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, 07.07.2020

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:

1. Step: 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.

2. Step: Strong Form
   This is the standard formulation of the PDE with a Dirichlet boundary condition:

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

3. Step: 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 $\mathbf{L}$.

4. Step: 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".

5. Step: 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.

6. Step: 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. Typically, finer meshes tend to 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) (as developed by Schwefel 1977 and Rechenberg 1978) 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 approximate 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) (Holland 1962) 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 adaptations 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 (Ponnuthurai Nagaratnam 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 some vectors in the population - hence the name *differential* evolution; the */current-to-pbest/1* mutation operator can be seen in equation 2.12; x_i is the current individual, x^p_{best} is one random vector of the p% top vectors, x_{r1} is a random vector of the population while $\tilde{x}_{r2}$ is randomly chosen from the population and the archive;

$$v_i = x_i + F_i(x^p_{best} - x_i) + F_i(x_{r1} - \tilde{x}_{r2}) \tag{2.12}$$

- 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 solver and their application. The papers are sorted by relevance to the present work.

3 Problem Definition

This chapter describes the broader idea of the WRM, which is used to reformulate any differential equation into an optimisation problem and its application in the field of spectral methods (Shen, Tang, and Wang 2011). The fitness function originates from these approaches. Further, the different approximation schemes and numerical solution representation are depicted. The following paragraphs also introduce the mathematical notation used for this work.

3.1 Theoretical Foundation

The most general description of a PDE is displayed in equation 3.1.

$$\begin{aligned} \mathbf{L}u(\mathbf{x}) &= f(\mathbf{x}) \\ \text{subjected to: } \mathbf{B}u(\mathbf{x}) &= g(\mathbf{x}) \end{aligned} \tag{3.1}$$

This is similar to the formulation of the strong form in equation 2.2, but not limited to a Dirichlet boundary problem. The matrix $\mathbf{B}$ can include differentiation, effectively allowing i.e. Neumann boundary condition.

An approximate solution to this problem is expressed as a finite sum of basis functions $\phi_k(\mathbf{x})$, as denoted in equation 3.2. The coefficients a_k are to be determined. In classical approximation schemes, the basis functions are orthogonal, such as trigonometric functions or Lagrange basis polynomials.

$$u(\mathbf{x}) \approx u_{apx}(\mathbf{x}) = \sum_{k=0}^{N} a_k \phi_k(\mathbf{x}) \tag{3.2}$$

The residual of a PDE, as shown in equation 3.3, is formulated as the difference between the left and the right side of the equation. The residual of a solved PDE is zero. This is only possible for the $u_{ext}(x, y)$ analytical solution. For a numerical approximation $u_{apx}(x, y)$, the residual should be "small enough".

$$\mathbf{R}(\mathbf{x}) = \mathbf{L}u_{apx}(\mathbf{x}) - f(\mathbf{x}) \tag{3.3}$$

The WRM (Shen, Tang, and Wang 2011) tries to minimise this residual of a numerical candidate solution over the whole domain Ω. Therefore, R at every $\mathbf{x}_i \in \Omega$ is evaluated and added up, resulting in the following integral of equation 3.4. The residual at every $\mathbf{x}$ can be scaled by a weighting function over the domain as denoted with $W(\mathbf{x})$. The choice of the test function $\psi(\mathbf{x})$ is distinct for different methods. The actual optimum does not change, since the zero-product property holds. The WRM builds the basis for many solving strategies, including FEM.

$$WRF = \int_{\Omega} \mathbf{R}(\mathbf{x})\psi(\mathbf{x})W(\mathbf{x})dx \tag{3.4}$$

This integral must be evaluated numerically. There are many different integration schemes available, but they are typically computational expensive. A less extensive approach is to evaluate the integral argument at different "collocation points" and add up the results. This can be seen in equation 3.5. Although it is not an integral per se, it does assign a numerical "score" to the residual.

$$\sum_{k=0}^{N} \mathbf{R}(\mathbf{x}_k)\psi(\mathbf{x}_k)W(\mathbf{x}_k) \tag{3.5}$$

To ensure that negative and positive residuals do not cancel each other out, a common choice in heuristic methods from table 5.1 is to choose $\psi(\mathbf{x}_k)$ as the residual itself, effectively squaring it. The resulting "sum of squared residuals" forms the basis for nearly all fitness functions in the current literature, as represented in equation 3.6.

$$\sum_{k=0}^{N} \mathbf{R}(\mathbf{x}_k)^2 W(\mathbf{x}_k) \tag{3.6}$$

These methods are often called "spectral methods" (Shen, Tang, and Wang 2011). Their main advantage over finite difference methods is that they take the whole domain into account. The derivative of the function at one point is influenced by the function at every other point in the domain. This global approach can lead to a better accuracy.

3.2 Fitness Function

As mentioned above, the basis of the fitness function used in Chaquet and Carmona 2019 is the squared weighted residual in equation 3.6. They split the summation over the collocation points into two parts: the points on the boundary (nb) and the points on the inner domain (nc). Further, they divide the fitness value by the number of points used. This fitness function, as seen in equation 3.7, is adopted in this work.

$$F(u_{apx}) = \frac{\sum_{i=1}^{n_C} \xi(\mathbf{x}_i)||\mathbf{L}u(\mathbf{x}_i) - f(\mathbf{x}_i)||^2 + \phi \sum_{j=1}^{n_B} ||\mathbf{B}u(\mathbf{x}_j) - g(\mathbf{x}_j)||^2}{(n_C + n_B)} \tag{3.7}$$

The fitness function has two important properties:

- The fitness value of the exact solution must be 0:
  $F(u_{ext}(\mathbf{x})) \equiv 0$
- The fitness value of an approximate solution is greater or equal to 0:
  $F(u_{apx}(\mathbf{x})) \geq 0$

The weighting function is also split into two parts: ξ for the inner collocation points and ϕ as penalty factor for the boundary points. The weighting function ξ can be adapted by a parameter κ. For a $\kappa > 1$, ξ assigns larger values for collocation points closer to the boundary, effectively shifting the relative importance towards the boundary. These weights can be calculated a priori, so no computational effort is added to the fitness function. Again, the weights are directly extracted from Chaquet and Carmona 2019.

$$\xi(\mathbf{x}_i) = \frac{1 + \kappa \left(1 - \frac{min_{\forall \mathbf{x}_j \in n_B} ||\mathbf{x}_i - \mathbf{x}_j||}{max_{\forall \mathbf{x}_k \in n_C} (min_{\forall \mathbf{x}_j \in n_B} ||\mathbf{x}_k - \mathbf{x}_j||)}\right)}{1 + \kappa} \tag{3.8}$$

3.3 Candidate Representation

As Chaquet and Carmona 2019 suggest, an approximate solution should be represented as a summation of N scaled and shifted radial basis functions ϕ, as denoted in equation 3.2 RBF are functions that are point symmetric to a centre c and their function value only depends on the radius r.

cite approximation theorem

$$\mathbf{r} = ||\mathbf{x} - \mathbf{c}|| \tag{3.9}$$

This thesis mainly works with two different types of RBF. The first one is the classical Gauss RBF (gak). The second one, further called GSin RBF (gsk), is more complex. A candidate solution is represented as the parameters that shift and deform these RBF.

3.3.1 Gauss Kernel

In general, a Gauss Kernel (GaK) is the classical choice to approximate functions. It was first introduced in Broomhead and Lowe 1988. These kernels have been used in many different works, including Chaquet and Carmona 2019. Equation 3.10 shows the formulation of such a kernel. One kernel has 4 parameters, that change its shape and location. ω is a scaling factor for each and γ describes how sharp the e-function is. Depending on the space the solution is defined in, the number of $\mathbf{c}$ could change - one for each dimension. However, in this work only $\mathbf{R}^2$ is of interest. In figure 3.1 a single standard GaK is plotted.

cite approximation proof

$$gak(\mathbf{x}) = \omega e^{-\gamma \mathbf{r}^2} \tag{3.10}$$

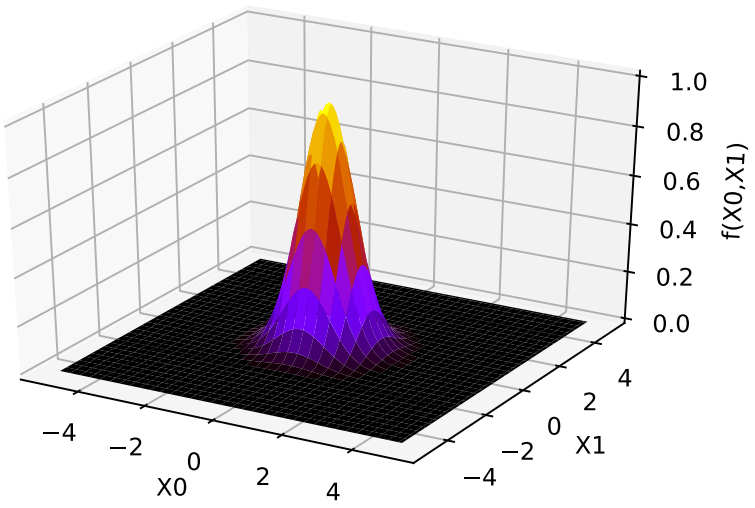


Figure 3.1: 3D plot of a single Gauss kernel with the parameters $\omega = 1$, $\gamma = 1$, $c_0 = 0$ and $c_1 = 0$

An approximate solution to the PDE in question is described as the superposition of N kernels as shown in equation 3.11.

$$u_{apx}(\mathbf{x}) = \sum_{i=0}^{N} \omega_i e^{-\gamma_i \mathbf{r}_i^2} \tag{3.11}$$

A solution is encoded as a vector of these $4 \cdot N$ parameters stacked together. This is shown in equation 3.12.

$$\vec{u_{apx}} = \left[\underbrace{\omega_0, \gamma_0, c_{00}, c_{01}}_{\text{kernel 0}}, \cdots \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}}_{\text{kernel i}}, \cdots \underbrace{\omega_N, \gamma_N, c_{N0}, c_{N1}}_{\text{kernel N}} \right]^T \tag{3.12}$$

To evaluate the residual of a PDE, the derivatives must be known. To that extend, the derivative of u_{apx} from equation 3.2 is calculated. The first order derivative with respect to x_0 is seen in equation 3.13. To solve the proposed testbed, also the second order derivatives are needed, as described in equation 3.14. Although the mixed term derivative is not used in this work, for the sake of completeness it is displayed in equation 3.15.

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_0} = -2 \sum_{i=0}^{N} \omega_i \gamma_i (x_0 - c_{i0}) e^{-\gamma_i \mathbf{r}_i^2} \tag{3.13}$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_0^2} = \sum_{i=0}^{N} \omega_i \gamma_i \left[4\gamma_i (x_0 - c_{i0})^2 - 2 \right] e^{-\gamma_i \mathbf{r}_i^2} \tag{3.14}$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_0 x_1} = 4 \sum_{i=0}^{N} \omega_i \gamma_i^2 (x_0 - c_{i0})(x_1 - c_{i1}) e^{-\gamma_i \mathbf{r}_i^2} \tag{3.15}$$

3.3.2 GSin Kernel

Additional to the Gauss kernel, a "GSin" kernel, further also abbreviated as Gauss Sine Kernel (GSK), is proposed. In essence, this is a multiplication of a GaK with a sine function, as seen in equation 3.16. It can be thought of a sine function, where its influence declines with the radius r. Both "sub-functions" are centred at the same $\mathbf{c}$. As shown in chapter 10.1, the universal approximation theorem for the GSin kernel holds true. The plot in figure 3.2 shows how the first half period is much larger than the second half. With the sine, two new parameters are introduced: f for the frequency of the sine wave and φ represents the phase shift.

approximation proof possible?

$$gsk(\mathbf{x}) = \omega e^{-\gamma \mathbf{r}^2} sin(f\mathbf{r}^2 - \varphi) \tag{3.16}$$

It is important to notice that, with the correct parameter choice, both traits - the sine and the e-function - can be retrieved. When $\gamma = 0$ the e-function evaluates to 1 and leaves the sine on its own. By setting $f = 0$ and $\varphi = \frac{\pi}{2}$, the sine vanishes and the e-function part is obtained. Thus, with this kernel it should be possible to approximate the PDEs 0A and 0B from the testbed arbitrarily close.

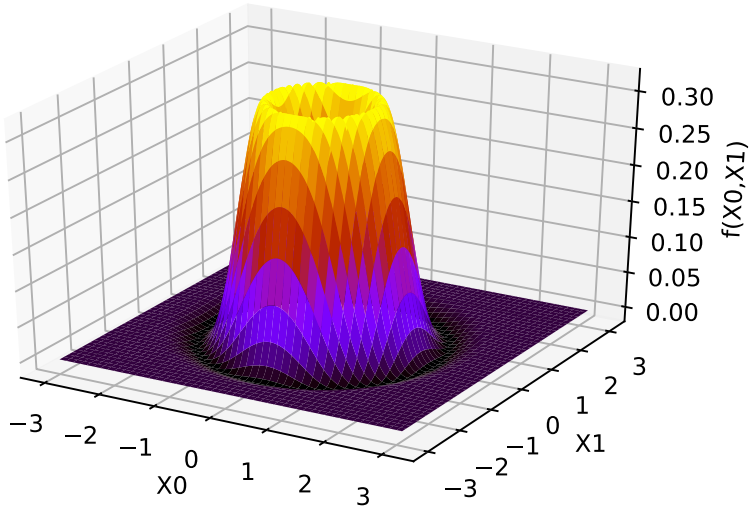


Figure 3.2: 3D plot of a single GSin kernel with the parameters $\omega = 1$, $\gamma = 1$, $c_0 = 0$, $c_1 = 0$, $f = 1$, $\varphi = 0$

Similar to the Gauss kernel, the approximate solution is described as the superposition of N GSin kernels, as seen in equation 3.17.

$$u_{apx}(\mathbf{x}) = \sum_{i=0}^{N} \omega_i e^{\gamma_i \mathbf{r}_i^2} sin(f_i \mathbf{r}_i^2 - \varphi_i) \tag{3.17}$$

Equation 3.18 shows the solution representation in stacked vector notation, including the two new parameters f and φ.

$$\vec{u_{apx}} = \Big[\underbrace{\omega_0, \gamma_0, c_{00}, c_{01}, f_0, \varphi_0}_{\text{kernel 0}}, \cdots \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}, f_i, \varphi_i}_{\text{kernel i}}, \cdots \underbrace{\omega_N, \gamma_N, c_{N0}, c_{N1}, f_N, \varphi_N}_{\text{kernel N}}\Big]^T \tag{3.18}$$

Again, the derivatives of the solution are needed, which are shown in the equations 3.19 through 3.21.

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_0} = \sum_{i=0}^{N} 2\omega_i(x_0 - c_{i0})e^{-\gamma_i \mathbf{r}_i^2}(\gamma_i sin(\varphi_i - f_i\mathbf{r}_i^2) + f_i cos(\varphi_i - f_i\mathbf{r}_i^2)) \tag{3.19}$$

$$\begin{aligned}\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_0^2} &= \sum_{i=0}^{N} 2\omega_i e^{-\gamma_i \mathbf{r}_i^2} \\ &[(2c_{i0}^2(f_i^2 - \gamma_i^2) + 4c_{i0}x(\gamma_i^2 - f_i^2) + 2f_i^2x^2 - 2\gamma_i^2x^2 + \gamma_i)sin(\varphi_i - f_i\mathbf{r}_i^2) + \\ &f_i(-4c_{i0}^2\gamma_i + 8c_{i0}\gamma_i x - 4\gamma_i x^2 + 1)cos(\varphi_i - f_i\mathbf{r}_i^2)]\end{aligned} \tag{3.20}$$

$$\begin{aligned}\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_0 x_1} &= \sum_{i=0}^{N} 4\omega_i(c_{i0} - x)(c_{i1} - y)e^{-\gamma_i \mathbf{r}_i^2} \\ &\left[(f_i^2 - \gamma_i^2)sin(\varphi_i - f_i\mathbf{r}_i^2) - 2f_i\gamma_i cos(\varphi_i - f_i\mathbf{r}_i^2)\right]\end{aligned} \tag{3.21}$$

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 underlying 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 papers are rather simple to approximate. Thus, more complicate equations are added to test the solvability over a wider variety of functions. The more complex equations are taken from the National Institute of Standards and Technology (NIST) website (Mitchell 2018) that provides benchmarking problems for FEM solvers 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 arbitrarily close, since the solution is a sum of 5 different GaK as represented in equation 3.10. The purpose of this PDE is to show that the algorithm converges towards the analytical solution.

PDE 0B: GSin Kernel (equation B.3) Similar to the PDE 0A, this problem is a sum of GSK, as seen in equation 3.16. 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 types.

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, Chaquet and Carmona 2012, Tsoulos and Lagaris 2006, Sobester, Nair, and Keane 2008, Panagant and Bureerat 2014)

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, Chaquet and Carmona 2012, Tsoulos and Lagaris 2006, Panagant and Bureerat 2014)

PDE 4: Sine Bump 2D (equation B.11) The sine bump occurs 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 tests have shown that this equation is one of the most difficult problems 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 GaK. 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 difficult 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. Additionally, 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 RBF. 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 implemented. A typical Gauss kernel as described in chapter 3.3.1 and a so called GSin kernel shown in chapter 3.3.2.

testbed in orange

The testbed holds the 11 differential equations used in all experiments. The testbed is abstracted 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 functions that take `FemPdeN` or `CiPdeN` objects and perform actions with them.

update implemented function description and class diagram in appendix

- `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 algorithm 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.
- `dict loadExpObjectFast(filename)`
  When solving a PDE with more function evaluation, the result file can become very large (>400Mb @ 10^6 #FE). For evaluating such large files, this function can be used. It does not load the generation data (meaning population, function value, F and CR). Since the standard json interpreter in python loads files to be serialized, a new interpreter is needed. To that extent, *bigjson* from Heino 2020 is used. This package accesses only those parts of a json file, that are actually needed.
- `bool drawGaussKernel(parameter, ggb)`
  Draws a solution approximated by Gauss kernels and with the specified parameters to a GeoGebra file. If the filename provided in the *ggb* argument does not exist, the function searches for a template and prints to a copy of that file.
- `bool drawGSinKernel(parameter, ggb)`
  This is similar to the `drawGaussKernel` method - but it takes parameters for a GSin kernel.
- `float calcRSME(solve_dict)`
  To compare the obtained results with previous work, the RMSE quality metric (as described in the chapter 4.3.3) must be computed. This is done from a single dictionary, as obtained by the functions `loadExpObject` or `loadExpObjectFast`.
- `None plotApprox3D(kernel, parameter, lD, uD, name=None)`
  The approximate solution of a PDE can be plotted over the domain with this function. Only square sized domains are accepted, as specified by the lower and the upper domain parameters `lD` and `uD`. If *name* is of type string, the plot is saved as this file.
- `string statsWilcoxon(a, b, alpha=0.05)`
  This function is a wrapper for the `scipy.stats.wilcoxon` (SciPy 2020). The default significance level is set to 0.05. A string is returned, that describes if

the mean/median of `a` is significantly smaller than the mean/median of `b`. The result is one of these strings:

  - sig. worse: the distributions are different; the mean and the median of `a` is larger than `b`
  - sig. better: the distributions are different; the mean and the median of `a` is smaller than `b`
  - unsig. worse: the distributions are similar; the mean and the median of `a` is larger than `b`
  - unsig. better: the distributions are similar; the mean and the median of `a` is smaller than `b`
  - unsig. undecided: the distributions are similar; the mean is larger, the median is smaller or vice versa

- `None plotFEDynamic(FEDynamic, name=None)`
  This method plots the function value dynamic of the population on a y-axis logarithmic plot. It can also cope with a varying population size. The plot can be saved with an optional argument.
- `None plotABSError3D(kernel, parameter, pdeName, lD, uD, name=None)`
  Similar to the `plotApprox3D` method, a 3D graph of the solution is plotted. Instead of the function value, the absolute error is shown. The error is calculated by $E_{abs} = |u_{apx}(x, y) - u_{ext}(x, y)| \, \forall x, y \in \Omega$.

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 Solving Time

The solving time is measured within the `solve()` method of either class. The time module of the T. 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 interpret the results. To make for a fair comparison, the same machine with a similar work-load must be used. The 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 an object with multiple attributes about the current state of the process. Of special interest is the Virtual Memory Size (VMS) field. This includes the Resident Set Size (RSS), the memory that is currently held within the main memory, and the memory that is currently swapped out to the hard drive.

Without assuming anything about the inner workings of the process, the memory usage 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. To remove outliers, it is helpful to create and solve one testbed object before recording the experiment. This sets up the necessary references and then are mingled into the actual solving process. This holds true for both, the FEM and the CI solver.

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 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 has 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. This is called an aliasing error. An approximation can fit the points, without correctly representing the space between them. This is shown in the following figure 4.1.

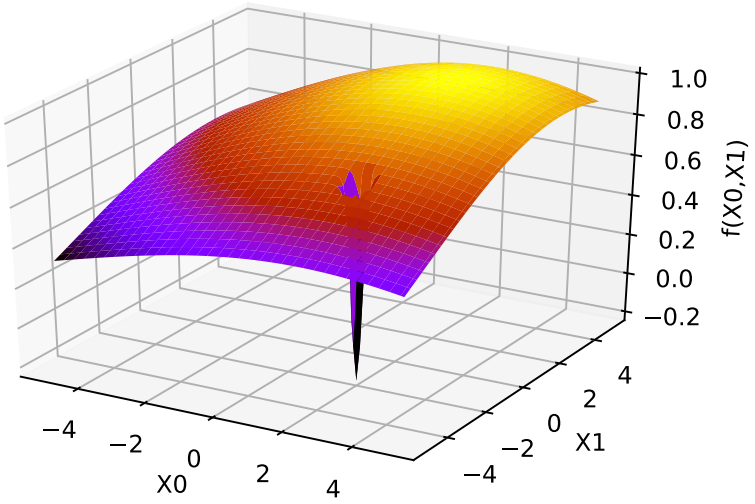


Figure 4.1: aliasing error: sharp inaccuracies in between collocation points

Finally, the FEM method doesn't use collocation points, so this quality measurement could not be calculated. A good quality measurement tool only uses 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. Referring 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 occur in the equations. This results in weak form as presented in equation 4.3, where the f is the RHS of the 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 interpret 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 4.2 and 4.3, respectively. In general, the solving time ranges from 2.5 to 5.0 seconds at about 50 to 80 Mbyte of 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.

The problem PDE 7 only needs around 2.5 seconds to be solved. 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.

Problem PDE	Distance
0A	$2.9670770746774782 \cdot 10^{-5}$
0B	$1.070854603999225 \cdot 10^{-5}$
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.

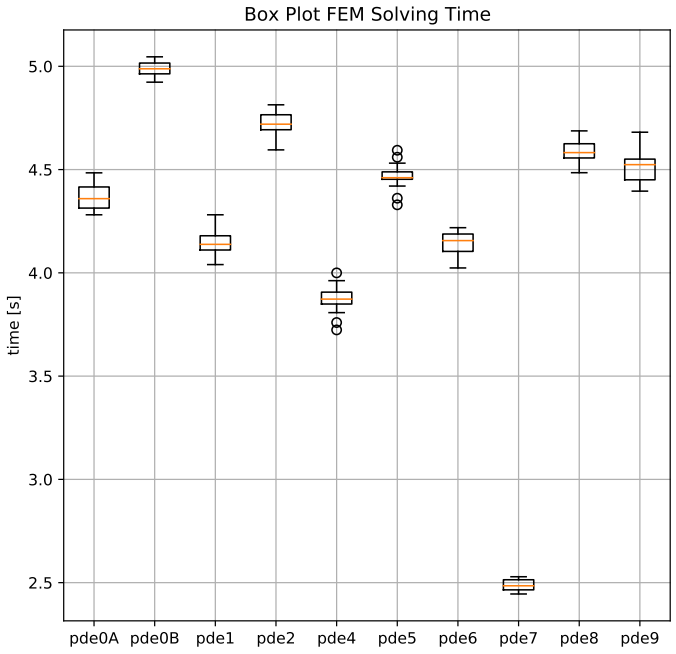


Figure 4.2: boxplot: time (in seconds) needed to solve the testbed PDE (without PDE3)

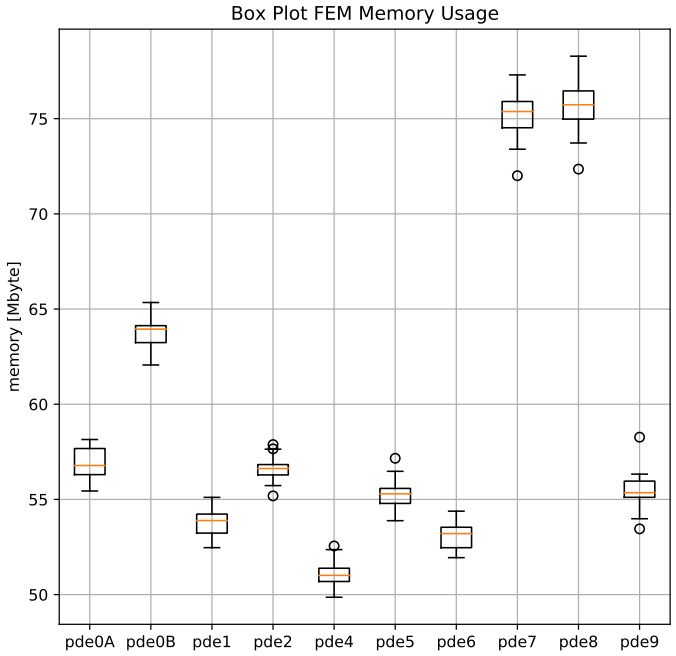


Figure 4.3: boxplot: memory (in Mbyte) needed to solve the testbed PDE (without PDE3)

The following images 4.4 and 4.5 show the boxplot 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: somewhere around 65.2 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.

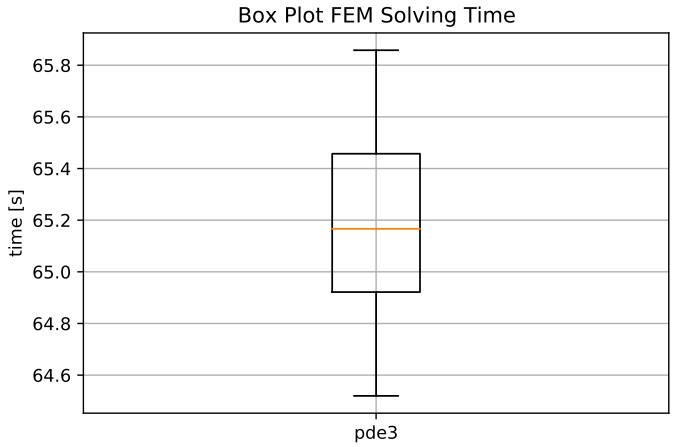


Figure 4.4: boxplot: time to solve testbed PDE3

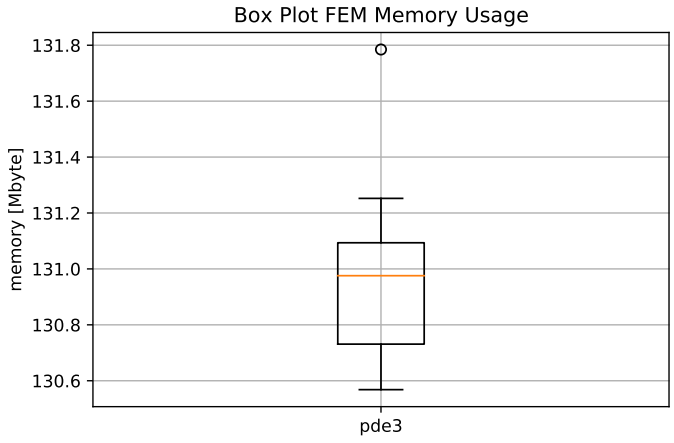


Figure 4.5: boxplot: memory to solve testbed PDE3

4.5 Default CI Parameter

Typically, heuristic optimisation algorithm use some parameters to tune the performance on specific testbed problems - for all further experiments this is the JADE algorithm form pseudocode A.1. Similarly, the reported CI solver has many parameters that could be adapted. However, adjusting every parameter in order to find the best combination is not an option, since that would take an extensive amount of computation time. Some parameters that might not have a great effect on the performance can be predefined. These values can be determined by preliminary tests or intuition, without any rigorous scientific justification. If not stated otherwise, the following parameters from table 4.2 are used in the subsequent experiments.

Parameter	Value	Chaquet
φ	100	300
κ	1	3
population size	$2 \cdot dim$	$\frac{3}{2}(4 + \lfloor 3 \cdot ln(dim) \rfloor)$
min error	0	-
p	0.3	-
c	0.5	-
replication	20	50
nb nc	40 81 121 = 11x11	100 equally spaced points over the domain
initialisation	$u_{apx}^{\rightarrow} \in \mathcal{N}(0,1)$	$\omega_i \in \mathcal{U}[-0.01, 0.01]$ $\gamma_i \in \mathcal{U}(0,1]$ $c_{ik} \in \mathcal{U}[2\Omega]$

Table 4.2: These predefined parameters are used for the following numerical experiments.

The parameters φ and κ are used in the fitness function (equation 3.7) for changing the relative importance of the boundary and the interior. Chaquet and Carmona 2019 take similar values. However, preliminary tests have shown that the current values perform slightly better on the present testbed. Figure 4.7 shows the values of ξ and φ. It further describes how the weighting factor emphasises the areas closer to the boundary.

The population size used by Chaquet and Carmona 2019 is based on the original formulation of the CMA-ES. However, they observed a better convergence when scaling the recommended population size by 3. Typically, DE uses larger population sizes. Mallipeddi and P. N. Suganthan 2008 describe an empirical study on choosing

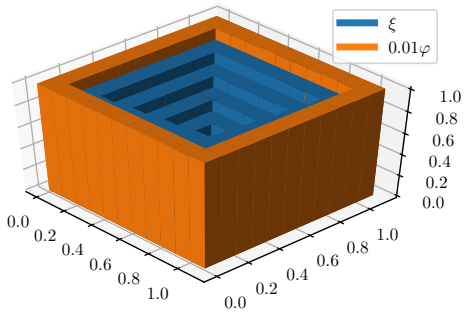


Figure 4.6: weighting factor φ and ξ on every collocation point

this parameter. They discuss the trade-off between premature convergence and computational effort. The results suggest that population sizes of $2 \cdot dim$ do get stuck in local optimums, but also converge faster. Since time is a critical resource, this population size is chosen.

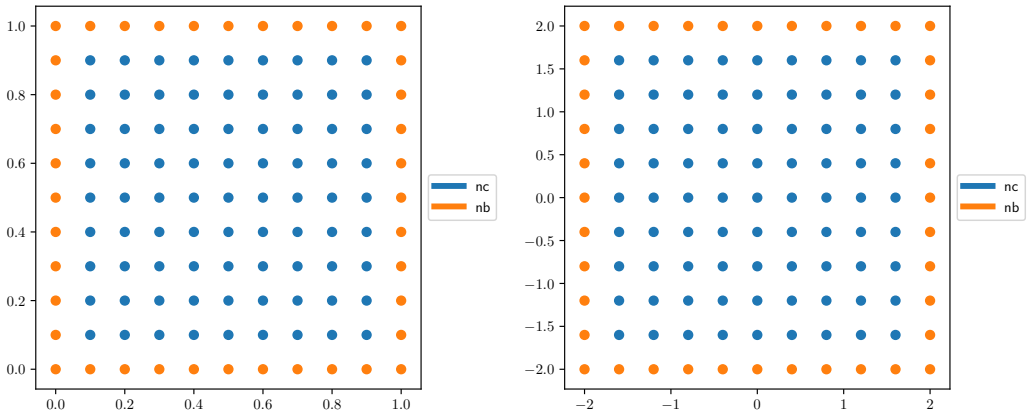
The termination condition for DE is either a maximum number of function evaluation, or a minimal function value to reach. Since the fitness function (equation 3.7) has its optimum at 0, this value is used as the termination condition. A helpful side-effect is that it ensures the same amount of function evaluation in every run, without early termination. This prevents outliers in the time and memory measurement.

The parameters p and c are specific to JADE (algorithm A.1). The mutation operator `mutationCurrentToPBest1` uses p to select the best individuals and c weights the parameter adaption mechanism. For all experiments these values are set at $p = 0.3$ and $c = 0.5$.

To account for the statistical influence, all experiments are restarted 20 times with independent initial guesses.

Chaquet and Carmona 2019 use 100 equally spaced collocation points over the domain to solve the PDE. Here, 121 points are created, as seen in figure 4.7.

The initialisation is done by a standard normal distribution. This means that every value in the $\vec{u_{apx}}$ vector is drawn from a normal distribution. On the contrary, Chaquet and Carmona 2019 initialise the values specifically tailored to each parameter of the kernel. This process assumes a priori information about the solution and thus should be dropped.



(a) points on $\Omega : x \in [0, 1]$ (b) points on $\Omega : x \in [-2, 2]$

Figure 4.7: collocation points used on the two domains of the testbed

4.6 Hardware Infrastructure

To increase the experimental throughput, two machines are used: a personal computer (machine 1) and a server on the cloud (machine 2). It is important that any time or memory comparisons must be performed between experiments on the same machine. In the experiments chapter, it is separately denoted, which comparisons are allowed. The following image 4.8, shows the properties both machines. It is important to note, that these information are only a reference point. This is probably not enough to actually recreate the exact time or memory data presented in this work.

Due to the Python incompatibilities, NGSolve can only be installed on machine 1. Thus, any time or memory comparisons between the CI solver and the FEM solver, must be done within machine 1.

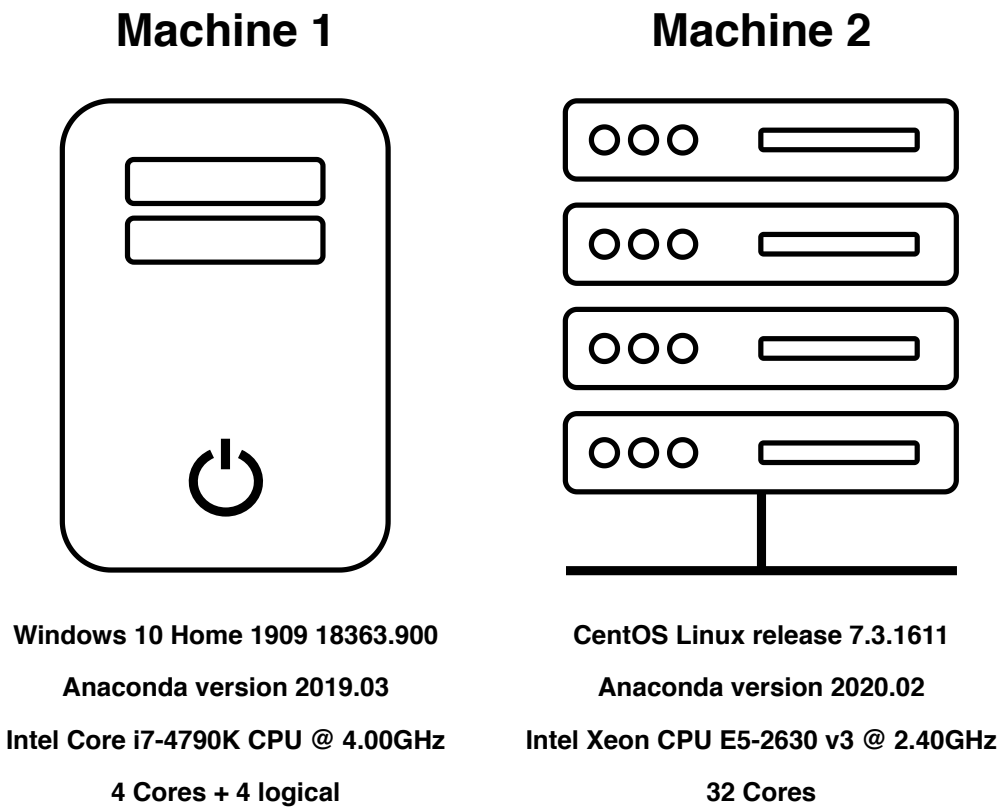


Figure 4.8: Comparison of the machines used for the experiments.

5 Experiment 0: Serial Memetic JADE

This chapter describes the results obtained by the most basic adaption of JADE for solving PDEs. The algorithm used here, builds on the concepts described in Chaquet and Carmona 2019. Aside from substituting some parameters, the only main difference is the usage of JADE instead of a CMA-ES.

5.1 Hypotheses

A memetic algorithm was first mentioned by Moscato 2000. In essence, it is a hybridisation of a population-based evolutionary algorithm and a deterministic direct or local search. The pseudocode 5.1 below shows the implementation of such a memetic JADE. At first, JADE performs the global search and places its population around the optimum. Than the DS exploits this area. The budget of #FE is split into two parts: JADE takes nearly all #FE and leaves $2 \cdot dim$ #FE for the DS.

Algorithm 5.1: memetic JADE Pseudocode

```
1 Function memeticJADE(X, funct, minErr, maxFE):
2     dim, popsize ← size(X)
3     p ← 0.3
4     c ← 0.5
5     pop, FE, F, CR ← JADE(X, p, c, funct, minErr, maxFE − 2dim )
6     bestIndex = argmin(FE)
7     bestSol = pop[bestIndex]
8     pop, FE = downhill simplex(funct, bestSol, minErr, 2dim)
9     return pop, FE, F, CR
```

This experiment provides first insight into the performance of the proposed algorithm. It tries to answer the question, if JADE is a suitable surrogate algorithm for a CMA-ES. Further, the memory usage and solving time is compared to the FEM results obtained in chapter 4.4.

5.2 Experiment Setup

The standard parameters from table 4.2 are taken. The memetic JADE is limited to either 10^4 #FE or 10^6 #FE. The experiment is done on two different machines. The first try with 10^4 #FE is run on the same machine ($\rightarrow$ machine 1) as the FEM experiment. This allows for a fair memory and solving time comparison. This comparison can not be performed with the data obtained by using 10^6 #FE ($\rightarrow$ machine 2). Further, 5 GaK are used which results in a dimension of 20 parameters. Thus, the population consists of 40 individuals.

5.3 Result

Table 5.1 shows a comparison on the common testbed functions PDE 2 and PDE 3 with numerical results obtained in other research papers. It is important to notice, that the parameters, used to obtain the results, might not coincide. Tsoulos and Lagaris 2006 also use these two PDE, but their paper did not provide any usable error metric.

Paper	parameter	RMSE PDE 2	RMSE PDE 3
Chaquet and Carmona 2019	4 kernel max #FE=10^6 50 replications	$(1.75 \pm 1.14)10^{-4}$	$(1.09 \pm 0.846)10^{-5}$
Chaquet and Carmona 2012	10 harmonics max #FE = $G \cdot \lambda = 1.2 \cdot 10^6$ 10 replications	$(6.37 \pm 0.733)10^{-3}$	$(5.90 \pm 0.799)10^{-3}$
Sobester, Nair, and Keane 2008	50 max tree lenght 12 generations 20 replications	$(6.9 \pm 8.3)10^{-4}$	–X–
Panagant and Bureerat 2014	unknowns: N/A #FE=$5 \cdot 10^5$ replications: N/A	7.25610^{-4}	9.48910^{-6}
serial memetic JADE	5 kernel max #FE = 10^6 20 replications	$(2.9798 \pm 1.5541)10^{-2}$	$(3.8225 \pm 1.9438)10^{-2}$

Table 5.1: This table compares the results obtained with the serial memetic JADE to the numerical results obtained by similar work in literature. The same metric must be used, thus the RMSE as defined in equation 4.1 is calculated.

The following table 5.2 lists the smallest L2 norm reached after 10^4 #FE and 10^6 #FE, respectively. A standard Wilcoxon test, as described in 4.2: post processing, is performed. Remarkable is that on PDE 5, more #FE result in a significantly worse solution.

#FE	10^4		10^6		
stat	mean	median	mean	median	Wilcoxon Test
PDE 0A	1.9415 ± 0.3321	1.8844	0.6596 ± 0.5510	0.9285	sig. better
PDE 0B	0.7137 ± 0.1979	0.6354	0.2027 ± 0.1302	0.1516	sig. better
PDE 1	0.1874 ± 0.0408	0.1938	0.0149 ± 0.0049	0.0151	sig. better
PDE 2	0.0890 ± 0.0334	0.0760	0.0257 ± 0.0140	0.0224	sig. better
PDE 3	0.2409 ± 0.1051	0.2309	0.0328 ± 0.0169	0.0285	sig. better
PDE 4	0.1102 ± 0.0367	0.0985	0.0378 ± 0.0083	0.0352	sig. better
PDE 5	0.6645 ± 0.1930	0.6263	1.1968 ± 0.0286	1.2056	sig. worse
PDE 6	1.9660 ± 1.3845	1.6540	0.4135 ± 1.2133	0.0018	sig. better
PDE 7	0.0457 ± 0.0137	0.0452	0.0221 ± 0.0019	0.0223	sig. better
PDE 8	0.2186 ± 0.0045	0.2191	0.2170 ± 0.0019	0.2175	unsig. better
PDE 9	0.0525 ± 0.0147	0.0516	0.0451 ± 0.0119	0.0459	unsig. better

Table 5.2: L2 norm reached with serial JADE at 10^4 #FE and 10^6 #FE

The following two images 5.1 and 5.2 show the time and memory usage for solving the testbed with 10^4 #FE. The images highlight the varying complexity of the testbed-PDEs and their corresponding fitness function. Although these results are not obtained for 10^6 #FE, they provide insight on how the solver scales with more #FE.

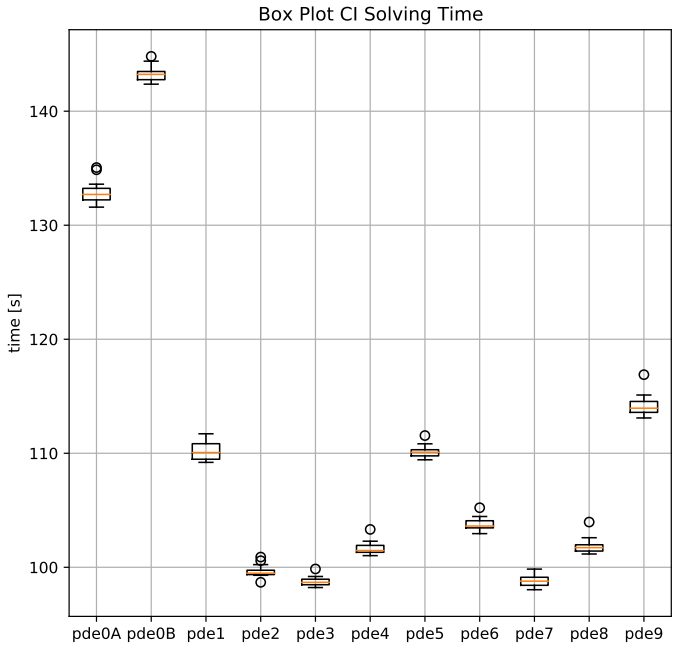


Figure 5.1: Solving time at 10^4 #FE.

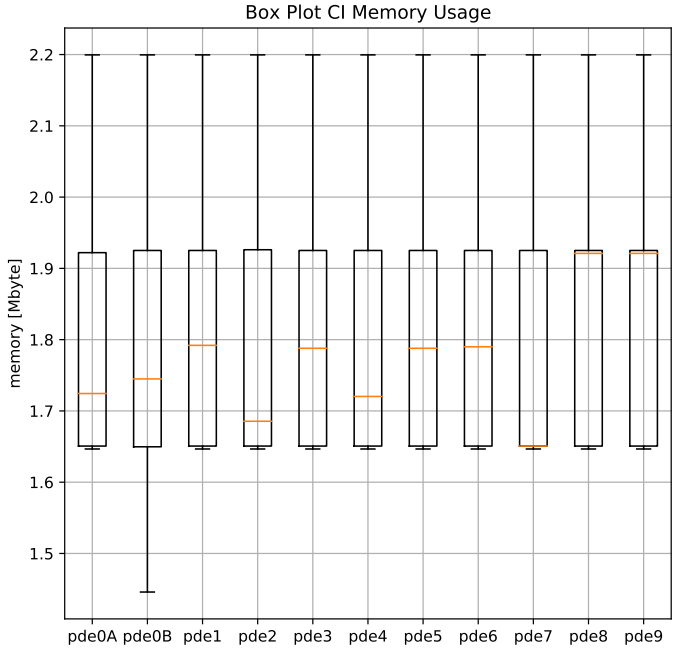


Figure 5.2: Memory usage with 10^4 #FE.

5.4 Discussion

The results presented in the chapter 5.3 above, are discussed on the following pages.

5.4.1 Comparison to Literature

The RMSE reached on the PDE 2 and 3, are clearly not as good as the results obtained in previous work, especially the results presented in Chaquet and Carmona 2019. There might be three reasons for that:

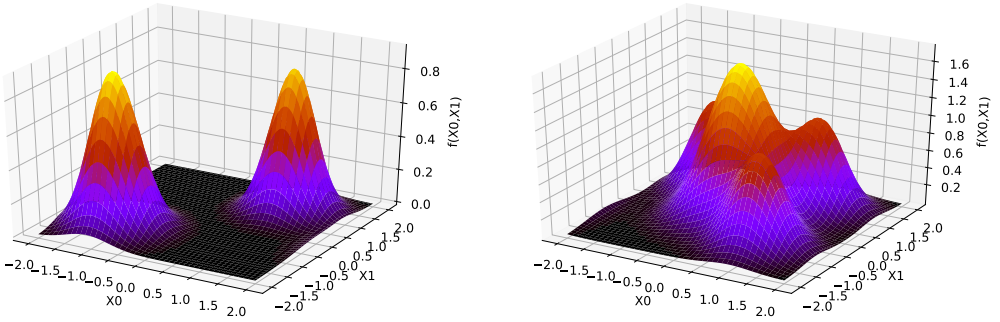
- As stated in the previous table 4.2, the penalty and weighting factor on the collocation points are different. Further, more kernels are used, resulting in a greater search dimension while simultaneously using a smaller population size. The combination of these parameters might influence the quality of the solution to the worse.
- Compared to Chaquet and Carmona 2019, the #FE-budget for the local DS search is smaller, thus the exploitation might not be as progressed.
- JADE might simply be not as well suited for the problem as a CMA-ES.

5.4.2 Time/Memory Usage

The results from table 5.2 show, that the CI solver can not nearly compete with the results obtained by the FEM solver from table 4.1. However, more interesting is the comparison of time and memory usage. In the current implementation, the population, the corresponding function values as well as the F and the CR history are recorded at every generation. This is not necessary for the performance of the algorithm, but helpful for evaluating the results. Therefore, the memory usage scales linearly with the number of function evaluations used $\mathcal{O}(n)$. In later implementation, this could be disregarded to further reduce the memory consumption. Every PDE takes about the same amount of memory to solve, somewhere between 1.5 and 2.2 Mbyte. The CI solver takes somewhere between 100 and 150 seconds (figure 5.1) to perform 10^4 #FE, depending on the PDE problem. This is always longer than the FEM solver, with the exception of PDE 3. More importantly, the CI solver uses less memory, on all problems(figure 5.1). An interesting observation is the distribution of the solving time within the testbed. The more commands a fitness function needs, the longer is its solving time. However, this does not necessarily correspond with the quality of the solutions. For example, PDE 0B takes the longest to evaluate but, compared to the other PDEs, it reaches a "fairly" good quality.

5.4.3 PDE 0A

The purpose of this PDE was to show that the solver would converge globally towards the analytical solution, if it could be represented by a finite number of kernels. Unfortunately, this can not be confirmed with the current implementation. While more function evaluation do tend to generate better results (as confirmed by the Wilcoxon test in table 5.2), it is not uncommon for the CI solver to result in different functions. A typical phenomenon is to miss out some of the five Gauss bumps, that build the solution. The comparison of two solutions with 10^4 #FE and 10^6 #FE in figure 5.3 shows this behaviour. Since the results do get a lot better from 10^4 to 10^6 #FE, it is possible that the results from 10^6 #FE can be improved. However, this is not tested due to the already extensive amount of computational effort.



(a) PDE 0A solution with 10^4 #FE:
L2 norm: 2.5074851055301095
FE value: 4.412933901588106
RMSE: 0.5699084204927098

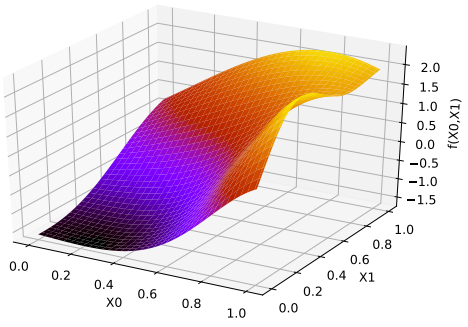
(b) PDE 0A solution with 10^6 #FE:
L2 norm: 0.9281243032816127
FE value: 1.3761997083143358
RMSE: 0.21103944621679302

Figure 5.3: Comparison of two typical PDE 0A solutions.

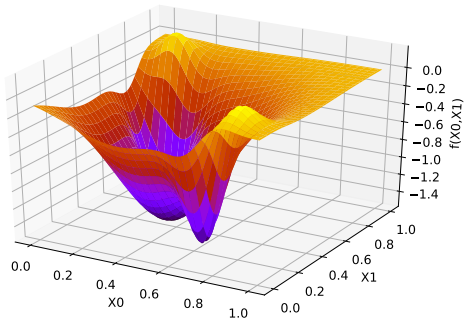
5.4.4 PDE 5

A fascinating property of PDE5 is observed: more function evaluation (from 10^4 to 10^6) result in a significantly worse solution quality. This is confirmed by a Wilcoxon test, as seen in the table 5.2. The corresponding histograms for the distribution of the L2 norm and the fitness value are depicted in figure 5.5. A comparison of the best solution after 10^4 #FE and the best solution after 10^6 #FE is shown in the figure 5.4. This can also be concluded from a visual perspective: the solution after 10^4 #FE describes the global structure better. It seems, that the correct description of the boundary points becomes less important when more #FE are allowed. This phenomenon can be explained by the structural difference between the fitness function and the L2 norm. The fitness of a candidate solution can only decrease or stay the same, thanks to the greedy selection used in JADE (line 15 in the pseudocode A.1). Because the L2 norm is not the property that gets optimised,

the quality of an individual at every generation is not necessarily monotonically decreasing. In the present case, this means that the fitness value does decrease, while the quality attribute gets worse, as confirmed by both histograms in figure 5.5. This indicates, that a fundamental different strategy might be needed to obtain better results.

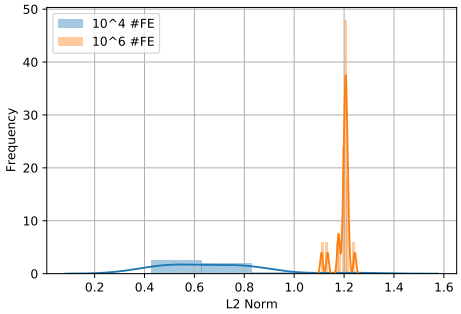


(a) PDE 5 solution with 10^4 #FE:
L2 norm: 0.4283407099978464
FE value: 2980.960124159222
RMSE: 0.4648836893027784

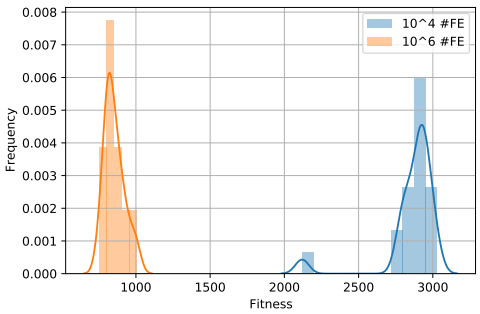


(b) PDE 5 solution with 10^6 #FE:
L2 norm: 1.1095483876690175
FE value: 1007.4408576392331
RMSE: 1.1456143814668234

Figure 5.4: Comparison of the best achieved quality (fitness function, L2 norm and RMSE) solution on PDE 5



(a) Histogram of the L2 norm reached on PDE 5.



(b) Histogram of the fitness value reached on PDE 5.

Figure 5.5: Histograms of the results reached with 10^4 #FE and 10^6 #FE on PDE 5. The paradox observation: the fitness value with 10^6 #FE is smaller, but the L2 norm tends to be larger.

6 Experiment 1: Parallel Population JADE

With the serial algorithm from the chapter above, evaluating a PDE takes way to long to be practically relevant (beyond $14.5 \cdot 10^3$ seconds at 10^6 #FE). To reduce the solving time and speed up all further experiments, a parallel JADE is implemented. This chapter describes the benefits and drawbacks the new algorithm.

6.1 Hypotheses

The actual parallel JADE algorithm is shown in the appendix E. The pseudocode E.1 is not much different from the JADE algorithm A.1. The main differences are laid out in the lines 11 to 17. In essence, the mutation, the crossover, the parameter adaption and the fitness evaluation are done in parallel for each individual in the population. The parallelism is based on the python module multiprocessing (M. Python Standard Library 2020), in particular the `multiprocessing.Pool.apply_async(func, *args)` method. This method takes a function, in this case an adapted version of the for-loop from line 11 to 17, and automatically maps it to a number of preallocated processes. After this is done for every individual in the population, the processes are synchronised and the results get returned to the main process. Therein lies the actual difference to the serial JADE. In the parallel algorithm new individuals are only available after each generation, whereas with the serial algorithm, the individuals are constantly updated, already before the next generation starts. This means, that in the parallel algorithm, information is withheld until the next generation - the same information can spread faster in the serial algorithm.

For the sake of completeness, the memetic parallel JADE pseudocode is shown in the algorithm 6.1 below. The only difference compared to the simple memetic JADE 5.1 is the usage of the pJADE in line 5.

This chapter tries to answer the question, if the explained distinctions actually influences the results, to the better or the worse. Further, the time-benefits of this strategy are examined and assessed.

Algorithm 6.1: memetic parallel JADE Pseudocode

```
1 Function memeticJADE(X, funct, minErr, maxFE):
2     dim, popsize ← size(X)
3     p ← 0.3
4     c ← 0.5
5     pop, FE, F, CR ← pJADE(X, p, c, funct, minErr, maxFE − 2dim )
6     bestIndex = argmin(FE)
7     bestSol = pop[bestIndex]
8     pop, FE = downhill simplex(funct, bestSol, minErr, 2dim)
9     return pop, FE, F, CR
```

6.2 Experiment Setup

Similar to the experiment 0 before, the standard parameters from table 4.2 are used. Again, two different machines with either 10^4 #FE ($\rightarrow$ machine 1) or 10^6 #FE ($\rightarrow$ machine 2) were used, where the time comparison is only allowed on machine 1. Also, 5 GaK are used, meaning the problem dimension is 20 and the population size is 40.

Additionally, a new parameter is introduced: the number of allocated parallel processes. It is recommended to set this parameter to the number of available processors, but this also depends on the general workload of the machine. When using too many processes, the system might be overwhelmed, effectively slowing down the task and resulting in an even longer solving time. On machine 1, 5 processes are implemented, while machine 2 is capable of handling 30 processes. With other machines, it might be possible to introduce even more processes, potentially increasing the speed-up even further.

6.3 Result

The two images 6.1 and 6.2 display a boxplot of the time- and memory consumption respectively. To ensure comparability with the results obtained for the serial JADE, these images represent the results at 10^4 #FE on machine 1.

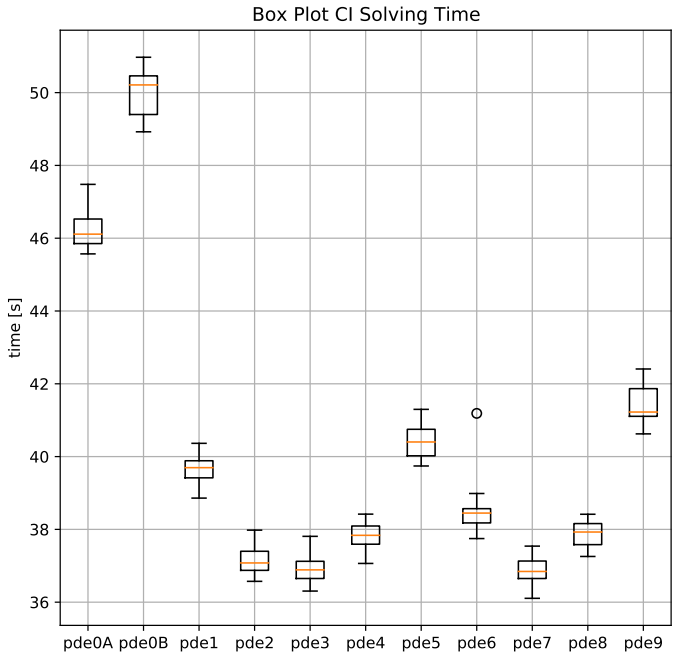


Figure 6.1: Solving time of the parallel memetic JADE algorithm at 10^4 #FE

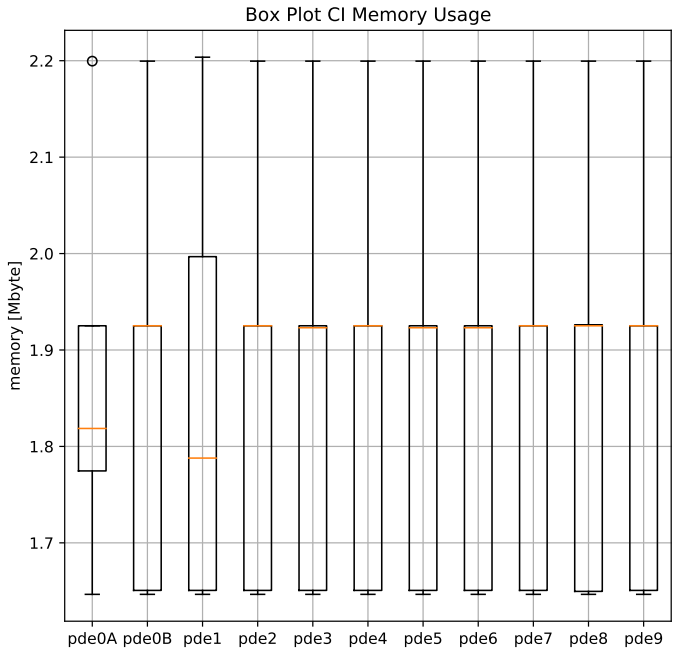


Figure 6.2: Memory consumption of the parallel memetic JADE at 10^4 #FE

Table 6.1 shows the mean and the median L2 norm reached on the testbed by both

the serial and the parallel algorithm. This is completed with a Wilcoxon test, to identify the difference in the performance. The term "undecided" in this context means, that the mean and the median paint different pictures: one of those values is smaller while the other one is larger. Most notably, on PDE 4, the parallel JADE performs significantly worse than the serial JADE.

algorithm	serial JADE		parallel JADE		
stat	mean	median	mean	median	Wilcoxon Test
PDE 0A	0.6596 ± 0.5510	0.9285	0.6939 ± 0.6635	0.9243	unsig. undecided
PDE 0B	0.2027 ± 0.1302	0.1516	0.2809 ± 0.3071	0.2035	unsig. worse
PDE 1	0.0149 ± 0.0049	0.0151	0.0239 ± 0.0467	0.0146	unsig. undecided
PDE 2	0.0257 ± 0.0140	0.0224	0.0300 ± 0.0157	0.0255	unsig. worse
PDE 3	0.0328 ± 0.0169	0.0285	0.0371 ± 0.0206	0.0295	unsig. worse
PDE 4	0.0378 ± 0.0083	0.0352	0.0505 ± 0.0121	0.0481	sig. worse
PDE 5	1.1968 ± 0.0286	1.2056	1.2030 ± 0.0465	1.2053	unsig. undecided
PDE 6	0.4135 ± 1.2133	0.0018	0.5814 ± 1.3550	0.0000	unsig. undecided
PDE 7	0.0221 ± 0.0019	0.0223	0.0228 ± 0.0025	0.0226	unsig. worse
PDE 8	0.2170 ± 0.0019	0.2175	0.2167 ± 0.0017	0.2169	unsig. better
PDE 9	0.0451 ± 0.0119	0.0459	0.0426 ± 0.0115	0.0463	unsig. undecided

Table 6.1: This tabular compares the results obtained by the parallel and the serial JADE. All results are obtained at 10^6 #FE. The serial data is the same as already presented in table 5.2.

6.4 Discussion

The results from above are examine in this chapter. The time and memory consumption as well as the significant worse results on PDE 4 are discussed.

6.4.1 Memory Utilization

As the boxplot 6.2 shows, the memory consumption is roughly at the same level, somewhere between 1.65 and 2.2 Mbyte. Only a few outliers are added, while others are removed. This is expected, since memory-wise no integral changes have been done in the implementation.

6.4.2 Solving Time

The main purpose of the parallelisation is to cut down the solving time and accelerate all further experiments. To that extent, the population is evaluated in parallel. As the solving time boxplot in figure 6.1 shows, this goal was achieved. The solving time ranks now between 36 and 52 seconds at 10^4 #FE. This is unquestionably faster than the solving time of the serial algorithm. Yet, these times can not compete with the solving time achieved by the FEM package NGSolve.

The image 6.3 shows the calculated mean speed-up (sp) over 20 replications, that is accomplish by substituting the serial JADE with the parallel version. The 95% confidence interval is shown in yellow.

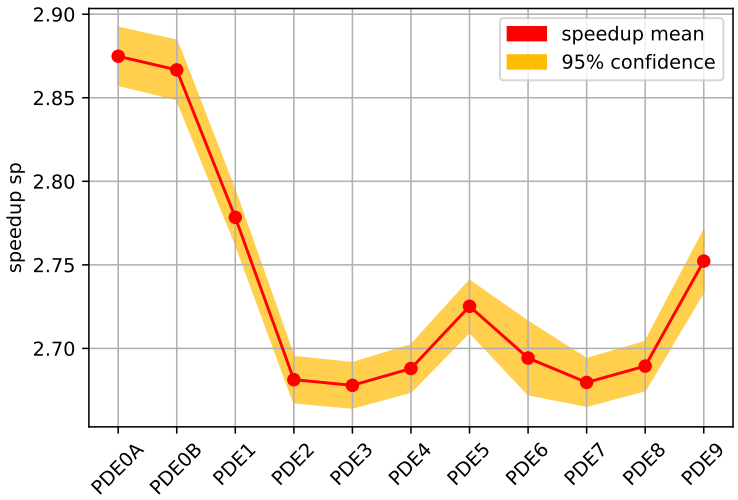


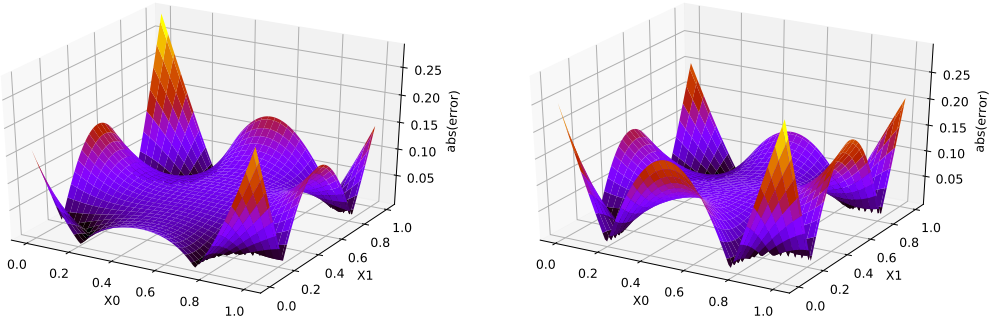
Figure 6.3: Mean speed-up of parallel JADE with 95% confidence interval.

As the plot 6.3 also shows, speed-up varies by testbed function. This can be directly linked to the "size" of the fitness function, especially the RHS of the PDE. The longer the fitness function is to evaluate, the greater is the speed-up that can be achieved ($time(fit\ evaln) \uparrow\rightarrow sp \uparrow$). The same correlation is true for the size of the population: the larger a population is, the better is the speed-up ($pop\ size \uparrow\rightarrow sp \uparrow$). Larger fitness functions are needed in on the testbed PDEs 0A, 0B, 1, 5, and 9. Looking at the definition of these equations in the appendix B, it is notable that the RHS are constructed of more terms compared to the other equations.

6.4.3 PDE 4

The table 6.1 shows, that the L2 norm achieved on PDE 4 is significantly worse when using the parallel JADE. This is not expected and hinders the justification of substituting the serial JADE with the parallel version. It is of interest to confirm the hypothesis and allow the usage of the parallel JADE in all further experiments.

The absolute error (calculated by $E_{abs} = |u_{apx}(x_0, x_1) - u_{ext}(x_0, x_1)|$) of the worst solution generated either by the serial or the parallel algorithm can be seen in the plot from figure 6.4. The plot suggests, that the underlying structure of the error is similar in both cases. Although, the L2 norm, the function value and the RMSE are lower with the serial JADE, the absolute error values from the plot are similar. Only on the boundary $x_1 = 0, x_0 \in [0, 1]$, the serial JADE generates visually smaller error.



(a) Absolute error, serial JADE
L2 norm: 0.06085492512786148
FE value: 0.4632201522148409
RMSE: 0.07521621060905691

(b) Absolute error, parallel JADE
L2 norm: 0.0732660037376775
FE value: 0.6976349252814954
RMSE: 0.08948976068654439

Figure 6.4: Comparison of the absolute error of the worst solution on PDE 4 by a parallel and a serial memetic JADE at 10^6 #FE

In order to make the results easier to interpret, the histogram from figure 6.5, shows the L2 norm data from the table 6.1. The best solution of both algorithms are at a similar quality level. The parallel sample introduces more results with a worse quality. The absolute numerical range of the quality is visualised in the boxplot 6.6. The Wilcoxon test asserts, that the two shown distributions are significantly different. While this can be confirmed visually, it can also be seen that the differences are only marginal.

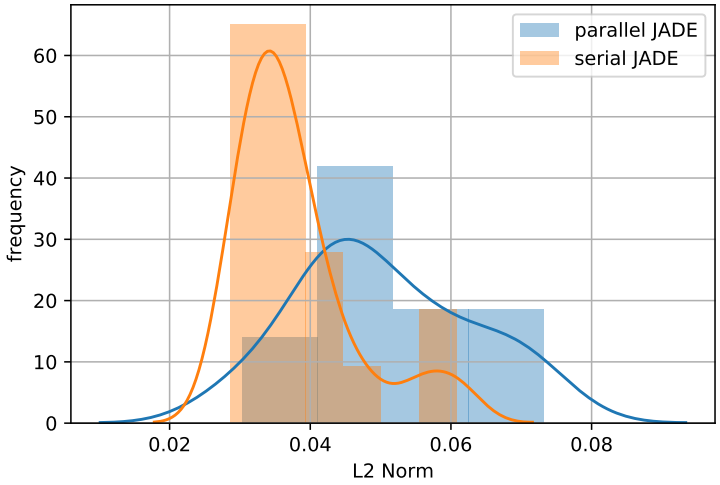


Figure 6.5: Histogram of the L2 norm data obtained by the serial and the parallel JADE on PDE 4 from table 6.1.

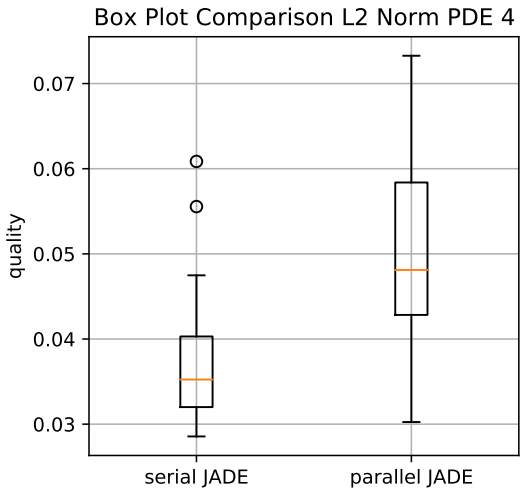


Figure 6.6: Boxplot of the L2 norm data from table 6.1 on PDE 4.

There is no definite certainty, but it is possible that the obtained data includes several unfortunate outliers. Increasing the number of independent replications could provide further clues to that matter. Considering, that quality-wise the parallel JADE makes no significant difference on 10 of 11 tested PDEs, and the compelling speed-up on all test problems, the parallel JADE is kept for the further experiments. At the very worst, the algorithm is faster by a factor of 2.5 and results in a minor decrease of the solution quality. This trade-off is acceptable.

7 Experiment 2: Adaptive Number of Kernels

Although, the parallel algorithm is effectively faster, the quality of the achieved solution is still not good enough. A common inaccuracy, especially with the testbed PDE 0A is, that the not all Gauss "bumps" are represented in the approximation, as described in chapter 5.4.3. A possible method to compensate that could be to adapte the number of kernels along the solving process, instead of arbitrarily using 5 kernels.

7.1 Hypotheses

The idea that this chapter tests is an adaptive scheme for the number of kernels used, which is directly linked to the search dimensionality of the optimisation problem.

This new concept requires a convergence based halting criterion in the JADE algorithm, which is not included in the original algorithm. The pseudocode F.1 in the appendix F is extended by a so called state detector. Ideally, the lines 37 to 40 should stop the overall optimisation loop as soon as the algorithm has converged and before the function evaluation budget is exceeded. Generally, this is done by checking if the function value has not changed for a certain amount of generations. The state detector introduces a new parameters: the Delay Time (dT), which represents the number of generations that the best function value must stay the same. It can also be thought of a buffer-time that allows the DE parameters F and CR to self-adapt. Further, the minError parameter has a new purpose: this is the minimal difference that the value is allowed to change over dT generations.

The new paJADE is wrapped into the memetic framework, already used in the chapters before. The new algorithm 7.1 is shown below, which also describes the kernel number adaption. Whenever a JADE/DS cycle is concluded, it is checked if the resulting function value is better than the previous best function value. If the new result is better (line 17), the problem dimensionality is increased by one kernel (line 19). Since the dimension is greater, the population size must be adapted by the same factor (line 20). If the function value could not be surpassed, a restart around

the previous best value solution is performed (line 27). Thus, the dimension can increase one kernel at a time, but it can only fall back one kernel.

Algorithm 7.1: memetic parallel JADE with adaptive kernels pseudocode

```
1  Function memeticJADE(X, funct, minErr, maxFE):
2      dim, popsize, kernelsize ← size(X)
3      p ← 0.3
4      c ← 0.5
5      delayTime ← 100
6      fecounter ← 0
7      bestFE ← inf
8      bestPop ← ∅
9      popFactor ← psize/dim
10     while fecounter < maxFE do
11         pop, FE, F, CR ← paJADE(X, p, c, dT, funct, minErr,
             maxFE − 2dim )
12         fecounter ← fecounter + size(pop) · psize
13         bestIndex ← argmin(FE)
14         bestSol ← pop[bestIndex]
15         pop, FE ← ds(funct, bestSol, minErr, 2dim)
16         fecounter ← fecounter + 2dim
17         if min(FE) < bestFE then
18             // increase dimension
19             X ← hstack(pop_g, N(psize, kernelsize))
20             X ← vstack(X, N((popFactor ∗ dim) − psize, dim))
21             bestFE ← min(FE)
22             bestPop ← pop_g
23             psize, dim ← size(X)
24         end
25         else
26             // reduce dimension
27             X ← bestPop + N
28             bestFE ← min(FE)
29             bestPop ← pop_g
30             psize, dim getssize(X)
31         end
32     end
33     return pop, FE, F, CR
```

This chapter tries to answer the question if this strategy is an effective method to increase the quality of obtained solutions. Also, the time and memory aspects are investigated.

7.2 Experiment Setup

7.3 Result

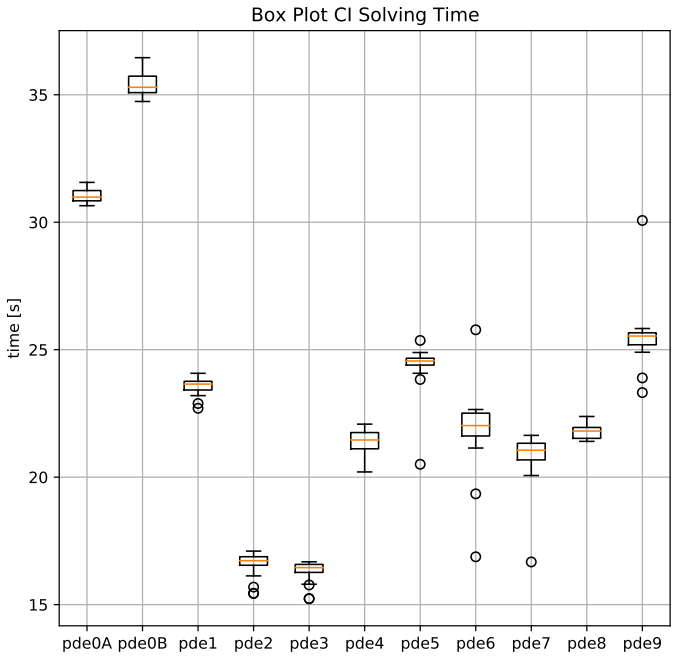


Figure 7.1: Solving time of the paJADE algorithm at 10^4 #FE

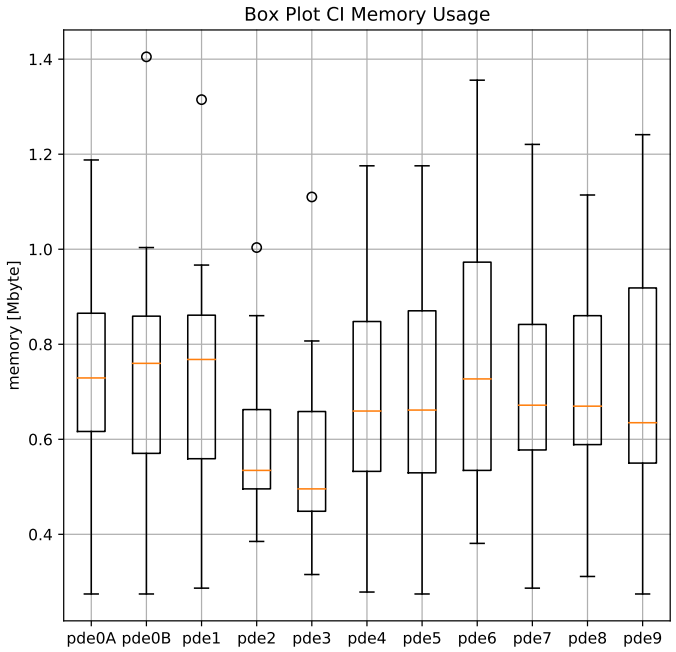


Figure 7.2: Memory consumption of the paJADE at 10^4 #FE

algorithm	parallel JADE		adaptive JADE		
stat	mean	median	mean	median	Wilcoxon Test
PDE 0A	0.6939 ± 0.6635	0.9243	9.6944E-16 ± 1.4867E-16	9.2559E-16	sig. better
PDE 0B	0.2809 ± 0.3071	0.2035	0.2380 ± 0.0572	0.2607	unsig. undecided
PDE 1	0.0239 ± 0.0467	0.0146	0.0116 ± 0.0061	0.0084	unsig. better
PDE 2	0.0300 ± 0.0157	0.0255	0.0735 ± 0.0358	0.1034	sig. worse
PDE 3	0.0371 ± 0.0206	0.0295	0.1731 ± 0.0395	0.1822	sig. worse
PDE 4	0.0505 ± 0.0121	0.0481	0.0707 ± 0.0053	0.0720	sig. worse
PDE 5	1.2030 ± 0.0465	1.2053	122.6312 ± 372.5676	1.1643	unsig. undecided
PDE 6	0.5814 ± 1.3550	0.0000	0.4428 ± 1.0980	0.0000	unsig. undecided
PDE 7	0.0228 ± 0.0025	0.0226	0.0513 ± 0.0442	0.0231	sig. worse
PDE 8	0.2167 ± 0.0017	0.2169	0.2144 ± 0.0044	0.2128	unsig. better
PDE 9	0.0426 ± 0.0115	0.0463	0.0483 ± 0.0149	0.0468	unsig. worse

Table 7.1: Comparison of the achieved L2 norm by the pJADE and the paJADE at 10^6 #FE.

7.4 Discussion

8 Experiment 3: GSin Kernel

8.1 Hypotheses

8.2 Experiment Setup

8.3 Result

8.4 Discussion

9 Limitations

9.1 Testbed

Only PDEs with Laplace operator

9.2 Fulfilment of Boundary Condition

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

10 Theoretical Observations

10.1 Universial Approximation Theorem

10.2 Multimodality and Symmetry

11 Conclusion

12 Further Work

technology:
implemented in python -> precompile fitness function

theoretical:
alaising error, tchebyshev collocation points

algorithmic:

13 Summary

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. 18, 20, 23, 28, 35, 36
CMA-ES Covariance Matrix Adaption Evolution Strategy. 5, 10, 29, 31, 35

DE differential evolution. 1, 2, 6–8, 10, 29, 30
DOF Degree of Freedom. 25, 27
DS Downhill-Simplex. 5, 7, 10, 31, 35

ES Evolution Strategy. 6, 10

FE Forward-Euler. 1
FEM finite element method. X, 1, 2, 12, 18, 20, 23, 24, 26, 27, 31, 32, 35

GA genetic algorithm. 7, 10
GaK Gauss Kernel. 14, 15, 18, 19, 32
GE grammatical evolution. 7, 10
GP genetic programming. 5–7, 10
GSK Gauss Sine Kernel. 15, 18

#FE Number of Function Evaluation. VIII, X, 31–38
HS Harmony Search. 6, 10

JSON JavaScript Object Notation. 21

ODE ordinary differential equation. 7

PDE partial differential equation. VIII, 1, 2, 7, 11, 14, 15, 18–21, 23, 25–28, 30–33, 35–38, 40

PSO particle swarm optimisation. 5, 6, 10

RBF radial basis function. 4, 13, 20

RHS right hand side. 2, 25

RMSE Root-Mean-Square Error. VIII, 21, 23, 35, 37

RSS Resident Set Size. 23

UML Unified Modeling Language. IX, 20, 76

VMS Virtual Memory Size. 23

WCA Water Cycle Algorithm. 6, 10

WRM weighted residual method. 4, 5, 11, 12

Bibliography

Babaei, M. (July 2013): "A general approach to approximate solutions of nonlinear differential equations using particle swarm optimization". en. 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).

Broomhead, David S. and Lowe, David (1988): "Multivariable Functional Interpolation and Adaptive Networks". In: *Complex Systems* (cit. on p. 14).

Chaquet, Jose M. and Carmona, Enrique J. (Sept. 2012): "Solving differential equations with Fourier series and Evolution Strategies". en. 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, 19, 34).

Chaquet, Jose M. and Carmona, Enrique J. (Mar. 2019): "Using Covariance Matrix Adaptation Evolution Strategies for solving different types of differential equations". en. 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, 12–14, 18, 19, 23, 24, 29, 30, 33, 34, 37).

Eskandar, Hadi et al. (Nov. 2012): "Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems". en. 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. 2019): "Differential evolution based computation intelligence solver for elliptic partial differential equations". en. 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. 2001): "A New Heuristic Optimization Algorithm: Harmony Search". en. 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).

Hansen, Nikolaus (2006): "The CMA Evolution Strategy: A Comparing Review". en. 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).

Heino, Henrik (May 2020): *henu/bigjson*. original-date: 2016-08-03T01:33:27Z. URL: https://github.com/henu/bigjson (visited on 06/13/2020) (cit. on p. 21).

Holland, John H. (July 1962): "Outline for a Logical Theory of Adaptive Systems". In: *Journal of the ACM* 9.3, pp. 297–314. ISSN: 0004-5411. DOI: 10.1145/321127.321128. URL: https://doi.org/10.1145/321127.321128 (visited on 05/29/2020) (cit. on p. 7).

Howard, Daniel; Brezulianu, Adrian, and Kolibal, Joseph (Jan. 2011): "Genetic programming of the stochastic interpolation framework: convection–diffusion equation". en. 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 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*. 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 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).

Mallipeddi, R. and Suganthan, P. N. (June 2008): "Empirical study on the effect of population size on Differential evolution Algorithm". In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. ISSN: 1941-0026, pp. 3663–3670. DOI: 10.1109/CEC.2008.4631294 (cit. on p. 29).

Mastorakis, Nikos E (Aug. 2006): "Unstable Ordinary Differential Equations: Solution via Genetic Algorithms and the method of Nelder-Mead". en. In: 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. 18, 19).

Moscato, Pablo (Oct. 2000): "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms". In: *Caltech Concurrent Computation Program* (cit. on p. 33).

Nelder, J. A. and Mead, R. (Jan. 1965): "A Simplex Method for Function Minimization". en. 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". en. In: *Mathematical Problems in Engineering* 2014.2014. Publisher: Hindawi, e747490. ISSN: 1024-123X. 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, 18, 19, 34).

Python Standard Library, Multiprocessing (July 2020): *multiprocessing — Process-based parallelism — Python 3.8.4rc1 documentation*. en. Documentation. URL: https://docs.python.org/3/library/multiprocessing.html (visited on 07/02/2020) (cit. on p. 40).

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. 22).

Rajeev S. and Krishnamoorthy C. S. (May 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).

Rechenberg, I. (1978): "Evolutionsstrategien". de. In: Schneider, Berthold and Ranft, Ulrich (Eds.): *Simulationsmethoden in der Medizin und Biologie*. Medizinische Informatik und Statistik. Berlin, Heidelberg: Springer, pp. 83–114. ISBN: 978-3-642-81283-5. DOI: 10.1007/978-3-642-81283-5_8 (cit. on p. 6).

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. 23).

Ryan, Conor; Collins, JJ, and Neill, Michael O. (1998): "Grammatical evolution: Evolving programs for an arbitrary language". en. 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 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. 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, 25).

Schwefel, Hans-Paul (1977): "Evolutionsstrategien für die numerische Optimierung". de. In: Schwefel, Hans-Paul (Ed.): *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Interdisciplinary Systems Research / Interdisziplinäre Systemforschung. Basel: Birkhäuser, pp. 123–176. ISBN: 978-3-0348-5927-1. DOI: 10.1007/978-3-0348-5927-1_5. URL: https://doi.org/10.1007/978-3-0348-5927-1_5 (visited on 05/29/2020) (cit. on p. 6).

SciPy, Reference Guide (June 2020): *scipy.stats.wilcoxon — SciPy v1.4.1 Reference Guide*. Documentation. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html#scipy.stats.wilcoxon (visited on 06/20/2020) (cit. on p. 21).

Shen, Jie; Tang, Tao, and Wang, Li-Lian (2011): *Spectral Methods: Algorithms, Analysis and Applications*. en. Springer Series in Computational Mathematics. Berlin Heidelberg: Springer-Verlag. ISBN: 978-3-540-71040-0. DOI: 10.1007/978-3-540-71041-7. URL: https://www.springer.com/de/book/9783540710400 (visited on 04/24/2020) (cit. on pp. 11, 12).

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, 19, 34).

Storn, Rainer and Price, Kenneth (Dec. 1997): "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". en. 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. 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*. ISSN: 1941-0026, pp. 71–78. DOI: 10.1109/CEC.2013.6557555 (cit. on p. 9).

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)*. ISSN: 1941-0026, pp. 1658–1665. DOI: 10.1109/CEC.2014.6900380 (cit. on p. 9).

Tsoulos, I. G. and Lagaris, I. E. (Mar. 2006): "Solving differential equations with genetic programming". en. 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, 18, 19, 34).

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

A.2 SHADE Pseudocode

Algorithm A.2: SHADE Pseudocode

Function SHADE($\mathbf{x}_{G=0}$, p, H, $function$, $minError$, $maxFE$):
  $M_{CR} \leftarrow 0.5$; $M_F \leftarrow 0.5$; $A \leftarrow \emptyset$; $G \leftarrow 0$; $k \leftarrow 1$
  $fValue_{G=0} \leftarrow function(\mathbf{x}_{G=0})$
  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, $function$, $minError$, $maxFE$)**:**
  $M_{CR} \leftarrow 0.5$; $M_F \leftarrow 0.5$; $A \leftarrow \emptyset$; $G \leftarrow 0$; $k \leftarrow 1$
  $fValue_{G=0} \leftarrow function(\mathbf{x}_{G=0})$; $NG_{init} \leftarrow size(\mathbf{x}_{G=0})$; $NG_{min} = \lceil 1/p \rceil$
  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:

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

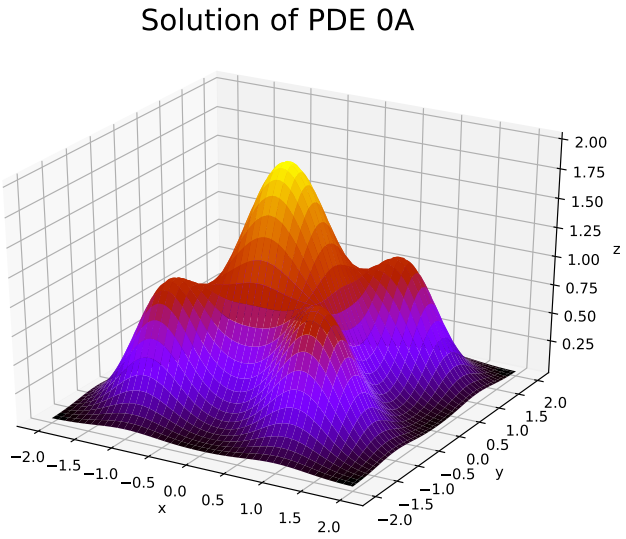


Figure B.1: PDE 0A Gauss Kernel solution plot

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:

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

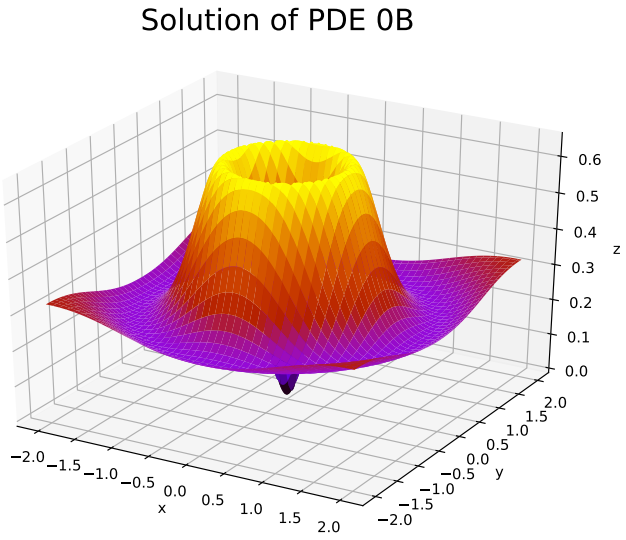


Figure B.2: PDE 0B Gsin Kernel solution plot

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:

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

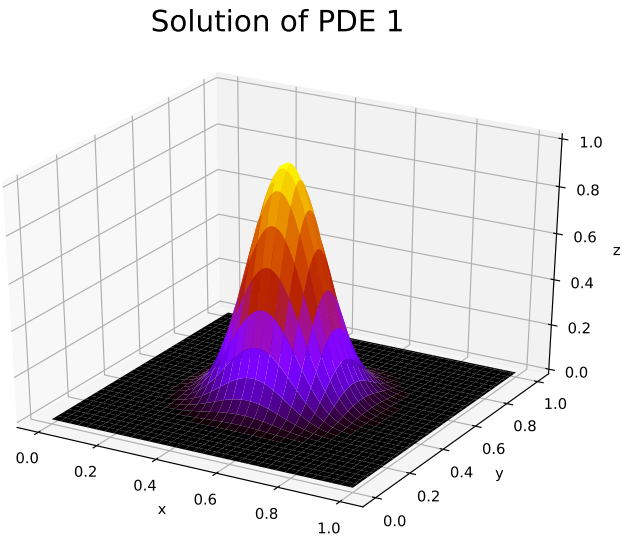


Figure B.3: PDE 1 Polynomial 2D solution plot

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:

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

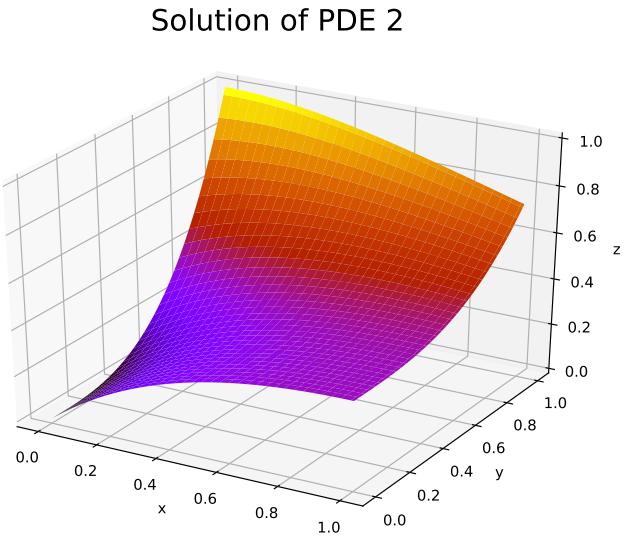


Figure B.4: PDE 2 Chaquet PDE 1 solution plot

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

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

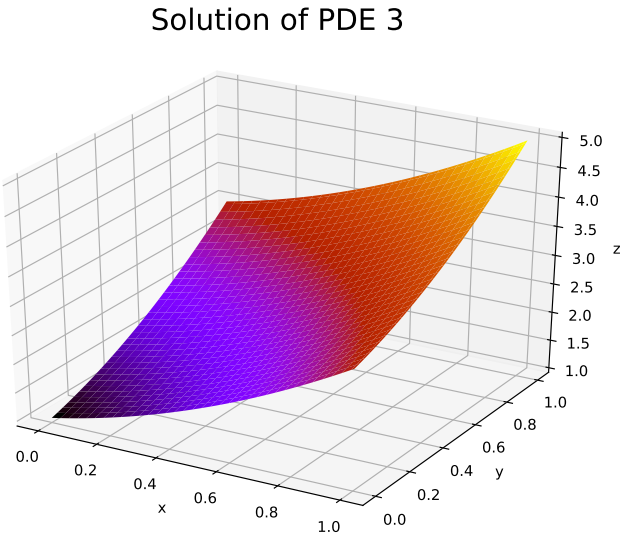


Figure B.5: PDE 3 Chaquet PDE 3 solution plot

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:

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

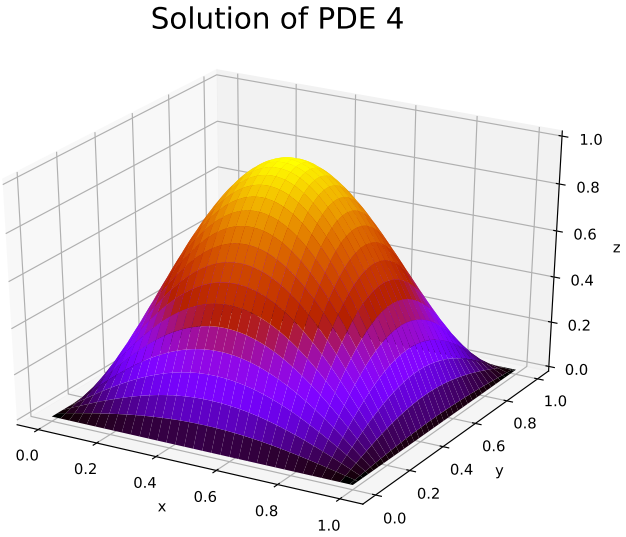


Figure B.6: PDE 4 Sine Bump 2D solution plot

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:

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

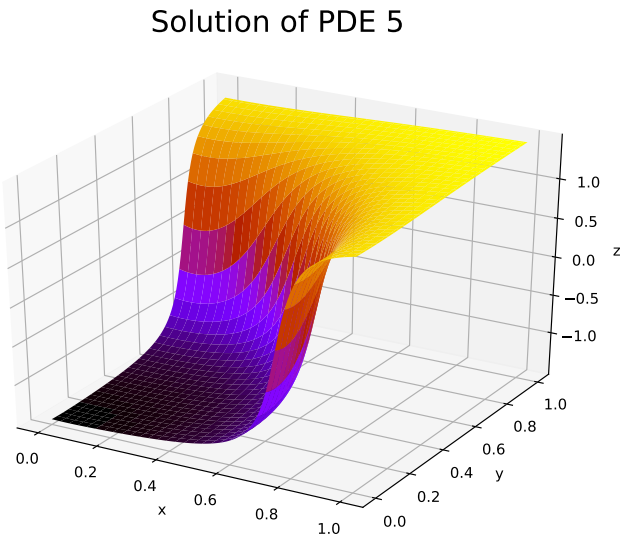


Figure B.7: PDE 5 Arctan Circular Wave Front solution plot

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:

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

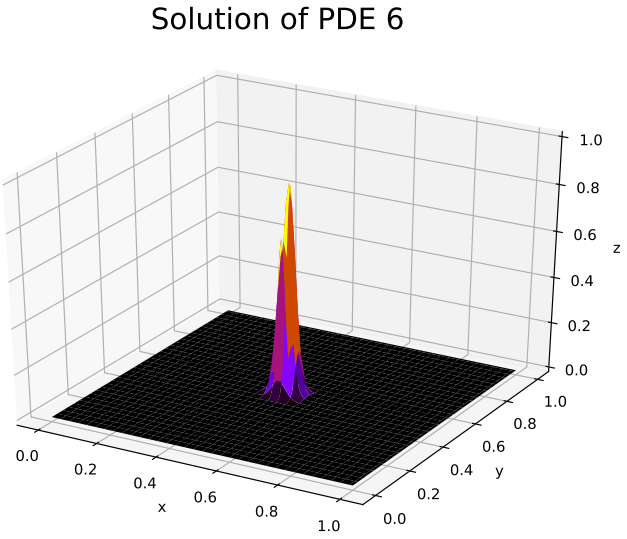


Figure B.8: PDE 6 Peak 2D solution plot

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:

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

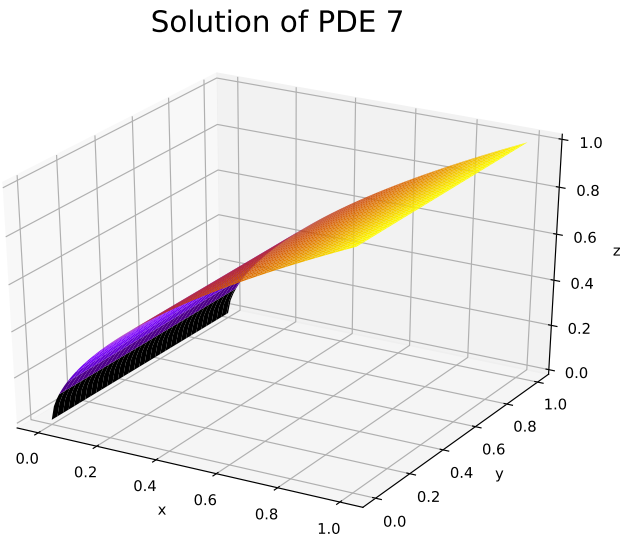


Figure B.9: PDE 7 Boundary Line Singularity solution plot

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:

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

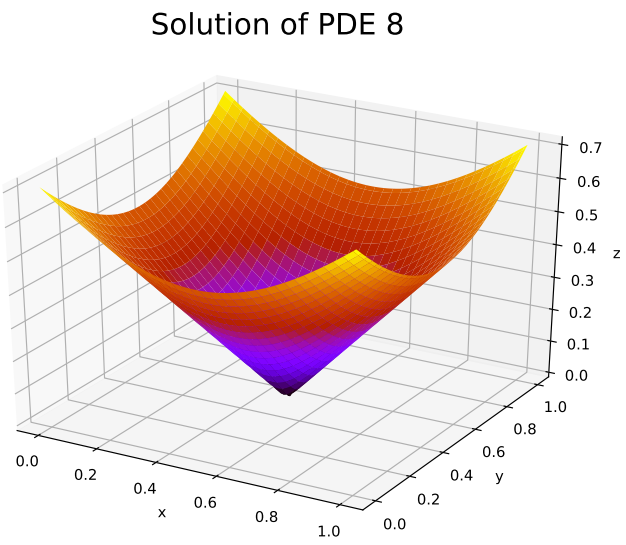


Figure B.10: PDE 8 Interior Point Singularity solution plot

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:

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

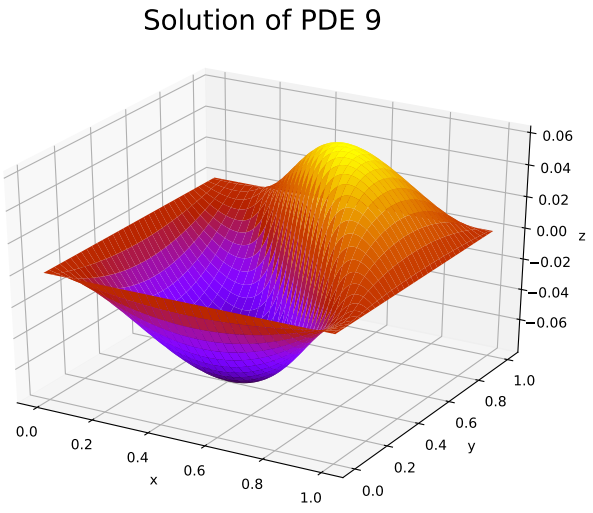


Figure B.11: PDE 9 Arctan Wave Front Homogeneous Boundary Conditions 2D solution plot

C Software Architecture

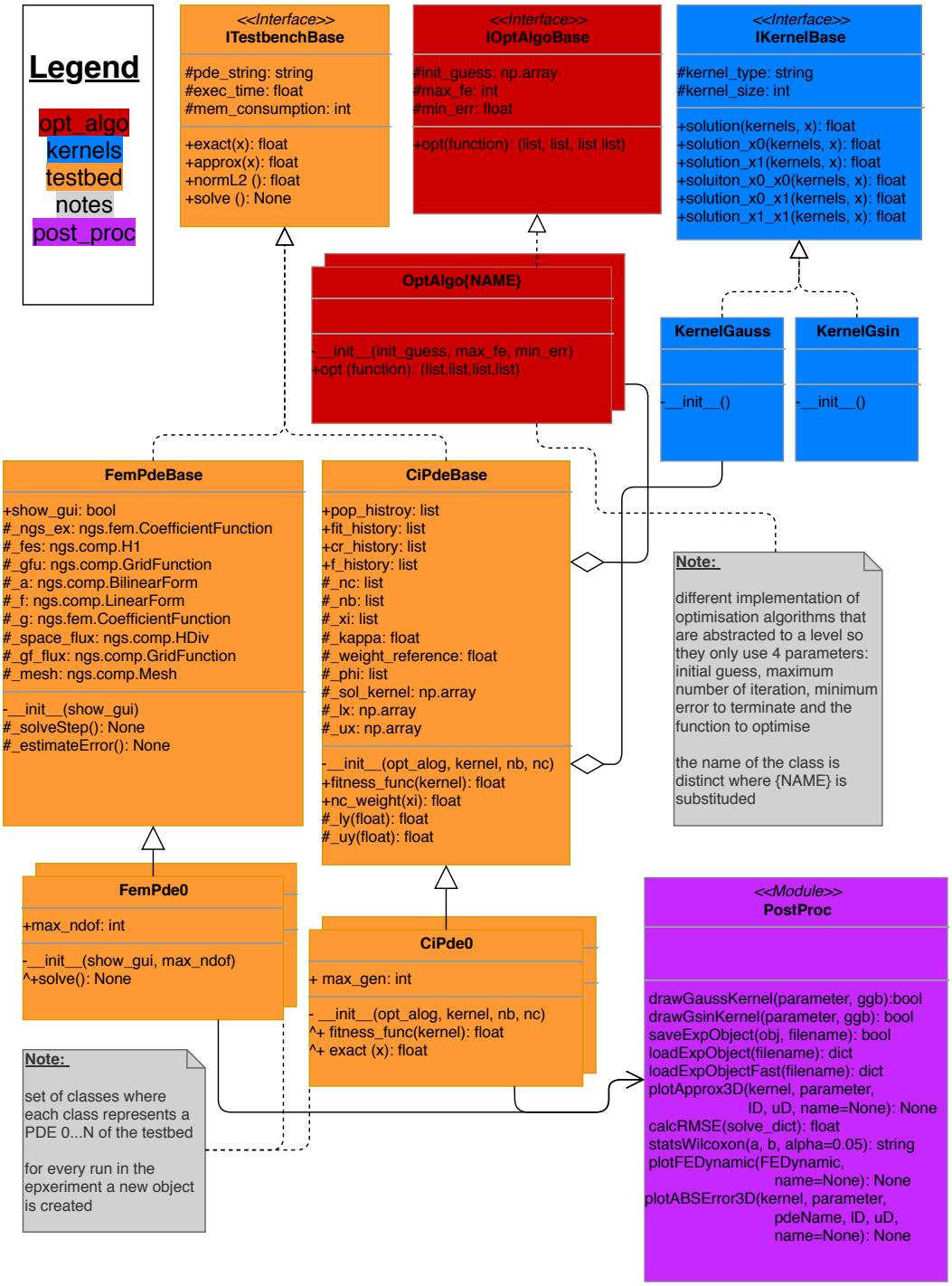


Figure C.1: This UML class diagram describes the software architecture defined to prepare, run and evaluate the experiments.

D Solve Method

Algorithm D.1: Solve Method 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().vms
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().vms − memstart
14     gc.enable()
15     gc.collect()
```

E pJADE

Algorithm E.1: pJADE Pseudocode

Function `pJADE(`$\mathbf{X}_{g=0}$, p, c, $function$, $minError$, $maxFE$`)`**:**
  $fValue_{g=0} \leftarrow function(\mathbf{x}_{g=0})$
  $\mu_{CR} \leftarrow 0.5$
  $\mu_F \leftarrow 0.5$
  $A \leftarrow \emptyset$
  while $fe \leq maxFE$ **do**
    $g \leftarrow g + 1$
    $S_F \leftarrow \emptyset$
    $S_{CR} \leftarrow \emptyset$
    $pResults \leftarrow \emptyset$
    for $i = 1$ *to* NP **do parallel**
      $F_i \leftarrow randc_i(\mu_F, 0.1)$
      $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,g}, A, fValue_g, F_i, p)$
      $CR_i \leftarrow randn_i(\mu_{CR}, 0.1)$
      $u_i \leftarrow crossoverBIN(\mathbf{x}_{i,g}, v_i, CR_i)$
      $(u_i, function(u_i)) \rightarrow pResults$
    end
    for $i = 1$ *to* NP **do**
      if $function(\mathbf{x}_{i,g}) \geq pResults_{i,f,g}$ **then**
        $\mathbf{x}_{i,g+1} \leftarrow \mathbf{x}_{i,g}$
      end
      else
        $\mathbf{x}_{i,g+1} \leftarrow pResults_{i,u,g}$
        $fValue_{i,g+1} \leftarrow pResults_{i,f,g}$
        $\mathbf{x}_{i,g} \rightarrow \mathbf{A}$
        $CR_i \rightarrow S_{CR}$
        $F_i \rightarrow S_F$
      end
    end
    `//` `resize` A `to size of` $\mathbf{x}_g$
    if $|A| > NP$ **then**
      $A \leftarrow A \setminus A_{rand_i}$
    end
    $fe \leftarrow fe + size(\mathbf{X})$
    $\mu_{CR} \leftarrow (1 - c) \cdot \mu_{CR} + c \cdot arithmeticMean(S_{CR})$
    $\mu_F \leftarrow (1 - c) \cdot \mu_F + c \cdot lehmerMean(S_F)$
  end

F paJADE

Algorithm F.1: paJADE Pseudocode

```
1  Function paJADE(X_{g=0}, p, c, dT, function, minError, maxFE):
2      fValue_{g=0} ← function(x_{g=0})
3      μ_CR ← 0.5
4      μ_F ← 0.5
5      A ← ∅
6      while fe ≤ maxFE do
7          g ← g + 1
8          S_F ← ∅
9          S_CR ← ∅
10         pResults ← ∅
11         for i = 1 to NP do parallel
12             F_i ← randc_i(μ_F, 0.1)
13             v_i ← mutationCurrentToPBest1(x_{i,g}, A, fValue_g, F_i, p)
14             CR_i ← randn_i(μ_CR, 0.1)
15             u_i ← crossoverBIN(x_{i,g}, v_i, CR_i)
16             (u_i, function(u_i)) → pResults
17         end
18         for i = 1 to NP do
19             if function(x_{i,g}) ≥ pResults_{i,f,g} then
20                 x_{i,g+1} ← x_{i,g}
21             end
22             else
23                 x_{i,g+1} ← pResults_{i,u,g}
24                 fValue_{i,g+1} ← pResults_{i,f,g}
25                 x_{i,g} → A
26                 CR_i → S_CR
27                 F_i → S_F
28             end
29         end
30         // resize A to size of x_g
31         if |A| > NP then
32             A ← A \ A_{rand_i}
33         end
34         fe ← fe + size(X)
35         μ_CR ← (1 − c) · μ_CR + c · arithmeticMean(S_CR)
36         μ_F ← (1 − c) · μ_F + c · lehmerMean(S_F)
37         // state detector
38         if min(function(X_{g−dT}) − function(X_g)) < minError then
39             break
40         end
41     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, 07.07.2020 Schwartze Nicolai, BSc