

# 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, 10.08.2020

# 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.

# Abstract

## 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

<b>List of Figures</b>	<b>VIII</b>
<b>List of Tables</b>	<b>XI</b>
<b>List of Algorithms</b>	<b>XII</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 State of the Art</b>	<b>2</b>
2.1 Finite Element Method . . . . .	2
2.2 Computational Intelligence Methods . . . . .	4
2.3 Differential Evolution . . . . .	10
<b>3 Problem Definition</b>	<b>12</b>
3.1 Theoretical Foundation . . . . .	12
3.2 Fitness Function . . . . .	13
3.3 Candidate Representation . . . . .	14
3.3.1 Gauss Kernel . . . . .	15
3.3.2 GSin Kernel . . . . .	16
<b>4 Experimental Design</b>	<b>19</b>
4.1 Testbed . . . . .	19
4.2 Software Architecture . . . . .	22
4.3 Performance Metric . . . . .	23
4.3.1 Solving Time . . . . .	24
4.3.2 Memory Usage . . . . .	24
4.3.3 Quality Measurement . . . . .	25
4.4 Baseline: NGsolve . . . . .	26
4.4.1 Setup . . . . .	27
4.4.2 Result . . . . .	27
4.5 Default CI Parameter . . . . .	30
4.6 Hardware Infrastructure . . . . .	32

<b>5 Experiment 0: Serial Memetic JADE</b>	<b>34</b>
5.1 Hypotheses . . . . .	34
5.2 Experiment Setup . . . . .	35
5.3 Results . . . . .	35
5.4 Discussion . . . . .	38
5.4.1 Comparison to Literature . . . . .	38
5.4.2 Solving Time/Memory Usage . . . . .	38
5.4.3 PDE 0A . . . . .	39
5.4.4 PDE 5 . . . . .	40
<b>6 Experiment 1: Parallel Population JADE</b>	<b>43</b>
6.1 Hypotheses . . . . .	43
6.2 Experiment Setup . . . . .	44
6.3 Results . . . . .	44
6.4 Discussion . . . . .	46
6.4.1 Memory Usage . . . . .	46
6.4.2 Solving Time . . . . .	47
6.4.3 PDE 4 . . . . .	48
<b>7 Experiment 2: Adaptive Number of Kernels</b>	<b>50</b>
7.1 Hypotheses . . . . .	50
7.2 Experiment Setup . . . . .	52
7.3 Result . . . . .	52
7.4 Discussion . . . . .	55
7.4.1 Solving Time . . . . .	55
7.4.2 Memory Usage . . . . .	56
7.4.3 PDE 0A . . . . .	56
7.4.4 PDE 2 3 4 7 . . . . .	57
7.4.5 PDE 5 . . . . .	60
7.4.6 PDE 6 . . . . .	61
<b>8 Experiment 3: Gauss-Sinus Kernel</b>	<b>64</b>
8.1 Hypotheses . . . . .	64
8.2 Experiment Setup . . . . .	64
8.3 Result . . . . .	65
8.4 Discussion . . . . .	67
8.4.1 Solving Time . . . . .	67
8.4.2 Memory Usage . . . . .	67
8.4.3 PDE 0A 6 . . . . .	67
8.4.4 PDE 0B . . . . .	68
8.4.5 PDE 5 . . . . .	68

<b>9 Limitations</b>	<b>71</b>
9.1 Testbed . . . . .	71
9.2 Fulfilment of Boundary Condition . . . . .	71
9.3 Computational Effort . . . . .	71
<b>10 Theoretical Notes</b>	<b>72</b>
10.1 Universial Approximation Theorem . . . . .	72
10.2 Multimodality and Symmetry . . . . .	72
<b>11 Conclusion</b>	<b>73</b>
<b>12 Further Work</b>	<b>74</b>
<b>13 Summary</b>	<b>75</b>
<b>Acronyms</b>	<b>76</b>
<b>Bibliography</b>	<b>78</b>
<b>Appendices</b>	<b>84</b>
<b>A Differential Evolution Pseudocodes</b>	<b>85</b>
A.1 JADE Pseudocode . . . . .	86
A.2 SHADE Pseudocode . . . . .	87
A.3 L-SHADE Pseudocode . . . . .	88
<b>B Testbed</b>	<b>89</b>
<b>C Software Architecture</b>	<b>101</b>
<b>D Post-Processing Module Description</b>	<b>103</b>
<b>E Solve Method</b>	<b>105</b>
<b>F pJADE</b>	<b>106</b>
<b>G paJADE</b>	<b>108</b>
<b>H Adaptive Kernel Scheme</b>	<b>110</b>
<b>I PDE 2 3 4 and 7 Kernel Adaption</b>	<b>111</b>

# List of Figures

3.1	3D plot of a single Gauss kernel with the parameters $\omega = 1, \gamma = 1, c_0 = 0$ and $c_1 = 0$ . . . . .	15
3.2	3D plot of a single GSin kernel with the parameters $\omega = 1, \gamma = 1, c_0 = 0, c_1 = 0, f = 1, \varphi = 0$ . . . . .	17
4.1	This limited Unified Modeling Language (UML) class diagram gives a small overview of the testbed design and its interfaces and classes.	23
4.2	Aliasing Error: sharp inaccuracies in between collocation points . .	26
4.3	Boxplot: time (in seconds) needed to solve the testbed Partial Eifferential Equation (PDE) (without PDE3) . . . . .	28
4.4	Boxplot: memory (in Mbyte) needed to solve the testbed PDE (without PDE3) . . . . .	29
4.5	PDE 3 performance metrics at different Degree of Freedom (DOF) budgets. . . . .	30
4.6	Comparison of time and memory effort on PDE 3 at $5 \cdot 10^4$ DOF. .	30
4.7	Weighting factor $\varphi$ and $\xi$ on every collocation point . . . . .	32
4.8	Collocation points used in the testbed. For the PDEs 0A and 0B with the larger domain, the lower limits $x_0L$ and $x_1L$ are replaced with -2 and the upper limits $x_0U$ and $x_1U$ are replaced with 2. The points are still equally spaced. . . . .	33
5.1	Relative solving time results of memetic JADE after $10^4$ Number of Function Evaluation (#FE). . . . .	37
5.2	Relative memory usage results of memetic JADE after $10^4$ #FE. .	37
5.3	Comparison of two typical PDE 0A solutions. . . . .	40
5.4	Comparison of best solution after $10^4$ and $10^6$ #FE. . . . .	40
5.5	Histograms of the fitness and the L2 norm reached with $10^4$ #FE and $10^6$ #FE on PDE 5. . . . .	41
5.6	Fitness value and L2 Norm of one individual at every generation on PDE 5. . . . .	42

6.1	Relative solving time results of parallel memetic JADE after $10^4$ #FE.	45
6.2	Relative memory usage results of parallel memetic JADE after $10^4$ #FE.	45
6.3	Mean speed-up of parallel JADE with 95% confidence interval.	47
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	48
6.5	Histogram of the L2 norm data obtained by the serial and the parallel JADE on PDE 4 from table 6.1.	49
6.6	Boxplot of the L2 norm data from table 6.1 on PDE 4.	49
7.1	Flowchart of the adaptive kernel scheme.	51
7.2	Relative solving time results of parallel memetic JADE with adaptive kernels after $10^4$ #FE.	53
7.3	Relative memory usage results of parallel memetic JADE with adaptive kernels after $10^4$ #FE.	54
7.4	Number of kernels of the proposed solution by the paJADE per PDE at $10^6$ #FE	55
7.5	3D plot and comparison of unnecessary 6 <sup>th</sup> kernels on PDE 0A.	57
7.6	Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are taken at $10^6$ #FE on the PDEs 2, 3, 4 and 7. The corresponding $R^2$ values of the linear regression are denoted.	57
7.7	Comparison of <i>minError</i> against the number of kernels and the achieved solution quality.	58
7.8	Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are produced with a <i>minError</i> = $10^{-1}$ after $10^6$ #FE.	59
7.9	Boxplot of solution quality on PDE 5 with a budget of $10^6$ #FE. Comparison of distribution with and without outliers.	60
7.10	3D plot of the outlier solution produced by the adaptive kernel scheme on PDE 5 after $10^6$ #FE.	61
7.11	Comparison of the L2 norm reached by the paJADE and a non-adaptive pJADE on PDE 6 after $10^6$ #FE.	62
7.12	These wrong solutions are generated with paJADE after $10^6$ #FE and represent local optima where JADE gets stuck.	62
7.13	Kernel bar plot of the false solution, represented in figure 7.12a	63
7.14	Kernel bar plot of the false solution, represented in figure 7.12b	63

8.1	Relative solving time results of parallel memetic JADE with Gauss Sine Kernel (GSK) after $10^4$ #FE. . . . .	65
8.2	Relative memory usage results of parallel memetic JADE with GSK after $10^4$ #FE. . . . .	66
8.3	Comparisons of best solution absolute error by memetic parallel JADE using Gauss Kernel (GaK) and GSK after $10^6$ #FE. . . . .	68
8.4	Comparison of the best and the worst result generated by memetic pJADE with GSK after $10^6$ #FE. . . . .	69
8.5	Histograms of GSK and GaK L2 norm and fitness value on PDE 5 after $10^6$ #FE. . . . .	69
8.6	Fitness value and L2 Norm of an exemplary individual at every generation on PDE 5 using a GSK. . . . .	70
B.1	PDE 0A Gauss Kernel solution plot . . . . .	90
B.2	PDE 0B Gsin Kernel solution plot . . . . .	91
B.3	PDE 1 Polynomial 2D solution plot . . . . .	92
B.4	PDE 2 Chaquet PDE 1 solution plot . . . . .	93
B.5	PDE 3 Chaquet PDE 3 solution plot . . . . .	94
B.6	PDE 4 Sine Bump 2D solution plot . . . . .	95
B.7	PDE 5 Arctan Circular Wave Front solution plot . . . . .	96
B.8	PDE 6 Peak 2D solution plot . . . . .	97
B.9	PDE 7 Boundary Line Singularity solution plot . . . . .	98
B.10	PDE 8 Interior Point Singularity solution plot . . . . .	99
B.11	PDE 9 Arctan Wave Front Homogeneous Boundary Conditions 2D solution plot . . . . .	100
C.1	This UML class diagram describes the software architecture defined to prepare, run and evaluate the experiments. . . . .	102
I.1	Kernel Bars Plot on the worst result of PDE2 in experiment 2. . .	111
I.2	Kernel Bars Plot on the best result of PDE2 in experiment 2. . .	111
I.3	Kernel Bars Plot on the worst result of PDE3 in experiment 2. . .	111
I.4	Kernel Bars Plot on the best result of PDE3 in experiment 2. . .	112
I.5	Kernel Bars Plot on the worst result of PDE4 in experiment 2. . .	112
I.6	Kernel Bars Plot on the best result of PDE4 in experiment 2. . .	112
I.7	Kernel Bars Plot on the worst result of PDE7 in experiment 2. . .	112
I.8	Kernel Bars Plot on the best result of PDE7 in experiment 2. . .	112

# List of Tables

2.1	Literature research on the general topic of stochastic solver and their application. The papers are sorted by relevance to the present work.	9
4.1	These are the results obtained by the Finite Element Method (FEM) solver in terms of distance to the analytical solution. The solver achieves the smallest deviation in PDE 3. . . . .	28
4.2	These predefined parameters are used for the following numerical experiments. . . . .	31
4.3	Comparison of the machines used for the experiments. . . . .	33
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. . . . .	35
5.2	L2 norm reached with serial JADE at $10^4$ #FE and $10^6$ #FE . . .	36
6.1	This table 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. . . . .	46
7.1	Comparison of the achieved L2 norm by the pJADE and the paJADE at $10^6$ #FE. The pJADE data is directly taken from table 6.1. . .	54
7.2	Statistical comparison of the achieved L2 norm by paJADE with $\minError = 0$ and $\minError = 10^{-1}$ after $10^6$ #FE. The data with $\minError = 0$ is directly taken from table 7.1. . . . .	59
8.1	Statistical comparison of the the parallel JADE using the GaK and the GSK. . . . .	66

# List of Algorithms

5.1	Pseudocode of memetic JADE . . . . .	34
6.1	Pseudocode of memetic parallel JADE . . . . .	44
A.1	JADE Pseudocode . . . . .	86
A.2	SHADE Pseudocode . . . . .	87
A.3	L-SHADE Pseudocode . . . . .	88
E.1	Solve Method Pseudocode . . . . .	105
F.1	Pseudocode of pJADE . . . . .	107
G.1	Pseudocode of paJADE . . . . .	109
H.1	Pseudocode of memetic parallel JADE with adaptive kernels . . . . .	110

# 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 modelled 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 equations. 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 algorithm. The main focus of this thesis lies on 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  $\Omega$  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: Strong Form

This is the standard formulation of the linear PDE.  $\mathbf{L}$  and  $\mathbf{B}$  are linear differential operators that include the derivatives.

$$\begin{aligned} \mathbf{x} &\in \mathbb{R}^2 \\ u(\mathbf{x}), f(\mathbf{x}), g(\mathbf{x}) &: \Omega \rightarrow \mathbb{R} \\ \mathbf{L}u(\mathbf{x}) &= f(\mathbf{x}) \text{ on } \Omega \\ \mathbf{B}u(\mathbf{x}) &= g(\mathbf{x}) \text{ on } \partial\Omega \end{aligned} \tag{2.1}$$

Further, only Dirichlet boundary conditions are considered, thus the boundary operator is always the identity matrix  $\mathbf{B} = \mathbb{I}$ . Therefore, the linear operator on the boundary  $\mathbf{B}$  can be disregarded, resulting in

$$u(\mathbf{x})|_{\partial\Omega} = g(\mathbf{x}). \tag{2.2}$$

2. Step: Weak Form

The next step is to reformulate the strong form into a usable weak form. This is equivalent to the strong form but written in an integral notation. In this

equation, the  $A$ ,  $b$  and  $c$  correspond to the constant factors of the derivatives in strong form. For the sake of completeness, this is kept abstract. In the PDEs considered in this work,  $A = \mathbb{I}$  and  $b = \mathbf{0}$ ,  $c = 0$ . Currently, the so-called test-function  $v(\mathbf{x})$  is an arbitrary function, but it has to be 0 on the boundary  $v(\mathbf{x})|_{\Omega} = 0$ . The choice of the test-function correspond to different FEM types. Shen, Tang, and Wang 2011, p. 6f

$$\underbrace{\int_{\Omega} -(\nabla^T A \nabla) u(\mathbf{x}) v(\mathbf{x}) dV - \int_{\Omega} b^T \nabla u(\mathbf{x}) v(\mathbf{x}) dV + \int_{\Omega} c u(\mathbf{x}) v(\mathbf{x}) dV}_{a(u,v)} = \underbrace{\int_{\Omega} f(\mathbf{x}) v(\mathbf{x}) dV}_{F(v)} \quad (2.3)$$

### 3. Step: Discretisation of $\Omega$

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

### 4. Step: Basis functions

Choose a basis function  $\Phi(\mathbf{x})$  that can be used to approximate the solution  $u(\mathbf{x}) \approx u_h(\mathbf{x}) = \sum_{i=1}^N u_i \Phi(\mathbf{x})_i$ . A common choice are Lagrange or Chebyshev polynomials. In the Galerkin type FEM, the test-function  $v(\mathbf{x})$  is the same as the trial-function, thus  $v(\mathbf{x}) = \sum_{j=1}^N v_j \Phi(\mathbf{x})_j$ . The choice of the basis function  $\Phi(\mathbf{x})$  largely influences the computational effort.  $\Phi(\mathbf{x})$  should have a small support, to produce a thinly populated matrix  $A$  in the linear system of equations (2.5) below.

### 5. Step: Solution

In the weak form, as seen in equation (2.3),  $a(u, v)$  is a continuous bilinear form and  $F(v)$  is a continuous linear functional. Substituting  $u$  and  $v$  with their corresponding approximation from step 4 results in

$$\sum_{j=1}^N v_j \sum_{i=1}^N u_i a(\Phi_i, \Phi_j) = \sum_{j=1}^N v_j F(\Phi_j). \quad (2.4)$$

Dividing by the  $v_j$  values on both sides results in a linear system of equations, where the constant factors  $u_i$  need to be determined.

$$\underbrace{\sum_{i=1}^N u_i a(\Phi_i, \Phi_j)}_{\mathbf{A}\mathbf{u}} = \underbrace{F(\Phi_j)}_{\mathbf{b}} \text{ for } j = 1, \dots, N \quad (2.5)$$

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. Static condensation, also called Guyan reduction (Guyan 1965), reduces this dimensionality by exploiting the structure of  $\mathbf{A}$ .

- 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.

- Adaptive Mesh Refinement:

The accuracy of a FEM-approximated solution mainly depends on the density 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)
- $\circlearrowleft$  until degrees of freedom exceed a specified  $N$

## 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 historically.

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 fitness function is the function to be optimised. It is also called objective function and these terms are used interchangeably. In short, the residual  $R$  is defined through the differential equation itself and can be calculated by  $R(u(\mathbf{x})) = \mathbf{L}u(\mathbf{x}) - f(\mathbf{x})$ . The residual can be thought of as a functional that substitutes  $u(\mathbf{x})$  with an approximate

solution  $u_{apx}(\mathbf{x})$  and returns a numerical score. The WRM method is further described in chapter 3.1.

Howard and Roberts 2001 is one of the first advances in this field. They approximate a subset of the convection-diffusion equations with Genetic Programming (GP) (Koza 1992). Their main idea is to use a polynomial of variable length as the candidate solution that is forced to satisfy the boundary condition. Their fitness value, as seen in equation (2.6), is calculated by squaring the residual  $R$  and integrating it over the domain. Since the polynomials are known, and the problems are restricted to a specific differential equation, the integral can be evaluated analytically.

$$F(u_{apx}(\mathbf{x})) = - \int_{\Omega} R(u_{apx}(\mathbf{x}))^2 dx \quad (2.6)$$

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 distinct points over the domain. These points are further called collocation points.

Tsoulos and Lagaris 2006 use 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. Because the GE step could result in virtually any function, the fitness integral from equation (2.6) can not be calculated analytically. Thus, the integral is approximated by evaluating the residual at collocation points within the domain. This is seen in equation (2.7). The algorithm was tested on multiple problems of Ordinary Differential Equation (ODE), system of ODEs and PDE. Only the results for ODEs were promising.

$$F(u_{apx}(\mathbf{x})) = \sum_{i=1}^{n_C} \|R(u_{apx}(\mathbf{x}_i))\|^2 \quad (2.7)$$

Mastorakis 2006 couples a Genetic Algorithm (GA) (Holland 1962) with a Downhill-Simplex (DS) method (Nelder and Mead 1965) 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 (2.7). The focus here lays on unstable ODEs, that can not be solved with finite difference methods.

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

$$u(\mathbf{x})_{apx} = u(\mathbf{x})_{GP} + u(\mathbf{x})_{RBF}. \quad (2.8)$$

At first, the GP step produced a trial solution according to the objective function (2.7). After the GP procedure, a linear combination of radial basis functions  $u(\mathbf{x})_{RBF} = \sum_{j=1}^{n_B} \alpha_j \Phi(||\mathbf{x} - \mathbf{x}_j||)$  is specifically tailored to  $u(\mathbf{x})_{GP}$ , that ensures the boundary condition at all  $\mathbf{x}_j$  points on  $\partial\Omega$ . Finding the parameters  $\alpha_j$  can be formulated as a least squares problem.

Howard, Brezulianu, and Kolibal 2011 use a GP scheme to find the solution to a specific set of simplified convection-diffusion equations. They represent a candidate as discrete function value points over the domain. The function between these points is interpolated. The fitness function is similar to equation (2.7) 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 fixed base points.

Chauquet 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 is expressed in equation (2.14). This is similar to the fitness function 2.7, but it extends the definition of the boundary to also include Neumann conditions by introducing the linear differential operator  $\mathbf{B}$ . The limit  $n_C$  denotes the number of inner collocation points  $\mathbf{x}_i$  within the domain  $\Omega$ , whereas  $n_B$  is the number of discrete points  $\mathbf{x}_j$  on the boundary  $\partial\Omega$ . Further, a penalty factor  $\phi$  shifts the focus of the fitness to the boundary. Additionally, this objective function can also represent systems of differential equations, where the number of equations is denoted by  $m$ . 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.

$$F(u_{apx}(\mathbf{x})) = \frac{\sum_{i=1}^{n_C} ||\mathbf{L}u_{apx}(\mathbf{x}_i) - f(\mathbf{x}_i)||^2 + \phi \sum_{j=1}^{n_B} ||\mathbf{B}u_{apx}(\mathbf{x}_j) - g(\mathbf{x}_j)||^2}{m(n_C + n_B)} \quad (2.9)$$

Babaei 2013 takes a similar approach. They approximate a solution using a partial Fourier series. 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 (2.10)) and one for the boundary (equation (2.11)). These are added together resulting in equation (2.12).

The weighted residual integral WRF is exactly the formulation of the WRM from chapter 3.1.  $W$  is an arbitrary weighting function. The absolute values of  $W$  and  $R$  ensure that only positive values count towards the fitness. Instead of using a sum over collocation points, the integral is evaluated using a numerical integration scheme.

$$WRF(u_{apx}(\mathbf{x})) = \int_{\Omega} |W(\mathbf{x})| |R(u_{apx}(\mathbf{x}))| dx \quad (2.10)$$

The boundary condition is incorporated by summing up its normed violations at distinct points  $\mathbf{x}_i$ .  $K_j$  are penalty multipliers that shift the focus to different points of the boundary. The concept of this penalty function originates from **rajeev\_s\_discrete\_1992**.

$$PFV(u_{apx}(\mathbf{x})) = WRF(u_{apx}(\mathbf{x})) \cdot \sum_{j=1}^{n_B} K_j \left( \frac{u_{apx}(\mathbf{x}_i)}{g(\mathbf{x}_i)} - 1 \right) \quad (2.11)$$

$$F(u_{apx}(\mathbf{x})) = WRF(u_{apx}(\mathbf{x})) + PFV(u_{apx}(\mathbf{x})) \quad (2.12)$$

Panagant and Bureerat 2014 use polynomials as a candidate representation. They do not specify the order or the type of the polynomial. Five different simple versions of DE (Storn and Price 1997) are 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.

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). They use the formulation in equation (2.10), where the weighting function is the same as the residual  $|W(\mathbf{x})| = |R(u_{apx}(\mathbf{x}))| \implies WRF = \int_{\Omega} |R(u_{apx}(\mathbf{x}))|^2 dx$ . The integral is again approximated using a numerical integration scheme. They find that the PSO is slightly better at producing low error solutions, however WCA is better at satisfying the boundary condition.

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:

$$u(\mathbf{x})_{apx} = \sum_{i=1}^N \omega_i e^{\gamma_i (\|\mathbf{x} - \mathbf{c}_i\|^2)} \quad (2.13)$$

The approximated function  $u(\mathbf{x})_{apx}$  can be fully determined by a finite number of parameters:  $\omega_i, \gamma_i, \mathbf{c}_i$ . These are stacked together into a vector  $param_{uapx}$  and called the decision variables which are optimised by the algorithm. The objective function

can be seen in equation (2.14). This is an update of the objective function in 2.9 where the inner collocation points also get scaled by a weighting function  $\xi(\mathbf{x}_i)$ .

$$F(u_{apx}(\mathbf{x})) = \frac{\sum_{i=1}^{n_C} \xi(\mathbf{x}_i) \|\mathbf{L}u_{apx}(\mathbf{x}_i) - f(\mathbf{x}_i)\|^2 + \phi \sum_{j=1}^{n_B} \|\mathbf{B}u_{apx}(\mathbf{x}_j) - g(\mathbf{x}_j)\|^2}{m(n_C + n_B)} \quad (2.14)$$

The multipliers  $\xi(\mathbf{x}_i)$  and  $\phi$  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 DS method to carry out the local search. The authors show empirically that the local search significantly improves the performance by testing the algorithm on a set of 32 differential equations.

Fateh et al. 2019 use a simple variant of DE where candidates are represented as discrete function value points within the domain. The function values between the grid points are linearly interpolated. 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 differential equation and the boundary condition is incorporated into the fitness function by taking the sum of squared residuals at every grid point, as seen in equation (2.15). The derivatives within the residual are calculated between two neighbouring points by the difference quotient.

$$F(\mathbf{x}) = \sqrt{\sum_{i=0}^n R(\mathbf{x}_i)^2} \quad (2.15)$$

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

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.

## 2.3 Differential Evolution

The differential evolution 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 GECCO 2019 (Suganthan 2020) shows.

The main DE framework consists of three necessary steps, that continuously update a population of possible solutions. The population can be interpreted as a matrix, where each row-vector  $\mathbf{x}_i$ , also called individual, represents a point within the search domain and has a fitness value corresponding to the fitness function  $f(\mathbf{x}_i) : \mathbb{R}^n \rightarrow \mathbb{R}$ . The goal is to minimise the fitness function. These steps are performed in a loop until some termination condition is reached. Each of these steps is controlled by a user-defined parameter:

- Mutation:

Mutation strength parameter  $F$ ;

The mutation 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 (2.16) where  $x_i$  is the current individual,  $x_{best}^p$  is one random vector of the p% top vectors,  $x_{r1}$  is a random vector from the population while  $\tilde{x}_{r2}$  is randomly chosen from the population and the archive.  $x_{r1}$  and  $\tilde{x}_{r2}$  must not describe the same individual.

$$v_i = x_i + F_i(x_{best}^p - x_i) + F_i(x_{r1} - \tilde{x}_{r2}) \quad (2.16)$$

- Crossover:

Crossover probability parameter CR;

The crossover procedure randomly mixes the information between the trial vector  $v_i$  and a random candidate from the population  $x_i$  to create a new trial vector  $u_i$ . The binomial crossover from equation (2.17) 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.

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } j = K \vee \text{rand}[0, 1] \leq CR \\ x_{ij}, & \text{otherwise} \end{cases} \quad (2.17)$$

- Selection:

Population size N;

The selection replaces 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, then the next generation is started.

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 improves 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 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.

# 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.

## 3.1 Theoretical Foundation

The most general description of a linear 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 (2.1), 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  $a_k$  need 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 differential equation. The residual of a solved PDE is zero. This is only possible for the analytical solution, further denoted as  $u_{ext}(x, y)$ . For a numerical approximation  $u_{apx}(x, y)$ , the residual should be "small enough".

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

The WRM (Shen, Tang, and Wang 2011) tries to minimise this residual of a numerical candidate solution over the whole domain  $\Omega$ . Therefore,  $R$  at every  $\mathbf{x} \in \Omega$  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  $W(\mathbf{x})$ , which is needed for numerical stability and is linked to the Gauss Quadrature integration scheme. 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.

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

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”. 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.

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

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).

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

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 ( $n_B$ ) and the points on the inner domain ( $n_C$ ). Further, they divide the fitness value by the

number of points used. This fitness function, as seen in equation (2.14), is adopted in this thesis. Because here, systems of differential equations are not considered, the denominator changes to  $m = 1$  resulting in the fitness function (3.7) below. The norm around the residual is not necessary for this thesis, since the fitness function does not have to work for System of PDEs.

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

A notable property of the fitness function is that it can not drop below 0. A solution, that satisfies the residual at every  $n_C$  and  $n_B$ , results in a fitness of 0. This is true for the analytical solution  $u_{apx}(\mathbf{x})$ . It is also important to notice, that aliasing errors (as described in the chapter 4.3.3) can occur with this fitness function.

The weighting function is also split into two parts:  $\xi$  for the inner collocation points and  $\phi$  as penalty factor for the boundary points. The weighting function  $\xi$  can be adapted by a parameter  $\kappa$ . For a  $\kappa > 0$ ,  $\xi$  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.

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

### 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  $\phi$ , as stated 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

$$r = \|\mathbf{x} - \mathbf{c}\| \quad (3.9)$$

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 GaK is the classical choice to approximate functions. It was first introduced in Broomhead and Lowe 1988. [Equation \(3.10\) shows the formulation of such a kernel.](#) One kernel has 4 parameters, that change its shape and location.  $\omega$  is a scaling factor for each and  $\gamma$  describes how sharp the e-function is. Depending on the space the solution is defined in, the number of values in the vector  $\mathbf{c}$  could change - one for each dimension. However, in this work only  $\mathbb{R}^2$  is of interest, thus  $\mathbf{c}$  always includes 2 values. In figure 3.1 a single standard GaK is plotted.

cite approximation proof

$$gak(\mathbf{x}) = \omega e^{-\gamma \|\mathbf{x}-\mathbf{c}\|^2} \quad (3.10)$$

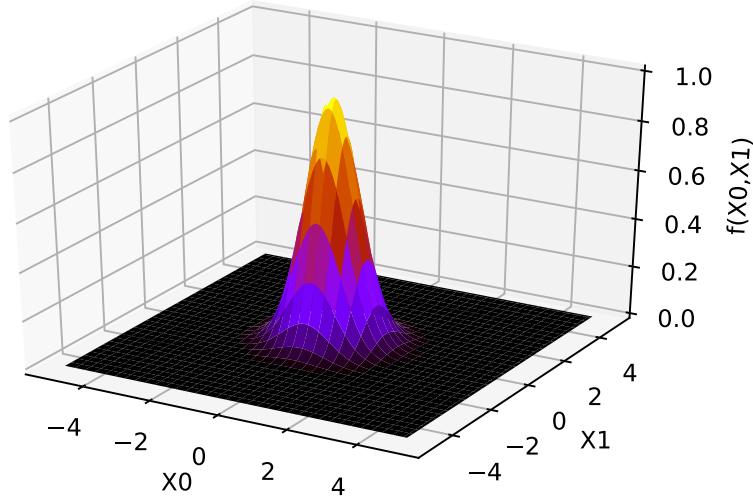


Figure 3.1: 3D plot of a single Gauss kernel with the parameters  $\omega = 1$ ,  $\gamma = 1$ ,  $c_0 = 0$  and  $c_1 = 0$

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

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

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

$$\mathbf{param}_{\mathbf{uapx}} = \left[ \underbrace{\omega_0, \gamma_0, c_{00}, c_{01}}_{\text{kernel 0}}, \dots, \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}}_{\text{kernel i}}, \dots, \underbrace{\omega_N, \gamma_N, c_{N0}, c_{N1}}_{\text{kernel N}} \right]^T \quad (3.12)$$

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).

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_j} = -2 \sum_{i=0}^N \omega_i \gamma_i (x_j - c_{ij}) e^{-\gamma_i r_i^2} \quad (3.13)$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j^2} = \sum_{i=0}^N \omega_i \gamma_i [4\gamma_i(x_j - c_{ij})^2 - 2] e^{-\gamma_i r_i^2} \quad (3.14)$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j \partial x_k} = 4 \sum_{i=0}^N \omega_i \gamma_i^2 (x_j - c_{ij})(x_k - c_{ik}) e^{-\gamma_i r_i^2} \quad (3.15)$$

### 3.3.2 GSin Kernel

Additional to the Gauss kernel, a “GSin” kernel, further also abbreviated as 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  $\varphi$  represents the phase shift.

approximation proof possible?

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

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 resolves to 1

### 3 Problem Definition

---

and the e-function part is obtained. Thus, with this kernel it should be possible to approximate more functions than the GaK.

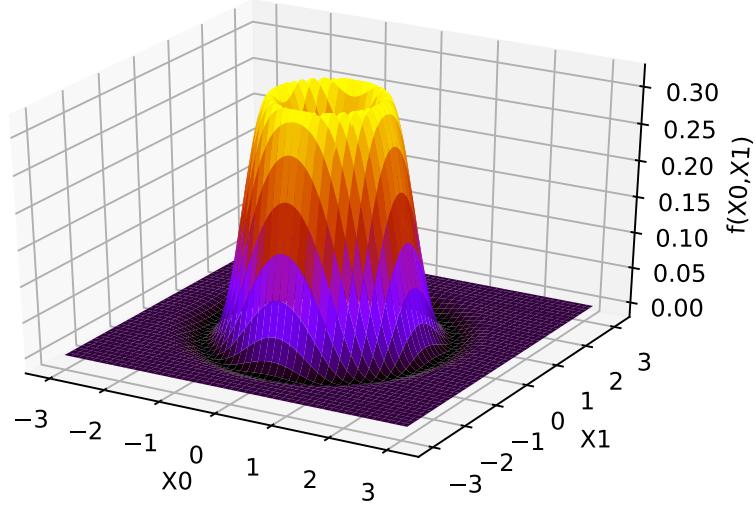


Figure 3.2: 3D plot of a single GSin kernel with the parameters  $\omega = 1$ ,  $\gamma = 1$ ,  $c_0 = 0$ ,  $c_1 = 0$ ,  $f = 1$ ,  $\varphi = 0$

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

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

Equation (3.18) shows the solution representation in stacked vector notation, including the two new parameters  $f$  and  $\varphi$ .

$$\mathbf{param}_{uapx} = \left[ \underbrace{\omega_0, \gamma_0, c_{00}, c_{01}, f_0, \varphi_0, \dots}_{\text{kernel 0}} \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}, f_i, \varphi_i, \dots}_{\text{kernel i}} \underbrace{\omega_N, \gamma_N, c_{N0}, c_{N1}, f_N, \varphi_N}_{\text{kernel N}} \right]^T \quad (3.18)$$

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

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_j} = \sum_{i=0}^N 2\omega_i(x_j - c_{ij})e^{-\gamma_i r_i^2} (f_i \cos(f_i r_i^2 - \varphi_i) - \gamma_i \sin(f_i r_i^2 - \varphi_i)) \quad (3.19)$$

$$\begin{aligned} \frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j^2} &= \sum_{i=0}^N 2\omega_i e^{-\gamma_i r_i^2} \\ &[-(2(f_i^2 - \gamma_i^2)(x_j - c_{ij})^2 + \gamma_i) \sin(f_i r_i^2 - \varphi_i) - (4f_i \gamma_i (x_j - c_{ij})^2 - f_i) \cos(f_i r_i^2 - \varphi_i)] \end{aligned} \quad (3.20)$$

$$\begin{aligned} \frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j \partial x_k} &= \sum_{i=0}^N -4\omega_i (c_{ij} - x_j)(c_{ik} - x_k) e^{-\gamma_i r_i^2} \\ &[(f_i^2 - \gamma_i^2) \sin(f_i r_i^2 - \varphi_i) + 2f_i \gamma_i \cos(f_i r_i^2 - \varphi_i)] \end{aligned} \quad (3.21)$$

# 4 Experimental Design

This chapter gives an overview on how the subsequent experiments are conducted. An integral part of solver comparison is to define a testbed that holds example problems with a known analytical solution. Further, a quality measurements must be defined that compares the numerical to the analytical solution. Since the memory consumption and the solving time is of interest, a robust method to measure these characteristics is needed. The baseline for all experiments is the FEM solver NGSolve (Mitchell 2018).

## 4.1 Testbed

The testbed is a collection of multiple 2-dimensional scalar PDEs that are analytically solved such that  $u(\mathbf{x}) : \mathbb{R}^2 \rightarrow \mathbb{R}$  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 equations and graphs are displayed in the appendix B. The equations used here are a mixture of different testbeds. Specifically, the equations PDE 2 and PDE 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 Reference: (B.1)

Characteristics: sum of 5 GaK, thus can be approximated arbitrarily close; is designed to show convergence towards analytical solution;

Difficulty: not especially difficult;

Boundary: function value of solution;

**PDE 0B: GSin Kernel**

Equation Reference: (B.3)

Characteristics: can be approximated arbitrarily close by 3 GSK; only 3 kernels are used to keep the dimension of the fitness function similar to PDE 0A;

Difficulty: the contrast between a dominant wave with a steep gradient in the middle and small waves with little influence on the boundary; simple approximations lay the focus on the middle, while good approximations also match the small waves;

Boundary: function value of solution;

**PDE 1: Polynomial 2D**

Equation Reference: (B.5)

Characteristics: polynomial of order 20

Difficulty: global structure similar to a GaK; thus, it should be simple to approximate;

Boundary: homogeneous boundary condition;

Reference: Mitchell 2018

**PDE 2: Chaquet PDE 1**

Equation Reference: (B.7)

Characteristics: used in other testbeds to compare the performance to other papers;

Difficulty: rather simple; slight curvature and no steep gradient;

Boundary: function value on boundary;

Reference: 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 Reference: (B.9)

Characteristics: polynomial of order 2;

Difficulty: well behaved, no steep gradient;

Boundary: function value on boundary;

Reference: Chaquet and Carmona 2019, Chaquet and Carmona 2012, Tsoulos and Lagaris 2006, Panagant and Bureerat 2014

**PDE 4: Sine Bump 2D**

Equation Reference: (B.11)

Characteristics: this PDE is used in both, the work of Chaquet and Carmona 2019 and Mitchell 2018 with slightly different boundary condition; preliminary test have shown that the Chaquet and Carmona 2019 PDE is easier to solve, thus this formulation is disregarded;

Difficulty: sharp boundary condition on the corners of  $\Omega$

Boundary: homogeneous boundary condition;

Reference: Mitchell 2018

**PDE 5: Arctan Circular Wave Front**

Equation Reference: (B.13)

Characteristics: quarter section of slightly shifted arctan;

Difficulty: transition from the flat plateaus to the steep gradient of the circular wave front; preliminary tests have shown that this problem is especially hard;

Boundary: function value on boundary

Reference: Mitchell 2018

**PDE 6: Peak 2D**

Equation Reference: (B.15)

Characteristics: solution is a single but sharp Gaussian “peak” at  $(0.5, 0.5)$ ;

Difficulty: large negative exponent in e-function results in a steep gradient and small region of interest;

Boundary: function value on boundary;

Reference: Mitchell 2018

**PDE 7: Boundary Line Singularity**

Equation Reference: (B.17)

Characteristics: the solution to this problem is a root function, that is only defined on  $x \in \mathbb{R}^+$ ;

Difficulty: line singularity on boundary with increasing gradient;

Boundary: function value with singularity on boundary  $x_0 = 0$ ;

Reference: Mitchell 2018

**PDE 8: Interior Point Singularity**

Equation Reference: (B.19)

Characteristics: root function; a singularity within the domain;

Difficulty: not defined at  $(0.5, 0.5)$  which results in numerical inaccuracies;

Boundary: function value on boundary;

**PDE 9: Arctan Wave Front Homogeneous Boundary Conditions 2D**

Equation Reference: (B.21)

Characteristics: arctan oscillation diagonal to the domain;

Difficulty: steep gradient on the transition between wave peak and trough; sharp corners on the boundary;

Boundary: homogeneous boundary condition;

Reference: 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 ???. The limited class diagram, without any methods and attributes is shown in the figure 4.1 below. The architecture is organised in 4 main segments.

### Optimisation Algorithm

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

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 the first and the 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

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/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

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. The methods are used in the experiments chapters to interpret the results. A detailed description of the implementation is given in the appendix ??.

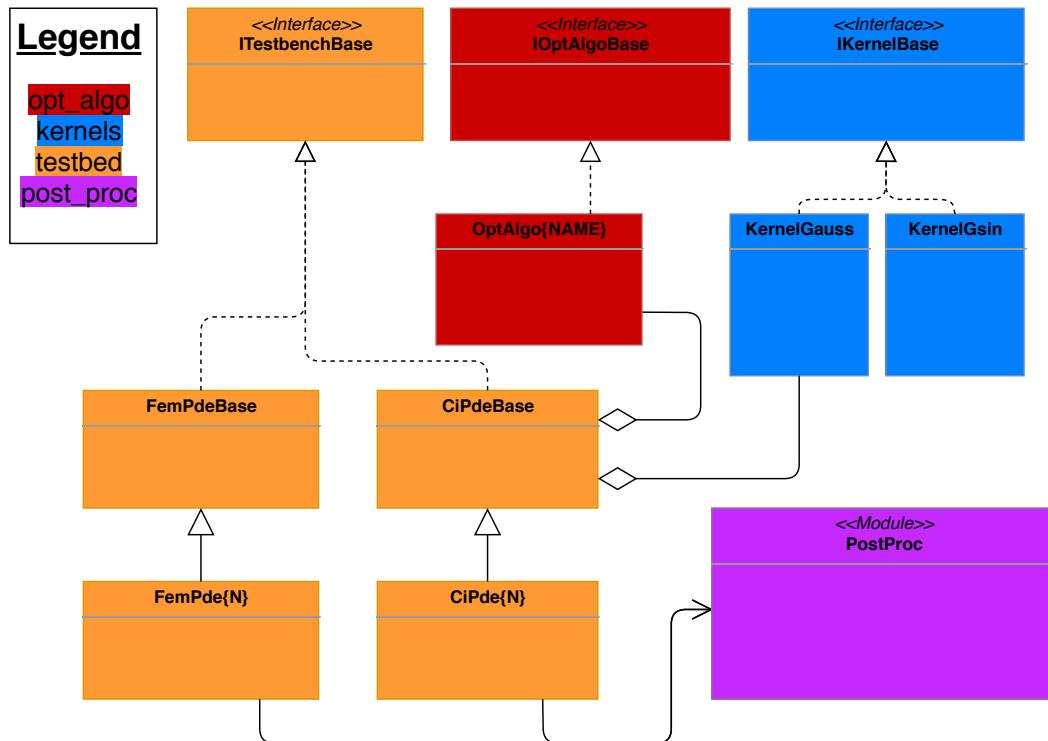


Figure 4.1: This limited UML class diagram gives a small overview of the testbed design and its interfaces and classes.

## 4.3 Performance 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 E.

### 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 E) 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  $\xi$  and  $\phi$
- the formulation of the PDE
- number of collocation points used
- number of kernels used

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

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

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, where the distribution of the points depends on random variants, subsequent measures of similar solutions could result in different quality values. Especially in solutions with a high condition number, the RMSE 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 4.2.

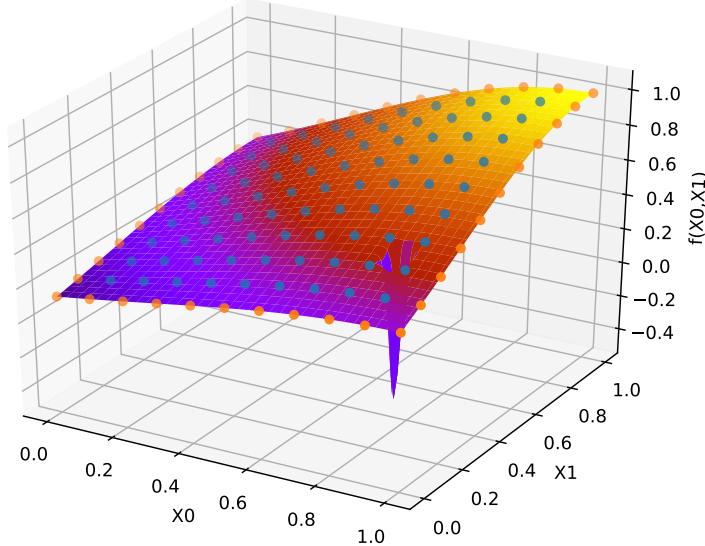


Figure 4.2: Aliasing Error: sharp inaccuracies in between collocation points

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.

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

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 Hochsteiger 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  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  or with a negative sign  $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$  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 inhomogeneous part of the PDE.

$$\int_{\Omega} \nabla u^T \nabla v dV = \int_{\Omega} f v dV \quad (4.3)$$

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  $5 \cdot 10^4$  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 4.3 and 4.4, respectively. These datasets provide a reference point for the performance of the CI solver. 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 within a readable scale, this PDE is omitted and plotted in a separate figure.

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	L2 Norm
0A	$2.967 \cdot 10^{-5}$
0B	$1.071 \cdot 10^{-5}$
1	$8.004 \cdot 10^{-7}$
2	$3.501 \cdot 10^{-8}$
3	$1.680 \cdot 10^{-9}$
4	$4.764 \cdot 10^{-7}$
5	$6.057 \cdot 10^{-6}$
6	$1.908 \cdot 10^{-7}$
7	$5.203 \cdot 10^{-5}$
8	$3.237 \cdot 10^{-7}$
9	$2.366 \cdot 10^{-7}$

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.

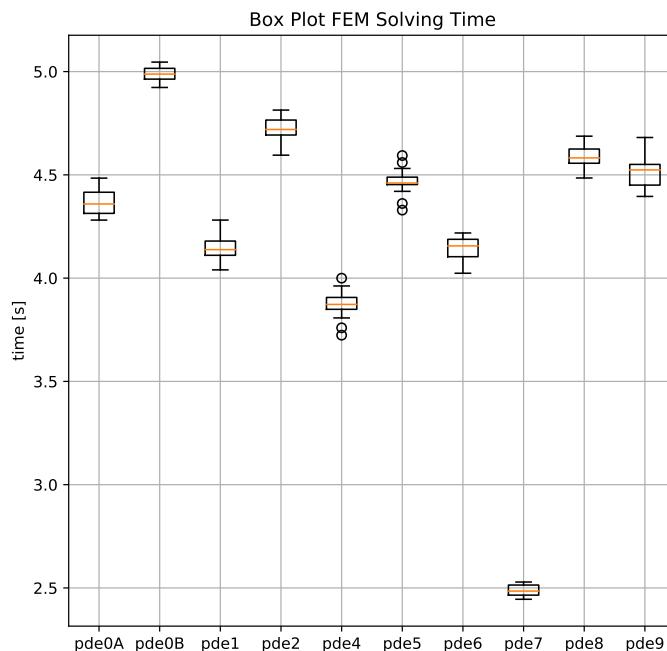


Figure 4.3: Boxplot: time (in seconds) needed to solve the testbed PDE (without PDE3)

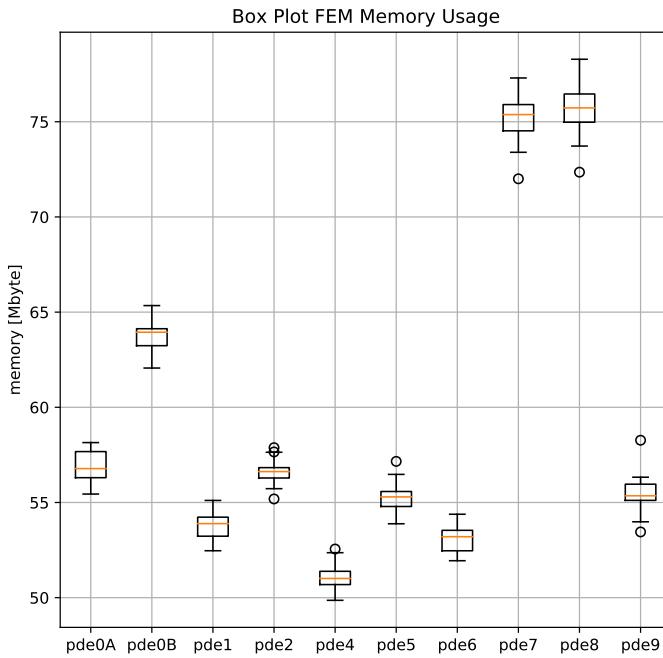


Figure 4.4: Boxplot: memory (in Mbyte) needed to solve the testbed PDE (without PDE3)

The following figures 4.6a and 4.6b show the boxplot of the time and memory consumption for the testbed PDE 3 with  $5 \cdot 10^4$  DOF. 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. One possible explanation for that is the fundamental 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. In fact, it should be possible to use less DOF and still reach a similar solution quality. Figure 4.5 shows the performance metrics at different DOF budgets. The plot confirms this hypothesis, that less DOF take less time and memory to solve the PDE and still end up with the same quality.

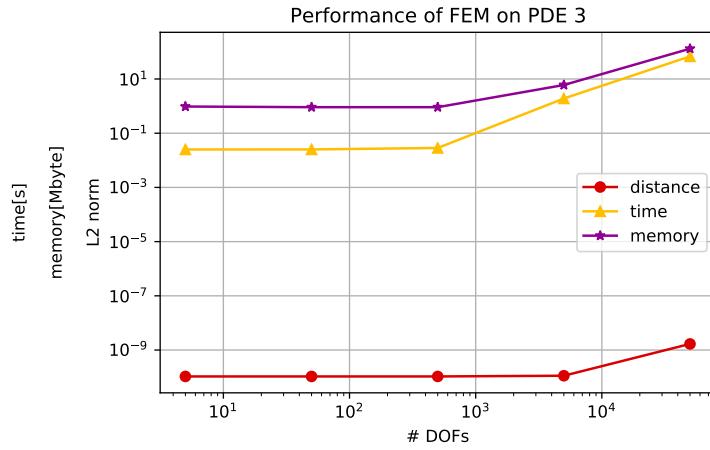
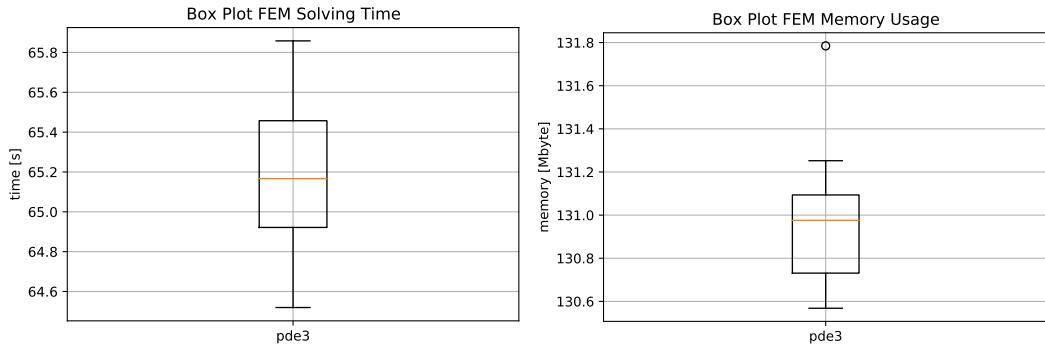


Figure 4.5: PDE 3 performance metrics at different DOF budgets.



(a) Boxplot: time to solve testbed PDE3 at  $5 \cdot 10^4$  DOF. (b) Boxplot: memory to solve testbed PDE3 at  $5 \cdot 10^4$  DOF.

 Figure 4.6: Comparison of time and memory effort on PDE 3 at  $5 \cdot 10^4$  DOF.

## 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 are be determined by preliminary test. If not stated otherwise, the following parameters from table 4.2 are used in the subsequent experiments.

Parameter	Value	Chaque and Carmona 2019
$\varphi$	100	300
$\kappa$	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	$\frac{81}{121 = 11 \times 11}$	100 equally spaced points over the domain
initialisation	$\vec{u}_{apx} \in \mathcal{N}(0, 1)$	$\omega_i \in \mathcal{U}[-0.01, 0.01]$ $\gamma_i \in \mathcal{U}(0, 1)$ $c_{ik} \in \mathcal{U}[2\Omega]$

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

The parameters  $\varphi$  and  $\kappa$  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  $\xi$  and  $\varphi$ . 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 Suganthan 2008 describe an empirical study on choosing 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 optima, 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.

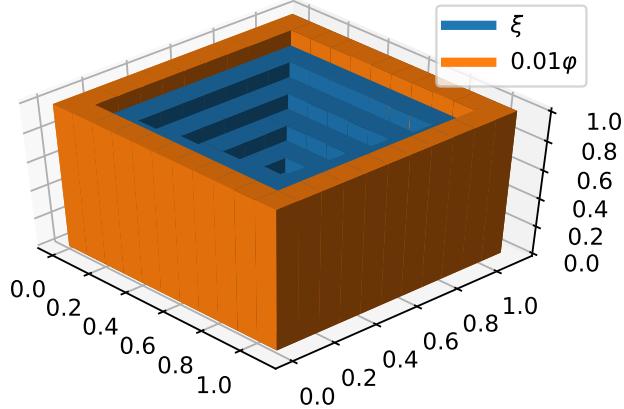


Figure 4.7: Weighting factor  $\varphi$  and  $\xi$  on every collocation point

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.

Chauquet 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 4.8.

The initialisation is done by a standard normal distribution. This means that every value in the  $param_{uapx}$  vector is drawn from a normal distribution. On the contrary, Chauquet 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.

## 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 table 4.3 shows the properties of both machines. It is important to note that these information are only a

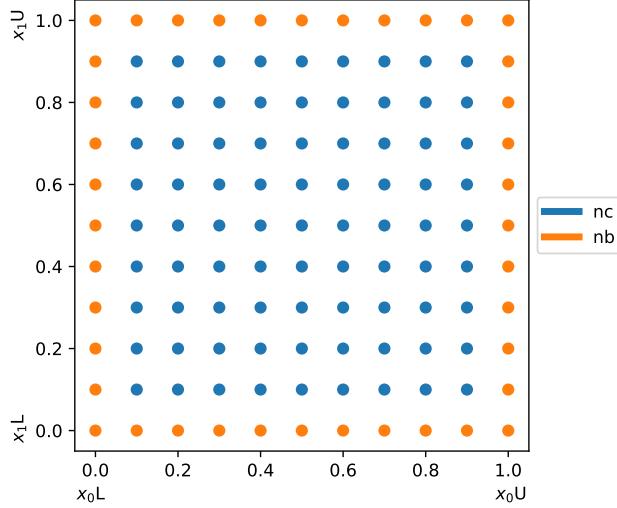


Figure 4.8: Collocation points used in the testbed. For the PDEs 0A and 0B with the larger domain, the lower limits  $x_0L$  and  $x_1L$  are replaced with -2 and the upper limits  $x_0U$  and  $x_1U$  are replaced with 2. The points are still equally spaced.

reference point. This is probably not enough to actually recreate the exact time or memory data presented in this work.

Property	Machine 1	Machine 2
Operating System	Windows 10 Home 1909 18363.900	CentOS Linux release 7.3.1611
Python	Anaconda version 2019.03	Anaconda version 2020.02
Processor	Intel Core i7-4790K CPU @ 4.00GHz	Intel Xeon CPU E5-2630 v3 @ 2.40GHz
Cores	4 Cores + 4 logical	32 Cores

Table 4.3: Comparison of the machines used for the experiments.

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.

# 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 Downhill Simplex (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. The SciPy implementation of the DS is used (SciPy 2020a).

---

Algorithm 5.1: Pseudocode of memetic JADE

---

```
1 Function memeticJADE(X, funct, minErr, maxFE):
2     dim, popsize  $\leftarrow$  size(X)
3     p  $\leftarrow$  0.3
4     c  $\leftarrow$  0.5
5     pop, FE, F, CR  $\leftarrow$  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 a 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 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.

To validate the results, a statistical significance test is performed. The test-function is based on the Wilcoxon test. The Wilcoxon test checks for significance at  $\alpha = 0.05$ . To decide if the results are better or worse, the mean and the median are compared. If both are smaller, the results are better. Contrary, if both are larger, the results are worse. If only one of the values is smaller, the results are undecided. The implementation is further explained in appendix D.

## 5.3 Results

Table 5.1 compares the obtained results with results from previous research papers. Therefore, the common testbed functions PDE 2 and PDE 3 are used. It is important to notice that the parameters used to obtain the results might not coincide. Tsoulos and Lagaris 2006 also use these two PDEs, but their paper did not provide any usable error metric.

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

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 the appendix D, is performed. Remarkable is that the results on PDE 5 get significantly worse when more #FE are used.

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

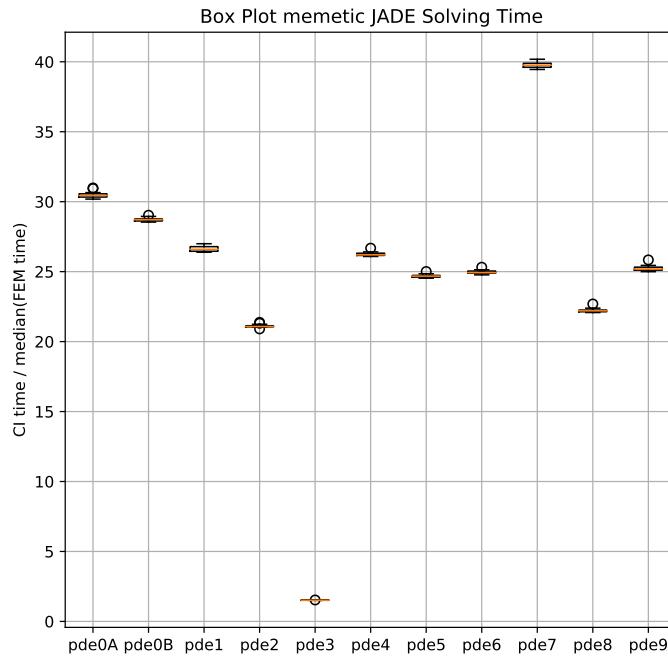


Figure 5.1: Relative solving time results of memetic JADE after  $10^4$  #FE.

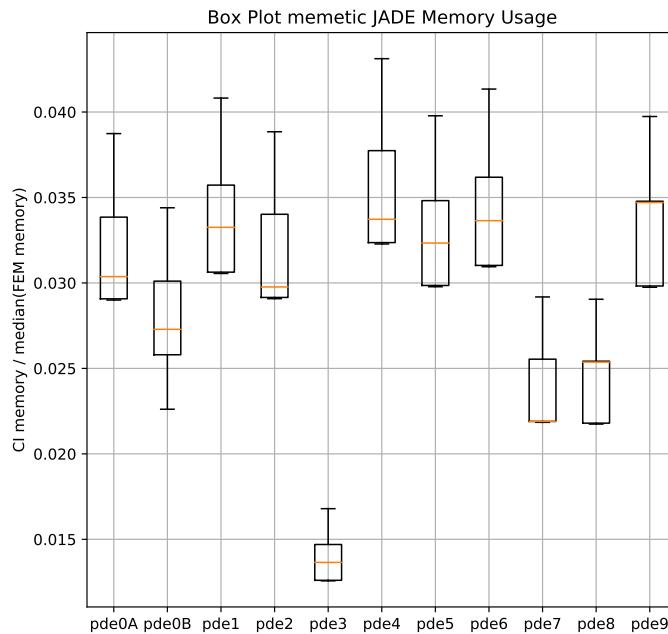


Figure 5.2: Relative memory usage results of memetic JADE after  $10^4$  #FE.

## 5.4 Discussion

The results presented in chapter 5.3 above are discussed on the following pages. A comparison to the current literature is drawn. The solving time and memory consumption in relation to the FEM solver NGSolve is examined. Especially interesting are the results obtained on PDE 0A and PDE 5.

### 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 papers, especially the results presented in Chaquet and Carmona 2019. There might be a few reasons for that:

- As stated in the previous table 4.2, the penalty and weighting factor on the collocation points are different. These settings have been chosen on the basis of preliminary experiments, but there is no guarantee that the choice of these parameters is optimal. Setting these values is not a trivial task and must be tailored to the optimisation algorithm and the differential equation.
- Since PDE 0A is defined as a combination of 5 GaK, at least so many kernels must be provided to the solver. This setup results in a greater search dimension, as compared in table 5.1. In general, more kernels result in a better solution, which is confirmed by experiments in Chaquet and Carmona 2019 and in chapter 7.4.4. To reduce the computational effort per generation, and thus allow JADE to adapt the internal parameters for a longer period, the population size is set to  $2 \cdot dim$ . According to Mallipeddi and Suganthan 2008, the population size should not be lower than that. Again, choosing these parameters is not necessarily simple and the current setting could influence the convergence to the worse.
- Compared to Chaquet and Carmona 2019, the #FE-budget for the local DS search is smaller. Thus, the algorithm puts more emphasis on the exploration. However, due to the internal parameter-adaption JADE can perform both - exploration and exploitation. Preliminary experiments have shown that the direct search converges fast with very little progress after more than 100 #FE.
- JADE might simply be not as well suited for the problem as e.g. a CMA-ES.

### 5.4.2 Solving 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 and the corresponding function values as well as the F and the CR history

are recorded at every generation. Therefore, the memory usage scales linearly with the number of function evaluations used. This information is not necessary for the actual algorithm, but helpful for evaluating the results. In later implementations, this could be disregarded in order to even further reduce the memory consumption. Depending on the testbed PDE, only 1.5 to 4.0 percent of the memory used by the FEM solver is needed.

The CI solver takes somewhere between 3 and 40 times as long as the FEM solver (figure 5.1) to perform  $10^4$  #FE. Since the solving time scales linearly,  $10^6$  #FE take about 10 times as long. More important, the CI solver uses less memory on all problems (figure 5.2). 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 requires the longest absolute evaluation time ( $29 \cdot 5$  s  $\approx$  145 s), compared to the other PDEs, it reaches a “fairly” good quality.

### 5.4.3 PDE 0A

The purpose of this PDE is to show that the solver would converge globally towards the analytical solution, if it can be represented by a finite number of kernels. However, 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 common for the CI solver to result in different functions. A typical phenomenon is that the obtained approximation has some of its Gauss “bumps” outside of the domain  $\Omega$ . The comparison of two solutions with  $10^4$  #FE and  $10^6$  #FE in figure 5.3 shows this behaviour. Since the results improve from  $10^4$  to  $10^6$ , it is possible that the results from  $10^6$  #FE can be refined with an even larger #FE budget. However, this is not tested due to the already extensive computational effort.

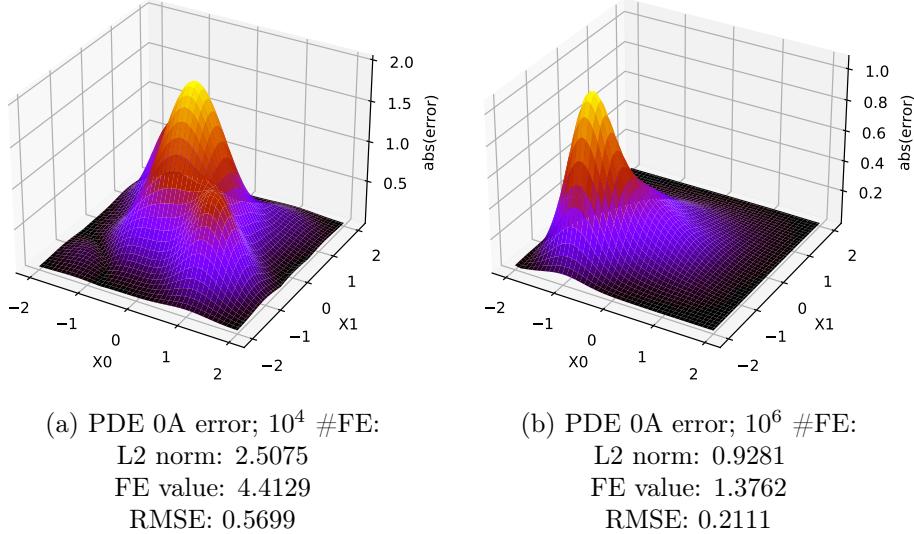
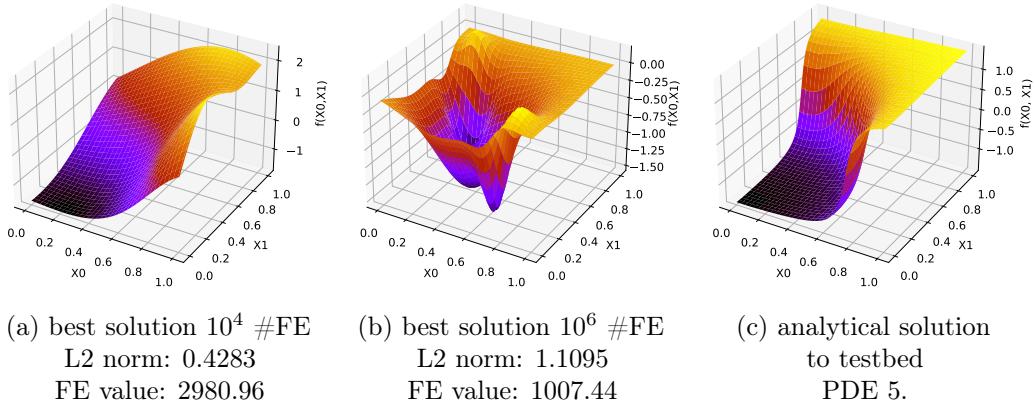


Figure 5.3: Comparison of two typical PDE 0A solutions.

#### 5.4.4 PDE 5

A fascinating property of PDE 5 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 table 5.2. This property can even be concluded from a visual perspective. A comparison of the best solution after  $10^4$  #FE and the best solution after  $10^6$  #FE is shown in figure 5.4. The solution after  $10^4$  #FE describes the global structure more accurately. It seems that the correct description of the boundary points gets lost with more #FE.


 Figure 5.4: Comparison of best solution after  $10^4$  and  $10^6$  #FE.

The distributions of the achieved fitness value after  $10^4$  and  $10^6$  #FE is shown in figure 5.5a. They are clearly separated with distinct mean and median. As expected, both the mean and the median after  $10^6$  #FE are significantly smaller than these values after  $10^4$  #FE. The corresponding L2 norm distributions are plotted in figure 5.5b. As described by the Wilcoxon test, the results are inverted. The mean and the median of the L2 norm after  $10^4$  #FE are smaller than the ones after  $10^6$  #FE.

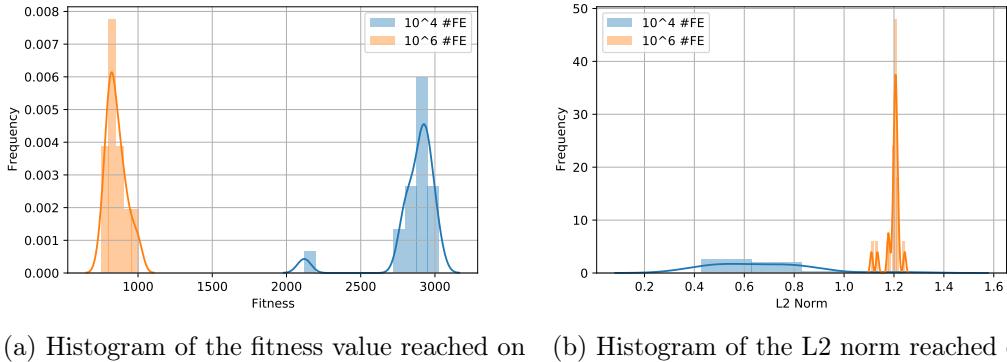


Figure 5.5: Histograms of the fitness and the L2 norm reached with  $10^4$  #FE and  $10^6$  #FE on PDE 5.

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, due to the greedy selection used in JADE (line 15 in the pseudocode A.1). The monotonically decreasing fitness value at every generation of an exemplary individual within the population is plotted in figure 5.6. Because the L2 norm is not the property that gets optimised, the quality of an individual at every generation is not necessarily monotonically decreasing. The corresponding L2 norm of the same individual is plotted alongside its fitness in figure 5.6.

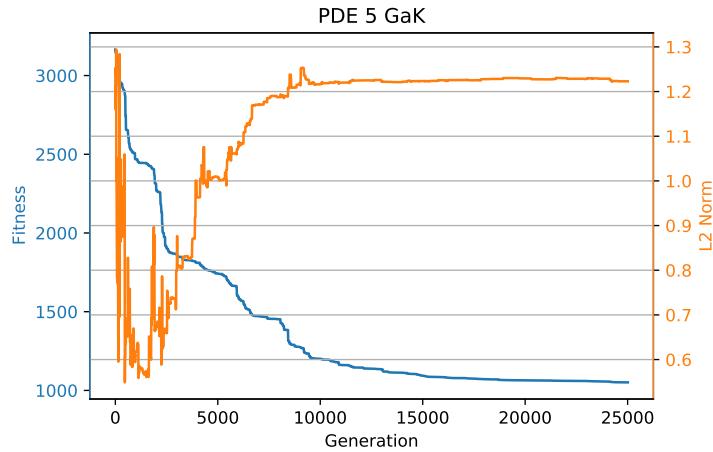


Figure 5.6: Fitness value and L2 Norm of one individual at every generation on PDE 5.

This indicates that the current fitness function does not fully describe the optimisation problem. Some possible solutions to this problem could be:

- An adaptive number of kernels reduces the dimensionality of the problem and only increases the kernel number if necessary.
- A different kernel type that better resembles the structure of the solution could be used.
- A denser grid of collocation points alters the fitness function. This introduces more points that might sit at more interesting positions within the domain. However, this also increases the computational effort to evaluate the fitness function. Alternatively, a scheme that adapts the collocation points during the optimisation process could balance out the computational effort and interesting probing locations.

An adaptive kernel scheme is proposed in chapter 7. The GSK kernel type is examined in chapter 8.

# 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 of the new algorithm.

## 6.1 Hypotheses

The actual parallel JADE algorithm is shown in appendix F. The pseudocode F.1 is very similar to the JADE algorithm A.1. The main differences are laid out in the lines 11 to 17. 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 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 influence the results, to the better or the worse. Further, the time-benefits of this strategy are examined and assessed.

---

Algorithm 6.1: Pseudocode of memetic parallel JADE

---

```

1 Function memeticJADE( $\mathbf{X}$ ,  $funct$ ,  $minErr$ ,  $maxFE$ ):
2    $dim, popsize \leftarrow size(\mathbf{X})$ 
3    $p \leftarrow 0.3$ 
4    $c \leftarrow 0.5$ 
5    $pop, FE, F, CR \leftarrow pJADE(\mathbf{X}, p, c, funct, minErr, maxFE - 2dim)$ 
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) are used, whereby 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. The standard Wilcoxon test (explained in appendix D) is used to proof statistical significance.

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 Results

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

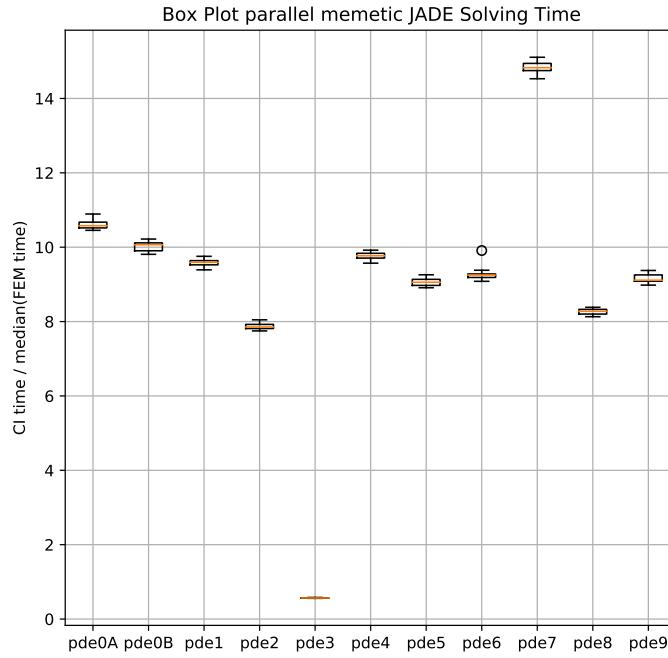


Figure 6.1: Relative solving time results of parallel memetic JADE after  $10^4$  #FE.

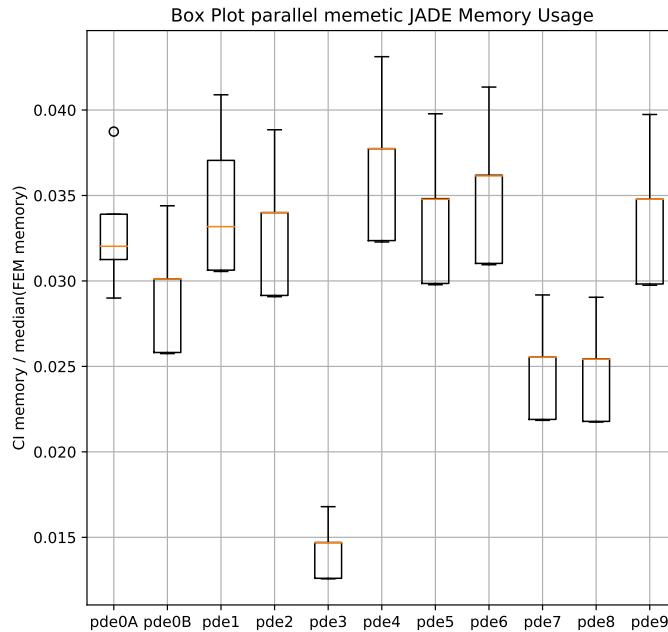


Figure 6.2: Relative memory usage results of parallel memetic JADE after  $10^4$  #FE.

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		Wilcoxon Test
	stat	mean	median	mean	median
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	1.266E-17	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 table 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 analysed in this chapter. The time and memory consumption as well as the speed-up for every testbed function is discussed. Further, the significant worse results on PDE 4 are investigated. Finally, the usage of the parallel JADE in all further experiments is justified.

### 6.4.1 Memory Usage

As the boxplot 6.2 shows, the memory consumption is roughly at the same level, somewhere between 1.5 to 4.0 percent of FEM solver. The memory usage is very similar to the serial experiment. This is expected, since memory-wise no integral changes have been performed.

### 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 execution time after  $10^4 \text{ #FE}$  now ranks between the 8 and 15-fold of the FEM solver. This is unquestionably faster than the solving time of the serial algorithm. Remarkable is that on PDE 3, the CI solver is even faster than NGsolve and takes only about  $0.56 \cdot 65.2 \text{ s} \approx 36.5 \text{ s}$ . Of course, to compete with the solution quality, more  $\text{#FE}$  are needed, which increases the execution time.

Image 6.3 shows the calculated mean Speed-Up (sp) over 20 replications, that is accomplished by substituting the serial JADE with the parallel version. The 95% confidence interval is shown in yellow.

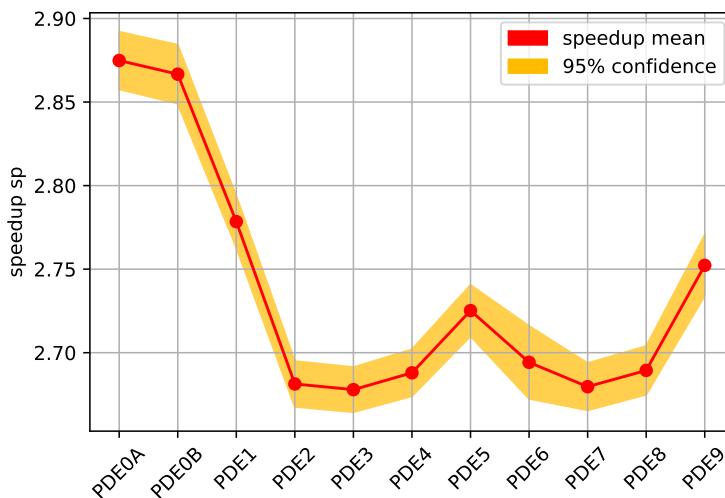


Figure 6.3: Mean speed-up of parallel JADE with 95% confidence interval.

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

### 6.4.3 PDE 4

Table 6.1 shows that the L2 norm achieved on PDE 4 is significantly worse when the parallel JADE is used. 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 fitness 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 a visually smaller error.

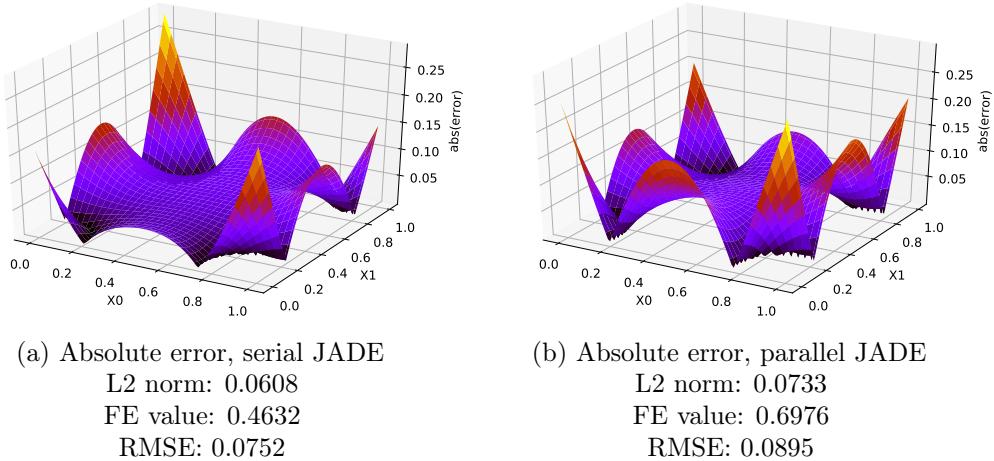


Figure 6.4: Comparison of the absolute error of the worst solution on PDE 4 by a parallel and a serial memetic JADE at  $10^6$  #FE

In order to make the results easier to interpret, the histogram from figure 6.5 shows the L2 norm data from table 6.1. The best solutions 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.

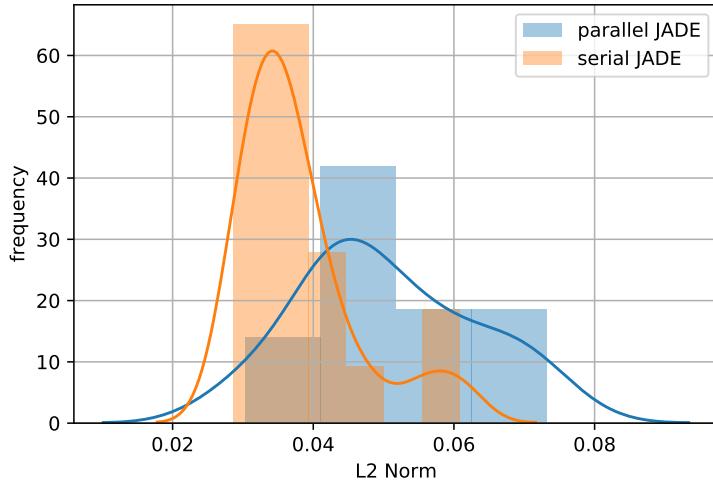


Figure 6.5: Histogram of the L2 norm data obtained by the serial and the parallel JADE on PDE 4 from table 6.1.

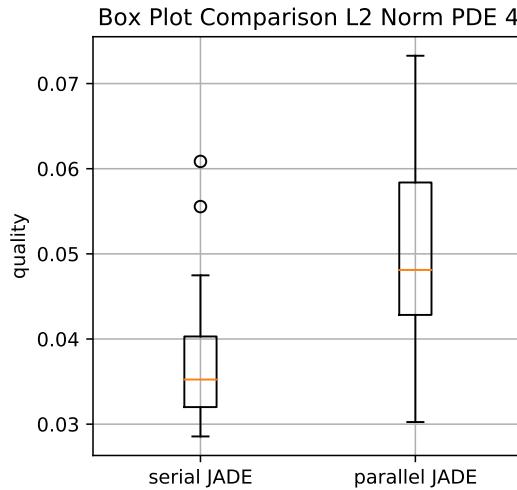


Figure 6.6: Boxplot of the L2 norm data from table 6.1 on PDE 4.

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 applied in all 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 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 adapt the number of kernels along the solving process, instead of arbitrarily using 5 kernels.

## 7.1 Hypotheses

The idea tested in this chapter 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 algorithm before. The pseudocode G.1 in the appendix G is extended by a so called “state detector”. Ideally, the state detector (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 parameter. The Delay Time ( $dT$ ) represents the number of generations over which the best function value must remain unchanged. It can also be thought of a buffer-time that allows the DE parameters  $F$  and  $CR$  to self-adapt. Further, the  $\text{minError}$  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. A flowchart of the process is shown in figure 7.1. The corresponding pseudocode H.1 in the appendix describes the implementation.

The algorithm always starts with one kernel. From there on, the number of kernels is increased. After the “state detector” has stopped the paJADE, the DS is employed on the best individual of the last population. If the last JADE/DS cycle was able to improve the function value, it is assumed that the best solution for that dimensionality is found. Thus, to further improve the results, the number of kernels must be increased. If the function value could not be decreased, a restart around

the previous best population is performed. Since the number of kernels is always increased by one, the previous best population consists of one kernel less.

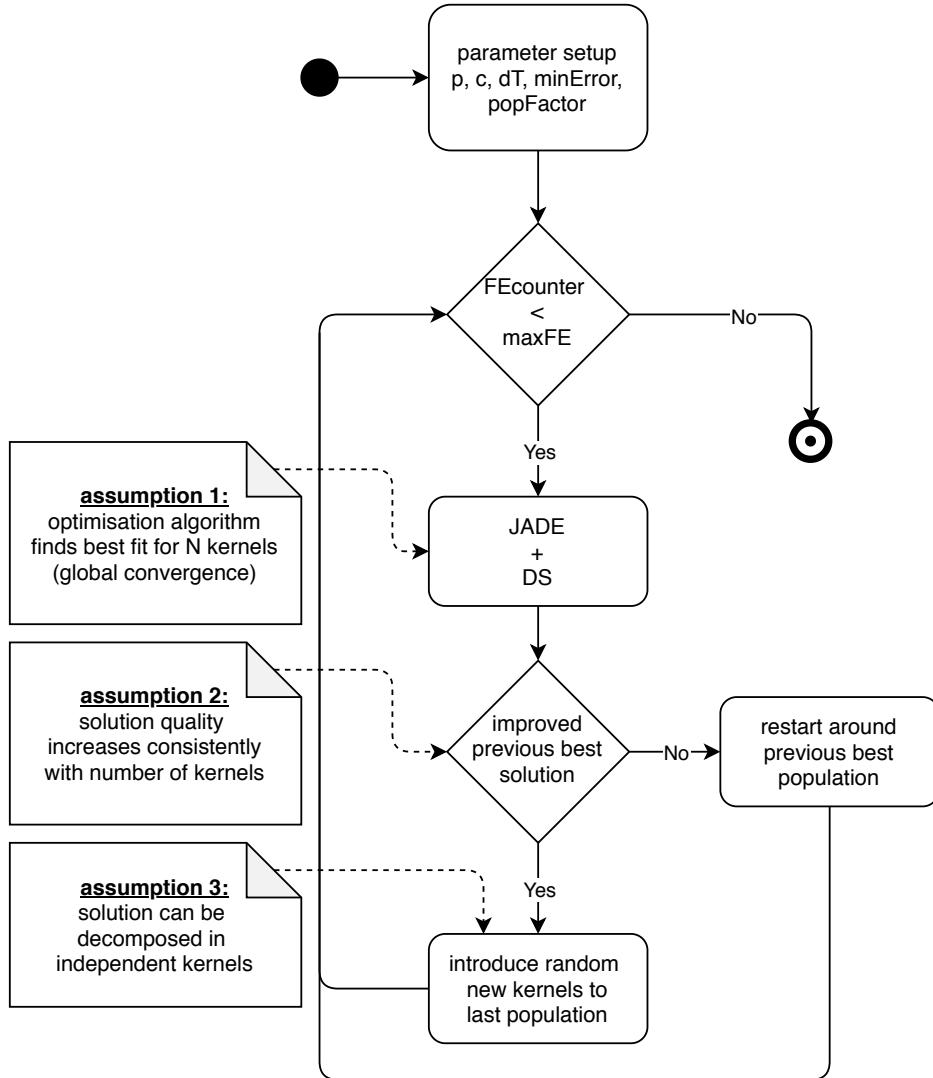


Figure 7.1: Flowchart of the adaptive kernel scheme.

This adaptive scheme operates under 3 strong assumptions. To reduce their possible negative impact, corresponding counter-strategies are implemented.

- **Assumption 1:** The optimisation algorithm (JADE + DS) finds (a close approximation to) the global optimum. This would be the best approximation of the solution by  $N$  kernels. Obviously, this property is not necessarily true. Currently, only a very limited amount of research papers exist, that discuss global convergence properties of DE (Hu et al. 2013, Opara and Arabas 2019). To counteract this assumption, restarts are performed.

- **Assumption 2:** The theoretically best achievable solution quality increases with the number of kernels. After a maximum number of kernels is reached, the quality can not be surpassed. Based on this assumption, the algorithm starts with one kernel and the dimensionality increases by only one kernel at a time. To catch the point after which too many kernels are used, the dimensionality reduction step is introduced.
- **Assumption 3:** The best approximation to a particular problem can be decomposed into independent kernels. This means, that the first kernel would not influence the next one. This premise would allow to simply include a new kernel without altering the kernels that have already been found. Again, this is not true for every PDE. The concept is similar to the one explained in Chaquet and Carmona 2012. Contrary to their work, the first kernels in the lower dimensions are not fixed. This means that the old kernels are merely used as a starting point and all kernels are allowed to adapt to the new setting.

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

Again, as in the experiments before, machine 1 runs at  $10^4$  #FE and machine 2 performs  $10^6$  #FE. The time- and memory-comparison are done on machine 1. The number of kernels is adapted, but the algorithm starts with 1 GaK. Thus, the dimension is 4 and the population size is 8. The population size gets corrected if the number of kernels changes. The Wilcoxon test from appendix D investigates the statistical significance.

The two new parameters  $dT$  and  $minError$  must be set. The  $minError$  is again set to 0. The delay time  $dT$  is set to 100. When the function value could not be improved over the course of 100 generations, a new kernel is introduced. This choice is rather arbitrary and depending on the PDE, different values might be more successful. However, this property is not analysed in the current experiment.

## 7.3 Result

Similar to the experiments before, the two images 7.2 and 7.3 illustrate the time and memory consumption of the described kernel adaptive JADE. The data to both plots is obtained on machine 1 at  $10^4$  #FE. PDE 7 takes the longest to solve, about 8.5 times longer than the FEM solver. Contrary, PDE 3 is solved the fastest, where the CI solver only needs about a quarter of the time used by FEM solver. The memory

effort of the CI solver over all testbed problems is between 0.5 and 2.5 percent of the memory needed by the FEM solver.

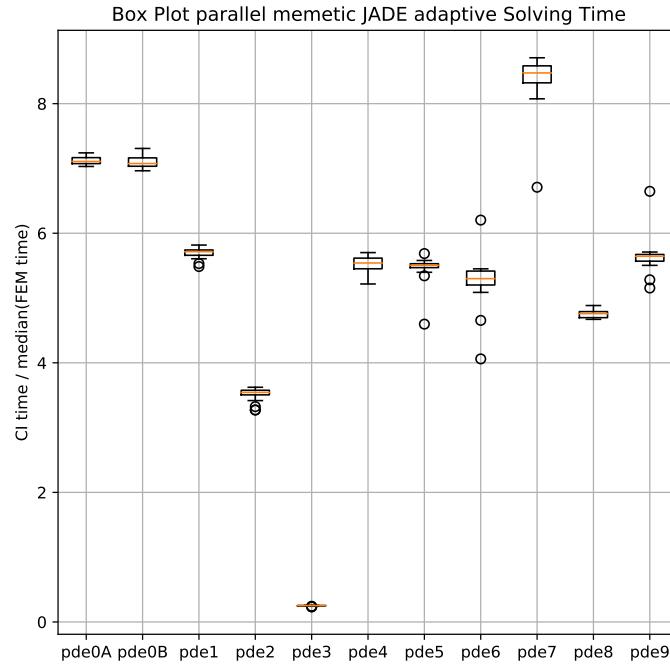


Figure 7.2: Relative solving time results of parallel memetic JADE with adaptive kernels after  $10^4$  #FE.

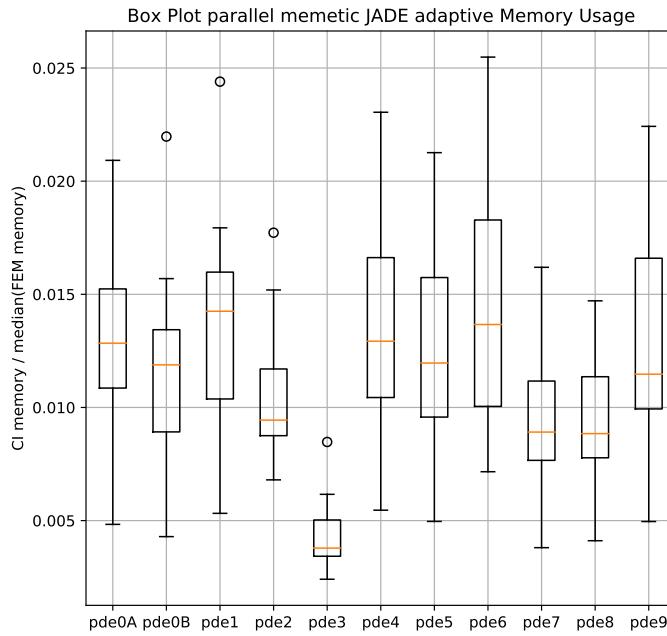


Figure 7.3: Relative memory usage results of parallel memetic JADE with adaptive kernels after  $10^4$  #FE.

The table 7.1 shows the L2 norm data obtained by the adaptive JADE and compares them against the results from the parallel JADE. The Wilcoxon test indicates mixed results. The adaptive kernel scheme works fine on the PDE 0A, but it also produces significantly worse results on the problems PDE 2, 3, 4 and 7.

Algorithm	parallel JADE $10^6$ #FE		adaptive JADE $10^6$ #FE		Wilcoxon Test
	stat	mean	median	mean	median
PDE 0A	$0.6939 \pm 0.6635$	0.9243	$9.694E-16 \pm 1.486E-16$	$9.255E-16$	sig. better
PDE 0B	$0.2809 \pm 0.3071$	0.2035	$0.2380 \pm 0.0572$	0.2607	unsig. undecided
PDE 1	$0.0239 \pm 0.0467$	0.0146	$0.0116 \pm 0.0061$	0.0084	unsig. better
PDE 2	$0.0300 \pm 0.0157$	0.0255	$0.0735 \pm 0.0358$	0.1034	sig. worse
PDE 3	$0.0371 \pm 0.0206$	0.0295	$0.1731 \pm 0.0395$	0.1822	sig. worse
PDE 4	$0.0505 \pm 0.0121$	0.0481	$0.0707 \pm 0.0053$	0.0720	sig. worse
PDE 5	$1.2030 \pm 0.0465$	1.2053	$122.6312 \pm 372.5676$	1.1643	unsig. undecided
PDE 6	$0.5814 \pm 1.3550$	1.266E-17	$0.4428 \pm 1.0980$	$1.266E-17$	unsig. undecided
PDE 7	$0.0228 \pm 0.0025$	0.0226	$0.0513 \pm 0.0442$	0.0231	sig. worse
PDE 8	$0.2167 \pm 0.0017$	0.2169	$0.2144 \pm 0.0044$	0.2128	unsig. better
PDE 9	$0.0426 \pm 0.0115$	0.0463	$0.0483 \pm 0.0149$	0.0468	unsig. worse

Table 7.1: Comparison of the achieved L2 norm by the pJADE and the paJADE at  $10^6$  #FE. The pJADE data is directly taken from table 6.1.

Since the number of kernels is not predefined, the resulting solution may have any amount of kernels. This is also a random variant, and the results are shown in the boxplot 7.4 below. As expected, this varies between different testbed problems, but it can not be less than 1 kernel.

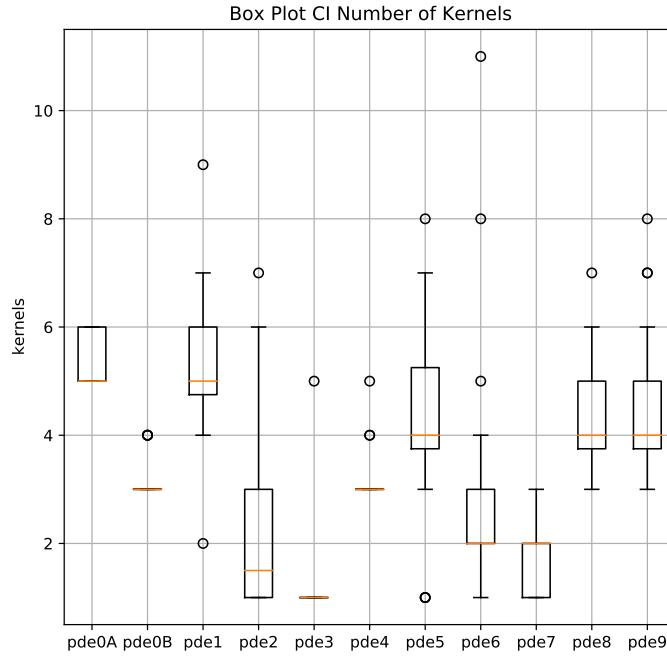


Figure 7.4: Number of kernels of the proposed solution by the paJADE per PDE at  $10^6 \#FE$

## 7.4 Discussion

This chapter analyses the presented results from section 7.3. The time and memory consumption are discussed. The significantly better results on PDE 0A are investigated. The adaptive kernel scheme performs significantly worse on the testbed problems PDE 2, 3, 4, and 7. Therefore the *minError* parameter is adapted and the results are critically examined.

### 7.4.1 Solving Time

This new adaptive strategy can significantly decrease the time that is necessary to solve any PDE of the testbed. This follows from the solving time box plot in figure 7.2. This effect can be explained by the reduction of the dimension and the adaptive population size. Depending on the number of kernels, the population can

grow. It always starts with 1 kernel and therefore the size of the population is 8. The boxplot shows that this breaks the pattern of simpler and more complex PDEs, that is exhibited in the previous experiments.

### 7.4.2 Memory Usage

The adaptive approach also leads to a reduction in the memory usage, as described by the memory boxplot in figure 7.3. The same explanation that describes the solving time reduction can applied here: smaller populations need less memory to be stored. Since not all PDEs result in the same dimension, the memory consumption over all testbed problems is not the same. Further, the adaptive scheme dismantles the linear scaling of memory usage with increasing #FE.

### 7.4.3 PDE 0A

As noted before, the testbed PDE is especially designed to be solved by 5 GaK. The common problem, that not all kernels are established, is solved by the adaptive strategy. The kernel boxplot in figure 7.4 shows that all 20 replications generate at least 5 kernels. However, some solutions are composed of 6 kernels, but this has only a limited effect on the numerical value of the solution quality. Generally, 6 kernels tend to produce worse solutions. The results by the CI solver can even compete with the FEM solver results from table 4.1.

There are mainly three things that can happen to the unnecessary kernel:

- Put the kernel far outside of  $\Omega$  with  $c_0$  and  $c_1$ , so that the influence within the domain is negligible.
- Reduce the scaling factor  $\varphi$  to near 0, therefore the kernel has a small influence compared to the others.
- Increase the exponent  $\gamma$  so that the kernel is very sharp and the influence dies out rapidly with  $r$ .

Figure 7.5 shows the 3D plot of two separate unnecessary kernels produced by the adaptive memetic pJADE. A kernel with increased  $\gamma$  is not included in the dataset.

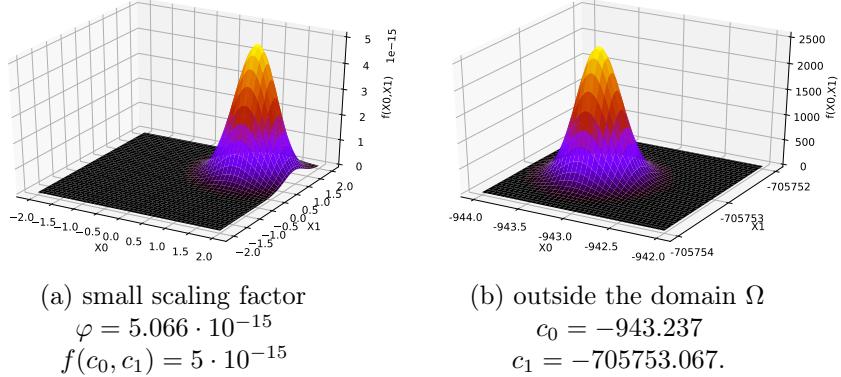


Figure 7.5: 3D plot and comparison of unnecessary 6<sup>th</sup> kernels on PDE 0A.

#### 7.4.4 PDE 2 3 4 7

The Wilcoxon significance test of table 7.1 shows that the adaptive scheme is worse for the PDEs 2, 3, 4 and 7. The boxplot 7.4 shows that on these test problems the solver frequently results in a smaller number of kernels, where the majority of runs even produce less than 5 GaK. This phenomenon points towards a shared problem where the solver does not increase the number of kernels consistently.

Figure 7.6 plots the solution quality against its number of kernels. It is clearly shown that on these PDEs, more kernels strongly correlate with a better quality.

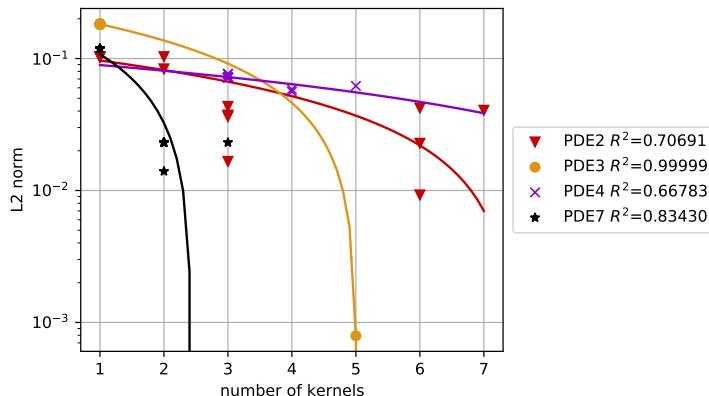


Figure 7.6: Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are taken at  $10^6$  #FE on the PDEs 2, 3, 4 and 7. The corresponding  $R^2$  values of the linear regression are denoted.

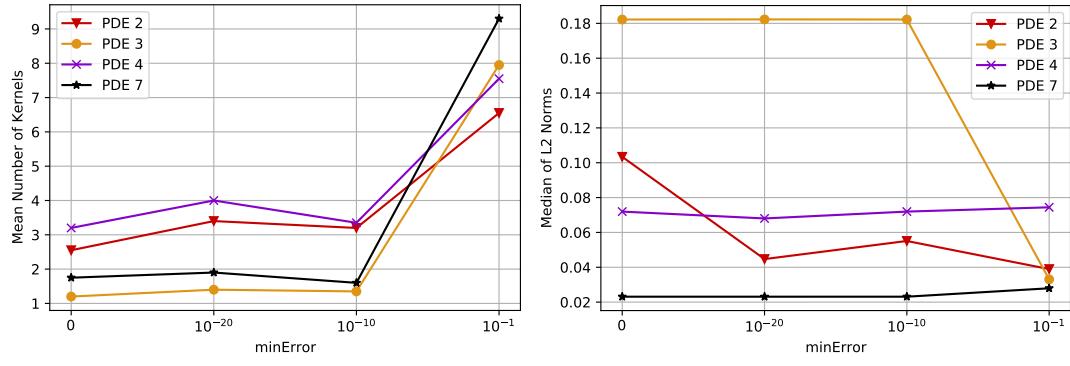
It seems that JADE exploits some areas long enough so that it does not terminate due to convergence. Thus, the number of kernels is not increased, which leads to a

poor approximation quality. The kernel bar plot of the best and the worst replication for these 4 PDEs is shown in the appendix I and confirms this hypothesis. A simple solution to mitigate this issue might be to adjust the parameters of the “state-detector”. In this experiment,  $\text{minError} = 0$  is used, however it might be beneficial to allow small changes in the function value and still terminate.

#### **Parameter Adaption: $\text{minError}$**

In this “sub-experiment” the effect of increasing the  $\text{minError}$  parameter is examined. Therefore, the same algorithm is rerun on the PDEs 2, 3, 4 and 7 at four different  $\text{minError}$  levels. Again, 20 replications are done. It is expected that the average number of kernels is increased. Simultaneously, the approximation quality should get better.

As expected, the average number of kernels in the solution gets increased. This is confirmed by the plot in figure 7.7a. Figure 7.7b shows the connection between the median L2 norm and the  $\text{minError}$ . The distance to the analytical solution decreases on PDE 2 and 3. However, this does not improve the results of PDE 4 and 7, where the quality stays roughly on the same level. This is supported statistically by the Wilcoxon test in table 7.2.



(a) Plot of the mean number of kernels against the  $\text{minError}$ . (b) Plot of the median L2 norm against the  $\text{minError}$ .

Figure 7.7: Comparison of  $\text{minError}$  against the number of kernels and the achieved solution quality.

Setup	$\minError = 0; 10^6 \text{ #FE}$	$\minError = 10^{-1}; 10^6 \text{ #FE}$			
stat	mean	median	mean	median	Wilcoxon Test
PDE 2	$0.0735 \pm 0.0358$	0.1034	$0.0418 \pm 0.0156$	0.0389	sig. better
PDE 3	$0.1731 \pm 0.0395$	0.1822	$0.0455 \pm 0.0406$	0.0331	sig. better
PDE 4	$0.0707 \pm 0.0053$	0.0720	$0.0726 \pm 0.0080$	0.0744	unsig. worse
PDE 7	$0.0513 \pm 0.0442$	0.0231	$0.0287 \pm 0.0045$	0.0279	unsig. undecided

Table 7.2: Statistical comparison of the achieved L2 norm by paJADE with  $\minError = 0$  and  $\minError = 10^{-1}$  after  $10^6 \text{ #FE}$ . The data with  $\minError = 0$  is directly taken from table 7.1.

Although the results on PDE 2 and 3 do get significantly better, the adaptive process with greater  $\minError$  introduces a larger spread of the results - both in the number of kernels and in the reached L2 norm. This can be seen in figure 7.8. Compared to the same plot at  $\minError = 0$  (figure 7.6), the coefficient of determination  $R^2$  is smaller, indicating a very poor correlation.

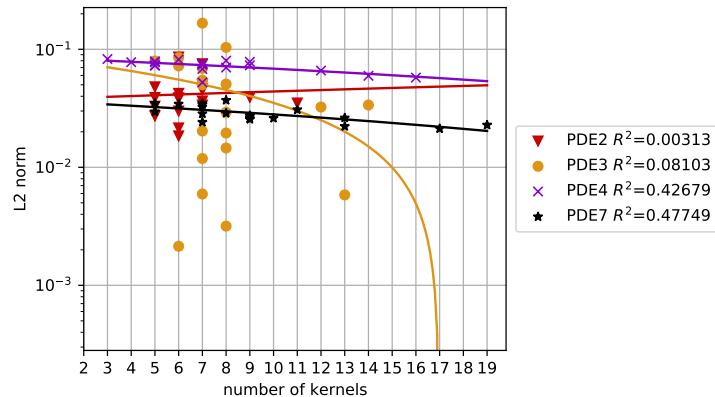


Figure 7.8: Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are produced with a  $\minError = 10^{-1}$  after  $10^6 \text{ #FE}$ .

Because JADE is terminated earlier, the exploitation is not as progressed. This trade-off needs to be managed by setting the “state-detector”-parameters. A small  $\minError$  ensures, that the exploitation is progressed sufficiently but it exceeds the  $\#FE$  budget before the number of kernels is increased. Contrary, larger  $\minError$ -values terminate faster and increase the number of kernels, but the solution is not fully exploited.

### 7.4.5 PDE 5

The results presented in table 7.1 shows an interesting observation for the testbed problem 5. The mean L2 norm of the adaptive scheme is very large, but the median is slightly smaller than the median of the non-adaptive JADE. The Wilcoxon test reveals an insignificant difference, which hints that the adaptive scheme includes some very large outliers. This is demonstrated by comparing the box plots of both L2 norm distributions in figure 7.9a. The same data is shown twice, with and without the outlier.

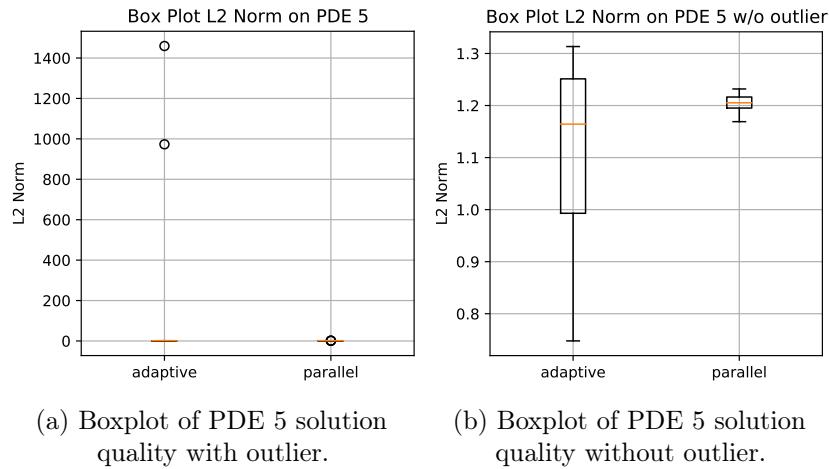


Figure 7.9: Boxplot of solution quality on PDE 5 with a budget of  $10^6 \text{ #FE}$ .  
Comparison of distribution with and without outliers.

The approximate solution of the outlier is shown in the two 3D plots in figure 7.10 below. It is clearly seen that both results have nothing in common with the actual solution from figure 7.10c. In general, it can be said that the adaptive scheme exhibits a greater spread in the quality of the solution.

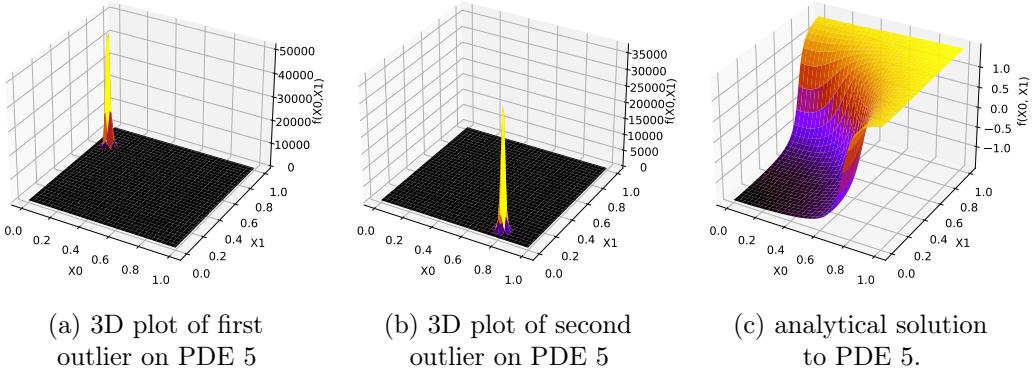


Figure 7.10: 3D plot of the outlier solution produced by the adaptive kernel scheme on PDE 5 after  $10^6$  #FE.

Due to the common structure of both outliers, a fundamental issue might be suggested. The adaptive strategy could not improve the results on PDE 5. A second possibility would be to use a different kernel type.

#### 7.4.6 PDE 6

The testbed PDE 6 could theoretically be solved exactly by a single GaK. As the Wilcoxon test shows, the adaptive kernel method does not improve the results significantly. This is confirmed in the subsequent boxplot of figure 7.11. Even good solutions are made of more than one kernel, as the boxplot in 7.4 indicates. This means that the adaptive kernel strategy is not successful on PDE 6. JADE terminates too early and introduces more kernels. The influence of these unnecessary kernels must be minimised by the same strategies as with the PDE 0A as presented in chapter 7.4.3.

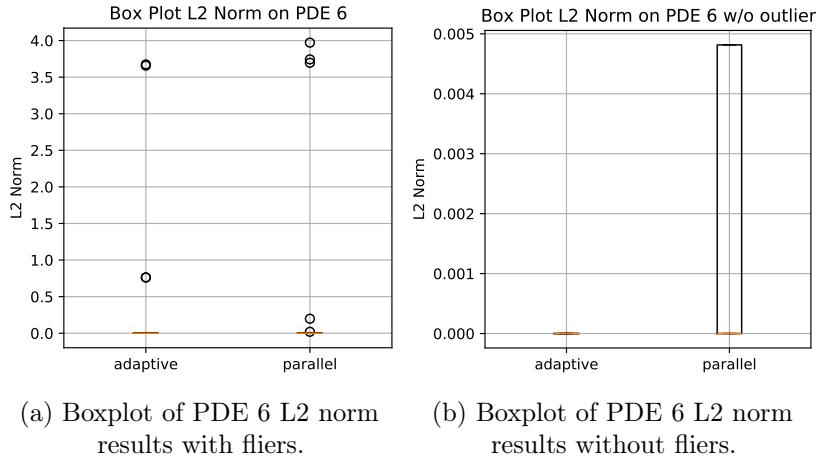


Figure 7.11: Comparison of the L2 norm reached by the paJADE and a non-adaptive pJADE on PDE 6 after  $10^6$  #FE.

Besides the good results, there are two recurring bad solutions as seen in figure 7.12. Their 3D plot shows no resemblance with the actual solution from figure 7.12c. The fact that these obviously bad approximations happen multiple times indicates that JADE converges to these local optima.

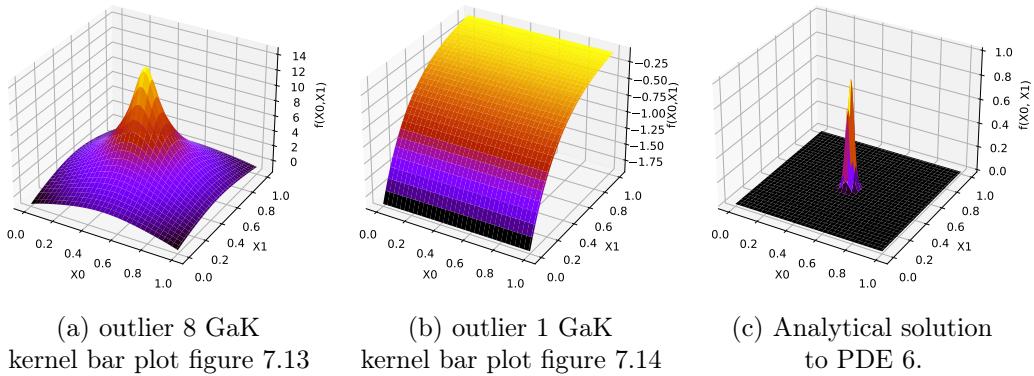


Figure 7.12: These wrong solutions are generated with paJADE after  $10^6$  #FE and represent local optima where JADE gets stuck.

Corresponding to these 3D plots, the kernel bar plot is depicted in figure 7.13 and 7.14. The kernel bar plot connects the decline of the fitness value with the adaption of the kernel number. Darker areas show a strong decline of the best fitness value over generations, lighter areas represent stagnation. A green bar indicates an increment by one kernel. Contrary, the red bar expresses a decrement by one kernel. This shows how figure 7.12a is created by increasing the kernel count, although one kernel should

## 7 Experiment 2: Adaptive Number of Kernels

---

be sufficient. Similar, figure 7.12b consists of a single kernel, where a local optimum is exploited. Both false solutions occur twice within the 20 replications.

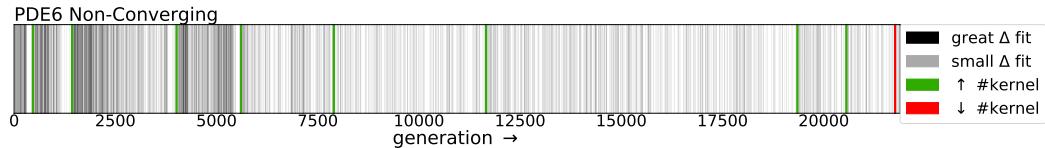


Figure 7.13: Kernel bar plot of the false solution, represented in figure 7.12a

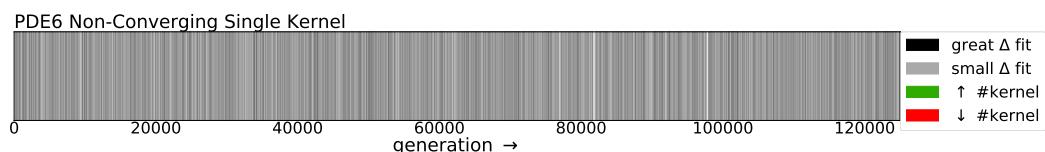


Figure 7.14: Kernel bar plot of the false solution, represented in figure 7.12b

# 8 Experiment 3: Gauss-Sinus Kernel

Up until now, the results on PDE 5 were always considerably worse than on all other testbed functions. The idea discussed in this chapter is the usage of a new kernel. The Gauss-Sinus Kernel, also abbreviated with GSK, was introduced in chapter 3.3.2. Theoretically, the kernel should be able to solve the testbed PDEs 0A and 0B exactly.

## 8.1 Hypotheses

As stated in chapter 5.4.4 attempting to solve PDE 5 with more #FE results in a worse solution quality. The adaptive kernel scheme could not improve the results. This means that the fitness function must be reconsidered. A simple approach to change the fitness function is by introducing a new kernel type. The GSK has the features of a Gauss kernel and a sine function and is potentially able to approximate more PDE solutions. The idea of the sine function is to help with the circular features of the wave front.

This experiment tries to answer the question if the new GSK kernel type can effectively improve the results on PDE 5. Further, it should at least maintain the solution quality on all other testbed PDEs.

## 8.2 Experiment Setup

As stated in the experiments chapters before, machine 1 performs the time and memory experiments at  $10^4$  #FE. Similar, machine 2 runs the experiment at the full computational budget of  $10^6$  #FE. The Wilcoxon test from appendix D is used to show statistical significance. Because the last experiment was not entirely conclusive, only a memetic pJADE without kernel-adaption is tested. Since the new kernel has 6 parameters, the dimension and the population size changes. To ensure that the algorithm is able to solve PDE 0A, 5 GSK are used. This results in a dimension of 30 parameters and a population size of 60. All other experimental parameters are taken from the table 4.2.

## 8.3 Result

The box plots in figure 8.1 compares the relative execution time of the memetic pJADE with the FEM solver. The results seen in this graph are obtained after  $10^4$  #FE on machine 1. Similar, the data in figure 8.2 compares the memory consumption of both algorithms.

The memory consumption boxplot exhibits many outliers, where some even indicate a memory usage of 0 Mbyte. This is not expected and might result from technical difficulties. Thus, when comparing the results, only the median should be considered.

The following table show the statistical test. The comparison between the GSK and the GaK on the memetic parallel JADE is shown in table 8.1. The comparison is done with a budget of  $10^6$  #FE.

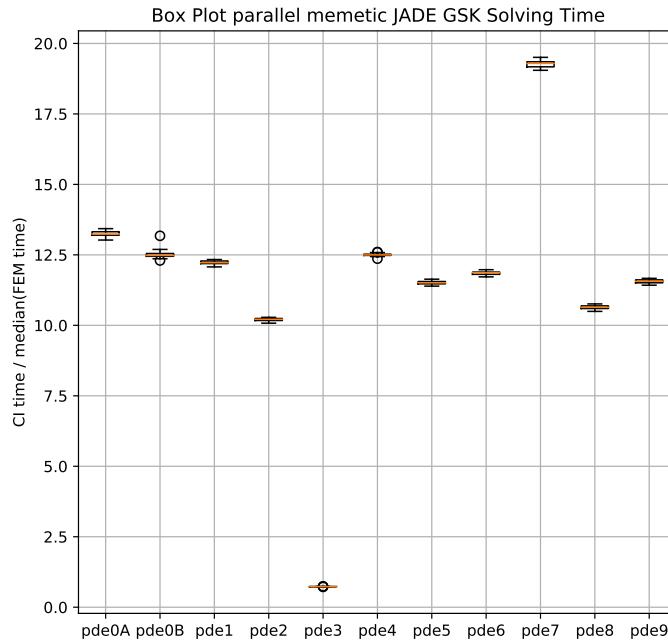


Figure 8.1: Relative solving time results of parallel memetic JADE with GSK after  $10^4$  #FE.

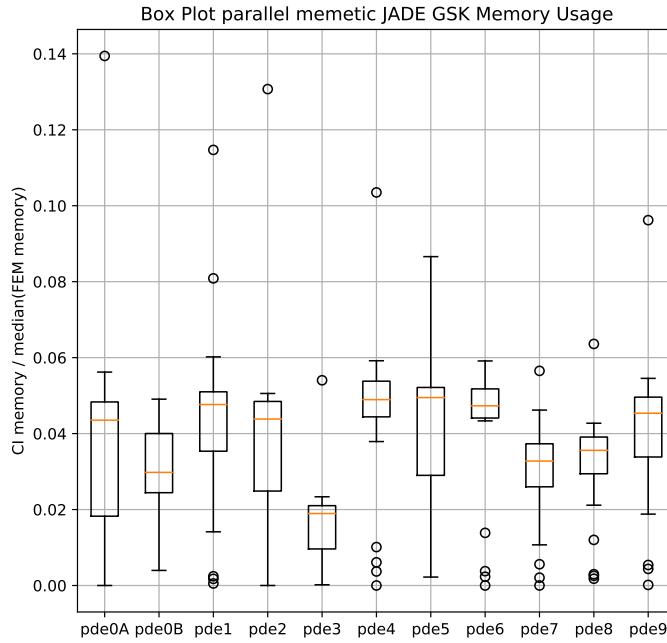


Figure 8.2: Relative memory usage results of parallel memetic JADE with GSK after  $10^4 \#FE$ .

Algorithm	parallel JADE GaK $10^6 \#FE$		parallel JADE GSK $10^6 \#FE$		Wilcoxon Test
	stat	mean	median	mean	median
PDE 0A	0.6939 $\pm$ 0.6635	0.9243	0.8106 $\pm$ 0.7929	0.6765	unsig. undecided
PDE 0B	0.2809 $\pm$ 0.3071	0.2035	0.0667 $\pm$ 0.0470	0.0614	sig. better
PDE 1	0.0239 $\pm$ 0.0467	0.0146	0.1665 $\pm$ 0.1015	0.1952	sig. worse
PDE 2	0.0300 $\pm$ 0.0157	0.0255	0.0448 $\pm$ 0.0224	0.0416	unsig. worse
PDE 3	0.0371 $\pm$ 0.0206	0.0295	0.0263 $\pm$ 0.0111	0.0269	unsig. better
PDE 4	0.0505 $\pm$ 0.0121	0.0481	0.0470 $\pm$ 0.0078	0.0458	unsig. better
PDE 5	1.2030 $\pm$ 0.0465	1.2053	0.5860 $\pm$ 0.2149	0.6841	sig. better
PDE 6	0.5814 $\pm$ 1.3550	0.0000	3.7321 $\pm$ 0.6397	3.9079	sig. worse
PDE 7	0.0228 $\pm$ 0.0025	0.0226	0.0243 $\pm$ 0.0046	0.0241	unsig. worse
PDE 8	0.2167 $\pm$ 0.0017	0.2169	0.2154 $\pm$ 0.0018	0.2150	sig. better
PDE 9	0.0426 $\pm$ 0.0115	0.0463	0.0351 $\pm$ 0.0099	0.0333	unsig. better

Table 8.1: Statistical comparison of the the parallel JADE using the GaK and the GSK.

## 8.4 Discussion

This chapter discusses the results obtained above with the GSK kernel. It is compared to the memetic parallel JADE using a GaK from experiment 1 in chapter 6. At first, the memory and time comparison is done. The performance on the PDEs 0A and 6 are discussed. The PDE 0B can be perfectly modelled with 3 GSK, thus the performance on this PDE is interesting. Finally, the behaviour of the GSK on PDE 5 is observed.

### 8.4.1 Solving Time

The solving time for the memetic pJADE with GSK forms the same pattern as the time consumption by the GaK from figure 7.2. The solving time is depending on the testbed problem. On PDE 7 the CI solver takes 20 times as long as the FEM solver. This is longer than for the GaK, which can be explained by the larger dimension and the greater population size.

### 8.4.2 Memory Usage

As mentioned above, the memory measurement includes many outliers. However, the median is quite stable. The CI solver needs only 2 to 5 percent of the memory used by the FEM solver. This is more than the corresponding GaK results from figure 6.2. The greater memory usage can be explained by the larger search dimension and the greater population size.

### 8.4.3 PDE 0A 6

Theoretically, PDEs 0A and 6 could be approximated perfectly by the GaK and the GSK. To that extend, the parameters of the GSK must evaluate to  $f = 0$  and  $\varphi = -\frac{\pi}{2}$ . Since the optimisation algorithm will always produce numerical inaccuracies and never actually result in these values, the solution quality is expected to be worse. As table 8.1 shows, the GSK results are not significantly different on PDE 0A.

However, the approximation quality for PDE 6 degrades by a lot. The mean as well as the median of the L2 norm is beyond 3.5. When looking at the 3D plots, a common theme is observed: one kernel is adapted quite well, but JADE fails to minimise the influence of the other 4 kernels. Thus, artefacts introduced by these kernels distort the function. A possible solution could be the adaptive kernel scheme.

#### 8.4.4 PDE 0B

Using a GSK produces significantly better solution qualities on PDE 0B. This is expected, since the variation generated by the sinus in the actual analytical solution can not be portrayed by the GaK.

The following comparison from figure 8.3 confirms this assumption. The left plot shows the absolute error to the analytical solution of the best result with a GaK. In contrast, the right figure shows the absolute error of best solution with a GSK. Remarkable is the structure of the error with the GaK: the error is clearly dominated by a domain-centred oscillation. This oscillation can not be portrayed by the GSK.

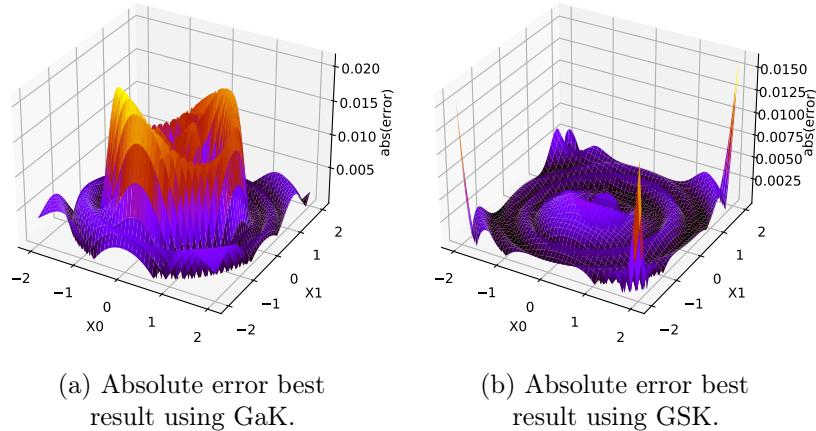


Figure 8.3: Comparisons of best solution absolute error by memetic parallel JADE using GaK and GSK after  $10^6$  #FE.

#### 8.4.5 PDE 5

The hypothesis of the GSK is that it significantly increases the approximation quality of PDE 5. Further, it should overcome the phenomenon where the L2 norm increases with more #FE.

As table 8.1 shows, the results are indeed significantly better. Again, this can be confirmed from a visual perspective by looking at the 3D plot of the approximation. Figure 8.4 depicts the best and the worst approximation of the 20 replications. Although the results are clearly better and the global structure is described more accurately, the CI solver can not compete with the FEM solver.

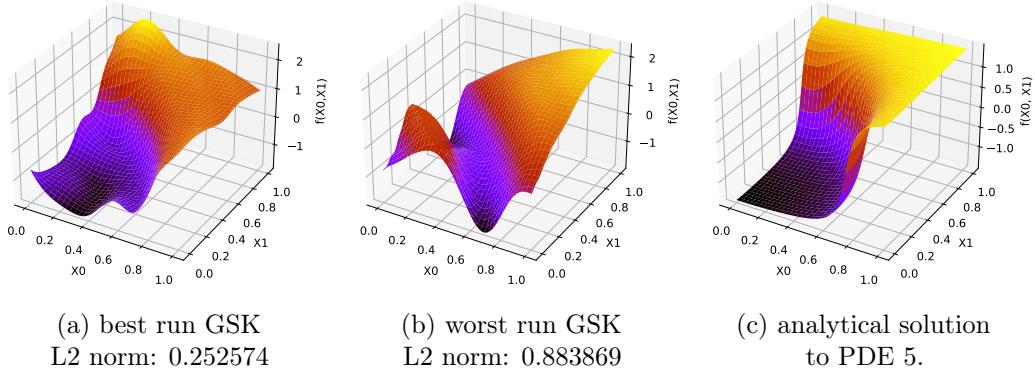


Figure 8.4: Comparison of the best and the worst result generated by memetic pJADE with GSK after  $10^6$  #FE.

The following histograms in figure 8.5b compare the experimental distribution of the L2 norm with the GaK and the GSK. As inferred from the Wilcoxon test, the distributions are significantly different. Further, the mean and the median of the “GSK-data” is smaller than the same statistical indicators of the “GaK-data”.

The histogram of the fitness values is shown in figure 8.5a. Contrary to the exception, the fitness values of the GaK are smaller than the fitness values of the GSK. Because the fitness function changed with the usage of the GSK, the numerical values can not be compared directly.

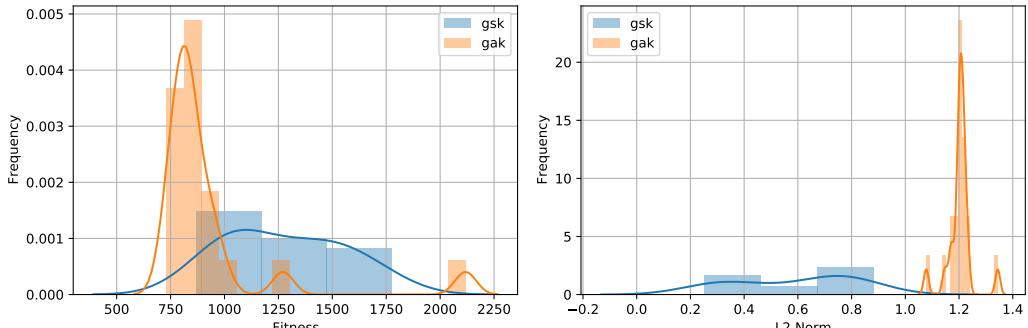


Figure 8.5: Histograms of GSK and GaK L2 norm and fitness value on PDE 5 after  $10^6$  #FE.

Similar to the plot in chapter 5.4.4, figure 8.6 connects the L2 norm of one individual with its fitness value at every generation. Although the effect of a raising L2 norm with increasing #FE is mitigated, it can be seen that the best quality is not reached

after  $10^6$  #FE. After generation 5000, the L2 norm settles in at around 0.4, while the fitness value continues to decrease.

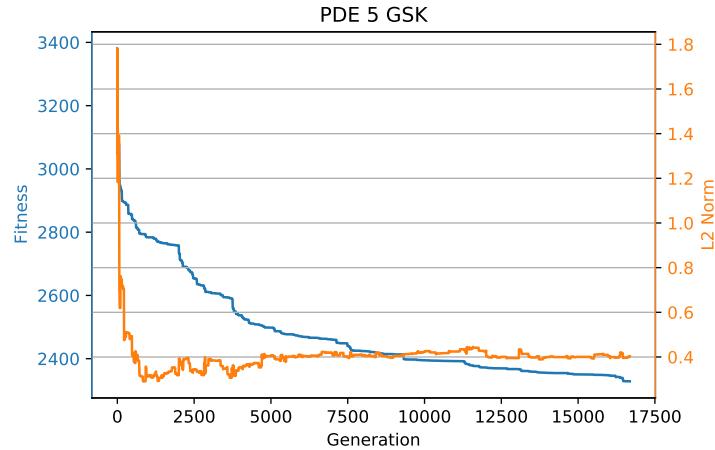


Figure 8.6: Fitness value and L2 Norm of an exemplary individual at every generation on PDE 5 using a GSK.

This again shows that a good fitness value does not necessarily indicate a good approximation quality. Although this property is only shown on PDE 5, it might also be true on other PDEs. This issue demonstrates that the fitness function suffers from a fundamental problem, but currently this is the best indicator for good solutions that only use the information posed in the original problem definition. It can be concluded that choosing an appropriate kernel is not trivial, especially in the common case where the analytical solution to the PDE is not known.

# 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

## 9.3 Computational Effort

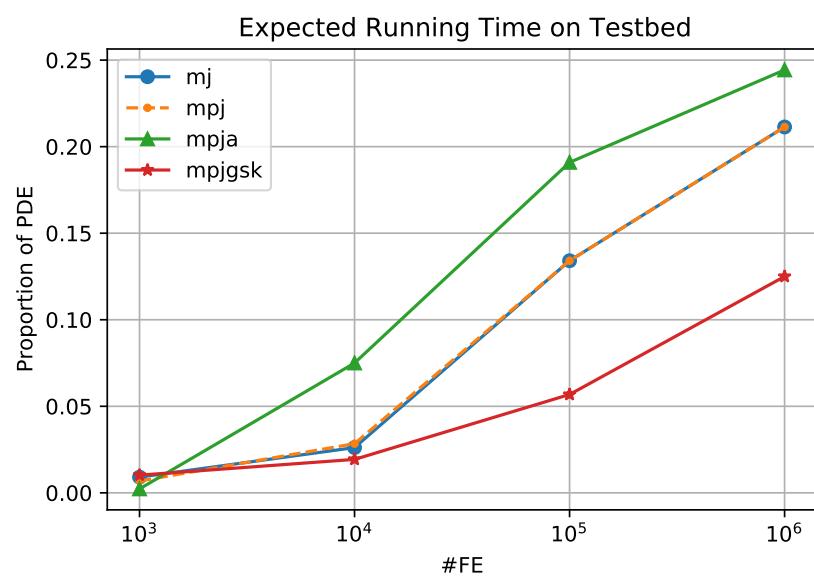


Figure 9.1:

# 10 Theoretical Notes

For the purpose of readability, some theoretical notes have been reserved for their own chapter. Here, these notes are aggregated and described.

## 10.1 Universial Approximation Theorem

## 10.2 Multimodality and Symmetry

# 11 Conclusion

## 12 Further Work

technology:

implemented in python -> precompile fitness function

theoretical:

alaisng error, tchebyshev collocation points

algorithmic:

adaptive collocation points, similar but not limited to Howard, Brezulianu, and Kolibal 2011

boundary condition fulfilment as described in Sobester, Nair, and Keane 2008  
state detector for DE that is better suited

# 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

<b>AB</b>	Adams–Bashforth. 1
<b>CI</b>	Computational Intelligence. 19, 22, 24, 27, 30, 33, 39, 40, 47, 53, 56, 67, 68
<b>CMA-ES</b>	Covariance Matrix Adaption Evolution Strategy. 8, 9, 31, 34, 39
<b>DE</b>	Differential Evolution. 1, 2, 7–11, 31, 50, 51, 74
<b>DOF</b>	Degree of Freedom. VIII, 27, 29, 30
<b>DS</b>	Downhill-Simplex. 5, 8, 9, 34, 39, 50, 51
<b>dT</b>	Delay Time. 50, 52
<b>ES</b>	Evolution Strategy. 6, 9
<b>FE</b>	Forward-Euler. 1
<b>FEM</b>	Finite Element Method. XI, 1–3, 13, 19, 22, 24, 26, 28, 29, 33–35, 37–39, 46, 47, 53, 56, 65, 67, 68
<b>GA</b>	Genetic Algorithm. 5, 9
<b>GaK</b>	Gauss Kernel. X, XI, 15–17, 19, 20, 35, 38, 44, 52, 56, 57, 61, 62, 65–69
<b>GE</b>	Grammatical Evolution. 5, 9
<b>GP</b>	Genetic Programming. 5, 6, 9
<b>GSK</b>	Gauss Sine Kernel. X, XI, 16, 20, 42, 64–70
<b>#FE</b>	Number of Function Evaluation. VIII–XI, 34–41, 43–48, 52–57, 59–62, 64–66, 68–70
<b>HS</b>	Harmony Search. 7, 9
<b>JSON</b>	JavaScript Object Notation. 95

---

*Acronyms*

---

<b>ODE</b>	Ordinary Differential Equation. 5
<b>PDE</b>	Partial Eifferential Equation. VIII–X, 1–3, 5, 12, 14–16, 19, 20, 22, 23, 25, 27–30, 32–43, 46–50, 52–62, 64, 66–70, 95, 96, 103, 104
<b>PSO</b>	Particle Swarm Optimisation. 6, 7, 9
<b>RBF</b>	Radial Basis Function. 7, 14, 22
<b>RHS</b>	Right Hand Side. 47
<b>RMSE</b>	Root-Mean-Square Error. 25, 38, 48, 95
<b>RSS</b>	Resident Set Size. 24
<b>sp</b>	Speed-Up. 47
<b>UML</b>	Unified Modeling Language. VIII, X, 22, 23, 94
<b>VMS</b>	Virtual Memory Size. 24
<b>WCA</b>	Water Cycle Algorithm. 7, 9
<b>WRM</b>	Weighted Residual Method. 4, 7, 12, 13

# 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](https://doi.org/10.1016/j.asoc.2013.02.005). URL: <http://www.sciencedirect.com/science/article/pii/S1568494613000598> (visited on 02/27/2020) (cit. on pp. 6, 9).
- Broomhead, David S. and Lowe, David (1988): “Multivariable Functional Interpolation and Adaptive Networks”. In: *Complex Systems* (cit. on p. 15).
- Chaque, 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](https://doi.org/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, 9, 20, 35, 52).
- Chaque, 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](https://doi.org/10.1007/s00500-017-2888-9). URL: <https://doi.org/10.1007/s00500-017-2888-9> (visited on 03/02/2020) (cit. on pp. 7, 9, 13, 14, 19, 20, 25, 31, 32, 34, 35, 38).
- 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](https://doi.org/10.1016/j.compstruc.2012.07.010). URL: <http://www.sciencedirect.com/science/article/pii/S0045794912001770> (visited on 05/05/2020) (cit. on p. 7).
- 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](https://doi.org/10.1631/FITEE.1900221). URL: <https://doi.org/10.1631/FITEE.1900221> (visited on 02/11/2020) (cit. on pp. 8, 9).

## Bibliography

---

- 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](https://doi.org/10.1177/003754970107600201). URL: <https://doi.org/10.1177/003754970107600201> (visited on 05/05/2020) (cit. on p. 7).
- Guyan, R. J. (1965): “Reduction of stiffness and mass matrices”. In: DOI: [10.2514/3-2874](https://doi.org/10.2514/3-2874) (cit. on p. 4).
- 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](https://doi.org/10.1007/3-540-32494-1_4). URL: [https://doi.org/10.1007/3-540-32494-1\\_4](https://doi.org/10.1007/3-540-32494-1_4) (visited on 05/05/2020) (cit. on p. 8).
- 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. 103).
- 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](https://doi.org/10.1145/321127.321128). URL: <https://doi.org/10.1145/321127.321128> (visited on 05/29/2020) (cit. on p. 5).
- 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](https://doi.org/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, 9, 74).
- 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. 5, 9).
- Hu, Zhongbo et al. (2013): *Sufficient Conditions for Global Convergence of Differential Evolution Algorithm*. en. Research Article. DOI: <https://doi.org/10.1155/2013/193196>. URL: <https://www.hindawi.com/journals/jam/2013/193196/> (visited on 02/11/2020) (cit. on p. 51).
- 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](https://doi.org/10.1109/ICNN.1995.488968) (cit. on p. 6).

## Bibliography

---

- 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](https://doi.org/10.1080/03081070500065676). URL: <https://doi.org/10.1080/03081070500065676> (visited on 03/02/2020) (cit. on pp. 5, 9).
- 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 and Suganthan (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](https://doi.org/10.1109/CEC.2008.4631294) (cit. on pp. 31, 38).
- 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](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. 5, 9).
- 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. 19–21).
- Moscato, Pablo (Oct. 2000): "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms". In: *Caltech Concurrent Computation Program* (cit. on p. 34).
- 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](https://doi.org/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).
- Opara, Karol R. and Arabas, Jarosław (Feb. 2019): "Differential Evolution: A survey of theoretical analyses". en. In: *Swarm and Evolutionary Computation* 44, pp. 546–558. ISSN: 2210-6502. DOI: [10.1016/j.swevo.2018.06.010](https://doi.org/10.1016/j.swevo.2018.06.010). URL: <http://www.sciencedirect.com/science/article/pii/S2210650217304224> (visited on 03/04/2020) (cit. on p. 51).

## Bibliography

---

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, 9, 19, 20, 35).

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. 43).

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. 24).

Rechenberg, I. (1978): “Evolutionsstrategien”. de. In: Schneider, Berthold and Ranft, Ulrich (Eds.): *Simulationstechniken 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](https://doi.org/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. 24).

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](https://doi.org/10.1007/BFb0055930) (cit. on p. 5).

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](https://doi.org/10.1080/0952813X.2016.1259271). URL: <https://doi.org/10.1080/0952813X.2016.1259271> (visited on 03/02/2020) (cit. on pp. 7, 9).

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, 26).

## Bibliography

---

- 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](https://doi.org/10.1007/978-3-0348-5927-1_5). URL: [https://doi.org/10.1007/978-3-0348-5927-1\\_5](https://doi.org/10.1007/978-3-0348-5927-1_5) (visited on 05/29/2020) (cit. on p. 6).
- SciPy, Reference Guide (Aug. 2020a): *scipy.optimize.fmin — SciPy v1.5.2 Reference Guide*. en. Documentation. URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin.html> (visited on 08/06/2020) (cit. on p. 34).
- SciPy, Reference Guide (June 2020b): *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. 104).
- 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](https://doi.org/10.1007/978-3-540-71041-7). URL: <https://www.springer.com/de/book/9783540710400> (visited on 04/24/2020) (cit. on pp. 3, 12, 13).
- 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](https://doi.org/10.1109/TEVC.2007.908467) (cit. on pp. 6, 9, 20, 35, 74).
- 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](https://doi.org/10.1023/A:1008202821328). URL: <https://doi.org/10.1023/A:1008202821328> (visited on 04/05/2019) (cit. on pp. 7, 10).
- Suganthan (Jan. 2020): *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. 10).
- 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](https://doi.org/10.1109/CEC.2013.6557555) (cit. on p. 11).

## Bibliography

---

- 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](https://doi.org/10.1109/CEC.2014.6900380) (cit. on p. 11).
- 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](https://doi.org/10.1007/s10710-006-7009-y). URL: <https://doi.org/10.1007/s10710-006-7009-y> (visited on 02/15/2020) (cit. on pp. 5, 9, 19, 20, 35).
- 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](https://doi.org/10.1109/TEVC.2009.2014613) (cit. on p. 11).

# Appendices



# A Differential Evolution Pseudocodes

## A.1 JADE Pseudocode

---

Algorithm A.1: JADE Pseudocode

---

```

1 Function JADE( $\mathbf{X}_{g=0}$ ,  $p$ ,  $c$ , function, minError, maxFE):
2    $fValue_{g=0} \leftarrow function(\mathbf{x}_{g=0})$ 
3    $\mu_{CR} \leftarrow 0.5$ 
4    $\mu_F \leftarrow 0.5$ 
5    $A \leftarrow \emptyset$ 
6   while  $fe \leq maxFE$  do
7      $g \leftarrow g + 1$ 
8      $S_F \leftarrow \emptyset$ 
9      $S_{CR} \leftarrow \emptyset$ 
10    for  $i = 1$  to  $NP$  do
11       $F_i \leftarrow randc_i(\mu_F, 0.1)$ 
12       $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,g}, A, fValue_g, F_i, p)$ 
13       $CR_i \leftarrow randn_i(\mu_{CR}, 0.1)$ 
14       $u_i \leftarrow crossoverBIN(\mathbf{x}_{i,g}, v_i, CR_i)$ 
15      if function( $\mathbf{x}_{i,g}$ )  $\geq function(\mathbf{u}_{i,g})$  then
16         $\mathbf{x}_{i,g+1} \leftarrow \mathbf{x}_{i,g}$ 
17      end
18      else
19         $\mathbf{x}_{i,g+1} \leftarrow \mathbf{u}_{i,g}$ 
20         $fValue_{i,g+1} \leftarrow function(\mathbf{u}_{i,g})$ 
21         $\mathbf{x}_{i,g} \rightarrow \mathbf{A}$ 
22         $CR_i \rightarrow S_{CR}$ 
23         $F_i \rightarrow S_F$ 
24      end
25    end
26    // resize  $A$  to size of  $\mathbf{x}_g$ 
27    if  $|A| > NP$  then
28       $A \leftarrow A \setminus A_{rand_i}$ 
29    end
30     $fe \leftarrow fe + size(\mathbf{X})$ 
31     $\mu_{CR} \leftarrow (1 - c) \cdot \mu_{CR} + c \cdot arithmeticMean(S_{CR})$ 
32     $\mu_F \leftarrow (1 - c) \cdot \mu_F + c \cdot lehmerMean(S_F)$ 
33  end

```

---

## A.2 SHADE Pseudocode

---

Algorithm A.2: SHADE Pseudocode

---

```

1 Function SHADE( $\mathbf{x}_{G=0}$ ,  $p$ ,  $H$ , function, minError, maxFE):
2    $M_{CR} \leftarrow 0.5$ ;  $M_F \leftarrow 0.5$ ;  $A \leftarrow \emptyset$ ;  $G \leftarrow 0$ ;  $k \leftarrow 1$ 
3    $fValue_{G=0} \leftarrow function(\mathbf{x}_{G=0})$ 
4   while termination condition not met do
5      $S_{CR} \leftarrow \emptyset$ ;  $S_F \leftarrow \emptyset$ 
6     for  $i = 1$  to  $N$  do
7        $r_i \leftarrow rand_{int}(1, H)$ 
8        $CR_{i,G} \leftarrow randn_i(M_{CR,r_i}, 0.1)$ ;  $F_{i,G} \leftarrow randc_i(M_{F,r_i}, 0.1)$ 
9        $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,G}, A, fValue_G, F_i, p)$ 
10       $u_i \leftarrow crossoverBIN(pop, v_i, CR)$ 
11    end
12    for  $i = 1$  to  $N$  do
13      if function( $u_{i,G}$ )  $\leq$  function( $x_{i,G}$ ) then
14         $x_{i,G+1} \leftarrow u_{i,G}$ ;  $fValue_{i,G+1} \leftarrow function(\mathbf{u}_{i,G})$ 
15      end
16      else
17         $x_{i,G+1} \leftarrow x_{i,G}$ 
18      end
19      if function( $u_{i,G}$ )  $<$  function( $x_{i,G}$ ) then
20         $x_{i,G} \rightarrow A$ ;  $CR_{i,G} \rightarrow S_{CR}$ ;  $F_{i,G} \rightarrow S_F$ 
21      end
22    end
23    if  $|A| > N$  then
24       $A \leftarrow A \setminus A_{rand_i}$ 
25    end
26    if  $S_{CR} \neq \emptyset \wedge S_F \neq \emptyset$  then
27      
$$M_{CR,k,G+1} = \begin{cases} arithmeticMean(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k,G} & \text{otherwise} \end{cases}$$

28      
$$M_{F,k,G+1} = \begin{cases} lehmerMean(S_F) & \text{if } S_F \neq \emptyset \\ M_{F,k,G} & \text{otherwise} \end{cases}$$

29       $k \leftarrow k + 1$ 
30      if  $k > H$  then
31         $k \leftarrow 1$ 
32      end
33    end
34     $G \leftarrow G + 1$ 
35  end

```

---

### A.3 L-SHADE Pseudocode

---

Algorithm A.3: L-SHADE Pseudocode

---

```

1 Function LSHADE( $\mathbf{x}_{G=0}$ ,  $p$ ,  $H$ , function, minError, maxFE):
2    $M_{CR} \leftarrow 0.5$ ;  $M_F \leftarrow 0.5$ ;  $A \leftarrow \emptyset$ ;  $G \leftarrow 0$ ;  $k \leftarrow 1$ 
3    $fValue_{G=0} \leftarrow function(\mathbf{x}_{G=0})$ ;  $NG_{init} \leftarrow size(\mathbf{x}_{G=0})$ ;  $NG_{min} = \lceil 1/p \rceil$ 
4   while termination condition not met do
5      $S_{CR} \leftarrow \emptyset$ ;  $S_F \leftarrow \emptyset$ 
6     for  $i = 1$  to  $N$  do
7        $r_i \leftarrow rand_{int}(1, H)$ 
8        $CR_{i,G} \leftarrow randn_i(M_{CR,r_i}, 0.1)$ ;  $F_{i,G} \leftarrow randc_i(M_{F,r_i}, 0.1)$ 
9        $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,G}, A, fValue_G, F_i, p)$ 
10       $u_i \leftarrow crossoverBIN(pop, v_i, CR)$ 
11    end
12    for  $i = 1$  to  $N$  do
13      if function( $u_{i,G}$ )  $\leq$  function( $x_{i,G}$ ) then
14         $x_{i,G+1} \leftarrow u_{i,G}$ ;  $fValue_{i,G+1} \leftarrow function(\mathbf{u}_{i,G})$ 
15      end
16      else
17         $x_{i,G+1} \leftarrow x_{i,G}$ 
18      end
19      if function( $u_{i,G}$ )  $<$  function( $x_{i,G}$ ) then
20         $x_{i,G} \rightarrow A$ ;  $CR_{i,G} \rightarrow S_{CR}$ ;  $F_{i,G} \rightarrow S_F$ 
21      end
22    end
23    if  $|A| > N$  then
24       $A \leftarrow A \setminus A_{rand_i}$ 
25    end
26    if  $S_{CR} \neq \emptyset \wedge S_F \neq \emptyset$  then
27      
$$M_{CR,k,G+1} = \begin{cases} arithmeticMean(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k,G} & \text{otherwise} \end{cases}$$

28      
$$M_{F,k,G+1} = \begin{cases} lehmerMean(S_F) & \text{if } S_F \neq \emptyset \\ M_{F,k,G} & \text{otherwise} \end{cases}$$

29       $k \leftarrow k + 1$ 
30      if  $k > H$  then
31         $k \leftarrow 1$ 
32      end
33    end
34     $\mathbf{x}_{G+1} \leftarrow popSizeRed(\mathbf{x}_{G+1}, fValue, G, maxGen, NG_{init}, NG_{min})$ 
35     $G \leftarrow G + 1$ 
36  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  $\Omega$
  - Dirichlet boundary condition obtained by evaluating the solution on the boundary
- solution
- plot of the solution over the domain

**PDE 0A: Gauss Kernel**

Problem PDE:

$$\begin{aligned}
 \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= (18x^2 - 6)e^{-1.5(x^2+y^2)} + (18y^2 - 6)e^{-1.5(x^2+y^2)} \\
 &+ 6(6x^2 + 12x + 5)e^{-3((x+1)^2+(y+1)^2)} + 6(6y^2 + 12y + 5)e^{-3((x+1)^2+(y+1)^2)} \\
 &+ 6(6x^2 - 12x + 5)e^{-3((x-1)^2+(y+1)^2)} + 6(6y^2 + 12y + 5)e^{-3((x-1)^2+(y+1)^2)} \\
 &+ 6(6x^2 + 12x + 5)e^{-3((x+1)^2+(y-1)^2)} + 6(6y^2 - 12y + 5)e^{-3((x+1)^2+(y-1)^2)} \\
 &+ 6(6x^2 - 12x + 5)e^{-3((x-1)^2+(y-1)^2)} + 6(6y^2 - 12y + 5)e^{-3((x-1)^2+(y-1)^2)}
 \end{aligned}$$

on the domain  $\Omega : x, y \in [-2, 2]$   
subjected to:  

$$\begin{aligned}
 u(x, 2) &= 2e^{-1.5(x^2+4)} + e^{-3((x+1)^2+9)} + e^{-3((x+1)^2+1)} + e^{-3((x-1)^2+9)} + e^{-3((x-1)^2+1)} \\
 u(x, -2) &= 2e^{-1.5(x^2+4)} + e^{-3((x+1)^2+1)} + e^{-3((x+1)^2+9)} + e^{-3((x-1)^2+1)} + e^{-3((x-1)^2+9)} \\
 u(2, y) &= 2e^{-1.5(4+y^2)} + e^{-3(9+(y+1)^2)} + e^{-3(9+(y-1)^2)} + e^{-3(1+(y+1)^2)} + e^{-3(1+(y-1)^2)} \\
 u(-2, y) &= 2e^{-1.5(4+y^2)} + e^{-3(1+(y+1)^2)} + e^{-3(1+(y-1)^2)} + e^{-3(9+(y+1)^2)} + e^{-3(9+(y-1)^2)}
 \end{aligned} \tag{B.1}$$

Solution:

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

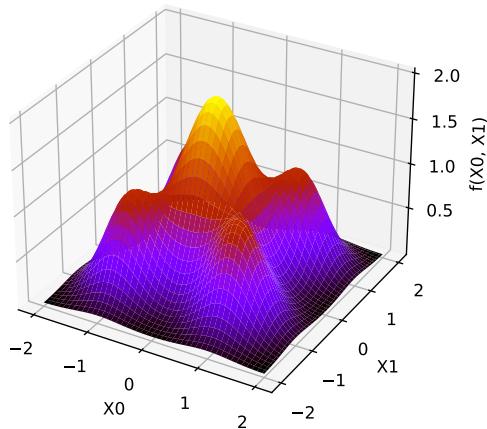


Figure B.1: PDE 0A Gauss Kernel solution plot

**PDE 0B: GSin Kernel**

Problem PDE:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} =$$

$$2e^{-2(x^2+y^2)}(2\sin(-2(x^2+y^2)) + 2(1-8x^2)\cos(-2(x^2+y^2))) +$$

$$2e^{-2(x^2+y^2)}(2\sin(-2(x^2+y^2)) + 2(1-8y^2)\cos(-2(x^2+y^2))) +$$

$$2e^{-1(x^2+y^2)}(1\sin(-1(x^2+y^2)) + 1(1-4x^2)\cos(-1(x^2+y^2))) +$$

$$2e^{-1(x^2+y^2)}(1\sin(-1(x^2+y^2)) + 1(1-4y^2)\cos(-1(x^2+y^2))) +$$

$$2e^{-0.1(x^2+y^2)}(0.1\sin(-0.1(x^2+y^2)) + 0.1(1-0.4x^2)\cos(-0.1(x^2+y^2))) +$$

$$2e^{-0.1(x^2+y^2)}(0.1\sin(-0.1(x^2+y^2)) + 0.1(1-0.4y^2)\cos(-0.1(x^2+y^2)))$$

on the domain  $\Omega : x, y \in [-2, 2]$   
subjected to:

$$u(x, 2) = e^{-2(x^2+4)}\sin(2((x^2+4))) + e^{-1(x^2+4)}\sin(1((x^2+4))) + e^{-0.1(x^2+4)}\sin(0.1((x^2+4)))$$

$$u(x, -2) = e^{-2(x^2+4)}\sin(2((x^2+4))) + e^{-1(x^2+4)}\sin(1((x^2+4))) + e^{-0.1(x^2+4)}\sin(0.1((x^2+4)))$$

$$u(2, y) = e^{-2(4+y^2)}\sin(2((4+y^2))) + e^{-1(4+y^2)}\sin(1((4+y^2))) + e^{-0.1(4+y^2)}\sin(0.1((4+y^2)))$$

$$u(-2, y) = e^{-2(4+y^2)}\sin(2((4+y^2))) + e^{-1(4+y^2)}\sin(1((4+y^2))) + e^{-0.1(4+y^2)}\sin(0.1((4+y^2)))$$

(B.3)

Solution:

$$u_{ext}(x, y) = e^{-2(x^2+y^2)}\sin(2((x^2+y^2))) + e^{-1(x^2+y^2)}\sin(1((x^2+y^2)))$$

$$+ e^{-0.1(x^2+y^2)}\sin(0.1((x^2+y^2)))$$

(B.4)

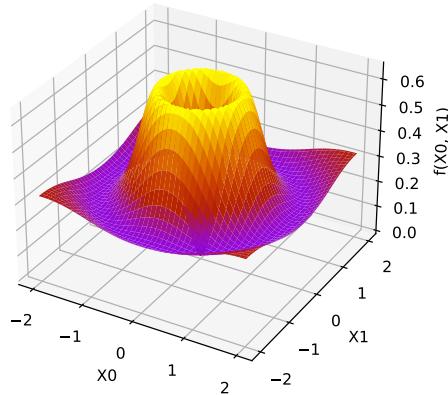


Figure B.2: PDE 0B Gsin Kernel solution plot

### PDE 1: Polynomial 2D

Problem PDE:

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

$$\begin{aligned}
 u(x, 1) &= 0 \\
 u(x, 0) &= 0 \\
 u(1, y) &= 0 \\
 u(0, y) &= 0
 \end{aligned}$$

Solution:

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

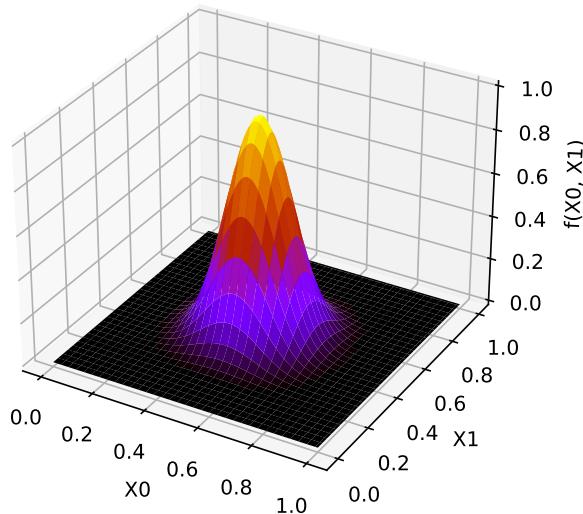


Figure B.3: PDE 1 Polynomial 2D solution plot

### PDE 2: Chaquet PDE 1

Problem PDE:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = e^{-x}(x - 2 + y^3 + 6y)$$

on the domain  $\Omega : \mathbf{x} \in [0, 1]$   
subjected to:

$$u(x, 0) = xe^{-x}u(x, 1) = (x + 1)e^{-x}u(0, y) = y^3$$

$$u(1, y) = (1 + y^3)e^{-1}$$
(B.7)

Solution:

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

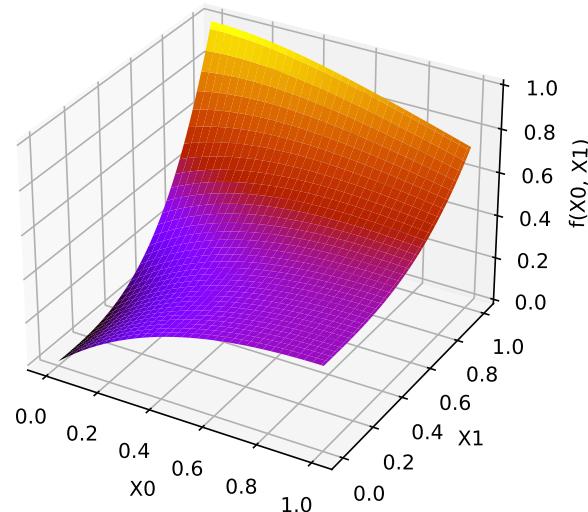


Figure B.4: PDE 2 Chaquet PDE 1 solution plot

### **PDE 3: Chaquet PDE 3**

Problem PDE:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 4$$

on the domain  $\Omega : \mathbf{x} \in [0, 1]$

subjected to:

$$u(x, 0) = x^2 + x + 1 \quad (\text{B.9})$$

$$u(x, 1) = x^2 + x + 3$$

$$u(1, y) = y^2 + y + 3$$

$$u(0, y) = y^2 + y + 1$$

Solution

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

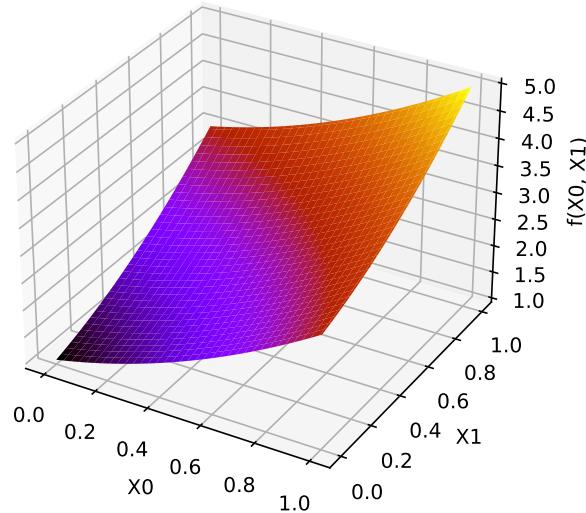


Figure B.5: PDE 3 Chaquet PDE 3 solution plot

#### **PDE 4: Sine Bump 2D**

Problem PDE:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 2\pi^2 \sin(\pi x) \sin(\pi y)$$

on the domain  $\Omega : \mathbf{x} \in [0, 1]$

subjected to:

$$u(x, 0) = 0 \quad (\text{B.11})$$

$$u(x, 1) = 0$$

$$u(0, y) = 0$$

$$u(1, y) = 0$$

Solution:

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

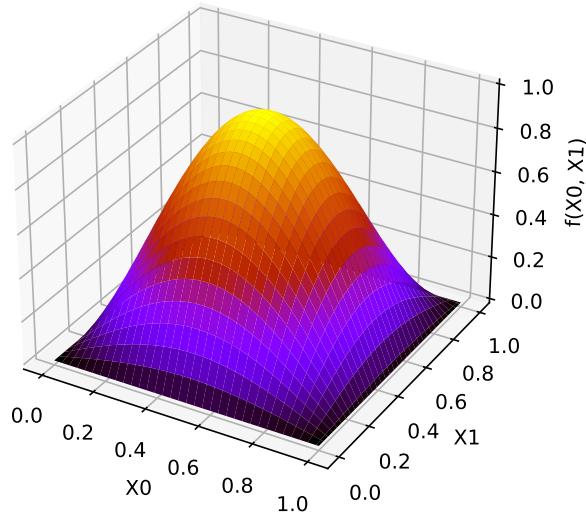


Figure B.6: PDE 4 Sine Bump 2D solution plot

**PDE 5: Arctan Circular Wave Front**

Problem PDE:

$$\begin{aligned}
 -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= \frac{16000(\sqrt{(x-0.05)^2 + (y-0.05)^2} - 0.7)}{(1+400(-0.7 + \sqrt{(x-0.05)^2 + (y-0.05)^2})^2)^2} \\
 + \frac{20(x-0.05)^2 + 20(y-0.05)^2}{(1+400(\sqrt{(x-0.05)^2 + (y-0.05)^2} - 0.7)^2)((x-0.05)^2 + (y-0.05)^2)^{3/2}} \\
 - \frac{40}{(1+400(\sqrt{(y-0.05)^2 + (x-0.05)^2} - 0.7)^2)\sqrt{(y-0.05)^2 + (x-0.05)^2}}
 \end{aligned}$$

on the domain  $\Omega : \mathbf{x} \in [0, 1]$   
subjected to:

$$\begin{aligned}
 u(x, 0) &= \tan^{-1} \left( 20 \left( \sqrt{(x-0.05)^2 + 0.0025} - 0.7 \right) \right) \\
 u(x, 1) &= \tan^{-1} \left( 20 \left( \sqrt{(x-0.05)^2 + 0.9025} - 0.7 \right) \right) \\
 u(0, y) &= \tan^{-1} \left( 20 \left( \sqrt{0.0025 + (y-0.05)^2} - 0.7 \right) \right) \\
 u(1, y) &= \tan^{-1} \left( 20 \left( \sqrt{0.9025 + (y-0.05)^2} - 0.7 \right) \right)
 \end{aligned} \tag{B.13}$$

Solution:

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

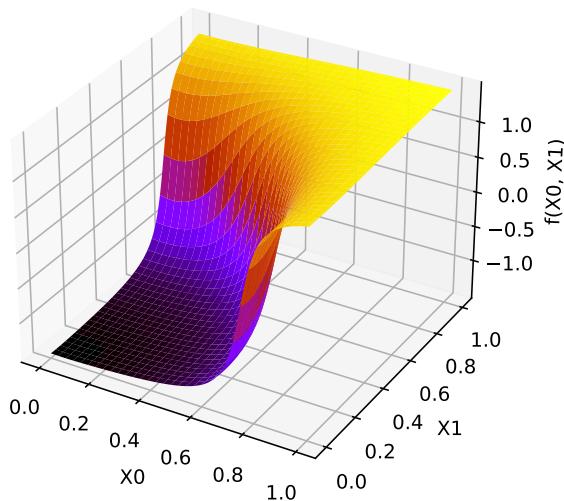


Figure B.7: PDE 5 Arctan Circular Wave Front solution plot

**PDE 6: Peak 2D**

Problem PDE:

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

Solution:

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

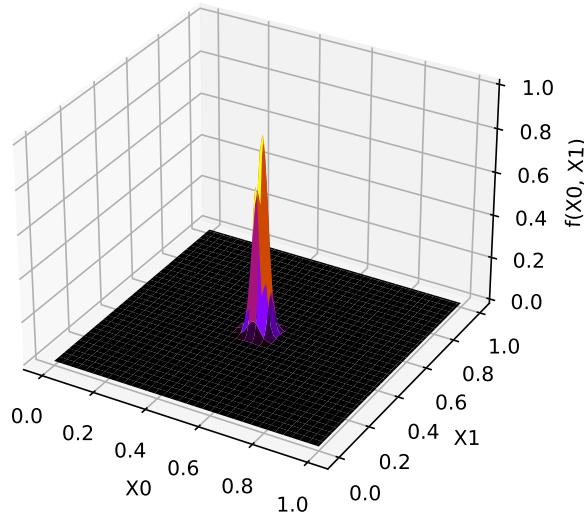


Figure B.8: PDE 6 Peak 2D solution plot

**PDE 7: Boundary Line Singularity**

Problem PDE:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0.24x^{-1.4}$$

on the domain  $\Omega : \mathbf{x} \in [0, 1]$

subjected to:

$$u(x, 0) = x^{0.6} \quad (\text{B.17})$$

$$u(x, 1) = x^{0.6}$$

$$u(0, y) = 0$$

$$u(1, y) = 1^{0.6}$$

Solution:

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

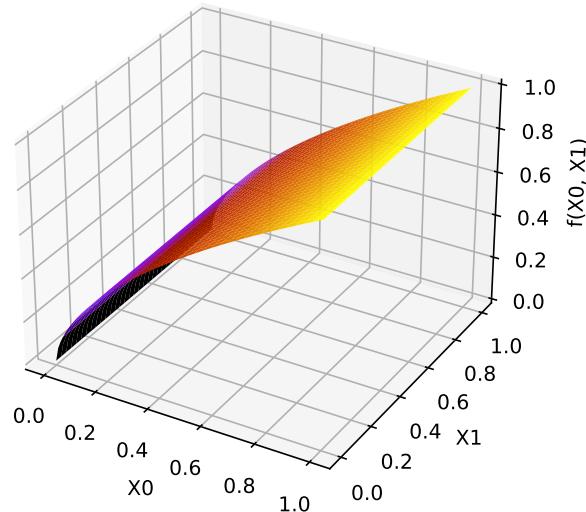


Figure B.9: PDE 7 Boundary Line Singularity solution plot

### PDE 8: Interior Point Singularity

Problem PDE:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{\sqrt{x^2 - x + y^2 - y + 0.5}} \\ \Omega : \mathbf{x} \in [0, 1]$$

on the domain subjected to:

$$u(x, 0) = \sqrt{(x - 0.5)^2 + 0.25} \quad (\text{B.19}) \\ u(x, 1) = \sqrt{(x - 0.5)^2 + 0.25} \\ u(0, y) = \sqrt{0.25 + (y - 0.5)^2} \\ u(1, y) = \sqrt{0.25 + (y - 0.5)^2}$$

Solution:

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

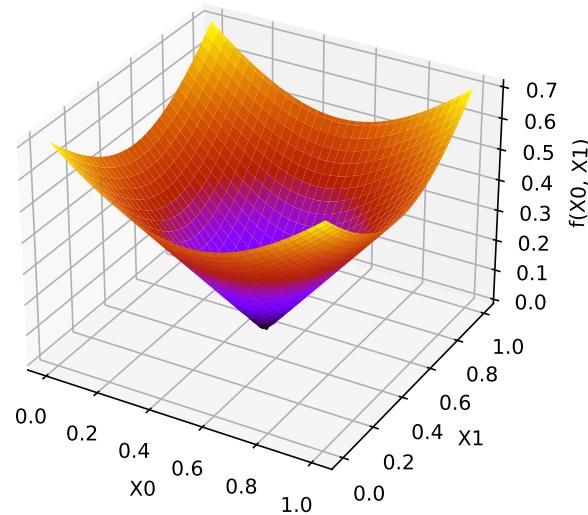


Figure B.10: PDE 8 Interior Point Singularity solution plot

### PDE 9: Arctan Wave Front Homogeneous Boundary Conditions 2D

Problem PDE:

$$\begin{aligned}
 & -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = \\
 & \frac{20\sqrt{2}(x^2 + y^2 - 2x^2y - 2xy^2 + 4xy - x - y)}{400(\frac{x+y}{\sqrt{2}} - 0.8)^2 + 1} \\
 & + \frac{16000(1-x)x(1-y)y(\frac{x+y}{\sqrt{2}} - 0.8)}{(400(\frac{x+y}{\sqrt{2}} - 0.8)^2 + 1)^2} \\
 & + \tan^{-1}\left(20\left(\frac{x+y}{\sqrt{2}} - 0.8\right)\right)(2(1-y)y + 2(1-x)x) \quad (B.21)
 \end{aligned}$$

on the domain  $\Omega : \mathbf{x} \in [0, 1]$   
subjected to:

$$\begin{aligned}
 u(x, 0) &= 0 \\
 u(x, 1) &= 0 \\
 u(0, y) &= 0 \\
 u(1, y) &= 0
 \end{aligned}$$

Solution:

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

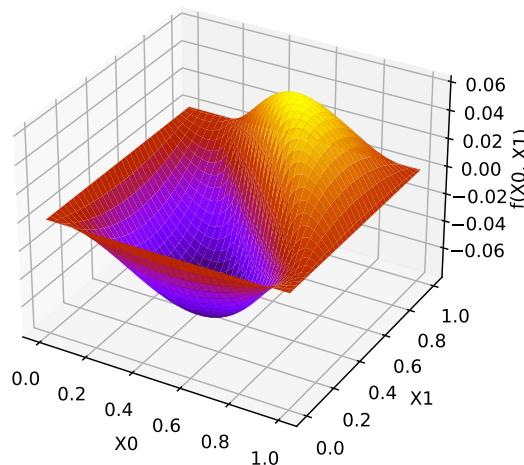


Figure B.11: PDE 9 Arctan Wave Front Homogeneous Boundary Conditions 2D  
solution plot



# C Software Architecture

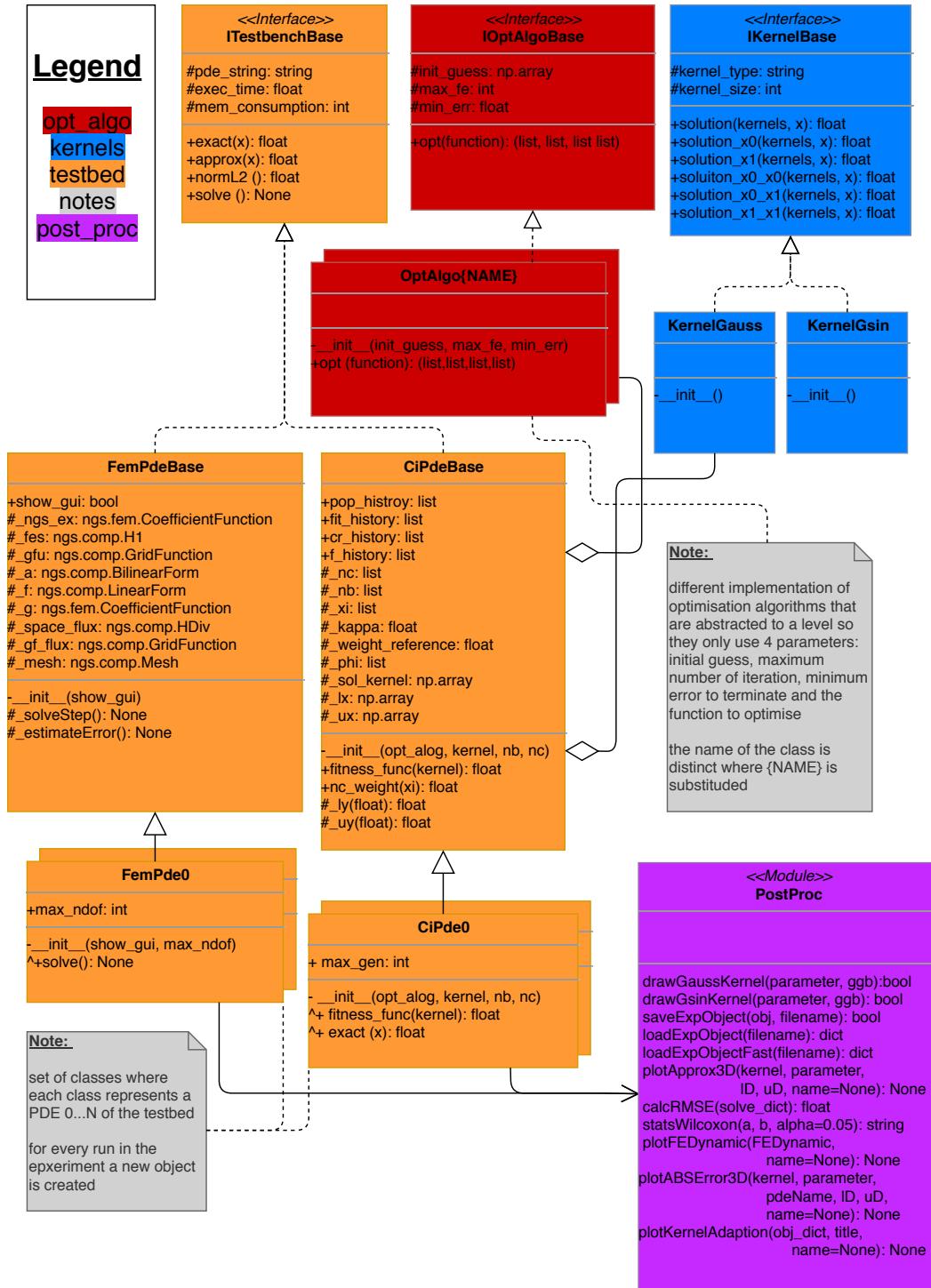


Figure C.1: This UML class diagram describes the software architecture defined to prepare, run and evaluate the experiments.

# D Post-Processing Module Description

The post-processing module includes these functions. Their implementation is described in the following list. These functions are used to interpret the results in the experiments chapters.

- `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 ( $>400\text{Mb} @ 10^6 \#FE$ ). 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

update implemented function description and class diagram in appendix

from a single dictionary, as obtained by the functions `loadExpObject` or `loadExpObjectFast`.

- `None plotApprox3D(kernel, parameter, 1D, 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 `1D` 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 2020b). The default significance level is set to 0.05. A string is returned, that describes if the mean and median of `a` is significantly smaller than the mean and 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, 1D, 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  $E_{abs} = |u_{apx}(x, y) - u_{ext}(x, y)| \forall x, y \in \Omega$ .

- `None plotKernelAdaption(obj_dict, title, 'green', 'red', name=None)`

This function plot the fitness difference over all generations. Additionally, it marks if the number of kernels (and thus the search dimension) is increased or reduced.

# E Solve Method

---

Algorithm E.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()
```

---



# F pJADE

Algorithm F.1: Pseudocode of pJADE

---

```

1 Function pJADE( $\mathbf{X}_{g=0}$ ,  $p$ ,  $c$ , function, minError, maxFE):
2    $fValue_{g=0} \leftarrow function(\mathbf{x}_{g=0})$ 
3    $\mu_{CR} \leftarrow 0.5$ 
4    $\mu_F \leftarrow 0.5$ 
5    $A \leftarrow \emptyset$ 
6   while  $fe \leq maxFE$  do
7      $g \leftarrow g + 1$ 
8      $S_F \leftarrow \emptyset$ 
9      $S_{CR} \leftarrow \emptyset$ 
10     $pResults \leftarrow \emptyset$ 
11    for  $i = 1$  to  $NP$  do parallel
12       $F_i \leftarrow randc_i(\mu_F, 0.1)$ 
13       $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,g}, A, fValue_g, F_i, p)$ 
14       $CR_i \leftarrow randn_i(\mu_{CR}, 0.1)$ 
15       $u_i \leftarrow crossoverBIN(\mathbf{x}_{i,g}, v_i, CR_i)$ 
16       $(u_i, function(u_i)) \rightarrow pResults$ 
17    end
18    for  $i = 1$  to  $NP$  do
19      if function( $\mathbf{x}_{i,g}$ )  $\geq pResults_{i,f,g}$  then
20         $\mathbf{x}_{i,g+1} \leftarrow \mathbf{x}_{i,g}$ 
21      end
22      else
23         $\mathbf{x}_{i,g+1} \leftarrow pResults_{i,u,g}$ 
24         $fValue_{i,g+1} \leftarrow pResults_{i,f,g}$ 
25         $\mathbf{x}_{i,g} \rightarrow \mathbf{A}$ 
26         $CR_i \rightarrow S_{CR}$ 
27         $F_i \rightarrow S_F$ 
28      end
29    end
30    // resize  $A$  to size of  $\mathbf{x}_g$ 
31    if  $|A| > NP$  then
32       $A \leftarrow A \setminus A_{rand_i}$ 
33    end
34     $fe \leftarrow fe + size(\mathbf{X})$ 
35     $\mu_{CR} \leftarrow (1 - c) \cdot \mu_{CR} + c \cdot arithmeticMean(S_{CR})$ 
36     $\mu_F \leftarrow (1 - c) \cdot \mu_F + c \cdot lehmerMean(S_F)$ 
37  end

```

---



# G paJADE

---

Algorithm G.1: Pseudocode of paJADE

---

```

1 Function paJADE( $\mathbf{X}_{g=0}$ ,  $p$ ,  $c$ ,  $dT$ ,  $function$ ,  $minError$ ,  $maxFE$ ):
2    $fValue_{g=0} \leftarrow function(\mathbf{x}_{g=0})$ 
3    $\mu_{CR} \leftarrow 0.5$ 
4    $\mu_F \leftarrow 0.5$ 
5    $A \leftarrow \emptyset$ 
6   while  $fe \leq maxFE$  do
7      $g \leftarrow g + 1$ 
8      $S_F \leftarrow \emptyset$ 
9      $S_{CR} \leftarrow \emptyset$ 
10     $pResults \leftarrow \emptyset$ 
11    for  $i = 1$  to  $NP$  do parallel
12       $F_i \leftarrow randc_i(\mu_F, 0.1)$ 
13       $v_i \leftarrow mutationCurrentToPBest1(\mathbf{x}_{i,g}, A, fValue_g, F_i, p)$ 
14       $CR_i \leftarrow randn_i(\mu_{CR}, 0.1)$ 
15       $u_i \leftarrow crossoverBIN(\mathbf{x}_{i,g}, v_i, CR_i)$ 
16       $(u_i, function(u_i)) \rightarrow pResults$ 
17    end
18    for  $i = 1$  to  $NP$  do
19      if  $function(\mathbf{x}_{i,g}) \geq pResults_{i,f,g}$  then
20         $\mathbf{x}_{i,g+1} \leftarrow \mathbf{x}_{i,g}$ 
21      end
22      else
23         $\mathbf{x}_{i,g+1} \leftarrow pResults_{i,u,g}$ 
24         $fValue_{i,g+1} \leftarrow pResults_{i,f,g}$ 
25         $\mathbf{x}_{i,g} \rightarrow \mathbf{A}$ 
26         $CR_i \rightarrow S_{CR}$ 
27         $F_i \rightarrow S_F$ 
28      end
29    end
30    // resize  $A$  to size of  $\mathbf{x}_g$ 
31    if  $|A| > NP$  then
32       $A \leftarrow A \setminus A_{rand_i}$ 
33    end
34     $fe \leftarrow fe + size(\mathbf{X})$ 
35     $\mu_{CR} \leftarrow (1 - c) \cdot \mu_{CR} + c \cdot arithmeticMean(S_{CR})$ 
36     $\mu_F \leftarrow (1 - c) \cdot \mu_F + c \cdot lehmerMean(S_F)$ 
37    // state detector
38    if  $min(function(\mathbf{X}_{g-dT}) - function(\mathbf{X}_g)) < minError$  then
39      break
40    end
41  end
```

---

# H Adaptive Kernel Scheme

---

Algorithm H.1: Pseudocode of memetic parallel JADE with adaptive kernels

---

```

1 Function memeticpJADEadaptive(X, funct, minErr, maxFE):
2   dim, popsize, kernelsize  $\leftarrow$  size(X)
3   p  $\leftarrow$  0.3
4   c  $\leftarrow$  0.5
5   dT  $\leftarrow$  100
6   fecounter  $\leftarrow$  0
7   bestFE  $\leftarrow \infty$ 
8   bestPop  $\leftarrow \emptyset$ 
9   popFactor  $\leftarrow$  popsize/dim
10  while fecounter  $<$  maxFE do
11    pop, FE, F, CR  $\leftarrow$  paJADE(X, p, c, dT, funct, minErr,
12      maxFE  $- 2 \cdot \text{dim})
13    fecounter  $\leftarrow$  fecounter  $+ \text{len}(F) \cdot \text{popsize}$ 
14    bestIndex  $\leftarrow \text{argmin}(\mathbf{FE})$ 
15    bestSol  $\leftarrow \mathbf{pop}[\text{bestIndex}]$ 
16    pop, FE  $\leftarrow$  ds(funct, bestSol, minErr, 2 · dim)
17    fecounter  $\leftarrow$  fecounter  $+ 2 \cdot \text{dim}$ 
18    if min(FE)  $<$  bestFE then
19      // increase dimension
20      X  $\leftarrow$  hstack(pop, N(popsize, kernelsize))
21      X  $\leftarrow$  vstack(X, N((popFactor · dim) - popsize, dim))
22      bestFE  $\leftarrow \text{min}(\mathbf{FE})$ 
23      bestPop  $\leftarrow \mathbf{pop}$ 
24      dim, popsize  $\leftarrow$  size(X)
25    end
26    else
27      // reduce dimension
28      X  $\leftarrow$  bestPop  $+ \mathcal{N}(\text{size(bestPop)})$ 
29      dim, popsize  $\leftarrow$  size(X)
30    end
31  end
32  return pop, FE, F, CR$ 
```

---

# I PDE 2 3 4 and 7 Kernel Adaption

The following plots show the fitness difference in relation to the kernel adaption. Darker grey and black areas represent a strong decline of the fitness value over multiple generations, while lighter areas mean that the fitness value is stagnating. Following lighter areas, often comes a green bar, which means that a new kernel is introduced. Similarly, red lines represent the reduction by one kernel. The plots compare the best and the worst replications produced in the experiment chapter 7.

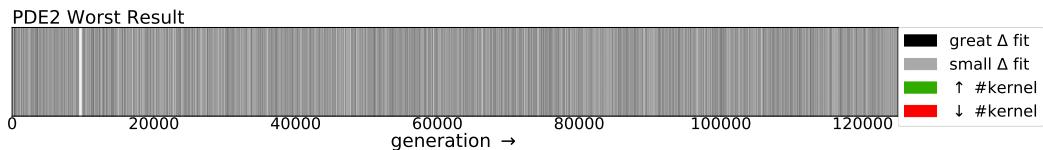


Figure I.1: Kernel Bars Plot on the worst result of PDE2 in experiment 2.

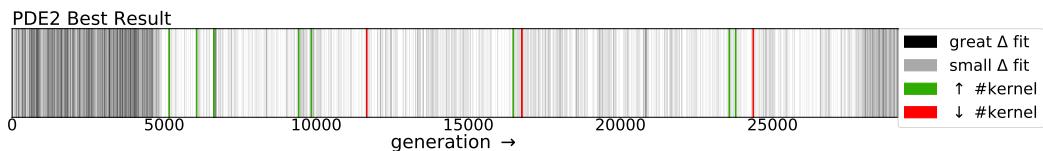


Figure I.2: Kernel Bars Plot on the best result of PDE2 in experiment 2.

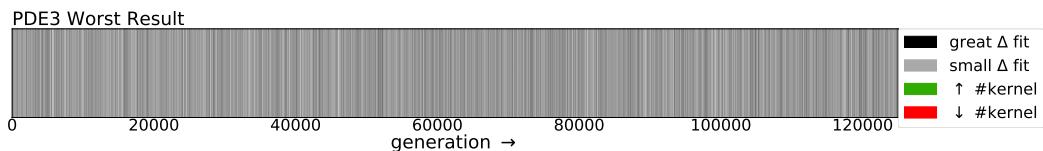


Figure I.3: Kernel Bars Plot on the worst result of PDE3 in experiment 2.

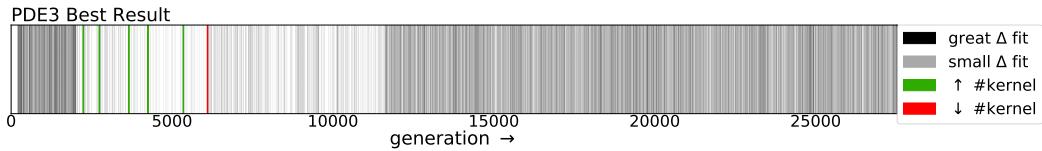


Figure I.4: Kernel Bars Plot on the best result of PDE3 in experiment 2.

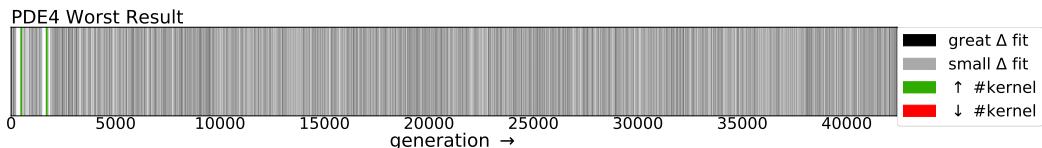


Figure I.5: Kernel Bars Plot on the worst result of PDE4 in experiment 2.

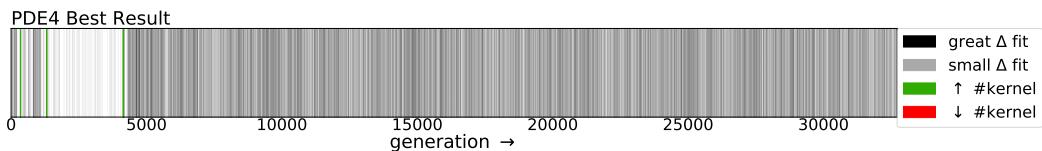


Figure I.6: Kernel Bars Plot on the best result of PDE4 in experiment 2.

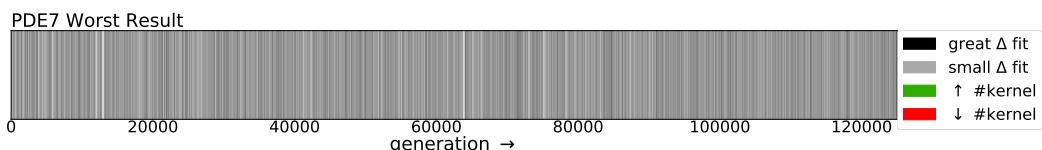


Figure I.7: Kernel Bars Plot on the worst result of PDE7 in experiment 2.

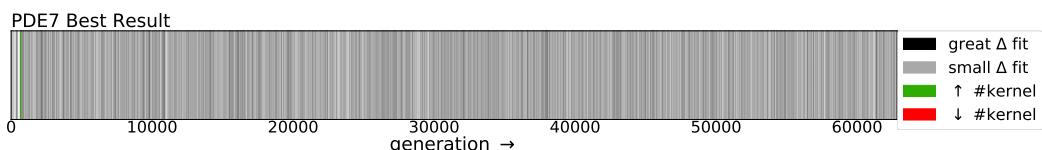


Figure I.8: Kernel Bars Plot on the best result of PDE7 in experiment 2.

# **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, 10.08.2020

Schwartze Nicolai, BSc