
519      * @param colCount number of matrix columns
520      * @return double result
521      */
522     private static double[][] parseMatrix(String[] strarr, int colCount) {
523
524         // number of rows
525         int rowCount = strarr.length / colCount;
526
527         // input variables
528         double[][] res = new double[rowCount][colCount];
529
530         // parse each entry
531         int k = 0;
532         for (int i = 0; i < rowCount; i++)
533             for (int j = 0; j < colCount; j++)
534                 res[i][j] = Double.parseDouble(strarr[k++].substring(4));
535
536         // return result
537         return res;
538     }
539
540     private static String validVarName(String name) {
541         String res = "";
542         String tmp = name;
543         if (name.length() > 0) {
```

```
544            if (!Character.isLetter(name.charAt(0)))
545               tmp = "V" + name;
546         } else {
547            return "var";
548         }
549         for (int i = 0; i < tmp.length(); i++) {
550            char c = tmp.charAt(i);
551            if (Character.isLetter(c)  Character.isDigit(c)) {
552               res += c;
553            } else {
554               res += "_";
555            }
556         }
557         return res;
558      }
559   }
```