FH Vorarlberg
University of Applied Sciences

Title

Subtitle

Master Thesis
Submitted in Fulfillment of the Degree

Master of Science in Engineering, MSc

Vorarlberg University of Applied Sciences
Master's in Mechatronics

Support Vorarlberg University of Applied Sciences

Dr.-Ing. Finck Steffen

Handed in by

Schwartze Nicolai, BSc

matrikelnummer

Dornbirn, 29.05.2020

Kurzreferat

Deutscher TITEL

Ein Kurzreferat macht die Relevanz der Arbeit sowie die innovativen Gedankengänge ersichtlich. Alleiniges Ziel ist es, in jeweils einem Absatz einen gerafften Überblick der Arbeit zu geben, so dass die Nutzer/innen entscheiden können, ob die vorliegende Arbeit für das eigene Forschungsvorhaben relevant ist oder nicht. Dementsprechend müssen darin die zentralen Abschnitte in neutraler, nicht wertender Perspektive beschrieben werden, vergleichbar einem Text über den Text von einem imaginierten Dritten.

Das Abstract muss für sich alleine verständlich sein. Es sollte zudem die zentralen Schlagwörter, die das Thema der Arbeit treffend umreißen, enthalten, um eine Indexierung in einer bibliographischen Referenzdatei zu erleichtern. Der Umfang von 1200 Anschlägen (d.h. Zeichen mit Leerzeichen; ca. 20 Zeilen) sollte nicht überschritten werden.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Abstrait

Français TITLE

The abstract must be relevant to the topic and show the innovative ideas of the work. It should summarize your research in order to inform the reader, and give him/her the choice of whether it should/not be selected for his/her work. It should be a factual and impartial presentation of the main ideas.

The abstract should be only one paragraph long. It must provide the required information so that it is not necessary to read the other parts or even the complete work. It must also include key words to outline the subject for indexing and for bibliographic reference.

The length should not be more than 1200 characters (i.e. symbols and spaces; ca. 20 lines).

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Abstract

English TITLE

The abstract must be relevant to the topic and show the innovative ideas of the work. It should summarize your research in order to inform the reader, and give him/her the choice of whether it should/not be selected for his/her work. It should be a factual and impartial presentation of the main ideas.

The abstract should be only one paragraph long. It must provide the required information so that it is not necessary to read the other parts or even the complete work. It must also include key words to outline the subject for indexing and for bibliographic reference.

The length should not be more than 1200 characters (i.e. symbols and spaces; ca. 20 lines).

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Differential equations are a very powerful mathematical tool to describe the universe. From fluid dynamic and heat flow in mechanical engineering or electrodynamics and circuit-design in electrical engineering to the fluctuation of stock market prices and even the movement of astronomical objects - most dynamic processes can be formulated by them. But only a tiny fraction of them are actually analytically solvable. The rise of the computer paved the way for approximating the solution of a differential equation numerically. There are plenty of different solver methods ranging from simple single step solvers like the Forward-Euler (FE) method over more complex multistep solvers like the Adams–Bashforth (AB) method to whole FEM solver packages. Most of these solvers are specialised for one kind of differential equation, leveraging some special properties of the problem to be most efficient in time and error. But there are very few generalised methods that can solve many different types of equations. This lack of a universal solver is the main motivation of the present master thesis.

There is already a small, yet steadily growing research community interested in tying together the concepts of computational intelligence and numerical solver for differential equation. In chapter 2.2 a brief overview of related work done within the last 20 years is given. The main idea of all listed papers is to reformulate the original problem into an optimisation problem, which in turn is then solved using an evolutionary optimisation algorithm. The main focus of this thesis lies two-dimensional PDE. The results are then compared to the analytical solution as well as a state of the art FEM solver.

This master thesis tries to answer the following questions: Is it possible to improve the results obtained in other research papers by using a modern variant of the differential evolution (DE) optimisation algorithm? How well does this method stack up against classical FEM package for solving PDEs considering time and memory usage as well as numerical error? Is there a meaningful way to reduce the time consumption by making use of a parallel computation architecture?

2 State of the Art

This chapter provides an overview of the current state of the art in solving PDE. Included are the widely used FEM as well as heuristic optimisation methods. Further, an introduction to the DE framework is given, which provides the basis for the algorithms described in this thesis.

2.1 Finite Element Method

Currently the finite element method is the go-to approach to solve partial differential equations. The domain Ω on which the PDE is posed, is discretised into multiple smaller elements - as the name suggests. Thus, FEM counts to the category of meshed methods. The underlying solution function $u(\mathbf{x})$ to the PDE is then approximated by so called "basis-functions" $\Phi(\mathbf{x})$ limited to these finite elements. This thesis uses the open-source Netgen/NGSolve FEM package (Schöberl, Lackner, and Hochsteger 2020).

The general steps taken to solve a PDE with an FEM solver are:

- Setup
  At first a Hilbertspace V is needed. The principal idea is to find a bilinear form $a : VxV \rightarrow \mathbf{R}$ and a linear functional $F : V \rightarrow \mathbf{R}$ so that for

  $$u \in V \; a(u, v) = F(v) \; \forall v \in V \tag{2.1}$$

  This can be achieved by reformulating the strong form into the weak form as seen in equation 2.3. There, the left-hand side represents the $a(u, v)$ and the right hand side (RHS) is $F(v)$. In this case u is often called the trial-function and v is named the test-function.

- Strong Form
  This is the standard formulation of the PDE with a Dirichlet boundary condition:

  $$\begin{aligned} u, f, g : \Omega &\rightarrow \mathbf{R} \\ \mathbf{L}u(\mathbf{x}) &= f(\mathbf{x}) \text{ on } \Omega \\ u(\mathbf{x})|_{\partial\Omega} &= g(\mathbf{x}) \end{aligned} \tag{2.2}$$

- Weak Form
  This is equivalent to the strong form but written in an integral formulation.

$$\begin{aligned}\int_\Omega -(\nabla^T A \nabla) u(\mathbf{x}) v(\mathbf{x}) dV - \int_\Omega b^T \nabla u(\mathbf{x}) v(\mathbf{x}) dV \\ + \int_\Omega c u(\mathbf{x}) v(\mathbf{x}) dV = \int_\Omega f(\mathbf{x}) v(\mathbf{x}) dV\end{aligned} \tag{2.3}$$

  In this equation, the A, b and c correspond to the constant factors of the derivatives in strong form. The derivatives are hidden in the differential operator $\mathbf{L}$.

- Discretisation of Ω
  Create a mesh of finite elements that span the whole domain. Usually these are triangles. Thus, this step is sometimes called "triangulation".

- Basis functions
  Choose a basis function $\Phi(\mathbf{x})$ that can be used to approximate the solution $u(\mathbf{x}) \approx u_h(\mathbf{x}) = \sum_{i=1}^{N} u_i \Phi(\mathbf{x})$. A common choice are Lagrange or Chebyshev polynomials.

- Solution
  Solve the resulting linear system of equations to determine the factors u_i. In the simple case where only second order derivatives are used in the strong from, the resulting linear system looks like this:

$$\begin{aligned}\sum_{j=1}^{N} v_j \sum_{i=1}^{N} u_i a(\Phi_i, \Phi_j) &= \sum_{j=1}^{N} v_j \mathbf{L}(\Phi_j) \\ \underbrace{\sum_{i=1}^{N} u_i a(\Phi_i, \Phi_j)}_{\mathbf{Au}} &= \underbrace{\mathbf{L}(\Phi_j)}_{\mathbf{b}}\end{aligned} \tag{2.4}$$

Modern solvers include more complex and advanced techniques to further improve the solution error and the computation time. Some of the most important concepts that are also available in NGSolve are listed here.

- Static Condensation:
  Depending on the number of discrete elements, the $\mathbf{A}$ matrix can be very large. Inverting large matrices is very time consuming. Condensation reduces this dimensionality by exploiting the structure of $\mathbf{A}$.

- Adaptive Mesh Refinement:
  The accuracy of a FEM-approximated solution mainly depends on the granularity of the mesh. Finer meshes produce more accurate solutions, but the computation time takes longer. This trade-off can be overcome by a self-adaptive mesh. NGSolve implements that in an adaptive loop that executes:

  - Solve PDE (with coarse mesh)
  - Estimate Error (for every element)
  - Mark Elements (that have the greatest error)
  - Refine Elements (that were previously marked)
  - ↻ until degrees of freedom exceed a specified N

- Preconditioner:
  Instead of solving the $\mathbf{A}^{-1}$ exactly, this can also be approximated by a matrix that is similar to $\mathbf{A}^{-1}$. The actual inverse can be iteratively approximated. NGSolve implements multiple different preconditioners and it even allows to create your own method.

2.2 Computational Intelligence Methods

The research community interested in computational intelligence solvers for differential equations has been steadily growing over the past 20 years. This chapter summarises the most important works done in the general field of development and application of such statistical numerical solvers. The following table 2.1 gives a brief overview of these papers and sorts them by relevance.

In general, all of these papers from the table use the weighted residual method (WRM), or some variant of that concept, to transform their differential equation into an optimisation problem. This serves as the fitness function and is necessary to evaluate a possible candidate solution and perform the evolutionary selection. The WRM method is further described in chapter 3.1.

In their paper Chaquet and Carmona 2019 describe an algorithm that approximates a solution with a linear combination of gaussian radial basis function (RBF) as kernels:

$$u(\mathbf{x})_{apx} = \sum_{i=1}^{N} \omega_i e^{\gamma_i (|\mathbf{x}-\mathbf{c}_i|^2)} \tag{2.5}$$

The approximated function $u(\mathbf{x})_{approx}$ can be fully determined by a finite number of parameters: $\omega_i, \gamma_i, c0_i,$ and $c1_i$. These are stacked together into a vector $\vec{u}_{apx}$ and called the decision variables which are optimised by the algorithm. The objective function can be seen in equation 2.6.

$$F(\vec{u}_{apx}) = \frac{\sum_{i=1}^{n_C} \xi(\mathbf{x}_i) ||\mathbf{L}u(\mathbf{x}_i) - f(\mathbf{x}_i)||^2 + \phi \sum_{j=1}^{n_B} ||\mathbf{B}u(\mathbf{x}_j) - g(\mathbf{x}_j)||^2}{m(n_C + n_B)} \tag{2.6}$$

The limit of the sum n_C denotes the number of inner collocation points $\mathbf{x}_i$ within the boundary Ω, whereas n_B is the number of discrete points $\mathbf{x}_j$ on the boundary

$\partial\Omega$. The first term represents the differential equation itself where $\mathbf{L}$ is a differential operator and $\mathbf{f}(\mathbf{x})$ is the inhomogeneous part. Similar, the second term stands for the boundary condition, where $\mathbf{B}$ is the differential operator and $\mathbf{g}(\mathbf{x})$ is the value on the boundary. The multipliers ξ and ϕ are weighting factors for either the inner or the boundary term. The whole term is normalised with the number of collocation points. The parameter of the kernels are determined via a Covariance Matrix Adaption Evolution Strategy (CMA-ES) (Hansen 2006). To further improve the solution, the evolutionary algorithm is coupled with a Downhill-Simplex (DS) method (Nelder and Mead 1965) to carry out the local search. The authors can show empirically that the local search significantly improves the performance by testing the algorithm on a set of 32 differential equations.

Babaei 2013 takes a different approach. They approximate a solution using a partial fourier series. The main advantage of this candidate representation is, that it is backed up with a solid foundation of convergence theory. The optimal parameters for the candidates are found using a particle swarm optimisation (PSO) algorithm (Kennedy and Eberhart 1995). The fitness function consists of two parts, one for the inner area (equation 2.7) and one for the boundary (equation 2.8). These are added together resulting in $F(u(\mathbf{x})_{apx}) = WRF + PFV$.

$$WRF = \int_{\Omega} |\mathbf{W}(\mathbf{x})||\mathbf{R}(\mathbf{x})|dx \tag{2.7}$$

This is exactly the formulation of the WRM, where $\mathbf{R}$ is the residual that originates directly from the differential equation $\mathbf{R} = \mathbf{L}u(\mathbf{x}_i) - f(\mathbf{x}_i)$ and $\mathbf{W}$ is an arbitrary weighting function. Instead of using a sum of collocation points, the integral is evaluated using a numerical integration scheme.

$$PFV = WRF \cdot \sum_{m=1}^{n_B} K_m g_m \tag{2.8}$$

In the penalty function from equation 2.8 the g_m stands for the boundary condition and K_m are penalty multipliers. The concept of this penalty function originates from Rajeev S. and Krishnamoorthy C. S. 1992.

Sobester, Nair, and Keane 2008 tried a radical different approach to incorporate the boundary condition into the solution. They found, that using genetic programming (GP) (Koza 1992) for the inner domain is only effective if the algorithm does not have to consider the boundary. They split the solution $u(\mathbf{x})_{apx}$ into two parts where $u(\mathbf{x})_{GP}$ represents the solution for the inner domain and $u(\mathbf{x})_{RBF}$ ensures the boundary condition, as seen in equation 2.9.

$$u(\mathbf{x})_{apx} = u(\mathbf{x})_{GP} + u(\mathbf{x})_{RBF} \tag{2.9}$$

At first, the GP step produced a trial solution according to the objective function

2.10.

$$F(u(\mathbf{x})_{apx}) = \sum_{i=1}^{n_C} ||\mathbf{L}u(\mathbf{x}_i) - f(\mathbf{x}_i)||^2 \tag{2.10}$$

After the GP procedure, a linear combination of radial basis functions $u(\mathbf{x})_{RBF} = \sum_{i=1}^{n_b} \alpha_i \Phi(||\mathbf{x} - \mathbf{x}_{i+n_d}||)$ is specifically tailored to $u(\mathbf{x})_{GP}$, that ensures the boundary condition at all $\mathbf{x}_{i+n_d}$ points on $\partial\Omega$. Finding the parameters α_i can be formulated as a least squares problem.

Chaquet and Carmona 2012 use a simple self-adaptive Evolution Strategy (ES) to evolve the coefficients of a partial Fourier series. The fitness function from equation 2.6 is reused. To reduce the search dimension (represented by the number of harmonics), they developed a scheme that only optimises one harmonic at a time and freezes the other coefficients. This scheme is based on the often observed principle that lower frequencies are more important in reconstructing a signal than higher ones. Albeit this concept might not be valid for all possible functions, it worked on all differential equations of their testbed.

Sadollah et al. 2017 compares three different optimisation algorithms to approximate differential equations: PSO, Harmony Search (HS) (Geem, Kim, and Loganathan 2001) and Water Cycle Algorithm (WCA) (Eskandar et al. 2012). The objective function for all algorithms is the same. They use the formulation in equation 2.7, where the weighting function is the same as the residual $|\mathbf{W}(\mathbf{x})| = |\mathbf{R}(\mathbf{x})| \implies WRF = \int_\Omega |\mathbf{R}(\mathbf{x})|^2 dx$. The integral is again approximated using a numerical integration scheme. They find that the PSO is slightly better at producing low error solutions, however the WCA is better at satisfying the boundary condition.

Fateh et al. 2019 use a simple variant of DE (Storn and Price 1997) where the candidates are extended to matrices. They take discrete function value points within the domain to appximate a solution of elliptic partial differential equations. This is a radical brute force approach that results in a massive search space dimension. Yet the main advantage is, that the solution is not limited to a decomposition of kernel functions and thus even non-smooth functions can be approximated. Since this approach does not produce an analytical solution, the boundary condition can be easily incorporated into the fitness function by simply taking the sum of squared residuals at every grid point, as seen in equation 2.11.

$$F(u(\mathbf{x})_{apx}) = \sqrt{\sum_{i=0}^{n} R(\mathbf{x}_i)^2} \tag{2.11}$$

Howard, Brezulianu, and Kolibal 2011 use a GP scheme to find the solution to a specific set of simplified convection-diffusion equations. They also represent a candidate as function value points over the domain. The function between these points is interpolated. The fitness function is similar to equation 2.10 with the exception that the n_C points are not predetermined. These points are sampled

randomly in the domain, thus allowing the algorithm to approximate the solution aside from the base points.

Panagant and Bureerat 2014 use polynomials as a candidate representation. They did not specify the order or the type of the polynomial. Five different simple versions of DE were tested. Further, they introduce a so called DE-New that increases the population size after every generation. Their proposition is that greater population sizes are better at finding good solutions.

Tsoulos and Lagaris 2006 use a grammatical evolution (GE) (Ryan, Collins, and Neill 1998) to find solutions to various differential equations. In contrary to GP, GE uses vectors instead of trees to represent the candidate string. The solution is evaluated as an analytical string, constructed of the functions sin, cos, exp and log, as well as all digits and all four basic arithmetic operations. The solution is measured by the same idea as displayed in equation 2.10. The algorithm was tested on multiple problems of ordinary differential equation (ODE), system of ODEs and PDE. Only the results for ODEs were promising.

Mastorakis 2006 couples a genetic algorithm (GA) (Goldberg, Korb, and Deb 1989) with a DS method for the local solution refinement. The candidates are represented as polynomials of the order 5 where the coefficients are optimised. The boundary condition is directly incorporated into the candidate, thus simplifying the objective function to equation 2.10. The focus here lays on unstable ODEs, that can not be solved with finite difference methods.

Kirstukas, Bryden, and Ashlock 2005 proposes a three-step procedure. The first step is time consuming and employs GP techniques to find basis functions that span the solution space. The second step is faster and uses a Gram–Schmidt algorithm to compute the basis function multipliers to develop a complete solution for a given set of boundary conditions. Using linear solver methods, a set of coefficients is found that produces a single function that both satisfies the differential equation and the boundary or initial conditions at all collocation points.

Howard and Roberts 2001 is one of the first advances in this field. They approximate a subset of the convection-diffusion equations with GP. Their main idea is to use a polynomial of variable length as the candidate representation. They use the same fitness function as already described in equation 2.9.

2.3 Differential Evolution

The differential evolution optimisation framework was first introduced in Storn and Price 1997. Due to its simple and flexible structure, it quickly became one of the most successful evolutionary algorithm. Over the years, several adaptation to the original framework have been proposed and some of them currently count to the best

performing algorithms, as the 100-Digit Challenge at the GECCO 2019 (Suganthan 2020) shows.

The main DE framework consists of three necessary steps, that continuously update a population of possible solutions. The population can be interpreted as a matrix, where each vector is a possible candidate for the optimum of the fitness function $f : \mathbf{R}^n \to \mathbf{R}$. These steps are performed in a loop until some termination condition is reached. Each of these steps are controlled by a user-defined parameter:

- mutation:
  mutation strength parameter F;
  uses the information from within the population to create a trial vector v_i;
  this is done by scaling the difference between random vectors x_r in the population - hence the name *differential* evolution; a simple */rand/1* mutation operator can be seen in equation 2.12;

$$v_i = x_i + F(x_{r1} - x_{r2}) \tag{2.12}$$

- crossover:
  crossover probability parameter CR;
  randomly mix the information between the trial vector v_i and a random candidate from the population x_i to a new trial vector u_i; the binomial crossover from equation 2.13 randomly takes elements from both vectors, where K is a random index to ensure that at least one element from the trial vector v_i is taken;

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } j = K \vee rand[0,1] \leq CR \\ x_{ij}, & \text{otherwise} \end{cases} \tag{2.13}$$

- selection:
  population size N;
  replace the old candidate x_i if the trial candidate u_i is better (according to the fitness function); this is performed for every individual in the population;

In modern DE variants, these parameters are self-adapted during the evolutionary process. This means that the algorithms can balance out between exploration of the search-space and exploitation of promising locations.

A prominent example of a modern DE with self-adaption is JADE, which was developed by Zhang and Sanderson 2009. The adaption is performed by taking successful F and CR parameter of the last generation into account. If a certain setting is successful in generating better candidates, newly selected F and CR tend towards that setting. The pseudocode is represented in the appendix A.1.

This idea was later refined by Tanabe and A. Fukunaga 2013. They propose a similar self-adaptive scheme but extend the "memory" for good F and CR parameters over

multiple generations. This idea should improve the robustness as compared to JADE. The pseudocode in appendix A.2 shows the outline of this so called SHADE algorithm.

The latest iteration of SHADE is called L-SHADE (Tanabe and A. S. Fukunaga 2014), which further improves the performance by including a deterministic adaptive concept for the population size. At first, L-SHADE starts with a big population size, and reduces the number of individuals in a linear fashion by deleting bad candidates. This has the effect of reducing the number of unnecessary function evaluation. The code is displayed in the appendix A.3.

Paper	Algorithm	Coding	Problems
Chaquet and Carmona 2019	CMA-ES (global); DS (local)	linear combination of Guassian kernals	testbench of ODEs system of ODEs and PDEs
Babaei 2013	PSO	partial sum of Fourier series	integro-differential equation systme of linear ODEs Brachistochrone nonlinear Bernoulli
Sobester, Nair, and Keane 2008	GP and RBF-NN	algebraic term for inner; RBF for boundary	Elliptic PDEs
Chaquet and Carmona 2012	self-adaptive ES	partial sum of Fourier series	testbench of ODEs system of ODEs and PDEs
Sadollah et al. 2017	PSO HS WCA	partial sum of Fourier series	singular BVP
Fateh et al. 2019	DE	function value gird	Elliptic PDEs
Howard, Brezulianu, and Kolibal 2011	GP	function value grid	convection–diffusion equation at different Peclet numbers
Panagant and Bureerat 2014	DE	polynomial of unspecified order	set of 6 different PDEs
Tsoulos and Lagaris 2006	GE	algebraic term	set of ODEs system of ODEs and PDEs
Mastorakis 2006	GA (global); DS (local)	5th order polynomial	unstable ODEs
Kirstukas, Bryden, and Ashlock 2005	GP	algebraic expression	heating of thin rod heating by current
Howard and Roberts 2001	GP	polynomial of arbitrary lengt	one-dimensional steady-state model of convection-diffusion equation

Table 2.1: Literature research on the general topic of stochastic slovers and their application. The papers are sorted by relevance to the present work.

3 Problem Definition

3.1 Optimisation Problem

introduce notation: $u_{apx}(x,y) = approximatedsolution$, $\vec{u_{apx}} = designvariables$, $u_{ext}(x,y) = exactsolution$, $F(\vec{u_{apx}}) = fitnessofapproximatedsolution$

There is no easy way to approach this optimisation problem. It can not be transformed into a least squares problem, also calculating the gradient is not an option, thus justifying the usage of a heuristic algorithm.

3.2 Fitness Function

The Fitness function is formulated as ...

3.3 Candidate Representation

4 Experimental Design

This chapter serves as an introduction to the experiments and gives an overview on how these are conducted. An integral part of solver comparison is to define a testbed that holds example problems with a known analytical solution. Further, quality measurements must be defined that compares the numerical to the analytical solution. The baseline for all experiments is the FEM solver NGSolve (Mitchell 2018).

4.1 Testbed

The testbed is a collection of multiple different 2-dimensional scalar PDEs that are analytically solved such that $u(\mathbf{x}) : \mathbf{R}^2 \to \mathbf{R}$ is a solution to the underlaying PDE. These can be used to demonstrate the correct implementation of a solver. The testbed can also be used to compare the performance of the classical FEM solver (NGSolve) with the computational intelligence (CI) solver. The actual equations are displayed in the appendix B. The equations used here are a mixture of multiple different testbeds. Specifically, the equations 2 and 3 are picked from the testbed in Chaquet and Carmona 2019. These problems were also used by Tsoulos and Lagaris 2006 and Panagant and Bureerat 2014. Preliminary tests have shown that the equations used in these paper are rather simple to approximate. Thus, more complicate equations are added to test the solvability over a wider variety of functions. The more complicated equations are taken from the National Institute of Standards and Technology (NIST) website (Mitchell 2018) that provides benchmarking problems for FEM slovers with adaptive mesh refinement methods. The other equations 0A, 0B and 8 are specially created to show different properties of the solver.

PDE 0A: Gauss Kernel (equation B.1) The analytic solution of this problem can be approximated arbitrarely close, since the solution is a sum of 5 different Gauss kernels. The purpose of this PDE is to show that the algorithm converges.

link Gauss kernel equations

PDE 0B: GSin Kernel (equation B.3) Similar to the PDE 0A, this problem is a sum of GSin kernels. To keep the search dimensionality roughly the same as in PDE 0A, the problem can be solved with 3 kernels. Again, the purpose of this equation is to determine the convergence with other kernel.

link Gsin chapter here

PDE 1: Polynomial 2D (equation B.5) The solution of this equation is a polynomial of order 20. The function is 0 on the boundary of the unit square. (Mitchell 2018)

PDE 2: Chaquet PDE 1 (equation B.7) This is the problem of the Chaquet testbed, that is also used by several other authors. Its main purpose is to build the bridge to those papers so that the results can be compared. Chaquet and Carmona 2019

PDE 3: Chaquet PDE 3 (equation B.9) This equation is solved by a polynomial of order 2. Again, the purpose is to compare the results to other papers. Chaquet and Carmona 2019

PDE 4: Sine Bump 2D (equation B.11) The sine bump occures in a similar fashion in both, the Chaquet testbed (PDE 8 in Chaquet and Carmona 2019) as well as the NIST (Mitchell 2018) testbed. This means that the underlying solution function is the same, but the PDE is posed differently. Preliminary results have shown that the formulation of the NIST testbed is harder to solve. Thus, the formulation of Chaquet is disregarded and the NIST problem is implemented.

PDE 5: Arctan Circular Wave Front (equation B.13) The main difficulty of this problem is the transition from the flat plateaus to the steep gradient of the circular wave front. Preliminary test have shown that this equation is one of the hardest in this testbed. (Mitchell 2018)

PDE 6: Peak 2D (equation B.15) The solution to this problem is described by a single Gaussian "peak" at $(0.5, 0.5)$ with a large exponent. This solution could be approximated by a single Gaussian kernel. The difficulty here is the steep gradient and a small region of interest. (Mitchell 2018)

PDE 7: Boundary Line Singularity (equation B.17) This equation is only determined on $x \in \mathbf{R}^+$, which results in a singularity line at $x = 0$. Towards this line the gradient increases. (Mitchell 2018)

PDE 8: Interior Point Singularity (equation B.19) The idea of this PDE is similar to the problem 7, but the singularity is located on the inner domain. The solution to this problem is not defined at $(0.5, 0.5)$, resulting in a very hard problem.

PDE 9: Arctan Wave Front Homogeneous Boundary Conditions 2D (equation B.21) Similar to PDE 5, the difficulty of this problem is the steep gradient. Additionaly, the boundary condition is zero wich results in sharp corners

on the boundary, that are hard to approximate. (Mitchell 2018)

4.2 Software Architecture

To simplify the preparation, execution and evaluation of the experiments, a comprehensive software architecture is defined. The UML class diagram can be seen in the appendix C. The architecture is organised in 4 main segments.

optimisation algorithm in red

The `IOptAlgoBase` interface must be implemented by every `OptAlgo` class to ensure the compatibility with `CiPdeBase` class of the testbed. A nice side-effect is that it reduces the number of user-defined parameters. An optimisation algorithm of this class must only take an initial guess (e.g. the starting population) as well as two stopping criteria: the maximum number of function evaluation or a minimum error to reach. Also a fitness function (i.e. the function to be optimised) must be provided. Four lists of the same length are returned: the optimum-guess, the function value, the crossover probability and the scale factor per each generation. The actual implementation of the algorithm is not predefined.

kernels in blue

As described in chapter 3.3, a candidate solution is defined as a sum of radial basis function. In order to test different candidate representations, different classes must be implemented. Again, to ensure compatibility with the `CiPdeBase` class, all representations must implement the `IKernelBase` interface. This assures that all classes have a method that can calculate the solution as well as first and second order derivatives. Here, only two kernels are imlemented.

testbed in orange

The testbed holds the 11 differential equations used in all experiments. The testbed is abstraced in such a way that an experiment is as simple as creating an PDE object and calling its solve method. All testbed classes must implement the `ITestbenchBase` interface. This ensures the minimal functionality of every subsequent class. Currently two classes implement this interface, the `FemPdeBase` and the `CiPdeBase`. These are the base classes that provide the specific attributes and methods needed for the FEM solver and the CI solver. The actual PDE problems are implemented in the classes `FemPde0` or `CiPde0` which inherit from the `FemPdeBase` and the `CiPdeBase`, respectively. The number in their name is representative for all different testbench problems and every PDE has its own class. Since all PDE classes have the same methods and attributes and only differ in their implementation and name, they do not have to be displayed separately. Therefore, they are symbolised together by a "stacked notation" used in the class diagram. Some methods in these classes must be overridden and adapted to the current PDE problem, which is indicated by the $\wedge$ character.

post processing in purple
Although the post processing block is not actually a class, it is still represented in this diagram. This module provides funtions that take `FemPdeN` or `CiPdeN` objects and performs actions with them.

update implemented function description

- `bool saveExpObj(obj, filename)`
  Save an `CiPdeN` object as a JavaScript Object Notation (JSON) file. The filename parameter can include a path, but it must end with .json. The results: execution time, memory usage, solution quality and all intergenerational data of the optimisation algoritm are stored in the file.
- `dict loadExpObject(filename)`
  Loads the JSON file located at the specified filename, which again can include a path. A dictionary with the saved `CiPdeN` parameters is returned.
- `bool drawGaussKernel(parameter, ggb)`
  Draw Gauss kernel to
- `bool drawGSinKernel(parameter, ggb)`
  Draw Gsin kernel to

4.3 Metric

In order to scientifically compare the results produced by the different solvers, some metrics are necessary. Three important solver-properties are measured: the execution time, the memory usage and the quality of the numerical solution. The following chapters describe the measurement process in greater detail.

4.3.1 Execution Time

The solving time is measured within the `solve()` method of either class. The time module of the Python Standard Library 2020 is used to interact with the system clock. The resolution, that the time module can access, depends on the system it is running on. Specifically, on the machine used in all further experiments, `time.time()` returns a 24 byte float that represents the time passed since 1st of January 1970. Usually, consecutive calls of this function return increasing values - changing the system time could interfere with the correctness of this value.

As the execution time of a program depends on many other factors, such as the current system load, the CPU temperature and the process scheduler, it is necessary to view it as a random variant. Thus, multiple replications have to be done before trying to interprete the results. These replications are not done within the `solve()` function and must be applied during the experiment. To reduce the random effects

and prevent possible outlier, the Python garbage collector is switched off during the time measurement. For a step-by-step description, the pseudocode is displayed in the appendix D.

4.3.2 Memory Usage

Similar to the solving time measurement, the memory usage is determined within the `solve()` method. The psutil module (Rodola 2020) provides the functionality to read the amount of memory attached to a process at a given time. The function call `process.memory_info()` returns a dictionary with multiple indicators about the current state of the process. Of special interest is the `rss` field. Resident Set Size (RSS) is the memory that is currently allocated by a process and actually held within the main memory. This means that memory that is currently swapped out, is not regarded with this measurement. Preliminary tests have shown that although the garbage collector was turned off, consecutive calls of `process.memory_info()` could result in a decreasing amount of rss-memory allocated by the process. A reason for this phenomenon could be that currently allocated memory is swapped out. When evaluating the results, these datapoints must be disregarded or treated as outliers.

Without assuming anything about the inner workings of the process, the memory is also a random variant. Thus, similar to the time measurement, replications have to be performed. The same pseudocode as for the time measurement (appendix D) also applies here.

4.3.3 Quality Measurement

Although the fitness function is the criterion that is optimised, it is not applicable as an objective quality measure. As Chaquet and Carmona 2019 describe, it depends on multiple factors:

- user-defined parameters ξ and ϕ
- the formulation of the PDE
- number of collocation points used
- number of kernels used

Thus, Chaquet and Carmona 2019 define a new quality measurement based on the Root-Mean-Square Error (RMSE) over the collocation points as seen in equation 4.1.

$$RMSE^2 = \frac{\sum_{i=1,\mathbf{x}_i \in C}^{n_C} ||\mathbf{u}(\mathbf{x}_i) - \mathbf{u_{ext}}(\mathbf{x}_i)||^2 + \sum_{j=1,\mathbf{x}_j \in B}^{n_B} ||\mathbf{u}(\mathbf{x}_j) - \mathbf{u_{ext}}(\mathbf{x}_j)||^2}{m(n_C + n_B)} \tag{4.1}$$

This quality criterion also three inherent issues. At first, it firmly depends on the number of collocation points used. In an algorithm that uses self-adaptive collocation points, this measurement would be rendered useless. Further, the quality is only measured on the collocation points and not in between. However, a good solution fits not only these discrete points, but the whole domain. Finally, the FEM method does not know anything about collocation points. A good quality measurement tool only use the numerical and analytical solution, independently of the solving method.

This leads to the quality measurement formulation used in this thesis: the L2 norm defined for functions as denoted in equation 4.2. This actually measures the distance between the analytical solution and the numerical approximation.

$$||u_{ext} - u_{apx}|| = \sqrt{\int_{\Omega} (u_{ext}(\mathbf{x}) - u_{apx}(\mathbf{x}))^2 d\mathbf{x}} \tag{4.2}$$

Although this integral is numerically evaluated, the discretisation is much finer than the resolution of the collocation points used in the fitness function - thus also regarding the areas between these points.

4.4 Baseline: NGSolve

As mentioned above, the NGSolve framework (Schöberl, Lackner, and Hochsteger 2020) is used as the baseline for all experiments. NGSolve is a state of the art FEM solver, that is in part developed and maintained by numerous well-known institutes such as Vienna University of Technology, University of Göttingen and Portland State University. This chapter describes the results obtained by running NGSolve on the testbed. The metrics from chapter 4.3 are applied.

4.4.1 Setup

At first the PDEs must be transformed into their corresponding weak form. As all testbed problems are Poisson equations and only differ in their algebraic sign and inhomogeneous part, the weak form is similar for all problems. Refering to equation 2.3, the terms $\vec{b} = 0$ and $c = 0$, thus these parts vanish. The matrix A is either $\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$ or with a negative sign $\left[\begin{smallmatrix} -1 & 0 \\ 0 & -1 \end{smallmatrix}\right]$ as only the non-mixed second order derivatives occure in the equations. This results in weak form as presented in equation 4.3, where the f is the RHS of problem.

$$\int_{\Omega} \nabla u^T \nabla v dV = \int_{\Omega} f v dV \tag{4.3}$$

To enhance the performance of NGSolve, static condensation is turned on. Further, a multigrid preconditioner is used. All problems are approximated by second order polynomials. The automatic mesh refinement is performed until a maximum of $5 \cdot 10^4$ Degree of Freedom (DOF) is reached. To properly interprete the results, 20 replications for each problem are performed. This is only needed for regarding time and memory, the solution itself is not a random variant. To further reduce the memory consumption, the GUI of NGSolve is switched off.

4.4.2 Result

With the parameters described above, the following results are produced. The solving time as well as the memory usage are displayed in the boxplots 4.1 and 4.2, respectively. In general, the solving time ranges from 2.5 to 5.5 seconds at about 45 to 70 MByte of main memory. Only problem 3 stands out as a notable exception. To keep the diagram visually appealing, this PDE is omitted and plotted in a separate figure.
The table 4.1 presents the achieved distances between the exact and the approximated solutions. On PDE 3 the best numerical quality is achieved.

Problem PDE	Distance
0A	$7.417692686693704 \cdot 10^{-6}$
0B	$2.6771365099980733 \cdot 10^{-6}$
1	$8.004152462854497 \cdot 10^{-7}$
2	$3.5013418621193666 \cdot 10^{-8}$
3	$1.6795224037775289 \cdot 10^{-9}$
4	$4.765830679060112 \cdot 10^{-7}$
5	$6.056858428283682 \cdot 10^{-6}$
6	$1.9078788449490833 \cdot 10^{-}7$
7	$5.202739901395381 \cdot 10^{-5}$
8	$3.237437258132996 \cdot 10^{-7}$
9	$2.3655968008139198 \cdot 10^{-7}$

Table 4.1: These are the results obtained by the FEM solver in terms of distance to the analytical solution. The solver achieves the smallest deviation in PDE 3.

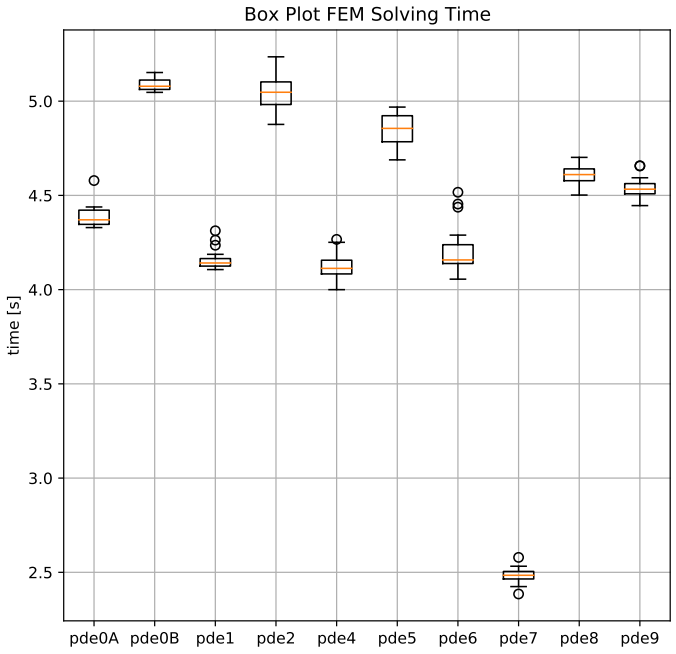

Figure 4.1: boxplot: time (in seconds) needed to solve the testbed PDE (without PDE3)

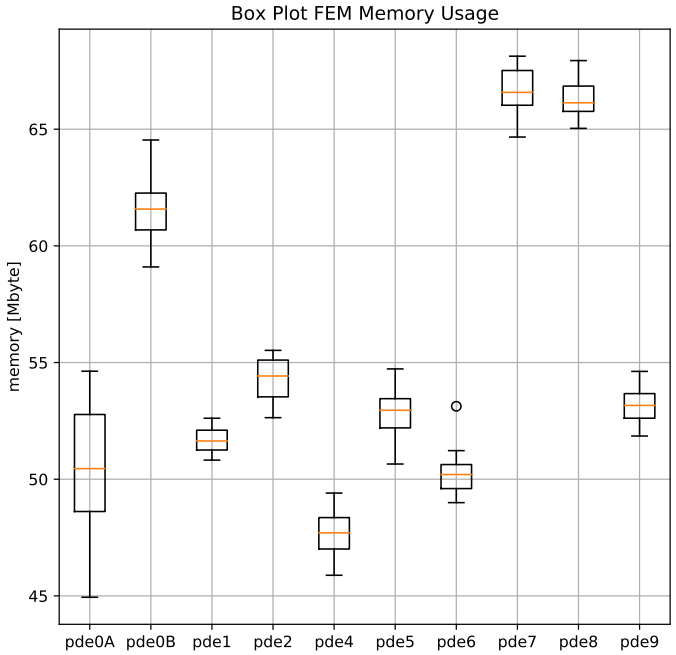

Figure 4.2: boxplot: memory (in Mbyte) needed to solve the testbed PDE (without PDE3)

The problem PDE 7 only needs around 2.5 seconds to solve. A reason could be that the solution only depends on one variable and the derivative with respect to y $\frac{\partial u}{\partial y} = 0$ vanishes. This does not effect the memory usage, since all $5 \cdot 10^4$ DOF must be created to terminate.
The following images 4.3 and 4.4 show the boxplots of the time and memory consumption for the testbed PDE 3. Compared to the other equations, this problem takes longer to solve while also needing more memory: a little less then 70 seconds at about 130 Mbyte. A possible explanation for that is the underlying structure of the PDE. As described in equation B.10, the exact solution to this problem is a polynomial of second order. This can be approximated perfectly by the FEM solver, since it also uses second order polynomials as basis functions. The mesh-refinement step takes more iteration to produce the $5 \cdot 10^4$ DOF as compared to the other testbed problems.

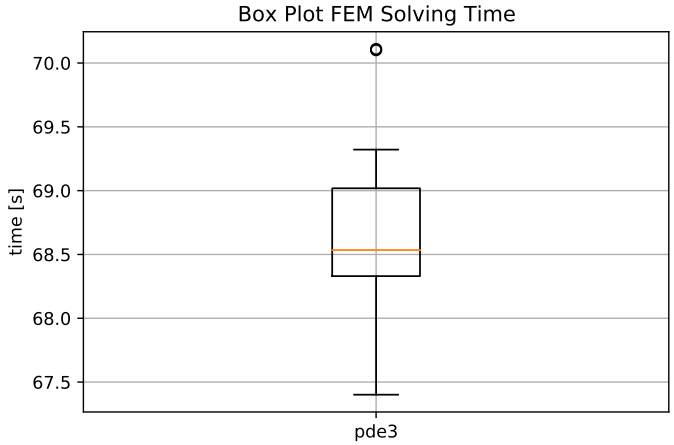

Figure 4.3: boxplot: time to solve testbed PDE3

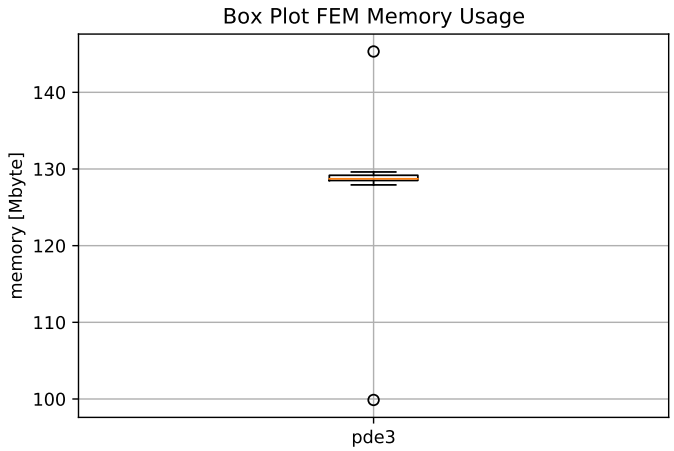

Figure 4.4: boxplot: memory to solve testbed PDE3

5 Experiment 0: Gauss Kernel

5.1 Hypotheses

5.2 Experiment Setup

5.3 Result

5.4 Interpretation

6 Experiment 1: Parallel Population JADE

6.1 Hypotheses

6.2 Experiment Setup

6.3 Result

6.4 Interpretation

7 Experiment 2: GSin Kernel

7.1 Hypotheses

7.2 Experiment Setup

7.3 Result

7.4 Interpretation

8 Experiment 3: Adaptive Number of Kernels

8.1 Hypotheses

8.2 Experiment Setup

8.3 Result

8.4 Interpretation

9 Experiment 4

9.1 Hypotheses

9.2 Experiment Setup

9.3 Result

9.4 Interpretation

10 Limitations

10.1 Testbed

Only PDEs with Laplace operator

10.2 Sine Bump 2D on Unit Square

Sensitive towards the boundary condition:
Chaquet PDE8 as described in dissertation vs. same PDE but with 0 on boundary

11 Conclusion

12 Summary and Further Work

Advantage: Can be applied to nearly any PDE without restrictions to a specific problem set Problematic: Might be applied to a problem although, there is an algorithm that is faster and has better convergence

Acronyms

AB Adams–Bashforth. 1

CI computational intelligence. 12, 14
CMA-ES Covariance Matrix Adaption Evolution Strategy. 5, 10

DE differential evolution. 1, 2, 6–8, 10
DOF Degree of Freedom. 17, 20
DS Downhill-Simplex. 5, 7, 10

ES Evolution Strategy. 6, 10

FE Forward-Euler. 1
FEM finite element method. IX, 1, 2, 12, 14, 16–18, 20

GA genetic algorithm. 7, 10
GE grammatical evolution. 7, 10
GP genetic programming. 5–7, 10

HS Harmony Search. 6, 10

ODE ordinary differential equation. 7

PDE partial differential equation. VIII, 1, 2, 7, 12–14, 16–20
PSO particle swarm optimisation. 5, 6, 10

RBF radial basis function. 4
RHS right hand side. 2, 17
RMSE Root-Mean-Square Error. 16
RSS Resident Set Size. 16

UML Unified Modeling Language. VIII, 14, 55

WCA Water Cycle Algorithm. 6, 10

WRM weighted residual method. 4, 5

Bibliography

Babaei, M. (July 1, 2013): "A general approach to approximate solutions of nonlinear differential equations using particle swarm optimization". In: *Applied Soft Computing* 13.7, pp. 3354–3365. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2013.02.005. URL: http://www.sciencedirect.com/science/article/pii/S1568494613000598 (visited on 02/27/2020) (cit. on pp. 5, 10).

Chaquet, Jose M. and Carmona, Enrique J. (Sept. 1, 2012): "Solving differential equations with Fourier series and Evolution Strategies". In: *Applied Soft Computing* 12.9, pp. 3051–3062. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2012.05.014. URL: http://www.sciencedirect.com/science/article/pii/S1568494612002505 (visited on 03/02/2020) (cit. on pp. 6, 10).

Chaquet, Jose M. and Carmona, Enrique J. (Mar. 1, 2019): "Using Covariance Matrix Adaptation Evolution Strategies for solving different types of differential equations". In: *Soft Computing* 23.5, pp. 1643–1666. ISSN: 1433-7479. DOI: 10.1007/s00500-017-2888-9. URL: https://doi.org/10.1007/s00500-017-2888-9 (visited on 03/02/2020) (cit. on pp. 4, 10, 12, 13, 16).

Eskandar, Hadi et al. (Nov. 1, 2012): "Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems". In: *Computers & Structures* 110-111, pp. 151–166. ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2012.07.010. URL: http://www.sciencedirect.com/science/article/pii/S0045794912001770 (visited on 05/05/2020) (cit. on p. 6).

Fateh, Muhammad Faisal et al. (Oct. 1, 2019): "Differential evolution based computation intelligence solver for elliptic partial differential equations". In: *Frontiers of Information Technology & Electronic Engineering* 20.10, pp. 1445–1456. ISSN: 2095-9230. DOI: 10.1631/FITEE.1900221. URL: https://doi.org/10.1631/FITEE.1900221 (visited on 02/11/2020) (cit. on pp. 6, 10).

Geem, Zong Woo; Kim, Joong Hoon, and Loganathan, G.V. (Feb. 1, 2001): "A New Heuristic Optimization Algorithm: Harmony Search". In: *SIMULATION* 76.2. Publisher: SAGE Publications Ltd STM, pp. 60–68. ISSN: 0037-5497. DOI: 10.1177/003754970107600201. URL: https://doi.org/10.1177/003754970107600201 (visited on 05/05/2020) (cit. on p. 6).

Goldberg, David E.; Korb, Bradley, and Deb, Kalyanmoy (1989): "Messy Genetic Algorithms: Motivation, Analysis, and First Results". In: *Complex Systems* (cit. on p. 7).

Hansen, Nikolaus (2006): "The CMA Evolution Strategy: A Comparing Review". In: Lozano, Jose A. et al. (Eds.): *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*. Studies in Fuzziness and Soft Computing. Berlin, Heidelberg: Springer, pp. 75–102. ISBN: 978-3-540-32494-2. DOI: 10.1007/3-540-32494-1_4. URL: https://doi.org/10.1007/3-540-32494-1_4 (visited on 05/05/2020) (cit. on p. 5).

Howard, Daniel; Brezulianu, Adrian, and Kolibal, Joseph (Jan. 1, 2011): "Genetic programming of the stochastic interpolation framework: convection–diffusion equation". In: *Soft Computing* 15.1, pp. 71–78. ISSN: 1433-7479. DOI: 10.1007/s00500-009-0520-3. URL: https://doi.org/10.1007/s00500-009-0520-3 (visited on 03/14/2020) (cit. on pp. 6, 10).

Howard, Daniel and Roberts, Simon C. (July 7, 2001): "Genetic Programming solution of the convection-diffusion equation". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO'01. San Francisco, California: Morgan Kaufmann Publishers Inc., pp. 34–41. ISBN: 978-1-55860-774-3. (Visited on 02/27/2020) (cit. on pp. 7, 10).

Kennedy, J. and Eberhart, R. (Nov. 1995): "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Proceedings of ICNN'95 - International Conference on Neural Networks. Vol. 4, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968 (cit. on p. 5).

Kirstukas, Steven J.; Bryden, Kenneth M., and Ashlock, Daniel A. (June 1, 2005): "A hybrid genetic programming approach for the analytical solution of differential equations". In: *International Journal of General Systems* 34.3. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/03081070500065676, pp. 279–299. ISSN: 0308-1079. DOI: 10.1080/03081070500065676. URL: https://doi.org/10.1080/03081070500065676 (visited on 03/02/2020) (cit. on pp. 7, 10).

Koza, John R. (1992): *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press. ISBN: 978-0-262-11170-6 (cit. on p. 5).

Mastorakis, Nikos E (Aug. 21, 2006): "Unstable Ordinary Differential Equations: Solution via Genetic Algorithms and the method of Nelder-Mead". In: WSEAS Int. Conf. on Systems Theory & Scientific Computation. Elounda, Greece, p. 7. URL: https://www.researchgate.net/profile/Nikos_Mastorakis2/publication/261859052_Unstable_ordinary_differential_equations_Solution_via_genetic_algorithms_and_the_method_of_Nelder-Mead/links/573b254e08ae9f741b2d7853.pdf (visited on 02/19/2020) (cit. on pp. 7, 10).

Mitchell, William F. (Mar. 2018): *NIST AMR Benchmarks*. en. URL: https://math.nist.gov/amr-benchmark/index.html (visited on 02/21/2020) (cit. on pp. 12–14).

Nelder, J. A. and Mead, R. (Jan. 1, 1965): "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4. Publisher: Oxford Academic, pp. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308. URL: https://academic.oup.com/comjnl/article/7/4/308/354237 (visited on 05/05/2020) (cit. on p. 5).

Panagant, Natee and Bureerat, Sujin (2014): "Solving Partial Differential Equations Using a New Differential Evolution Algorithm". In: *Mathematical Problems in Engineering* 2014. ISSN: 1024-123X Library Catalog: www.hindawi.com Pages: e747490 Publisher: Hindawi Volume: 2014. DOI: https://doi.org/10.1155/2014/747490. URL: https://www.hindawi.com/journals/mpe/2014/747490/ (visited on 02/27/2020) (cit. on pp. 7, 10, 12).

Python Standard Library, Time (May 2020): *time — Time access and conversions — Python 3.8.3 documentation*. en. Documentation. URL: https://docs.python.org/3/library/time.html (visited on 05/22/2020) (cit. on p. 15).

Rajeev S. and Krishnamoorthy C. S. (May 1, 1992): "Discrete Optimization of Structures Using Genetic Algorithms". In: *Journal of Structural Engineering* 118.5. Publisher: American Society of Civil Engineers, pp. 1233–1250. DOI: 10.1061/(ASCE)0733-9445(1992)118:5(1233). URL: https://ascelibrary.org/doi/abs/10.1061/(ASCE)0733-9445(1992)118:5(1233) (visited on 03/23/2020) (cit. on p. 5).

Rodola, Giampaolo (May 2020): *psutil documentation — psutil 5.7.1 documentation*. en. Documentation. https://github.com/giampaolo/psutil/blob/master/docs/index.rst. URL: https://psutil.readthedocs.io/en/latest/ (visited on 05/21/2020) (cit. on p. 16).

Ryan, Conor; Collins, JJ, and Neill, Michael O. (1998): "Grammatical evolution: Evolving programs for an arbitrary language". In: Banzhaf, Wolfgang et al. (Eds.): *Genetic Programming*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 83–96. ISBN: 978-3-540-69758-9. DOI: 10.1007/BFb0055930 (cit. on p. 7).

Sadollah, Ali et al. (July 4, 2017): "Metaheuristic optimisation methods for approximate solving of singular boundary value problems". In: *Journal of Experimental & Theoretical Artificial Intelligence* 29.4. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/0952813X.2016.1259271, pp. 823–842. ISSN: 0952-813X. DOI: 10.1080/0952813X.2016.1259271. URL: https://doi.org/10.1080/0952813X.2016.1259271 (visited on 03/02/2020) (cit. on pp. 6, 10).

Schöberl, Joachim; Lackner, Christopher, and Hochsteger, Matthias (Apr. 2, 2020): *NGSolve/ngsolve*. original-date: 2017-07-18T08:47:19Z. URL: https://github.com/NGSolve/ngsolve (visited on 04/02/2020) (cit. on pp. 2, 17).

Sobester, AndrÁs; Nair, Prasanth B., and Keane, Andy J. (Aug. 2008): "Genetic Programming Approaches for Solving Elliptic Partial Differential Equations". In: *IEEE Transactions on Evolutionary Computation* 12.4. Conference Name: IEEE Transactions on Evolutionary Computation, pp. 469–478. ISSN: 1941-0026. DOI: 10.1109/TEVC.2007.908467 (cit. on pp. 5, 10).

Storn, Rainer and Price, Kenneth (Dec. 1, 1997): "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11.4, pp. 341–359. ISSN: 1573-2916. DOI: 10.1023/A:1008202821328. URL: https://doi.org/10.1023/A:1008202821328 (visited on 04/05/2019) (cit. on pp. 6, 7).

Suganthan, Ponnuthurai Nagaratnam (Jan. 22, 2020): *P-N-Suganthan/CEC2019*. original-date: 2019-07-01T11:46:13Z. URL: https://github.com/P-N-Suganthan/CEC2019 (visited on 03/20/2020) (cit. on p. 8).

Tanabe, Ryoji and Fukunaga, Alex (June 2013): "Success-history based parameter adaptation for Differential Evolution". In: *2013 IEEE Congress on Evolutionary Computation*. 2013 IEEE Congress on Evolutionary Computation. ISSN: 1941-0026, pp. 71–78. DOI: 10.1109/CEC.2013.6557555 (cit. on p. 8).

Tanabe, Ryoji and Fukunaga, Alex S. (July 2014): "Improving the search performance of SHADE using linear population size reduction". In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. 2014 IEEE Congress on Evolutionary Computation (CEC). ISSN: 1941-0026, pp. 1658–1665. DOI: 10.1109/CEC.2014.6900380 (cit. on p. 9).

Tsoulos, I. G. and Lagaris, I. E. (Mar. 1, 2006): "Solving differential equations with genetic programming". In: *Genetic Programming and Evolvable Machines* 7.1, pp. 33–54. ISSN: 1573-7632. DOI: 10.1007/s10710-006-7009-y. URL: https://doi.org/10.1007/s10710-006-7009-y (visited on 02/15/2020) (cit. on pp. 7, 10, 12).

Zhang, Jingqiao and Sanderson, Arthur C. (Oct. 2009): "JADE: Adaptive Differential Evolution With Optional External Archive". In: *IEEE Transactions on Evolutionary Computation* 13.5. Conference Name: IEEE Transactions on Evolutionary Computation, pp. 945–958. ISSN: 1941-0026. DOI: 10.1109/TEVC.2009.2014613 (cit. on p. 8).

Appendices

A Differential Evolution Pseudocodes

A.1 JADE Pseudocode

Algorithm A.1: JADE Pseudocode

```
1  Function JADE(x_{g=0}, p, c, function, minError, maxGen):
2      fValue_{g=0} ← function(x_{g=0})
3      μ_CR ← 0.5
4      μ_F ← 0.5
5      A ← ∅
6      for g = 1 to G do
7          S_F ← ∅
8          S_CR ← ∅
9          for i = 1 to NP do
10             F_i ← randc_i(μ_F, 0.1)
11             v_i ← mutationCurrentToPBest1(x_{i,g}, A, fValue_g, F_i, p)
12             CR_i ← randn_i(μ_CR, 0.1)
13             u_i ← crossoverBIN(x_{i,g}, v_i, CR_i)
14             if function(x_{i,g}) ≥ function(u_{i,g}) then
15                 x_{i,g+1} ← x_{i,g}
16             end
17             else
18                 x_{i,g+1} ← u_{i,g}
19                 fValue_{i,g+1} ← function(u_{i,g})
20                 x_{i,g} → A
21                 CR_i → S_CR
22                 F_i → S_F
23             end
24         end
25         // resize A to size of x_g
26         if |A| > NP then
27             A ← A \ A_{rand_i}
28         end
29         μ_CR ← (1 − c) · μ_CR + c · arithmeticMean(S_CR)
30         μ_F ← (1 − c) · μ_F + c · lehmerMean(S_F)
31     end
```

A.2 SHADE Pseudocode

Algorithm A.2: SHADE Pseudocode

Function SHADE($\mathbf{x}_{G=0}$, p, H, $function$, $minError$, $maxGen$):
  $M_{CR} \leftarrow 0.5$; $M_F \leftarrow 0.5$; $A \leftarrow \emptyset$; $G \leftarrow 0$; $k \leftarrow 1$
  $fValue_{G=0} \leftarrow function(\mathbf{x}_{G=0})$
  while *termination condition not met* **do**
    $S_{CR} \leftarrow \emptyset$; $S_F \leftarrow \emptyset$
    for $i = 1$ *to* N **do**
      $r_i \leftarrow rand_{int}(1, H)$
      $CR_{i,G} \leftarrow randn_i(M_{CR,r_i}, 0.1)$; $F_{i,G} \leftarrow randc_i(M_{F,r_i}, 0.1)$
      $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,G}, A, fValue_G, F_i, p)$
      $u_i \leftarrow crossoverBIN(pop, v_i, CR)$
    end
    for $i = 1$ *to* N **do**
      if $function(u_{i,G}) \leq function(x_{i,G})$ **then**
        $x_{i,G+1} \leftarrow u_{i,G}$; $fValue_{i,G+1} \leftarrow function(\mathbf{u}_{i,G})$
      end
      else
        $x_{i,G+1} \leftarrow x_{i,G}$
      end
      if $function(u_{i,G}) < function(x_{i,G})$ **then**
        $x_{i,G} \rightarrow A$; $CR_{i,G} \rightarrow S_{CR}$; $F_{i,G} \rightarrow S_F$
      end
    end
    if $|A| > N$ **then**
      $A \leftarrow A \setminus A_{rand_i}$
    end
    if $S_{CR} \neq \emptyset \wedge S_F \neq \emptyset$ **then**
      $$M_{CR,k,G+1} = \begin{cases} arithmeticMean(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k,G} & otherwise \end{cases}$$
      $$M_{F,k,G+1} = \begin{cases} lehmerMean(S_F) & \text{if } S_F \neq \emptyset \\ M_{F,k,G} & otherwise \end{cases}$$
      $k \leftarrow k + 1$
      if $k > H$ **then**
        $k \leftarrow 1$
      end
    end
    $G \leftarrow G + 1$
  end

A.3 L-SHADE Pseudocode

Algorithm A.3: L-SHADE Pseudocode

Function `LSHADE`($\mathbf{x}_{G=0}$, p, H, $function$, $minError$, $maxGen$)**:**
  $M_{CR} \leftarrow 0.5$; $M_F \leftarrow 0.5$; $A \leftarrow \emptyset$; $G \leftarrow 0$; $k \leftarrow 1$
  $fValue_{G=0} \leftarrow function(\mathbf{x}_{G=0})$; $NG_{init} \leftarrow size(\mathbf{x}_{G=0})$; $NG_{min} = \lceil 1/p \rceil$
  while *termination condition not met* **do**
    $S_{CR} \leftarrow \emptyset$; $S_F \leftarrow \emptyset$
    for $i = 1$ *to* N **do**
      $r_i \leftarrow rand_{int}(1, H)$
      $CR_{i,G} \leftarrow randn_i(M_{CR,r_i}, 0.1)$; $F_{i,G} \leftarrow randc_i(M_{F,r_i}, 0.1)$
      $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,G}, A, fValue_G, F_i, p)$
      $u_i \leftarrow crossoverBIN(pop, v_i, CR)$
    end
    for $i = 1$ *to* N **do**
      if $function(u_{i,G}) \leq function(x_{i,G})$ **then**
        $x_{i,G+1} \leftarrow u_{i,G}$; $fValue_{i,G+1} \leftarrow function(\mathbf{u}_{i,G})$
      end
      else
        $x_{i,G+1} \leftarrow x_{i,G}$
      end
      if $function(u_{i,G}) < function(x_{i,G})$ **then**
        $x_{i,G} \rightarrow A$; $CR_{i,G} \rightarrow S_{CR}$; $F_{i,G} \rightarrow S_F$
      end
    end
    if $|A| > N$ **then**
      $A \leftarrow A \setminus A_{rand_i}$
    end
    if $S_{CR} \neq \emptyset \wedge S_F \neq \emptyset$ **then**
      $$M_{CR,k,G+1} = \begin{cases} arithmeticMean(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k,G} & otherwise \end{cases}$$
      $$M_{F,k,G+1} = \begin{cases} lehmerMean(S_F) & \text{if } S_F \neq \emptyset \\ M_{F,k,G} & otherwise \end{cases}$$
      $k \leftarrow k + 1$
      if $k > H$ **then**
        $k \leftarrow 1$
      end
    end
    $\mathbf{x}_{G+1} \leftarrow popSizeRed(\mathbf{x}_{G+1}, fValue, G, maxGen, NG_{init}, NGmin)$
    $G \leftarrow G + 1$
  end

B Testbed

The following pages describe the testbed that is used for all experiments. The problems are structured in these major points:

- differential equation
  - differential equation
  - domain Ω
  - Dirichlet bounday condition obtained by evaluating the solution on the boundary
- solution
- plot of the solution over the domain

PDE 0A: Gauss Kernel

Problem PDE:

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2} &= (18x^2-6)e^{-1.5(x^2+y^2)}+(18y^2-6)e^{-1.5(x^2+y^2)}\\ &+6(6x^2+12x+5)e^{-3((x+1)^2+(y+1)^2)}+6(6y^2+12y+5)e^{-3((x+1)^2+(y+1)^2)}\\ &+6(6x^2-12x+5)e^{-3((x-1)^2+(y+1)^2)}+6(6y^2+12y+5)e^{-3((x-1)^2+(y+1)^2)}\\ &+6(6x^2+12x+5)e^{-3((x+1)^2+(y-1)^2)}+6(6y^2-12y+5)e^{-3((x+1)^2+(y-1)^2)}\\ &+6(6x^2-12x+5)e^{-3((x-1)^2+(y-1)^2)}+6(6y^2-12y+5)e^{-3((x-1)^2+(y-1)^2)}\\ &\text{on the domain } \Omega: x,y\in[-2,2]\\ &\text{subjected to:}\\ u(x,2) &= 2e^{-1.5(x^2+4)}+e^{-3((x+1)^2+9)}+e^{-3((x+1)^2+1)}+e^{-3((x-1)^2+9)}+e^{-3((x-1)^2+1)}\\ u(x,-2) &= 2e^{-1.5(x^2+4)}+e^{-3((x+1)^2+1)}+e^{-3((x+1)^2+9)}+e^{-3((x-1)^2+1)}+e^{-3((x-1)^2+9)}\\ u(2,y) &= 2e^{-1.5(4+y^2)}+e^{-3(9+(y+1)^2)}+e^{-3(9+(y-1)^2)}+e^{-3(1+(y+1)^2)}+e^{-3(1+(y-1)^2)}\\ u(-2,y) &= 2e^{-1.5(4+y^2)}+e^{-3(1+(y+1)^2)}+e^{-3(1+(y-1)^2)}+e^{-3(9+(y+1)^2)}+e^{-3(9+(y-1)^2)}\end{aligned} \tag{B.1}$$

Solution:

$$\begin{aligned}u_{ext}(x,y) = 2e^{-1.5(x^2+y^2)} &+ e^{-3((x+1)^2+(y+1)^2)}+e^{-3((x+1)^2+(y-1)^2)}\\ &+ e^{-3((x-1)^2+(y+1)^2)}+e^{-3((x-1)^2+(y-1)^2)}\end{aligned} \tag{B.2}$$

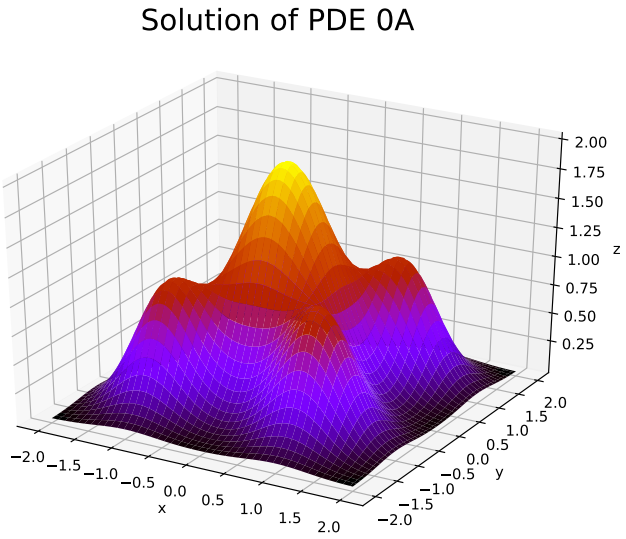

Figure B.1: PDE 0A Gauss Kernel solution plot

PDE 0B: GSin Kernel

Problem PDE:

$$\begin{aligned}
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \\
2e^{-2(x^2+y^2)}(2sin(-2(x^2+y^2)) + 2(1-8x^2)cos(-2(x^2+y^2)))+ \\
2e^{-2(x^2+y^2)}(2sin(-2(x^2+y^2)) + 2(1-8y^2)cos(-2(x^2+y^2)))+ \\
2e^{-1(x^2+y^2)}(1sin(-1(x^2+y^2)) + 1(1-4x^2)cos(-1(x^2+y^2)))+ \\
2e^{-1(x^2+y^2)}(1sin(-1(x^2+y^2)) + 1(1-4y^2)cos(-1(x^2+y^2)))+ \\
2e^{-0.1(x^2+y^2)}(0.1sin(-0.1(x^2+y^2)) + 0.1(1-0.4x^2)cos(-0.1(x^2+y^2)))+ \\
2e^{-0.1(x^2+y^2)}(0.1sin(-0.1(x^2+y^2)) + 0.1(1-0.4y^2)cos(-0.1(x^2+y^2))) \\
\text{on the domain } \Omega : x, y \in [-2, 2] \\
\text{subjected to:} \\
u(x,2) = e^{-2(x^2+4)}sin(2((x^2+4))) + e^{-1(x^2+4)}sin(1((x^2+4))) + e^{-0.1(x^2+4)}sin(0.1((x^2+4))) \\
u(x,-2) = e^{-2(x^2+4)}sin(2((x^2+4))) + e^{-1(x^2+4)}sin(1((x^2+4))) + e^{-0.1(x^2+4)}sin(0.1((x^2+4))) \\
u(2,y) = e^{-2(4+y^2)}sin(2((4+y^2))) + e^{-1(4+y^2)}sin(1((4+y^2))) + e^{-0.1(4+y^2)}sin(0.1((4+y^2))) \\
u(-2,y) = e^{-2(4+y^2)}sin(2((4+y^2))) + e^{-1(4+y^2)}sin(1((4+y^2))) + e^{-0.1(4+y^2)}sin(0.1((4+y^2)))
\end{aligned} \tag{B.3}$$

Solution:

$$\begin{aligned} u_{ext}(x,y) =& e^{-2(x^2+y^2)} sin(2((x^2+y^2))) \\ & e^{-1(x^2+y^2)} sin(1((x^2+y^2))) \\ & e^{-0.1(x^2+y^2)} sin(0.1((x^2+y^2))) \end{aligned} \tag{B.4}$$

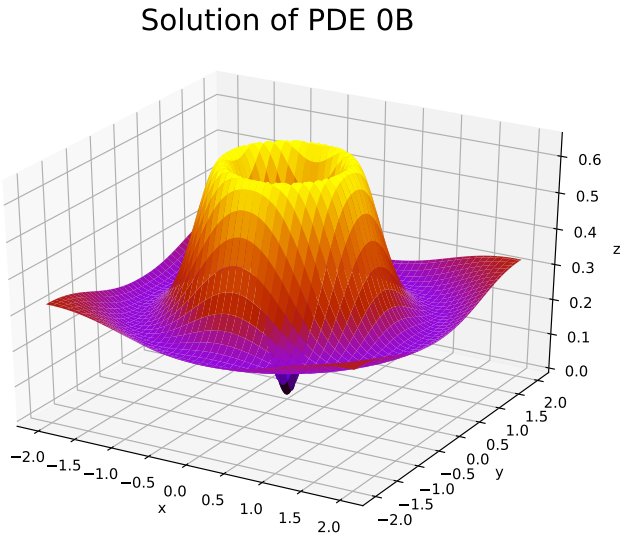

Figure B.2: PDE 0B Gsin Kernel solution plot

PDE 1: Polynomial 2D

Problem PDE:

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = \\ -2^{40} y^{10}(1-y)^{10}[90x^8(1-x)^{10} - 200x^9(1-x)^9 + 90x^{10}(1-x)^8] \\ -2^{40} x^{10}(1-x)^{10}[90y^8(1-y)^{10} - 200y^9(1-y)^9 + 90y^{10}(1-y)^8] \\ \text{on the domain } \Omega : x, y \in [0,1] \\ \text{subjected to:} \\ u(x,1) = 0 \\ u(x,0) = 0 \\ u(1,y) = 0 \\ u(0,y) = 0 \end{aligned} \tag{B.5}$$

Solution:

$$u_{ext}(x,y) = 2^{40} x^{10}(1-x)^{10} y^{10}(1-y)^{10} \tag{B.6}$$

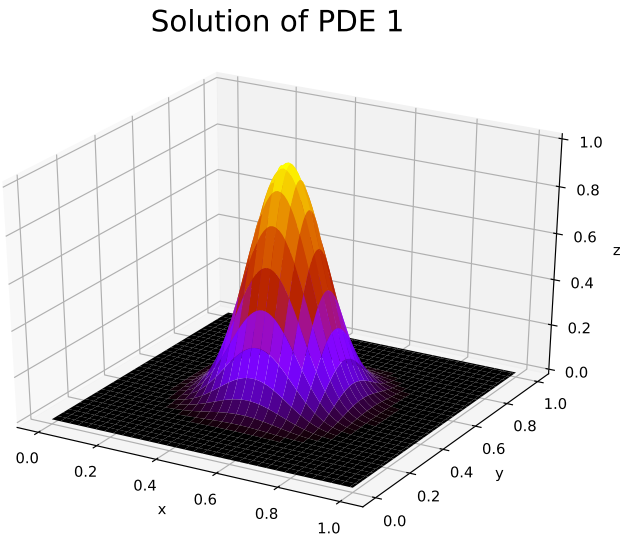

Figure B.3: PDE 1 Polynomial 2D solution plot

PDE 2: Chaquet PDE 1

Problem PDE:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= e^{-x}(x - 2 + y^3 + 6y) \\ \text{on the domain } \Omega : \mathbf{x} &\in [0, 1] \\ \text{subjected to:} & \\ u(x,0) = xe^{-x} u(x,1) = (x+1)e^{-x} u(0,y) &= y^3 \\ u(1,y) &= (1+y^3)e^{-1} \end{aligned} \tag{B.7}$$

Solution:

$$u_{ext}(x,y) = (x + y^3)e^{-x} \tag{B.8}$$

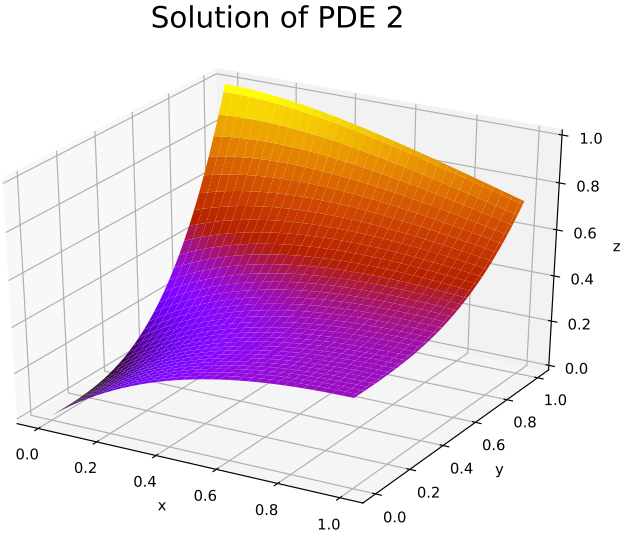

Figure B.4: PDE 2 Chaquet PDE 1 solution plot

PDE 3: Chaquet PDE 3

Problem PDE:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= 4 \\ \text{on the domain } \Omega : \mathbf{x} &\in [0,1] \\ \text{subjected to:} & \\ u(x,0) &= x^2 + x + 1 \\ u(x,1) &= x^2 + x + 3 \\ u(1,y) &= y^2 + y + 3 \\ u(0,y) &= y^2 + y + 1 \end{aligned} \tag{B.9}$$

Solution

$$u_{ext}(x,y) = x^2 + y^2 + x + y + 1 \tag{B.10}$$

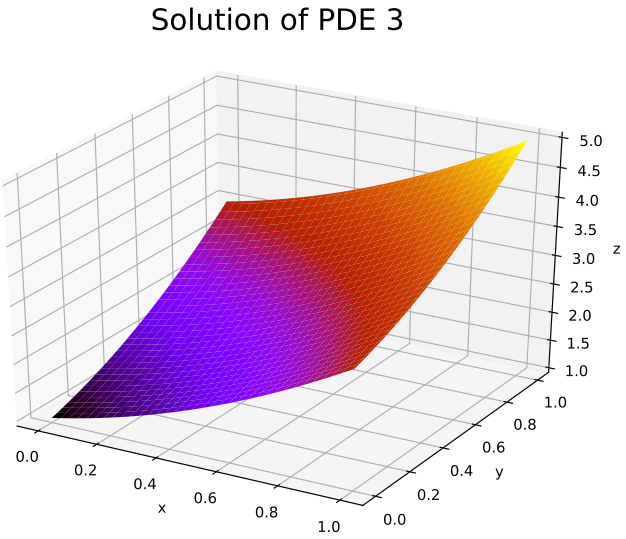

Figure B.5: PDE 3 Chaquet PDE 3 solution plot

PDE 4: Sine Bump 2D

Problem PDE:

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= 2\pi^2 sin(\pi x) sin(\pi y) \\ \text{on the domain } \Omega : \mathbf{x} &\in [0,1] \\ \text{subjected to:} & \\ u(x,0) &= 0 \\ u(x,1) &= 0 \\ u(0,y) &= 0 \\ u(1,y) &= 0 \end{aligned} \tag{B.11}$$

Solution:

$$u_{ext}(x,y) = sin(\pi x) sin(\pi y) \tag{B.12}$$

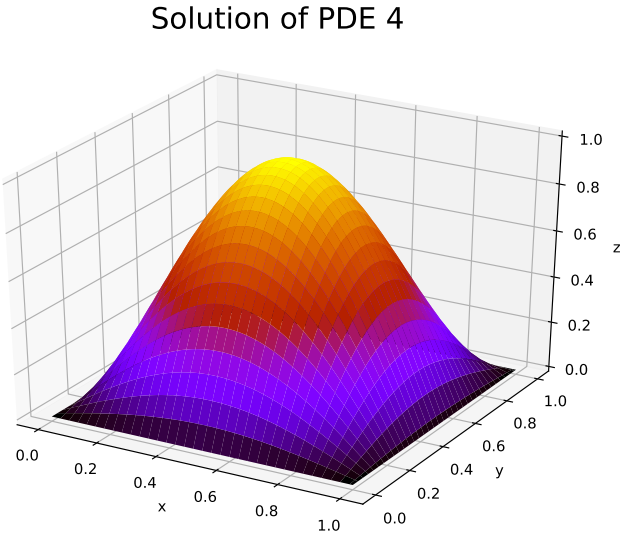

Figure B.6: PDE 4 Sine Bump 2D solution plot

PDE 5: Arctan Circular Wave Front

Problem PDE:

$$
\begin{aligned}
-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= \frac{16000(\sqrt{(x-0.05)^2+(y-0.05)^2}-0.7)}{(1+400(-0.7+\sqrt{(x-0.05)^2+(y-0.05)^2})^2)^2} \\
&+ \frac{20(x-0.05)^2+20(y-0.05)^2}{(1+400(\sqrt{(x-0.05)^2+(y-0.05)^2}-0.7)^2)((x-0.05)^2+(y-0.05)^2)^{3/2}} \\
&- \frac{40}{(1+400(\sqrt{(y-0.05)^2+(x-0.05)^2}-0.7)^2)\sqrt{(y-0.05)^2+(x-0.05)^2}} \\
&\text{on the domain } \Omega : \mathbf{x} \in [0,1] \\
&\text{subjected to:} \\
u(x,0) &= tan^{-1}\left(20\left(\sqrt{(x-0.05)^2+0.0025}-0.7\right)\right) \\
u(x,1) &= tan^{-1}\left(20\left(\sqrt{(x-0.05)^2+0.9025}-0.7\right)\right) \\
u(0,y) &= tan^{-1}\left(20\left(\sqrt{0.0025+(y-0.05)^2}-0.7\right)\right) \\
u(1,y) &= tan^{-1}\left(20\left(\sqrt{0.9025+(y-0.05)^2}-0.7\right)\right)
\end{aligned}
\tag{B.13}
$$

Solution:

$$
u_{ext}(x,y) = tan^{-1}\left(20\left(\sqrt{(x-0.05)^2+(y-0.05)^2}-0.7\right)\right) \tag{B.14}
$$

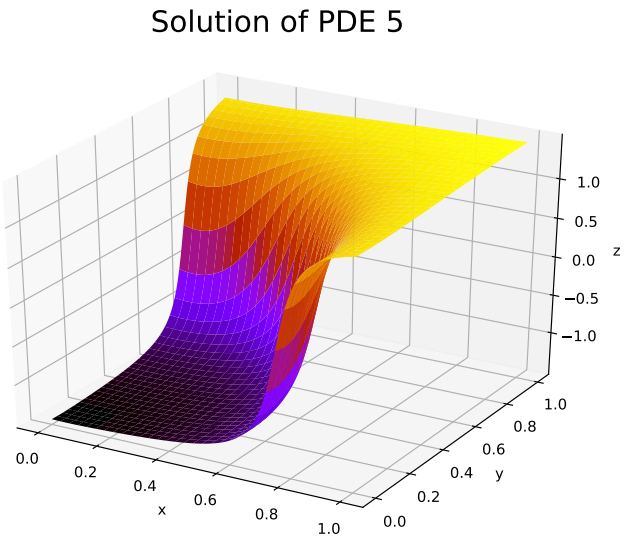

Figure B.7: PDE 5 Arctan Circular Wave Front solution plot

PDE 6: Peak 2D

Problem PDE:

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = \\ -(4 \cdot 10^6 x^2 - 4 \cdot 10^6 x + 998 \cdot 10^3) e^{-1000((x-0.5)^2+(y-0.5)^2)} \\ -(4 \cdot 10^6 y^2 - 4 \cdot 10^6 y + 998 \cdot 10^3) e^{-1000((x-0.5)^2+(y-0.5)^2)} \\ \text{on the domain } \Omega : \mathbf{x} \in [0,1] \\ \text{subjected to:} \\ u(x,0) = e^{-1000((x-0.5)^2+0.25)} \\ u(x,1) = e^{-1000((x-0.5)^2+0.25)} \\ u(0,y) = e^{-1000(0.25+(y-0.5)^2)} \\ u(1,y) = e^{-1000(0.25+(y-0.5)^2)} \end{aligned} \tag{B.15}$$

Solution:

$$u_{ext}(x,y) = e^{-1000((x-0.5)^2+(y-0.5)^2)} \tag{B.16}$$

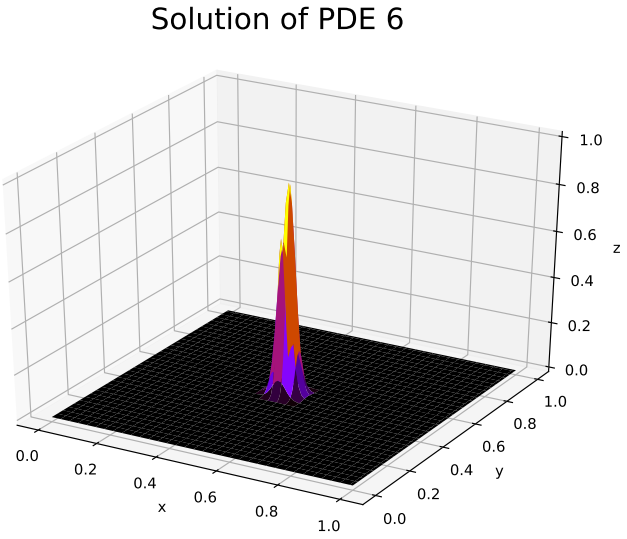

Figure B.8: PDE 6 Peak 2D solution plot

PDE 7: Boundary Line Singularity

Problem PDE:

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= 0.24x^{-1.4} \\ \text{on the domain } \Omega : \mathbf{x} &\in [0,1] \\ \text{subjected to:} & \\ u(x,0) &= x^{0.6} \\ u(x,1) &= x^{0.6} \\ u(0,y) &= 0 \\ u(1,y) &= 1^{0.6} \end{aligned} \tag{B.17}$$

Solution:

$$u_{ext}(x,y) = x^{0.6} \tag{B.18}$$

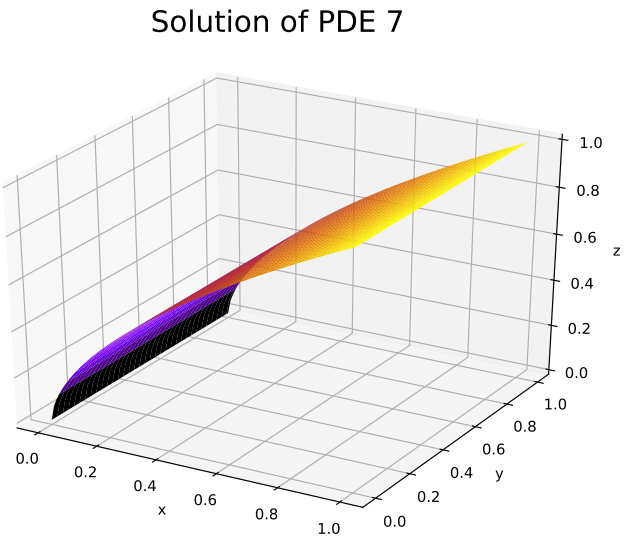

Figure B.9: PDE 7 Boundary Line Singularity solution plot

PDE 8: Interior Point Singularity

Problem PDE:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= \frac{1}{\sqrt{x^2 - x + y^2 - y + 0.5}} \\ \Omega : \mathbf{x} &\in [0,1] \\ \text{on the domain subjected to:} & \\ u(x,0) &= \sqrt{(x-0.5)^2 + 0.25} \\ u(x,1) &= \sqrt{(x-0.5)^2 + 0.25} \\ u(0,y) &= \sqrt{0.25 + (y-0.5)^2} \\ u(1,y) &= \sqrt{0.25 + (y-0.5)^2} \end{aligned} \tag{B.19}$$

Solution:

$$u_{ext}(x,y) = \sqrt{(x-0.5)^2 + (y-0.5)^2} \tag{B.20}$$

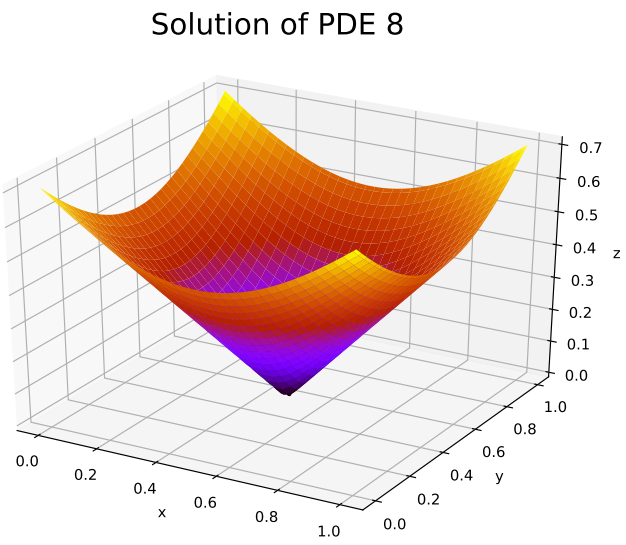

Figure B.10: PDE 8 Interior Point Singularity solution plot

PDE 9: Arctan Wave Front Homogeneous Boundary Conditions 2D

Problem PDE:

$$
\begin{aligned}
-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = \\
\frac{20\sqrt{2}(x^2+y^2-2x^2y-2xy^2+4xy-x-y)}{400(\frac{x+y}{\sqrt{2}}-0.8)^2+1} \\
+\frac{16000(1-x)x(1-y)y(\frac{x+y}{\sqrt{2}}-0.8)}{(400(\frac{x+y}{\sqrt{2}}-0.8)^2+1)^2} \\
+tan^{-1}\left(20\left(\frac{x+y}{\sqrt{2}}-0.8\right)\right)(2(1-y)y+2(1-x)x) \\
\text{on the domain } \Omega : \mathbf{x} \in [0,1] \\
\text{subjected to:} \\
u(x,0) = 0 \\
u(x,1) = 0 \\
u(0,y) = 0 \\
u(1,y) = 0
\end{aligned}
\tag{B.21}
$$

Solution:

$$
u_{ext}(x,y) = tan^{-1}\left(20\left(\frac{(x+y)}{\sqrt{2}}-0.8\right)\right)x(1-x)y(1-y) \tag{B.22}
$$

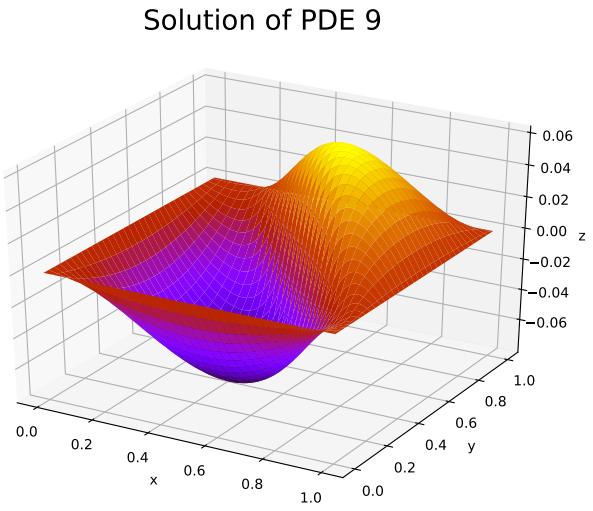

Figure B.11: PDE 9 Arctan Wave Front Homogeneous Boundary Conditions 2D solution plot

C Software Architecture

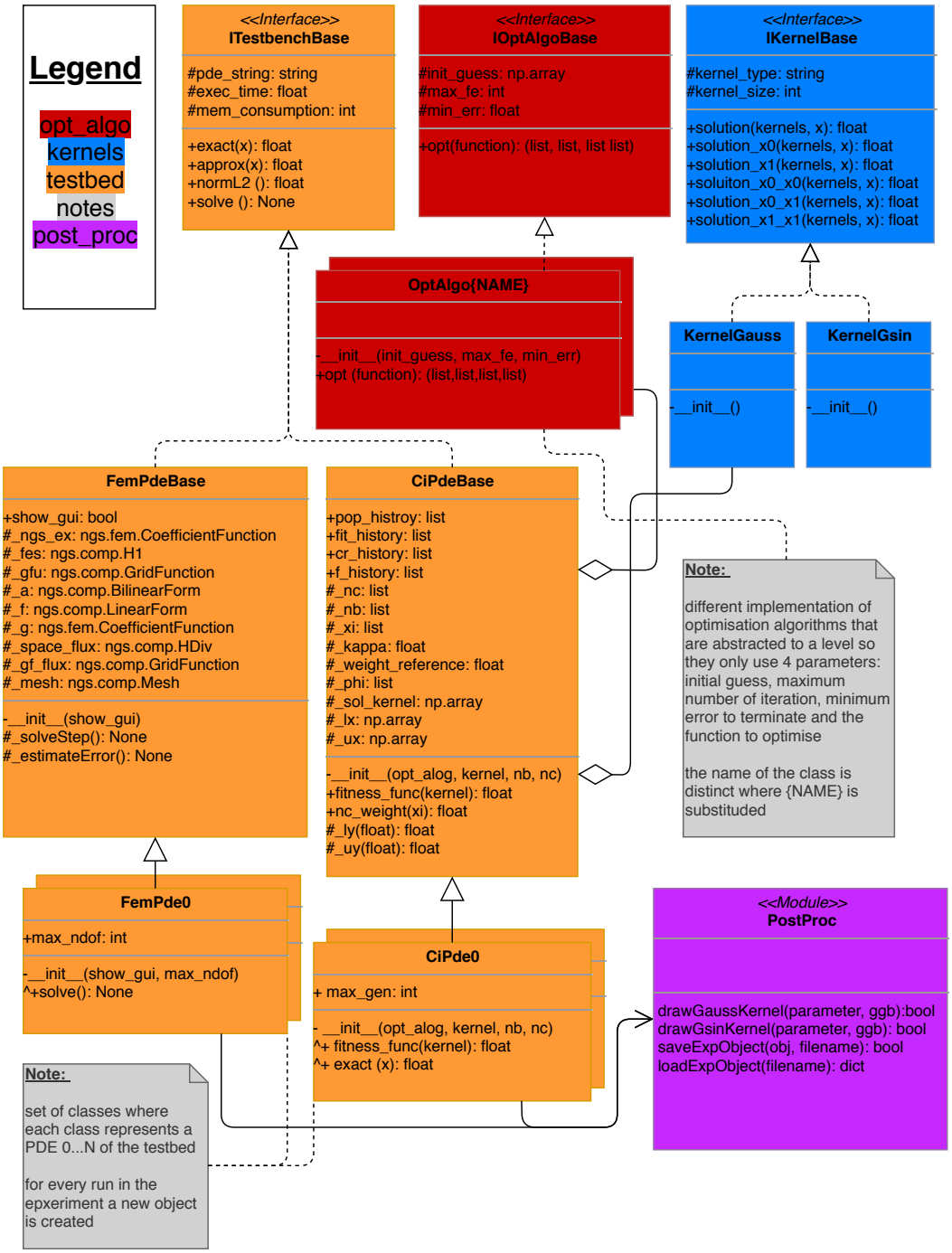

Figure C.1: This UML class diagram describes the software architecture defined to prepare, run and evaluate the experiments.

D Solve Method

Algorithm D.1: solve function pseudocode

```
1  Function solve():
2      gc.disable()
3      while gc.isenabled() do
4          time.sleep(0.1)
5      end
6      process = psutil.Process()
7      memstart = process.memory_info().rss
8      t_start = time.time()
9      // perform solver steps
10     // that are particular
11     // to FEM or CI solver
12     self._exec_time = time.time() − t_start
13     memstop = process.memory_info().rss − memstart
14     gc.enable()
15     gc.collect()
```

E pJADE

Algorithm E.1: JADE Pseudocode

```
1  Function pJADE(x_{g=0}, p, c, function, minError, maxGen):
2      fValue_{g=0} ← function(x_{g=0})
3      μ_CR ← 0.5
4      μ_F ← 0.5
5      A ← ∅
6      for g = 1 to G do
7          S_F ← ∅
8          S_CR ← ∅
9          for i = 1 to NP do
10             F_i ← randc_i(μ_F, 0.1)
11             v_i ← mutationCurrentToPBest1(x_{i,g}, A, fValue_g, F_i, p)
12             CR_i ← randn_i(μ_CR, 0.1)
13             u_i ← crossoverBIN(x_{i,g}, v_i, CR_i)
14         end
15         for i = 1 to NP do
16             if function(x_{i,g}) ≥ function(u_{i,g}) then
17                 x_{i,g+1} ← x_{i,g}
18             end
19             else
20                 x_{i,g+1} ← u_{i,g}
21                 fValue_{i,g+1} ← function(u_{i,g})
22                 x_{i,g} → A
23                 CR_i → S_CR
24                 F_i → S_F
25             end
26         end
27         // resize A to size of x_g
28         if |A| > NP then
29             A ← A \ A_{rand_i}
30         end
31         μ_CR ← (1 − c) · μ_CR + c · arithmeticMean(S_CR)
32         μ_F ← (1 − c) · μ_F + c · lehmerMean(S_F)
33     end
```

Statement of Affirmation

I hereby declare that this thesis was in all parts exclusively prepared on my own, without using other resources than those stated. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis or parts of it were not previously submitted to another academic institution and have also not yet been published.

Dornbirn, 29.05.2020 Schwartze Nicolai, BSc