

COMPUTATIONAL INTELLIGENCE METHOD FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Nicolai Schwartze

Abstract This master-thesis describes and tests a differential equation solver that is based on the heuristic black-box optimisation algorithm JADE. The solver works by reformulating the residual of the differential equation into a fitness function. The underlying solution function of the differential equation is approximated by a finite sum of Radial Basis Functions. The performance of the solver is evaluated on a testbed that consists of 11 two-dimensional Poisson equations. The solving-time, the memory usage and the accuracy are compared to the open-source Finite Element Solver NGSolve as well as similar work in the current literature. To optimise the solving time and speed up the experiments, JADE is parallelised. This work introduces two new strategies: First, the number of kernels is adjusted to the differential equation during the solving process. Furthermore, a new kernel is formed and tested. These concepts are checked for significant improvements. The structure of the fitness function is investigated. A fundamental symmetry is observed that could be used to develop better suited optimisation algorithms.

Keywords Finite Element Methods; Weighted Residual Method; Partial Differential Equations; Numerical Methods; Differential Evolution; JADE; Downhill Simplex, Memetic Algorithm

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-Basforth (AB) method to whole Finite Element Method (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 Partial Differential Equation (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.

2.1. Finite Element Method

Currently the finite element method is the go-to approach to solve partial differential equations. The domain Ω on which the PDE is posed, is discretised into multiple smaller elements. 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).

Nicolai Schwartze, BSc.
Vorarlberg University of Applied Science
Master Mechatronics 2018
nicolai.schwartzze@students.fhv.at

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.

$$\begin{aligned} \mathbf{x} &\in \mathbb{R}^2 \\ u(\mathbf{x}), f(\mathbf{x}), g(\mathbf{x}) &: \Omega \rightarrow \mathbb{R} \\ \mathbf{L}u(\mathbf{x}) &= f(\mathbf{x}) \text{ on } \Omega \\ \mathbf{B}u(\mathbf{x}) &= g(\mathbf{x}) \text{ on } \partial\Omega \end{aligned} \quad (1)$$

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

$$u(\mathbf{x})|_{\partial\Omega} = g(\mathbf{x}). \quad (2)$$

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. 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

3. Step: Discretisation of Ω

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

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 .

5. Step: Solution

The integral formulation is used to convert the differential equation into a system of linear equations, where the constant factors u_i of the approximate solution $u_h(\mathbf{x})$ need to be determined.

$$\mathbf{A}\mathbf{u} = \mathbf{b} \quad (3)$$

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)
- ⌂ 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 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 (4), 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.

$$F(u_{apx}(\mathbf{x})) = - \int_{\Omega} R(u_{apx}(\mathbf{x}))^2 dx \quad (4)$$

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 (4) 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 (5). The algorithm was tested on multiple problems of Ordinary Differential Equation (ODE), system of ODEs and PDE. Only the results for ODEs were promising.

$$F(u_{apx}(\mathbf{x})) = \sum_{i=1}^{n_C} \|R(u_{apx}(\mathbf{x}_i))\|^2 \quad (5)$$

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 (5). 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

$$u(\mathbf{x})_{apx} = u(\mathbf{x})_{GP} + u(\mathbf{x})_{RBF}. \quad (6)$$

At first, the GP step produced a trial solution according to the objective function (5). 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 α_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 (5) 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.

Chaquet and Carmona 2012 use a simple self-adaptive Evolution Strategy (ES) (as developed by Schwefel 1977 and Rechenberg 1978) to evolve the coefficients of a partial Fourier series. The fitness function is expressed in equation (12). This is similar to the fitness function 5, 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 Ω , whereas n_B is the number of discrete points \mathbf{x}_j on the boundary $\partial\Omega$. Further, a penalty factor ϕ 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.

$$\begin{aligned} F(u_{apx}(\mathbf{x})) &= \frac{1}{m(n_C + n_B)} \\ &\sum_{i=1}^{n_C} \|\mathbf{L}u_{apx}(\mathbf{x}_i) - f(\mathbf{x}_i)\|^2 + \phi \sum_{j=1}^{n_B} \|\mathbf{B}u_{apx}(\mathbf{x}_j) - g(\mathbf{x}_j)\|^2 \end{aligned} \quad (7)$$

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 (8)) and one for the boundary (equation (9)). These are added together resulting in equation (10).

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.

$$WRF(u_{apx}(\mathbf{x})) = \int_{\Omega} |W(\mathbf{x})| |R(u_{apx}(\mathbf{x}))| dx \quad (8)$$

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 and Krishnamoorthy 1992.

$$PFV(u_{apx}(\mathbf{x})) = WRF(u_{apx}(\mathbf{x})) \cdot \sum_{j=1}^{n_B} K_j \left(\frac{u_{apx}(\mathbf{x}_i)}{g(\mathbf{x}_i)} - 1 \right) \quad (9)$$

$$F(u_{apx}(\mathbf{x})) = WRF(u_{apx}(\mathbf{x})) + PFV(u_{apx}(\mathbf{x})) \quad (10)$$

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 (8), 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:

$$u(\mathbf{x})_{apx} = \sum_{i=1}^N \omega_i e^{\gamma_i (||\mathbf{x} - \mathbf{c}_i||^2)} \quad (11)$$

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 \mathbf{p}_{apx} and called the decision variables which are optimised by the algorithm. The objective function can be seen in equation (12). This is an update of the objective function in 7 where the inner collocation points also get scaled by a weighting function $\xi(\mathbf{x}_i)$.

$$F(u_{apx}(\mathbf{x})) = \frac{\sum_{i=1}^{n_C} \xi(\mathbf{x}_i) ||\mathbf{L}u_{apx}(\mathbf{x}_i) - f(\mathbf{x}_i)||^2 + \phi \sum_{j=1}^{n_B} ||\mathbf{B}u_{apx}(\mathbf{x}_j^{\text{Quadrat and } \text{Gauß}}) - g(\mathbf{x}_j)||^2}{m(n_C + n_B)} \quad (12)$$

The multipliers $\xi(\mathbf{x}_i)$ and ϕ are weighting factors for either the inner or the boundary term. The whole term is normalised with the number of collocation points. The parameter of the kernels are determined via a Covariance Matrix Adaption Evolution Strategy (CMA-ES) (Hansen, Müller, and Koumoutsakos 2003). 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 (13). The derivatives within the residual are calculated between two neighbouring points by the difference quotient.

$$F(\mathbf{x}) = \sqrt{\sum_{i=0}^n R(\mathbf{x}_i)^2} \quad (13)$$

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	(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, Brezilianu, and Kolibal 2011	GP	function value grid	convection-diffusion equation at different Peclet numbers
$F(u_{apx}(\mathbf{x}_j^{\text{Quadrat and } \text{Gauß}}) - g(\mathbf{x}_j))$		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
Chaquet 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 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 (14) 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.

$$v_i = x_i + F_i(x_{best}^p - x_i) + F_i(x_{r1} - \tilde{x}_{r2}) \quad (14)$$

- **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 (15) 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.

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } j = K \vee \text{rand}[0, 1] \leq CR \\ x_{ij}, & \text{otherwise} \end{cases} \quad (15)$$

- 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 ??.

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 ?? 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 ??.

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 (16).

$$\begin{aligned} \mathbf{L}u(\mathbf{x}) &= f(\mathbf{x}) \\ \text{subjected to: } \mathbf{B}u(\mathbf{x}) &= g(\mathbf{x}) \end{aligned} \quad (16)$$

This is similar to the formulation of the strong form in equation (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 (17). 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.

$$u(\mathbf{x}) \approx u_{apx}(\mathbf{x}) = \sum_{k=0}^N a_k \phi_k(\mathbf{x}) \quad (17)$$

The residual of a PDE, as shown in equation (18), 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“.

$$R(\mathbf{x}) = \mathbf{L}u_{apx}(\mathbf{x}) - f(\mathbf{x}) \quad (18)$$

The WRM (Shen, Tang, and Wang 2011) tries to minimise this residual of a numerical candidate solution over the whole domain Ω . Therefore, R at every $\mathbf{x} \in \Omega$ is evaluated and added up, resulting in the following integral of equation (19). 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.

$$WRF = \int_{\Omega} R(\mathbf{x})\psi(\mathbf{x})W(\mathbf{x})dx \quad (19)$$

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 (20). Although it is not an integral per se, it does assign a numerical “score” to the residual.

$$\sum_{k=0}^N R(\mathbf{x}_k)\psi(\mathbf{x}_k)W(\mathbf{x}_k) \quad (20)$$

To ensure that negative and positive residuals do not cancel each other out, a common choice in heuristic methods from table 5 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 (21).

$$\sum_{k=0}^N R(\mathbf{x}_k)^2 W(\mathbf{x}_k) \quad (21)$$

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 (21). 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

(12), 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 (22) 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.

$$F(u_{apx}(\mathbf{x})) = \frac{\sum_{i=1}^{n_C} \xi(\mathbf{x}_i) [\mathbf{L}u_{apx}(\mathbf{x}_i) - f(\mathbf{x}_i)]^2 + \phi \sum_{j=1}^{n_B} [\mathbf{B}u_{apx}(\mathbf{x}_j) - g(\mathbf{x}_j)]^2}{(n_C + n_B)} \quad (22)$$

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: ξ for the inner collocation points and ϕ as penalty factor for the boundary points. The weighting function ξ can be adapted by a parameter κ . For a $\kappa > 0$, ξ assigns larger values for collocation points closer to the boundary, effectively shifting the relative importance towards the boundary. These weights can be calculated a priori, so no computational effort is added to the fitness function. Again, the weights are directly extracted from Chaquet and Carmona 2019.

$$\xi(\mathbf{x}_i) = \frac{1 + \kappa \left(1 - \frac{\min_{\forall \mathbf{x}_j \in n_B} \|\mathbf{x}_i - \mathbf{x}_j\|}{\max_{\forall \mathbf{x}_k \in n_C} (\min_{\forall \mathbf{x}_j \in n_B} \|\mathbf{x}_k - \mathbf{x}_j\|)} \right)}{1 + \kappa} \quad (23)$$

3.3. Candidate Representation

As Chaquet and Carmona 2019 suggest, an approximate solution should be represented as a summation of N scaled and shifted radial basis functions ϕ , as stated in equation (17) Gauss kernels have frequently been used for the approximation of functions. A solid foundation on the approximation theorem can be seen in Park and Sandberg 1991. RBF are functions that are point symmetric to a centre c and their function value only depends on the radius r .

$$r = \|\mathbf{x} - \mathbf{c}\| \quad (24)$$

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

3.3.1. Gauss Kernel

In general, a Gauss Kernel (GaK) is the classical choice to approximate functions. It was first introduced in Broomhead and Lowe 1988. Equation (25) shows the formulation of such a kernel. One kernel has 4 parameters, that change its shape and location. ω is a scaling factor for each and γ describes how sharp the e-function is. Depending on the space the solution is defined in, the number of 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 1 a single standard GaK is plotted.

$$gak(\mathbf{x}) = \omega e^{-\gamma \|\mathbf{x} - \mathbf{c}\|^2} \quad (25)$$

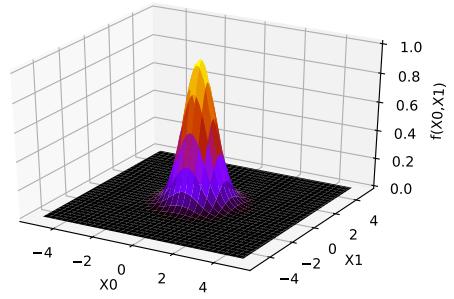


Figure 1: 3D plot of a single Gauss kernel with the parameters $\omega = 1$, $\gamma = 1$, $c_0 = 0$ and $c_1 = 0$

An approximate solution to the PDE in question is described as the superposition of N kernels as shown in equation (26).

$$u_{apx}(\mathbf{x}) = \sum_{i=0}^N \omega_i e^{-\gamma_i r_i^2} \quad (26)$$

A solution is encoded as a vector of these $4 \cdot N$ parameters stacked together. This is shown in equation (27).

$$\mathbf{p}_{apx} = \left[\underbrace{\omega_0, \gamma_0, c_{00}, c_{01}}_{\text{kernel 0}}, \underbrace{\dots, \omega_i, \gamma_i, c_{i0}, c_{i1}}_{\text{kernel i}}, \underbrace{\dots, \omega_N, \gamma_N, c_{N0}, c_{N1}}_{\text{kernel N}} \right]^T \quad (27)$$

To evaluate the residual of a PDE, the derivatives must be known. To that extend, the derivative of u_{apx} from equation (17) is calculated. The first order derivative with respect to x_0 is seen in equation (28). To solve the proposed testbed, also the second order derivatives are needed, as described in equation (29). Although the mixed term derivative is not used in this work, for the sake of completeness it is displayed in equation (30).

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_j} = -2 \sum_{i=0}^N \omega_i \gamma_i (x_j - c_{ij}) e^{-\gamma_i r_i^2} \quad (28)$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j^2} = \sum_{i=0}^N \omega_i \gamma_i [4\gamma_i (x_j - c_{ij})^2 - 2] e^{-\gamma_i r_i^2} \quad (29)$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j \partial x_k} = 4 \sum_{i=0}^N \omega_i \gamma_i^2 (x_j - c_{ij})(x_k - c_{ik}) e^{-\gamma_i r_i^2} \quad (30)$$

3.3.2. GSin Kernel

Additional to the Gauss kernel, a “GSin” kernel, further also abbreviated as Gauss Sine Kernel (GSK), is proposed. In essence, this is a multiplication of a GaK with a sine function, as seen in equation (31). It can be thought of a sine function, where its influence declines with the radius r . The plot in figure 2 shows how the first half period is much larger than the second half. Both “sub-functions” are centred at the same \mathbf{c} . As shown in chapter 11, the universal approximation theorem for the GSin kernel holds true. With the sine, two new parameters are introduced: f for the frequency of the sine wave and φ represents the

phase shift.

$$gsk(\mathbf{x}) = \omega e^{-\gamma ||\mathbf{x} - \mathbf{c}||^2} \sin(f ||\mathbf{x} - \mathbf{c}||^2 - \varphi) \quad (31)$$

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

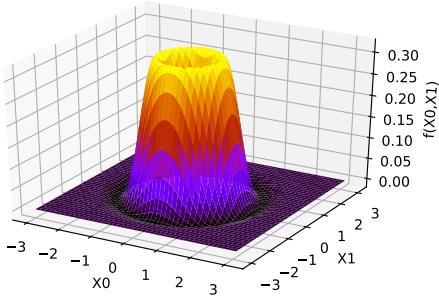


Figure 2: 3D plot of a single GSin kernel with the parameters $\omega = 1$, $\gamma = 1$, $c_0 = 0$, $c_1 = 0$, $f = 1$, $\varphi = 0$

Similar to the Gauss kernel, the approximate solution is described as the superposition of N GSin kernels, as seen in equation (32).

$$u_{apx}(\mathbf{x}) = \sum_{i=0}^N \omega_i e^{\gamma_i r_i^2} \sin(f_i r_i^2 - \varphi_i) \quad (32)$$

Equation (33) shows the solution representation in stacked vector notation, including the two new parameters f and φ .

$$\mathbf{p}_{apx} = \left[\underbrace{\omega_0, \gamma_0, c_{00}, c_{01}, f_0, \varphi_0, \dots}_{\text{kernel 0}} \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}, f_i, \varphi_i, \dots}_{\text{kernel i}} \underbrace{\omega_N, \gamma_N, c_{N0}, c_{N1}, f_N, \varphi_N}_{\text{kernel N}} \right] \quad (33)$$

Again, the derivatives of the solution are needed, which are shown in the equations (34) through (36).

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_j} = \sum_{i=0}^N 2\omega_i (x_j - c_{ij}) e^{-\gamma_i r_i^2} (f_i \cos(f_i r_i^2 - \varphi_i) - \gamma_i \sin(f_i r_i^2 - \varphi_i)) \quad (34)$$

$$\begin{aligned} \frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j^2} &= \sum_{i=0}^N 2\omega_i e^{-\gamma_i r_i^2} \\ &[-(2(f_i^2 - \gamma_i^2)(x_j - c_{ij})^2 + \gamma_i) \sin(f_i r_i^2 - \varphi_i) - (4f_i \gamma_i (x_j - c_{ij})^2 - f_i) \cos(f_i r_i^2 - \varphi_i)] \end{aligned} \quad (35)$$

$$\begin{aligned} \frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j x_k} &= \sum_{i=0}^N -4\omega_i (c_{ij} - x_j)(c_{ik} - x_k) e^{-\gamma_i r_i^2} \\ &[(f_i^2 - \gamma_i^2) \sin(f_i r_i^2 - \varphi_i) + 2f_i \gamma_i \cos(f_i r_i^2 - \varphi_i)] \end{aligned} \quad (36)$$

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 measurement 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 (Schöberl, Lackner, and Hochsteger 2020).

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 ???. 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: ???

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: ???

Characteristics: can be approximated arbitrarily close by 3 GSK; only 3 kernels are used to keep the minimal necessary dimension of the representation 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: ???

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: ???

Characteristics: compare the performance to other solvers;
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: (??)

Characteristics: compare the performance to other solvers; 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: (??)

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 Ω
Boundary: homogeneous boundary condition;
Reference: Mitchell 2018

PDE 5: Arctan Circular Wave Front

Equation Reference: (??)

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: (??)

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: (??)

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: (??)

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: (??)

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 Unified Modeling Language (UML) class diagram can be seen in the appendix ???. The limited class diagram, without any methods and attributes is shown in the figure 3 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 method-

s/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 ??.

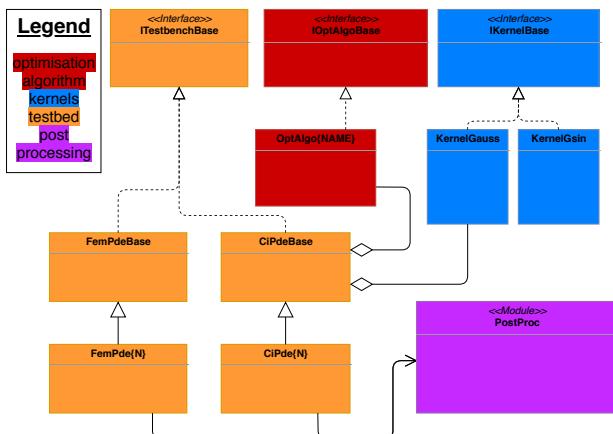


Figure 3: This reduced UML class diagram gives an overview of the testbed design and its interfaces and classes.

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 variable. 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 ??.

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 variable. Thus, similar to the time measurement, replications have to be performed. The same pseudocode as for the time measurement (appendix ??) also applies here. To remove outliers, it is helpful to create and solve one testbed object before recording the experiment. This sets up the necessary references and then are mingled into the actual solving process. This holds true for both, the FEM and the CI solver.

4.3.3. Quality Measurement

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

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

Thus, Chaquet and Carmona 2019 define a new quality measurement based on the Root-Mean-Square Error (RMSE) over the collocation points as seen in equation (37).

$$RMSE^2 = \frac{\sum_{i=1, \mathbf{x}_i \in C}^{n_C} \|u(\mathbf{x}_i) - u_{ext}(\mathbf{x}_i)\|^2 + \sum_{j=1, \mathbf{x}_j \in B}^{n_B} \|u(\mathbf{x}_j) - u_{ext}(\mathbf{x}_j)\|^2}{(n_C + n_B)} \quad (37)$$

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 variables, subsequent measures of similar solutions could result in different quality values. Especially in solutions with high condition numbers, the RMSE measurement would be rendered useless

since a small deviation of the point results in a large difference of the function value.

Further with the RMSE, 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.

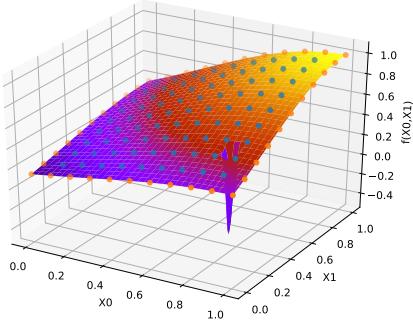


Figure 4: Aliasing Error: sharp inaccuracies in between collocation points

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 (38). This actually measures the distance between the analytical solution and the numerical approximation.

$$\|u_{ext} - u_{apx}\| = \sqrt{\int_{\Omega} (u_{ext}(\mathbf{x}) - u_{apx}(\mathbf{x}))^2 d\mathbf{x}} \quad (38)$$

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

4.4. Baseline: NGSolve

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

4.4.1. Setup

At first the PDEs must be transformed into their corresponding weak form. As all testbed problems are Poisson equations and only differ in their algebraic sign and inhomogeneous part, the weak form is similar for all problems. Referring to equation (??), 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 (39), where the f is the inhomogeneous part of the PDE.

$$\int_{\Omega} \nabla u^T \nabla v dV = \int_{\Omega} f v dV \quad (39)$$

To enhance the performance of NGSolve, static condensation is turned on. Further, a multigrid preconditioner is used. All problems are approximated by second order polynomials. The automatic mesh refinement is performed until a maximum of $5 \cdot 10^4$ Degree of Freedom (DOF) is reached. To properly interpret the results, 20 replications for each problem are performed. This is only needed for regarding time and memory, the solution itself is not a random variable. 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 5 and 6, 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 2 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 2: 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.

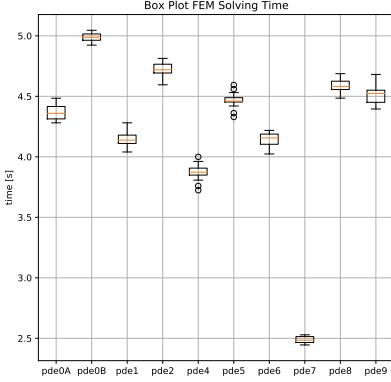


Figure 5: Boxplot: time (in seconds) needed to solve the testbed PDE (without PDE3)

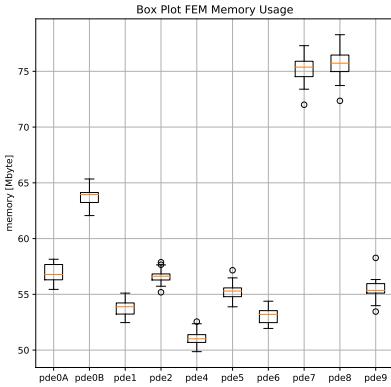


Figure 6: Boxplot: memory (in Mbyte) needed to solve the testbed PDE (without PDE3)

The following figures 8a and 8b 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 (??), 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 iterations to produce the $5 \cdot 10^4$ DOF as compared to the other testbed problems. In fact, since the numerical errors per finite element accumulate, more DOF should result in a larger approximation error. Figure 7 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 a better quality.

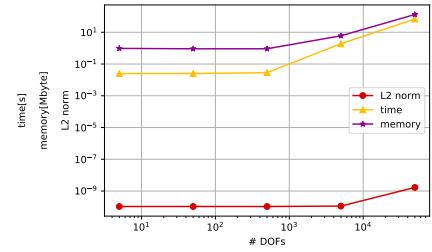
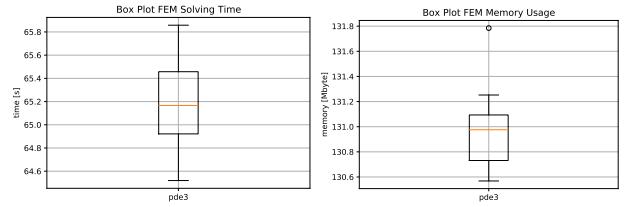


Figure 7: PDE 3 performance metrics at different DOF budgets.



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

5. Default CI Parameter

Typically, the performance of heuristic optimisation algorithms can be adjusted to specific testbed problems by tuning its parameters. For all further experiments the JADE algorithm from pseudocode ?? is used. 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 determined by preliminary test. If not stated otherwise, the following parameters from table 3 are used in the subsequent experiments.

Parameter	JADE	CMA-ES (Chaquet and Carmona 2019)
φ	100	300
κ	1	3
population size	$2 \cdot \text{dim}$	$\frac{3}{2}(4 + \lceil 3 \cdot \ln(\text{dim}) \rceil)$
min error	0	-
p	0.3	-
c	0.5	-
replication	20	50
nb	40	100 equally spaced points over the domain
nc	81	
	$121 = 11 \times 11$	
initialisation	$\vec{u}_{apx} \in \mathcal{N}(0, 1)$	$\omega_i \in \mathcal{U}[-0.01, 0.01]$ $\gamma_i \in \mathcal{U}(0, 1]$ $c_{ik} \in \mathcal{U}[20]$

Table 3: These predefined parameters are used for the following numerical experiments.

The parameters φ and κ are used in the fitness function (equation (22)) 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 9 shows the values of ξ and φ . It further describes

how the weighting factor emphasises the areas closer to the boundary.

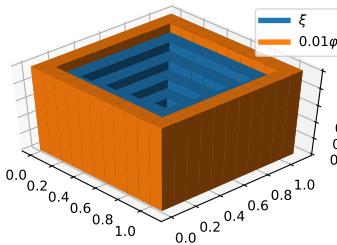


Figure 9: Weighting factor φ and ξ on every collocation point

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 $2 \cdot \text{dim}$ are more likely to stagnate, 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 (22)) has its optimum at 0, this value is used as the termination condition. A helpful side-effect is that it ensures the same amount of function evaluation in every run, without early termination. This prevents outliers in the time and memory measurement.

The parameters p and c are specific to JADE (algorithm ??). 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, every setup is reevaluated by 20 replications where each replication starts with independent initial guesses.

Chaquet and Carmona 2019 use 100 equally spaced collocation points over the domain to solve the PDE. Here, 121 points are created, as seen in figure 10.

The initialisation is done by a standard normal distribution. This means that every value in the `papx` vector is drawn from a normal distribution. On the contrary, Chaquet and Carmona 2019 initialise the values specifically tailored to each parameter of the kernel. This process assumes a priori information about the solution. In the typical application, this knowledge is not available and thus it should not be used.

5.1. Hardware Infrastructure

To increase the experimental throughput, two machines are used: a personal computer (machine 1) and a server in the cloud (machine 2). It is important that

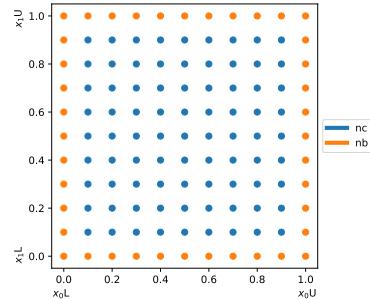


Figure 10: Collocation points used in the testbed. For the PDEs 0A and 0B with the larger domain, the lower limits x_0L and x_1L are replaced with -2 and the upper limits x_0U and x_1U are replaced with 2. The points are still equally spaced.

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 shows the properties of both machines. It is important to note that these information are only a reference point. This is probably not enough to actually recreate the exact time or memory data presented in this work.

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: 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.

6. 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.

6.1. 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 1 below shows the implementation of such a memetic JADE. At first, JADE performs the global search and places its population around the optimum. Then the Downhill Simplex (DS) exploits this area. The budget of Number of Function Evaluation (#FE) is split into two parts: JADE takes nearly all #FE and leaves $2 \cdot \text{dim} \cdot \#FE$ for the DS. The SciPy implementation of the DS is used (SciPy 2020).

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.

Algorithmus 1: Pseudocode of memetic 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  $\leftarrow$  JADE( $\mathbf{X}$ , 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

The standard parameters from table 3 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 ??.

6.3. Results

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

Table 5: 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 (37) is calculated.

The following table 6 lists the smallest L2 norm reached after 10^4 #FE and 10^6 #FE, respectively. A standard Wilcoxon test, as described in the appendix ??, is performed. Remarkable is that the results on PDE 5 get significantly worse when more #FE are used.

#FE stat	10^4		10^6		Wilcoxon Test
	mean	median	mean	median	
PDE 0A	1.9415 ± 0.3321	1.8844	0.6596 ± 0.5510	0.9285	sig. better
PDE 0B	0.7137 ± 0.1979	0.6354	0.2027 ± 0.1302	0.1516	sig. better
PDE 1	0.1874 ± 0.0408	0.1938	0.0149 ± 0.0049	0.0151	sig. better
PDE 2	0.0890 ± 0.0334	0.0760	0.0257 ± 0.0140	0.0224	sig. better
PDE 3	0.2409 ± 0.1051	0.2309	0.0328 ± 0.0169	0.0285	sig. better
PDE 4	0.1102 ± 0.0367	0.0985	0.0378 ± 0.0083	0.0352	sig. better
PDE 5	0.6645 ± 0.1930	0.6263	1.1968 ± 0.0286	1.2056	sig. worse
PDE 6	1.9660 ± 1.3845	1.6540	0.4133 ± 1.2133	0.0018	sig. better
PDE 7	0.0457 ± 0.0137	0.0452	0.0221 ± 0.0019	0.0223	sig. better
PDE 8	0.2186 ± 0.0045	0.2191	0.2170 ± 0.0019	0.2175	unsig. better
PDE 9	0.0525 ± 0.0147	0.0516	0.0451 ± 0.0119	0.0459	unsig. better

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

The following two images 11 and 12 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.

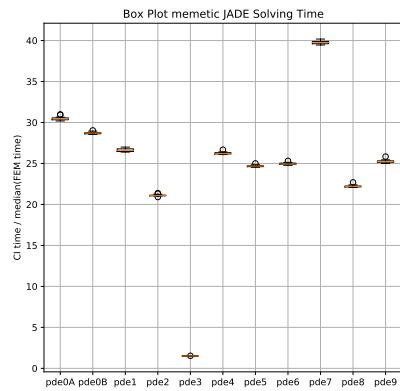


Figure 11: Relative solving time results of memetic JADE after 10^4 #FE.

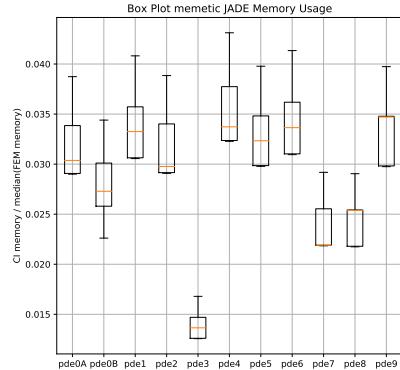


Figure 12: Relative memory usage results of memetic JADE after 10^4 #FE.

6.4. Discussion

The results presented in chapter 6.3 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.

6.4.1. 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 3, 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. In general, more kernels result in a better solution, which is confirmed by experiments in Chaquet and Carmona 2019 and in chapter 8.4.4. 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.

6.4.2. Solving Time/Memory Usage

The results from table 6 show that the CI solver can not nearly compete with the results obtained by the FEM solver from table 2. 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 11) 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 12). An interesting observation is the distribution of the solving time within the testbed. The more operations a fitness function needs, the more expensive is one function evaluation and thus 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 ($29 \cdot 5 \text{ s} \approx 145 \text{ s}$), compared to the other PDEs, it reaches a “fairly” good quality.

6.4.3. 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, his 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 6), 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 Ω . The comparison of two solutions with 10^4 #FE and 10^6 #FE in figure 13 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.

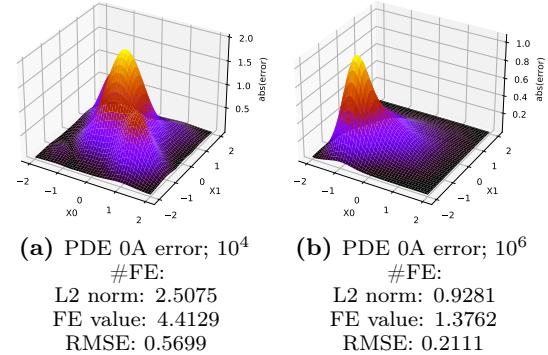


Figure 13: Comparison of two typical PDE 0A solutions.

6.4.4. 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 6. 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 14. 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.

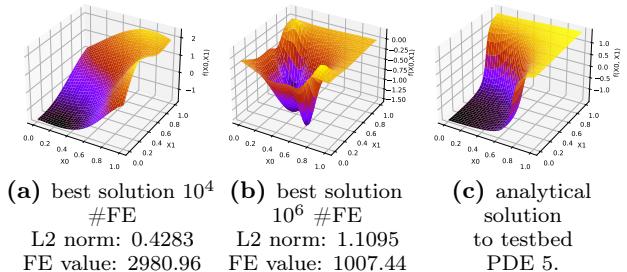


Figure 14: Comparison of best solution after 10^4 and 10^6 #FE.

The distributions of the achieved fitness value after 10^4 and 10^6 #FE is shown in figure 15a. 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 15b. 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.

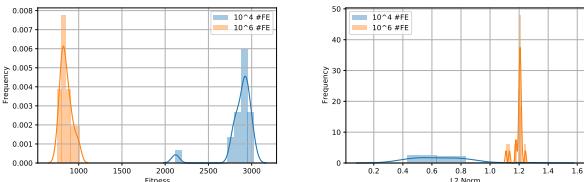


Figure 15: Histograms of the fitness and the L2 norm reached with 10^4 #FE and 10^6 #FE on PDE 5.

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 ??). The monotonically decreasing fitness value at every generation of an exemplary individual within the population is plotted in figure 16. 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 16.

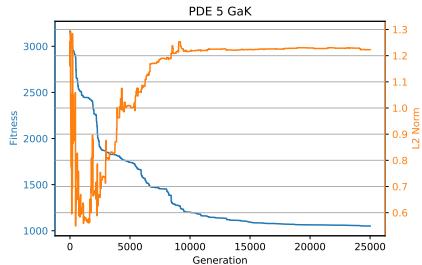


Figure 16: Fitness value and L2 Norm of one individual at every generation on PDE 5.

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 8. The GSK kernel type is examined in chapter 9.

7. 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.

7.1. Hypotheses

The actual parallel JADE algorithm is shown in appendix ???. The pseudocode ?? is very similar to the JADE algorithm ???. 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 2 below. The only difference compared to the simple memetic JADE 1 is the usage of the pJADE in line 5.

Algorithmus 2: 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  $pop, FE, F, CR$ 
```

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.

7.2. Experiment Setup

Similar to the experiment 0 before, the standard parameters from table 3 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 ???) 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.

7.3. Results

The two images 17 and 18 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.

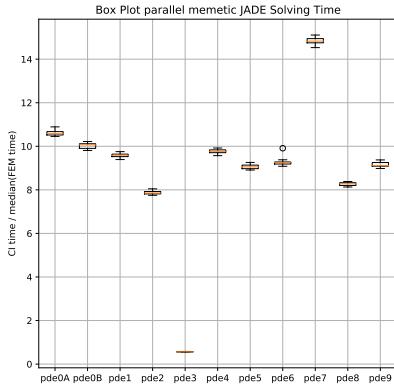


Figure 17: Relative solving time results of parallel memetic JADE after 10^4 #FE.

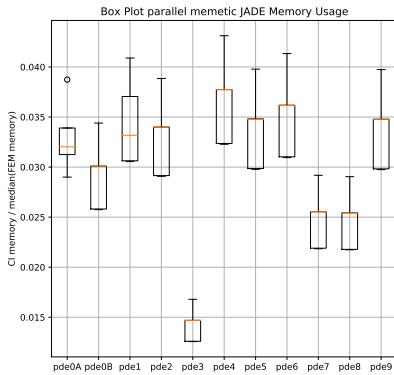


Figure 18: Relative memory usage results of parallel memetic JADE after 10^4 #FE.

Table 7 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 stat	serial JADE		parallel JADE		Wilcoxon Test
	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 ± 0.2133	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 7: 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 6.

7.4. 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.

7.4.1. Memory Usage

As the boxplot 18 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.

7.4.2. 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 17 shows, this goal was achieved. The execution time after 10^4 #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 $0.56 \cdot 65.2$ s \approx 36.5 s. Of course, to compete with the solution quality, more #FE are needed, which increases the execution time.

Image 19 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.

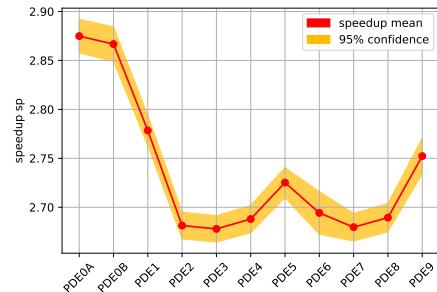


Figure 19: Mean speed-up of parallel JADE with 95% confidence interval.

The plot 19 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 ($time(fit evaln) \uparrow \rightarrow sp \uparrow$). One would expect the same correlation for the size of the population: the larger a population, the better is the speed-up ($pop\ size \uparrow \rightarrow sp \uparrow$). Larger fitness functions are needed by the testbed PDEs 0A, 0B, 1, 5, and 9. Looking at the definition of these equations in appendix ??, it is notable that the RHS are constructed of more terms compared to the other equations.

7.4.3. PDE 4

Table 7 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 20. 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.

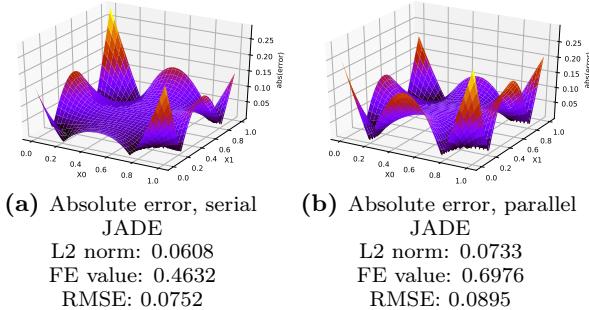


Figure 20: Comparison of the absolute error of the worst solution on PDE 4 by a parallel and a serial memetic JADE at 10^6 #FE

In order to make the results easier to interpret, the histogram from figure 21 shows the L2 norm data from table 7. 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 22. 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.

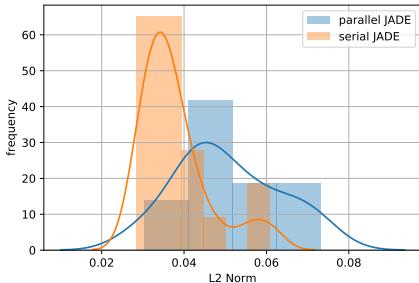


Figure 21: Histogram of the L2 norm data obtained by the serial and the parallel JADE on PDE 4 from table 7.

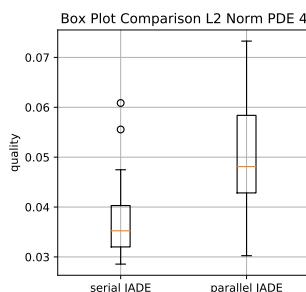


Figure 22: Boxplot of the L2 norm data from table 7 on PDE 4.

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.

8. 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 6.4.3. A possible method to compensate that could be to adapt the number of kernels along the solving process, instead of arbitrarily using 5 kernels.

8.1. 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 ?? in the appendix ?? 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 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 23. The corresponding pseudocode ?? 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.

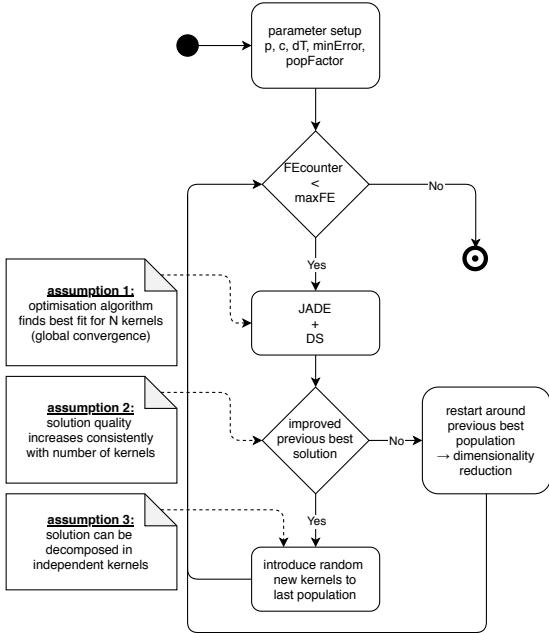


Figure 23: Flowchart of the adaptive kernel scheme.

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. Generally, the maximum number of kernels is not known except for PDE 0A and PDE 6.
- **Assumption 3:** The best approximation of e.g. 3 kernels to a particular problem is independent of the best approximation by 4 kernels. This means that from 3 to 4 kernels simply a new kernel is introduced while not altering the other 3. Again, this is not true for every PDE. Preliminary experiments on PDE 0A have confirmed this assumption, while on other PDEs, like PDE 2, the solution can not simply be decomposed into independent kernels.

The concept is similar to the one explained in Chaquet and Carmona 2012. They start the approximation process by optimising the first harmonic. During the search for the second harmonic, the first one is fixed in place. In this algorithm, the first kernels are allowed to change. When introducing a new random kernel, it is simply appended to the ever evolving \mathbf{p}_{apx} vector. Thus, the search for the 4th kernel starts where the best approximation for 3 kernels was found, but since the earlier kernels are allowed to readapt, other solutions can be retrieved.

This chapter tries to answer the question if this strat-

egy is an effective method to increase the quality of obtained solutions. Also, the time and memory aspects are investigated.

8.2. 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 ?? 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.

8.3. Result

Similar to the experiments before, the two images 24 and 25 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.

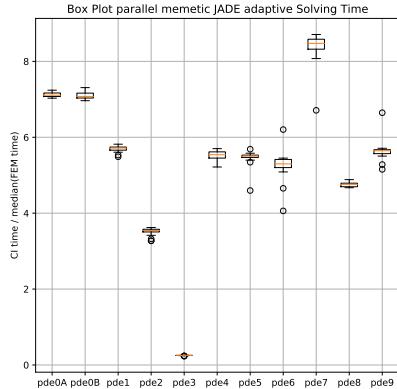


Figure 24: Relative solving time results of parallel memetic JADE with adaptive kernels after 10^4 #FE.

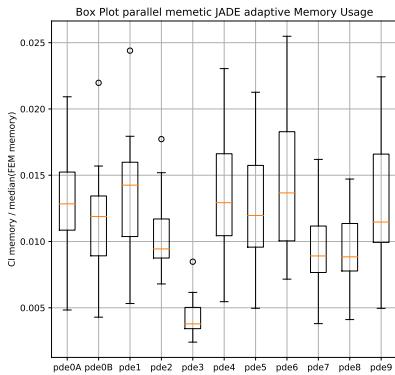


Figure 25: Relative memory usage results of parallel memetic JADE with adaptive kernels after 10^4 #FE.

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

Table 8: Comparison of the achieved L2 norm by the pjADE and the paJADE at 10^6 #FE. The pjADE data is directly taken from table 7.

Since the number of kernels is not predefined, the resulting solution may have any amount of kernels. This is also a random variable, and the results are shown in the boxplot 26 below. As expected, this varies between different testbed problems, but it can not be less than 1 kernel.

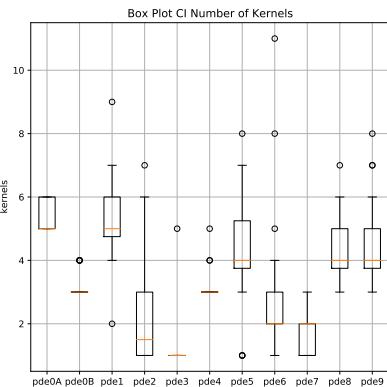


Figure 26: Number of kernels of the proposed solution by the paJADE per PDE at 10^6 #FE

8.4. Discussion

This chapter analyses the presented results from section 8.3. The time and memory the 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.

8.4.1. 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 24. 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.

8.4.2. Memory Usage

The adaptive approach also leads to a reduction in the memory usage, as described by the memory boxplot in figure 25. 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.

8.4.3. 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 26 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 2.

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

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

Figure 27 shows the 3D plot of two separate unnecessary kernels produced by the adaptive memetic pjADE. A kernel with increased γ is not included in the dataset.

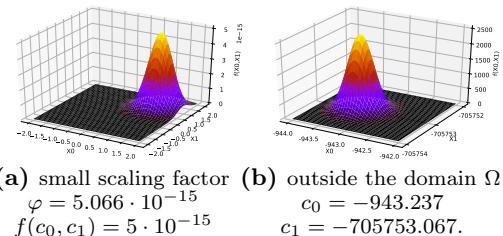


Figure 27: 3D plot and comparison of unnecessary 6th kernels on PDE 0A.

8.4.4. PDE 2 3 4 7

The Wilcoxon significance test of table 8 shows that the adaptive scheme is worse for the PDEs 2, 3, 4 and 7.

The boxplot 26 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 28 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.

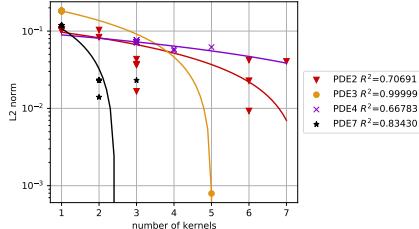


Figure 28: Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are taken at $10^6 \#FE$ on the PDEs 2, 3, 4 and 7. The corresponding R^2 values of the linear regression are denoted.

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 ?? and confirms this hypothesis. A simple solution to mitigate this issue might be to adjust the parameters of the “state-detector”. In this experiment, $minError = 0$ is used, however it might be beneficial to allow small changes in the function value and still terminate.

Parameter Adaption: $minError$

In this “sub-experiment” the effect of increasing the $minError$ parameter is examined. Therefore, the same algorithm is rerun on the PDEs 2, 3, 4 and 7 at four different $minError$ 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 29a. Figure 29b shows the connection between the median L2 norm and the $minError$. 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 9.

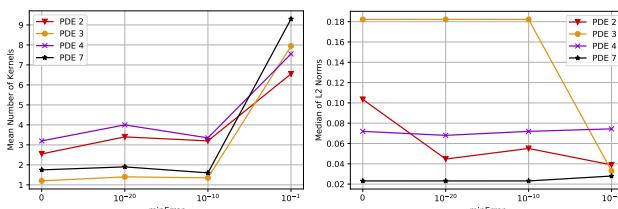


Figure 29: Comparison of $minError$ against the number of kernels and the achieved solution quality.

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

Table 9: 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 8.

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 30. Compared to the same plot at $minError = 0$ (figure 28), the coefficient of determination R^2 is smaller, indicating a very poor correlation.

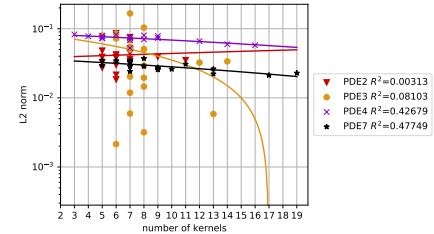
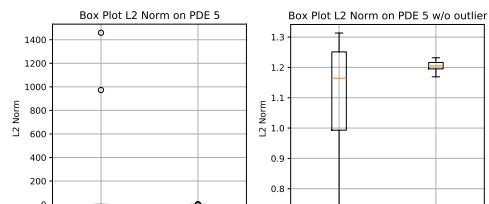


Figure 30: Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are produced with a $minError = 10^{-1}$ after $10^6 \#FE$.

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.

8.4.5. PDE 5

The results presented in table 8 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 31a. The same data is shown twice, with and without the outlier.



(a) Boxplot of PDE 5 solution quality with outlier. **(b)** Boxplot of PDE 5 solution quality without outlier.

Figure 31: Boxplot of solution quality on PDE 5 with a budget of $10^6 \#FE$. Comparison of distribution with and without outliers.

The approximate solution of the outlier is shown in

the two 3D plots in figure 32 below. It is clearly seen that both results have nothing in common with the actual solution from figure 32c. In general, it can be said that the adaptive scheme exhibits a greater spread in the quality of the solution.

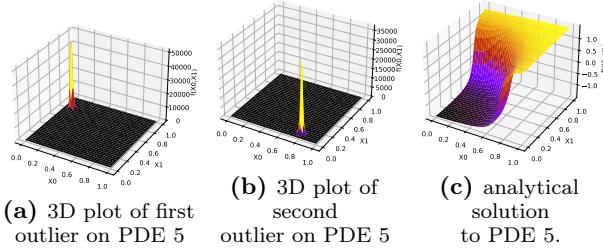


Figure 32: 3D plot of the outlier solution produced by the adaptive kernel scheme on PDE 5 after 10^6 #FE.

Both outliers have a very similar structure. This might indicate, that JADE diverges to areas where the fitness function does not give a proper signal towards the optimum. The adaptive strategy can not improve the results on PDE 5. A second possibility would be to use a different kernel type.

8.4.6. 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 33. Even good solutions are made of more than one kernel, as the boxplot in 26 indicates. This means that the adaptive kernel strategy is not successful on PDE 6. JADE terminates to 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 8.4.3.

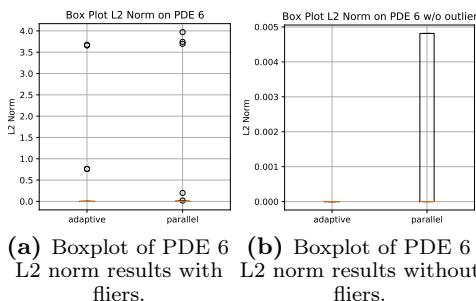
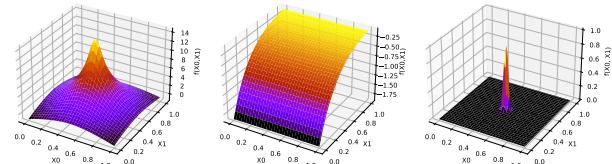


Figure 33: Comparison of the L2 norm reached by the adaptive and parallel paJADE on PDE 6 after 10^6 #FE.

Besides the good results, there are two recurring bad solutions as seen in figure 34. Their 3D plot shows no resemblance with the actual solution from figure 34c. The fact that these obviously bad approximations happen multiple times indicates that JADE converges to these local optima.



35 36

Figure 34: 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 35 and 36. 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 34a is created by increasing the kernel count, although one kernel should be sufficient. Similar, figure 34b consists of a single kernel, where a local optimum is exploited. Both false solutions occur twice within the 20 replications.

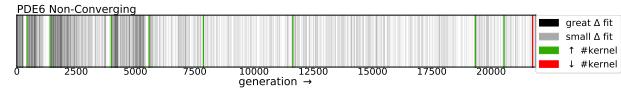


Figure 35: Kernel bar plot of the false solution, represented in figure 34a

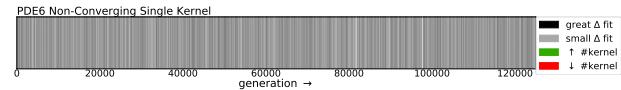


Figure 36: Kernel bar plot of the false solution, represented in figure 34b

9. 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.

9.1. Hypotheses

As stated in chapter 6.4.4 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.

9.2. 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 ?? 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 3.

9.3. Result

The box plots in figure 37 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 38 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 10. The comparison is done with a budget of 10^6 #FE.

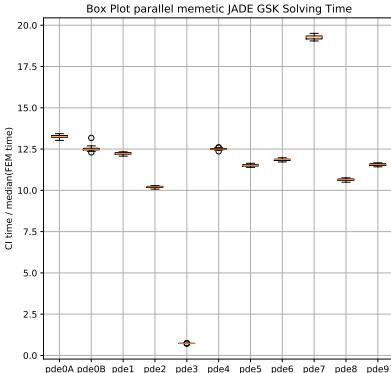


Figure 37: Relative solving time results of parallel memetic JADE with GSK after 10^4 #FE.

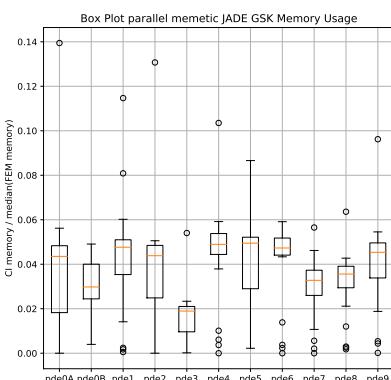


Figure 38: Relative memory usage results of parallel memetic JADE with GSK after 10^4 #FE.

Algorithm	parallel JADE	GaK	10^6 #FE	parallel JADE	GSK	10^6 #FE	Wilcoxon Test
	stat	mean	median	mean	median		
PDE 0A	0.6939 ± 0.6635	0.9243	0.8106 ± 0.7929	0.6765	0.6765	0.6765	unsig. undecided
PDE 0B	0.2809 ± 0.3071	0.2035	0.0667 ± 0.0470	0.0614	0.0614	0.0614	sig. better
PDE 1	0.0239 ± 0.0467	0.0146	0.1665 ± 0.1015	0.1952	0.1952	0.1952	sig. worse
PDE 2	0.0300 ± 0.0157	0.0255	0.0448 ± 0.0224	0.0416	0.0416	0.0416	unsig. worse
PDE 3	0.0371 ± 0.0206	0.0295	0.0263 ± 0.0111	0.0269	0.0269	0.0269	unsig. better
PDE 4	0.0504 ± 0.0121	0.0481	0.0470 ± 0.0078	0.0455	0.0455	0.0455	unsig. better
PDE 5	1.2030 ± 0.0465	1.2053	0.5860 ± 0.2149	0.6841	0.6841	0.6841	sig. better
PDE 6	0.5814 ± 1.3550	0.0000	3.7321 ± 0.6397	3.9079	3.9079	3.9079	sig. worse
PDE 7	0.0228 ± 0.0025	0.0226	0.0243 ± 0.0046	0.0241	0.0241	0.0241	unsig. worse
PDE 8	0.2167 ± 0.0017	0.2169	0.2154 ± 0.0018	0.2150	0.2150	0.2150	sig. worse
PDE 9	0.0426 ± 0.0115	0.0463	0.0351 ± 0.0099	0.0333	0.0333	0.0333	unsig. better

Table 10: Statistical comparison of the the parallel JADE using the GaK and the GSK.

9.4. 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 7. 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.

9.4.1. Solving Time

The solving time for the memetic pJADE with GSK forms the same pattern as the time consumption by the GaK from figure 24. 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.

9.4.2. 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 18. The greater memory usage can be explained by the larger search dimension and the greater population size.

9.4.3. PDE 0A and PDE 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 10 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.

9.4.4. 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 39 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.

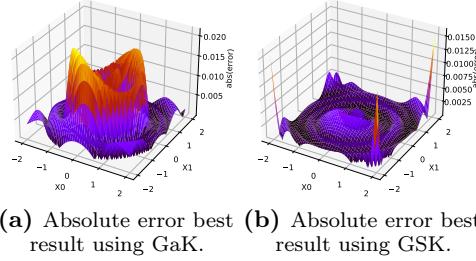


Figure 39: Comparisons of best solution absolute error by memetic parallel JADE using GaK and GSK after 10^6 #FE.

9.4.5. 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 10 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 40 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.

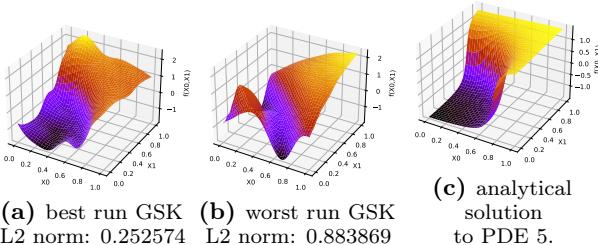
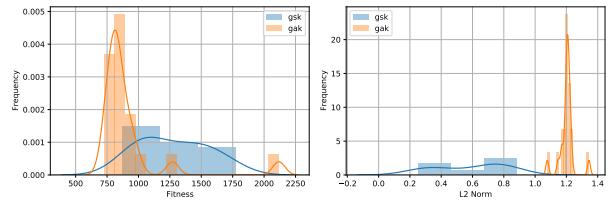


Figure 40: Comparison of the best and the worst result generated by memetic pJADE with GSK after 10^6 #FE.

The following histograms in figure 41b 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 41a. 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.



(a) Histogram of fitness value **(b)** Histogram of L2 norm using GSK and GaK.

Figure 41: Histograms of GSK and GaK L2 norm and fitness value on PDE 5 after 10^6 #FE.

Similar to the plot in chapter 6.4.4, figure 42 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.

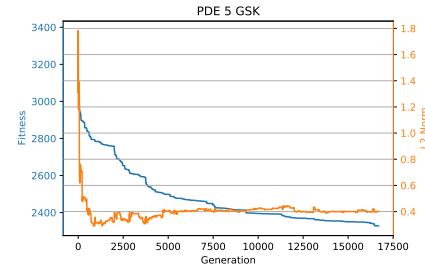


Figure 42: Fitness value and L2 Norm of an exemplary individual at every generation on PDE 5 using a GSK.

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.

10. Limitations

This work and its proposed solver suffer from several limiting factors. At first, the testbed only includes one type of equation. Further, the treatment of the boundary condition is critically examined. Finally, the achieved quality is strictly limited by the #FE-budget. Thus, the different algorithms are compared with their computational effort.

10.1. Testbed

Contrary to the testbed used by other authors, the test-equations used here are only second order PDEs in \mathbb{R}^2 . In particular, for all test-PDEs the Laplace operator Δ is applied to a solution function $u(\mathbf{x})$, resulting in various types of Poisson equations. Further, only Dirichlet type boundary conditions are used. This means that the testbed is not very diverse. Especially compared to Chaquet and Carmona 2019, the testbed falls short of ODE and systems of differential equations. However, the results presented in the experiment chapters already indicate mixed performances on different testbed PDEs. Thus, starting with a manageable variety of problems helps with assessing the

performance on a particular subset of differential equations.

10.2. Fulfilment of Boundary Condition

Contrary to the FEM solver, the described CI solver does not guarantee the boundary-condition fulfilment. There are ways to counteract the deviance on the boundary.

A simple possibility is to increase the penalty factor φ on the boundary collocation points n_B . This sets an emphasis on the boundary, however it still does not assure the fulfilment of the boundary condition.

As described in the literature overview, Kirstukas, Bryden, and Ashlock 2005 propose a strategy that repairs the boundary condition after the solver supposed an approximation to the interior domain. Introducing this strategy benefit the performance of the present solver.

As discussed in chapter 4.1, PDE 0A is used in the testbed by Chaquet and Carmona 2019 and Mitchell 2018. Since preliminary experiments have shown that the CI solver produced worse results on the Mitchell 2018 implementation, this version is adopted for the current testbed. The main difference between these variants of PDE 4 is the definition of the boundary condition:

- **Mitchell 2018:**

Homogeneous Dirichlet Boundary Condition:

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

- **Chaquet and Carmona 2019:**

Mixed Boundary Condition:

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

Experiment 1 investigates the error behaviour of PDE 0A. As seen in figure 20, the error is large on the boundary of the domain $\partial\Omega$, while the interior suffers of a relatively small error. Thus, improvements on the boundary can be achieved more easily. Therefore, it makes sense that the solver is sensitive to the implementation of the boundary condition. However, in the typical application the boundary condition can not be rewritten. Thus, better methods for ensuring the boundary condition are needed.

10.3. Computational Effort

The greatest limiting factor for the solver is the extensive computational effort. This is best measured by the Empirical Runtime Distribution (ERD). The ERD calculates the performance of heuristic optimisation algorithms and makes them comparable. ERD plots are often used in Black Box Optimisation Benchmarking (BBOB) contests, such as the annual Comparing Continuous Optimisers (COCO) workshop (Hansen, Brockhoff, et al. 2019). A similar plot is shown in figure 43 below.

The ERD plot shows the correlation between the #FE and the percentage of the solved functions in the testbed. Therefore, the #FE is increased from 10^3 to 10^6 . The testbed consists of 11 PDEs. Further, different target values in the L2 norm are inspected: $5 \cdot 10^{-2}$, $1 \cdot 10^{-2}$, $5 \cdot 10^{-3}$ and $1 \cdot 10^{-3}$. Thus, the vertical axis indicates how often a algorithm reaches these $4 \cdot 11 = 44$ target values.

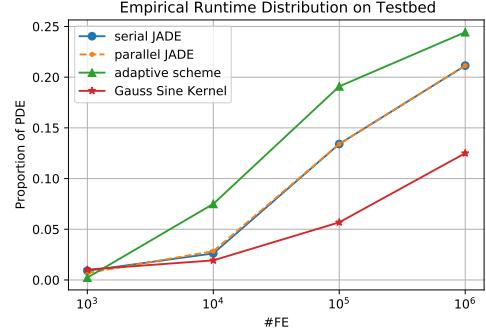


Figure 43: Empirical Runtime Distribution of all algorithms on the 11 testbed PDEs at target values $5 \cdot 10^{-2}$, $1 \cdot 10^{-2}$, $5 \cdot 10^{-3}$ and $1 \cdot 10^{-3}$.

Remarkable is the virtually non-existing performance difference between the serial and the parallel memetic JADE. Since “performance” in this graph does not mean “time-consumption”, this is expected.

The adaptive JADE scores continuously best and solves up to a quarter of the target values. This is largely thanks to the good performance on PDE 0A, which contributes 9 percent ($\frac{1}{11}$) to the whole testbed.

Extrapolating the ERD plots indicates a further performance increase with more #FE. Due to the already expensive algorithm, this is not tested. Substituting JADE for a faster converging algorithm might provide more clues to that matter.

11. Universal Approximation Theorem for GSK

The Gauss kernel is able to approximate all functions that are part of the Lebesgue space $f(\mathbf{x}) \in L^1(\mathbb{R}^n)$ arbitrarily close. This has been proven by various works (Park and Sandberg 1991, Hangelbroek and Ron 2010). In particular, Park and Sandberg 1991 extends the universal approximation theorem to other kernels. The following paragraphs show that the GSK fulfils the posed conditions and thus can benefit from the approximation theorem.

The Gauss-Sine kernel, as defined in equation (40), is a multiplication of the Gauss kernel with a sine function. Its main purpose is described in the experiment chapter 9.

$$gsk(\mathbf{x}) = \omega e^{-\gamma||\mathbf{x}-\mathbf{c}||^2} \sin(f||\mathbf{x}-\mathbf{c}||^2 - \varphi) \quad (40)$$

To prove that the universal approximation theorem is also applicable to the GSK, it must comply with the conditions placed by Park and Sandberg 1991. At first, a kernel, in this case the $gsk(\mathbf{x})$, must be continuous and bounded. This already restricts $\gamma > 0$. Secondly, the integral over the whole domain of the kernel $K(\mathbf{x})$ must not be 0. Thus, it needs to be shown that

$$\lim_{t \rightarrow \infty} \int_{\mathbf{x}=0}^t gsk(\mathbf{x}) d\mathbf{x} \neq 0 \quad (41)$$

To simplify the following calculations, the GSK is rewritten into polar coordinates. Further, the offsets by c_0 and c_1 are accounted for by an appropriate substitution. This results in

$$\lim_{t \rightarrow \infty} \int_{r=0}^t \int_{\theta=0}^{2\pi} e^{-\gamma(r^2)} \sin(fr^2 - \varphi) r dr d\theta \quad (42)$$

Since the kernel is radial symmetric and thus has no dependency on θ , the respective integral can be solved

immediately which results in a multiplicative factor of 2π . Substituting $r^2 = u$ gets rid of the multiplicative r.

$$= \pi \lim_{t \rightarrow \infty} \int_{u=0}^t e^{-\gamma u} \sin(fu - \varphi) du. \quad (43)$$

This integral can be solved with “integration by part” $\int f(u) \frac{g(u)}{du} = f(u)g(u) - \int g(u) \frac{f(u)}{du}$, which has to be applied twice until the original integral is obtained.

$$= \lim_{t \rightarrow \infty} \frac{\pi e^{-\gamma r^2} (-\gamma \sin(fr^2 - \varphi) - f \cos(fr^2 - \varphi))}{\gamma^2 + f^2} + C \Big|_{r=0}. \quad (44)$$

To resolve the limit of the function towards ∞ , the function value must be bounded. Therefore, γ must be positive, which is already required. Finally, this results in

$$\frac{-\pi(\gamma \sin(\varphi) + f \cos(\varphi))}{\gamma^2 + f^2} \neq 0. \quad (45)$$

This places more constraints on the parameter of the GSK. These restrictions on the parameters could be enforced during the optimisation process. However, Chaquet and Carmona 2019 found that not placing any limits to the parameters results in a better performance. Thus, these constraints are also not applied during the experiments in the chapters before.

12. Multimodality and Symmetry

The optimisation algorithm tries to find the best approximation of N kernels to the solution of a differential equation. Assume, that a kernel K is fully defined by a vector of parameters \mathbf{p} . The best fit is defined as

$$\hat{\mathbf{p}}_{apx} = \left[\underbrace{[\hat{\mathbf{p}}_{K_0}]}_{\text{kernel 0}}, \dots, \underbrace{[\hat{\mathbf{p}}_{K_i}]}_{\text{kernel i}}, \dots, \underbrace{[\hat{\mathbf{p}}_{K_N}]}_{\text{kernel N}} \right]^T \quad (46)$$

where the parameters $\hat{\mathbf{p}}$ of every kernel are chosen optimally. The optimal kernel functions $K(\hat{\mathbf{p}}_{K_i}, \mathbf{x})$ are summed up to form the optimal approximation $\hat{u}_{apx}(\mathbf{x})$.

$$\hat{u}_{apx}(\mathbf{x}) = \sum_{i=0}^N K(\hat{\mathbf{p}}_{K_i}, \mathbf{x}) \quad (47)$$

Since the order of the summation is irrelevant, any kernel-wise permutation describes an optimal solution $\hat{\mathbf{p}}_{apx}$. Thus, the fitness function $F(u_{apx}(\mathbf{x}))$ has at least $N!$ number of local optima.

Further, a symmetry in the location of the optima is observed. All optima lay on the surface of the hypersphere that is centred at the origin and has a radius of $r = \|\hat{\mathbf{p}}_{apx}\|$.

The following 3D plot in figure 44 shows an exemplary distribution of optima on the fitness function. For the sake of simplicity, a kernel now consists of only one parameter. As an example, the vector $\hat{\mathbf{p}}_{apx} = [2, 1, -1]^T$ describes an optimal solution that consists of 3 kernels. Any permutation of these three coordinates is itself a perfect fit.

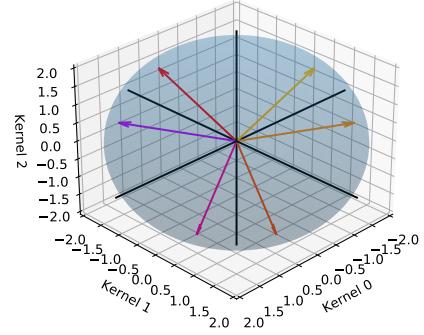


Figure 44: Distribution of exemplary optima on the fitness function in 3D space.

This symmetry is independent of the kernel type. Large parts of the fitness function, such as the weighting and penalty factors or the number of collocation points, have no influence on the actual radial arrangement of the optima.

13. Conclusion

Using JADE could not replicate the objectively good performance observed with CMA-ES in Chaquet and Carmona 2019 and other related work. The reason for that is not completely apparent. The poor convergence might be due to some bad parameter choices. Otherwise, JADE might not be as well suited for that sort of fitness function as a CMA-ES is.

The purpose of the adaptive kernels scheme is to reduce the search dimension and increase the number of kernels only when necessary. It works very well on the testbed PDE 0A. This PDE fulfils the assumptions that have been made for this algorithm. Mixed results have been observed on all other testbed problems. The experiments suggest, that the problems could be overcome with an algorithm that converges faster. Further experiments could couple the CMA-ES with the adaptive scheme.

The present fitness function might not be the perfect formulation of the problem. Ideally, the quality of the approximation should be tightly linked to the fitness value. Specifically, on PDE 5 more #FE result in a significantly worse approximation quality. Thus, a new kernel called GSK is created. The goal for this kernel is to alter the fitness function and thus enhance the correlation between fitness function and quality. The idea works as intended and the results on that PDE do get significantly better. Although the GSK exhibits all the features of the Gauss kernel, the quality on all other results are mixed. This might be due to the larger search space, that comes with the new kernel. More experiments, investigating the connection of the adaptive scheme with the GSK could provide more insight to that subject.

Obviously, this CI approach to solve PDEs is not as effective as the FEM method. FEM is specifically designed to work on elliptic PDEs and convergence is proven. However, the CI method is a universal approach, that should be able to solve all sorts of differential equations. Contrary, the “universal solver” attribute might also be the greatest disadvantage of this strategy. Currently, there is no guarantee for convergence. The simple implementation and the relatively straight forward approach could misguide the user into applying this strategy, although there might exist better solvers for the problem at hand.

The described symmetry should be further investi-

gated. Designing a variational operator to that circular feature could significantly improve the searching process. Also the search space could be restricted so that not the whole \mathbb{R}^N is taken into account. It might even be a starting point for further theoretical advances.

References

- 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).
- Broomhead, David S. and Lowe, David (1988): “Multivariable Functional Interpolation and Adaptive Networks”. In: *Complex Systems*.
- 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).
- 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).
- 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).
- 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).
- 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).
- Guyan, R. J. (1965): “Reduction of stiffness and mass matrices”. In: DOI: [10.2514/3.2874](https://doi.org/10.2514/3.2874).
- Hangelbroek, Thomas and Ron, Amos (July 2010): “Non-linear approximation using Gaussian kernels”. en. In: *Journal of Functional Analysis* 259.1, pp. 203–219. ISSN: 0022-1236. DOI: [10.1016/j.jfa.2010.02.001](https://doi.org/10.1016/j.jfa.2010.02.001). URL: <http://www.sciencedirect.com/science/article/pii/S0022123610000467> (visited on 02/28/2020).
- Hansen, Nikolaus; Brockhoff, Dimo, et al. (Mar. 2019): *CComparing Continuous Optimizers: numbo/COCO on Github*. Version v2.3. DOI: [10.5281/zenodo.2594848](https://doi.org/10.5281/zenodo.2594848). URL: <https://doi.org/10.5281/zenodo.2594848>.
- Hansen, Nikolaus; Müller, Sibylle D., and Koumoutsakos, Petros (Mar. 2003): “Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)”. In: *Evolutionary Computation* 11.1. Publisher: MIT Press, pp. 1–18. ISSN: 1063-6560. DOI: [10.1162/106365603321828970](https://doi.org/10.1162/106365603321828970). URL: <https://doi.org/10.1162/106365603321828970> (visited on 08/15/2020).
- 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).
- Horn, Berthold K. P. (Nov. 1970): “Shape from Shading A Method for Obtaining the Shape of a Smooth Opaque Object from One View”. en. In: URL: <https://dspace.mit.edu/handle/1721.1/6885> (visited on 08/13/2020).
- 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).
- 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).
- 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).
- 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).
- 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).
- 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.
- Mallipedi 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).

- Mastorakis, Nikos E (Aug. 2006): "Unstable Ordinary Differential Equations: Solution via Genetic Algorithms and the method of Nelder-Mead". en. In: Elounda, Greece, p. 7. URL: https://www.researchgate.net/profile/Nikos_Mastorakis2/publication/261859052_Unstable_ordinary_differential_equations_Solution_via_genetic_algorithms_and_the_method_of_Nelder-Mead/links/573b254e08ae9f741b2d7853.pdf (visited on 02/19/2020).
- Mitchell, William F. (Mar. 2018): *NIST AMR Benchmarks*. en. URL: <https://math.nist.gov/amr-benchmark/index.html> (visited on 02/21/2020).
- Moscato, Pablo (Oct. 2000): "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms". In: *Caltech Concurrent Computation Program*.
- 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).
- Opala, 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](https://www.sciencedirect.com/science/article/pii/S2210650217304224) (visited on 03/04/2020).
- 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).
- Park, J. and Sandberg, I. W. (June 1991): "Universal Approximation Using Radial-Basis-Function Networks". In: *Neural Computation* 3.2. Publisher: MIT Press, pp. 246–257. ISSN: 0899-7667. DOI: [10.1162/neco.1991.3.2.246](https://doi.org/10.1162/neco.1991.3.2.246). URL: <https://doi.org/10.1162/neco.1991.3.2.246> (visited on 08/11/2020).
- 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).
- 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).
- Rajeev and Krishnamoorthy (May 1992): "Discrete Optimization of Structures Using Genetic Algorithms". In: *Journal of Structural Engineering* 118.5. Publisher: American Society of Civil Engineers, pp. 1233–1250. DOI: [10.1061/\(ASCE\)0733-9445\(1992\)118:5\(1233\)](https://doi.org/10.1061/(ASCE)0733-9445(1992)118:5(1233)). URL: [https://ascelibrary.org/doi/abs/10.1061/\(ASCE\)0733-9445\(1992\)118:5\(1233\)](https://ascelibrary.org/doi/abs/10.1061/(ASCE)0733-9445(1992)118:5(1233)) (visited on 03/23/2020).
- Rechenberg, I. (1978): "Evolutionsstrategien". de. In: Schneider, Berthold and Ranft, Ulrich (Eds.): *Simulationsmethoden in der Medizin und Biologie*. Medizinische Informatik und Statistik. Berlin, Heidelberg: Springer, pp. 83–114. ISBN: 978-3-642-81283-5. DOI: [10.1007/978-3-642-81283-5_8](https://doi.org/10.1007/978-3-642-81283-5_8).
- Rodola, Giampaolo (May 2020): *psutil documentation — psutil 5.7.1 documentation*. en. Documentation. URL: <https://github.com/giampaolo/psutil/blob/master/docs/index.rst> (visited on 05/21/2020).
- 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).
- 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) (visited on 03/02/2020).
- 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).
- 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 (visited on 05/29/2020).
- SciPy, Reference Guide (Aug. 2020): *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).
- 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).
- 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).
- 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).

- 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).
- 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).
- 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).
- 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).
- 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).