COMPUTATIONAL INTELLIGENCE METHODS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Nicolai Schwartze

Abstract This master thesis investigates a Computational Intelligence-based method for solving PDEs. The proposed strategy formulates the residual of a PDE as a fitness function. The solution is approximated by a finite sum of Gauss kernels. An appropriate optimisation technique, in this case JADE, is deployed that searches for the best fitting parameters for these kernels. This field is fairly young, a comprehensive literature research reveals several past papers that investigate similar techniques. To evaluate the performance of the solver, a comprehensive testbed is defined. It consists of 11 different Poisson equations. The solving time, the memory consumption and the approximation quality are compared to the state of the art open-source Finite Element solver NGSolve. The first experiment tests a serial JADE. The results are not as good as comparable work in the literature. Further, a strange behaviour is observed, where the fitness and the quality do not match. The second experiment implements a parallel JADE, which allows to make use of parallel hardware. This significantly speeds up the solving time. The third experiment implements a parallel JADE with adaptive kernels. It starts with one kernel and introduce more kernels along the solving process. A significant improvement is observed on one PDE, that is purposely built to be solvable. On all other testbed PDEs the quality-difference is not conclusive. The last experiment investigates the discrepancy between the fitness and the quality. Therefore, a new kernel is defined. This kernel inherits all features of the Gauss kernel and extends it with a sine function. As a result, the observed inconsistency between fitness and quality is mitigated. The thesis closes with a proposal for further investigations. The concepts here should be reconsidered by using better performing optimisation algorithms from the literature, like CMA-ES. Beyond that, an adaptive scheme for the collocation points could be tested. Finally, the fitness function should be further examined.

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

Nicolai Schwartze, BSc.
Vorarlberg University of Applied Science
Master Mechatronics 2020
nicolai.schwartze@students.fhv.at

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 currently analytically solvable. The rise of the computer paved the way for approximating the solution of a differential equation numerically. There are plenty of different solver methods ranging from simple single step solvers like the Forward-Euler (FE) method over more complex multistep solvers like the Adams–Bashforth (AB) method to whole 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 section 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 Equations (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 packages 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

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 Partial Differential Equation (PDE) is posed, is discretised into multiple smaller elements - as the name suggests. Thus, Finite Element Method (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 [20].
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.

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

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

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

2. Step: Weak Form
The next step is to reformulate the strong form into a usable weak form. This is equivalent to the strong form but written in an integral notation. In this equation, the A, b and c correspond to the constant factors of the derivatives in strong form. For the sake of completeness, this is kept abstract. In the PDEs considered in this work, $A = \mathbb{I}$ and $b = \mathbf{0}$, $c = 0$. Currently, the so-called test-function $v(\mathbf{x})$ is an arbitrary function, but it has to be 0 on the boundary $v(\mathbf{x})|_\Omega = 0$. The choice of the test-function correspond to different FEM types ([21, p. 6f]).

$$\begin{aligned} a(u,v) \begin{cases} \\ \\ \\ \end{cases} & \int_\Omega -(\nabla^T A \nabla) u(\mathbf{x}) v(\mathbf{x}) dV \\ & - \int_\Omega b^T \nabla u(\mathbf{x}) v(\mathbf{x}) dV \\ & + \int_\Omega c u(\mathbf{x}) v(\mathbf{x}) dV \\ F(v) \begin{cases} \end{cases} & = \int_\Omega f(\mathbf{x}) v(\mathbf{x}) dV \end{aligned} \tag{3}$$

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 $u(\mathbf{x}) \approx u_h(\mathbf{x}) = \sum_{i=1}^N u_i \Phi_i(\mathbf{x})$. A common choice are Lagrange or Chebyshev polynomials. In the Galerkin type FEM, the test-function $v(\mathbf{x})$ is the same as the trail-function, thus $v(\mathbf{x}) = \sum_{j=1}^N v_j \Phi_j(\mathbf{x})$. The choice of the basis function $\Phi(\mathbf{x})$ largely influences the computational effort. $\Phi(\mathbf{x})$ should have a small support, to produce a thinly populated matrix A in the linear system of equations (5) below.
5. Step: Solution
In the weak form, as seen in equation (3), $a(u,v)$ is a continuous bilinear form and $F(v)$ is a continuous linear functional. Substituting u and v with their corresponding approximation from step 4 results in

$$\sum_{j=1}^N v_j \sum_{i=1}^N u_i a(\Phi_i, \Phi_j) = \sum_{j=1}^N v_j F(\Phi_j). \tag{4}$$

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

$$\underbrace{\sum_{i=1}^N u_i a(\Phi_i, \Phi_j)}_{\mathbf{Au}} = \underbrace{F(\Phi_j)}_{\mathbf{b}} \text{ for } j = 1, ...N \tag{5}$$

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

- Static Condensation:
Depending on the number of discrete elements, the $\mathbf{A}$ matrix can be very large. Inverting large matrices is very time consuming. Static condensation, also called Guyan reduction ([6]), 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 is 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)
  - Repeat until Degree of Freedom (DOF) exceed a specified N

2.2. Computational Intelligence Methods

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 also called objective function and these terms are used interchangeably in this paper. 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.
This paper mainly builds on the work presented in [4]. In their paper they describe an algorithm that approximates a solution with a linear combination of Gaussian Radial Basis Function (RBF) as kernels:

$$u(\mathbf{x})_{apx} = \sum_{i=1}^N \omega_i e^{\gamma_i (||\mathbf{x} - \mathbf{c}_i||^2)} \tag{6}$$

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 (7).

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

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 parameters of the kernels are determined via a Covariance Matrix Adaption Evolution Strategy (CMA-ES) [8]. To further improve the solution, the evolutionary algorithm is coupled with a Downhill-Simplex (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.

Paper	Algorithm	Representation	Problems
[10]	GP	polynomial of arbitrary length	one-dimensional steady-state model of convection-diffusion equation
[11]	GP	algebraic expression	heating of thin rod heating by current
[25]	GE	algebraic term	set of ODEs system of ODEs and PDEs
[14]	GA (global); DS (local)	5th order polynomial	unstable ODEs
[22]	GP and RBF-NN	algebraic term for inner; RBF for boundary	elliptic PDEs
[9]	GP	function value grid	convection–diffusion equation at different Peclet numbers
[3]	ES	partial sum of Fourier series	testbench of ODEs system of ODEs and PDEs
[1]	PSO	partial sum of Fourier series	integro-differential equation system of linear ODEs Brachistochrone nonlinear Bernoulli
[17]	DE	polynomial of unspecified order	set of 6 different PDEs
[19]	PSO HS WCA	partial sum of Fourier series	singular BVP
[4]	CMA-ES (global); DS (local)	linear combination of Gaussian kernels	testbench of ODEs system of ODEs and PDEs
[5]	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 date of release.

2.3. Differential Evolution

The differential evolution framework was first introduced in [23]. 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. Some of them currently count to the best performing algorithms, as the 100-Digit Challenge at GECCO 2019 [24] 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 according to the fitness function $f(\mathbf{x}_i) : \mathbb{R}^n \to \mathbb{R}$. The goal is to minimise the fitness function. These steps are performed in a loop until a predefined termination condition is reached. Each step 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. The */current-to-pbest/1* mutation operator can be seen in equation (8) where x_i is the current individual, x^p_{best} 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.

$$v_i = x_i + F_i(x^p_{best} - x_i) + F_i(x_{r1} - \tilde{x}_{r2}) \tag{8}$$

- 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 (9) randomly takes elements from both vectors, where K is a random index to ensure that at least one element from the trial vector v_i is taken.

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } j = K \vee rand[0,1] \leq CR \\ x_{ij}, & \text{otherwise} \end{cases} \tag{9}$$

- Selection:
  Population size N;
  The selection replaces the old candidate x_i if the trial candidate u_i is better as measured by 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 is described in [26]. The adaption is performed by taking successful F and CR values of the last generation into account. If a certain setting is successful in generating better candidates, newly selected F and CR gravitate towards that setting.

3. Problem Definition

This section 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 [21]. 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

$$\begin{aligned} \mathbf{L}u(\mathbf{x}) &= f(\mathbf{x}) \\ \text{subjected to: } \mathbf{B}u(\mathbf{x}) &= g(\mathbf{x}). \end{aligned} \tag{10}$$

This is not limited to a Dirichlet boundary problem. The matrix $\mathbf{B}$ can include differentiation, effectively allowing Neumann boundary conditions. An approximate solution to this problem is expressed as a finite sum of basis functions $\phi_k(\mathbf{x})$, as denoted in equation (11). The coefficients a_k need to be determined.

$$u(\mathbf{x}) \approx u_{apx}(\mathbf{x}) = \sum_{k=0}^{N} a_k \phi_k(\mathbf{x}) \tag{11}$$

The residual of a PDE, as shown in equation (12), 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 an analytical solution, further denoted as $u_{ext}(x, y)$. For a numerical approximation $u_{apx}(x, y)$, the residual should be small enough.

$$R(\mathbf{x}) = \mathbf{L}u_{apx}(\mathbf{x}) - f(\mathbf{x}) \tag{12}$$

The WRM ([21]) 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 (13). 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. 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.

$$WRF = \int_{\Omega} R(\mathbf{x})\psi(\mathbf{x})W(\mathbf{x})dx \tag{13}$$

This integral must be evaluated numerically. There are many different integration schemes available, but typically they are computational expensive. A less extensive approach is to evaluate the integral argument at different "collocation points" n_C. This can be seen in equation (14). Although it is not an integral per se, it does assign a numerical "score" to the residual.

$$\sum_{k=0}^{N} R(\mathbf{x}_k)\psi(\mathbf{x}_k)W(\mathbf{x}_k) \tag{14}$$

A common choice in heuristic methods from table 6 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 (15).

$$\sum_{k=0}^{N} R(\mathbf{x}_k)^2 W(\mathbf{x}_k) \tag{15}$$

These methods are often called "spectral methods" [21]. 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 higher accuracy.

3.2. Fitness Function

As mentioned above, the basis of the fitness function used in [4] is the squared weighted residual in equation (15). 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). This fitness function, as seen in equation (7), 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 (16) below. The norm of the residual is not necessary for this paper, since the fitness function does not have to work for system of PDEs.

$$\begin{aligned} F(u_{apx}(\mathbf{x})) = & \frac{1}{(n_C + n_B)} \\ & \left[\sum_{i=1}^{n_C} \xi(\mathbf{x}_i) \left[\mathbf{L}u_{apx}(\mathbf{x}_i) - f(\mathbf{x}_i) \right]^2 \right. \\ & \left. + \phi \sum_{j=1}^{n_B} \left[\mathbf{B}u_{apx}(\mathbf{x}_j) - g(\mathbf{x}_j) \right]^2 \right] \end{aligned} \tag{16}$$

A notable property of the fitness function is that it can not become negative. 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 $u_{ext}(\mathbf{x})$. It is also important to notice that aliasing errors 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 and the fitness function are directly extracted from [4].

$$\xi(\mathbf{x}_i) = \frac{1 + \kappa \left(1 - \frac{min_{\forall \mathbf{x}_j \in n_B} ||\mathbf{x}_i - \mathbf{x}_j||}{max_{\forall \mathbf{x}_k \in n_C} (min_{\forall \mathbf{x}_j \in n_B} ||\mathbf{x}_k - \mathbf{x}_j||)} \right)}{1 + \kappa} \tag{17}$$

3.3. Candidate Representation

Gauss kernels have frequently been used for the approximation of functions. A solid foundation on the approximation theorem can be seen in [18]. RBF are functions that are point symmetric to a centre c and their function value only depends on the radius r.

$$r = ||\mathbf{x} - \mathbf{c}|| \tag{18}$$

This paper mainly works with two different types of RBF. The first one is the classical Gauss RBF, also denoted as Gauss Kernel (GaK). The second one, further called Gauss Sine RBF, or Gauss Sine Kernel (GSK), is more complex. A candidate solution is represented as the parameters that shift and deform these RBF.

3.3.1. Gauss Kernel

The Gauss Kernel was first introduced in [2]. Equation (19) 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 kernel 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.

$$gak(\mathbf{x}) = \omega e^{-\gamma ||\mathbf{x} - \mathbf{c}||^2} \tag{19}$$

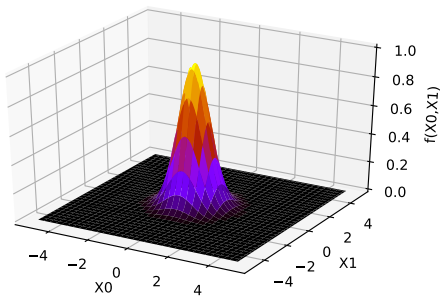


Figure 1: 3D plot of a single Gauss kernel with the parameters $\omega = 1$, $\gamma = 1$, $c_0 = 0$ and $c_1 = 0$

An approximate solution to the PDE in question is described as the superposition of N kernels as shown in equation (20).

$$u_{apx}(\mathbf{x}) = \sum_{i=0}^{N} \omega_i e^{-\gamma_i r_i^2} \tag{20}$$

A solution is encoded as a vector of these $4 \cdot N$ parameters stacked together. This is shown in equation (21).

$$\mathbf{p_{apx}} = \left[\cdots \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}}_{\text{kernel i}} \cdots \right]^T \tag{21}$$

To evaluate the residual of a PDE, the derivatives must be known. To that extend, the derivative of u_{apx} from equation (11) is calculated. The first order derivative with respect to x_0 is seen in equation (22). To solve the proposed testbed, also the second order derivatives are needed, as described in equation (23). Although the mixed term derivative is not used in this work, for the sake of completeness it is displayed in equation (24).

$$\frac{\partial u_{apx}(\mathbf{x})}{\partial x_j} = -2 \sum_{i=0}^{N} \omega_i \gamma_i (x_j - c_{ij}) e^{-\gamma_i r_i^2} \tag{22}$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j^2} = \sum_{i=0}^{N} \omega_i \gamma_i \left[4\gamma_i (x_j - c_{ij})^2 - 2 \right] e^{-\gamma_i r_i^2} \tag{23}$$

$$\frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j x_k} = 4 \sum_{i=0}^{N} \omega_i \gamma_i^2 (x_j - c_{ij})(x_k - c_{ik}) e^{-\gamma_i r_i^2} \tag{24}$$

3.3.2. Gauss Sine Kernel

Additional to the Gauss kernel, a "Gauss Sine" kernel, further also abbreviated as GSK, is proposed. In essence, this is a multiplication of a GaK with a sine function, as seen in equation (25). It can be thought of as a sine function, where its influence declines with the radius r. The plot in figure 2 shows how the first half period has a larger amplitude than the second half. As shown in section 10, the universal approximation theorem for the GSK holds true. With the sine, two new parameters are introduced: f for the frequency of the sine wave and φ for the phase shift.

$$gsk(\mathbf{x}) = \omega e^{-\gamma ||\mathbf{x}-\mathbf{c}||^2} sin(f||\mathbf{x} - \mathbf{c}||^2 - \varphi) \tag{25}$$

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, this kernel should be able to approximate some functions more efficiently than the GaK.

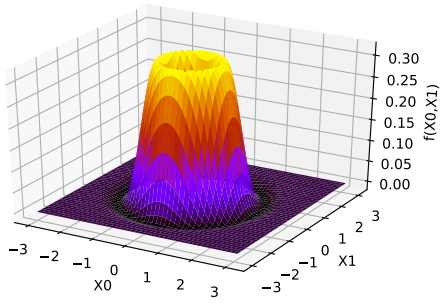


Figure 2: 3D plot of a single Gauss Sine kernel with the parameters $\omega = 1$, $\gamma = 1$, $c_0 = 0$, $c_1 = 0$, $f = 1$, $\varphi = 0$

Similar to the Gauss kernel, the approximate solution is described as the superposition of N GSK, as seen in equation (26).

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

Equation (27) shows the solution representation in stacked vector notation, including the two new parameters f and φ.

$$\mathbf{p_{apx}} = \left[\cdots \underbrace{\omega_i, \gamma_i, c_{i0}, c_{i1}, f_i, \varphi_i}_{\text{kernel i}} \cdots \right]^T \tag{27}$$

Again, the derivatives of the solution are needed, which are shown in the equations (28) through (30).

$$\begin{aligned} \frac{\partial u_{apx}(\mathbf{x})}{\partial x_j} = \sum_{i=0}^{N} & 2\omega_i (x_j - c_{ij}) e^{-\gamma_i r_i^2} \\ & (f_i cos(f_i r_i^2 - \varphi_i) - \gamma_i sin(f_i r_i^2 - \varphi_i)) \end{aligned} \tag{28}$$

$$\begin{aligned} \frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j^2} = \sum_{i=0}^{N} & 2\omega_i e^{-\gamma_i r_i^2} \\ & [-(2(f_i^2 - \gamma_i^2)(x_j - c_{ij})^2 + \gamma_i) sin(f_i r_i^2 - \varphi_i) \\ & -(4f_i \gamma_i (x_j - c_{ij})^2 - f_i) cos(f_i r_i^2 - \varphi_i)] \end{aligned} \tag{29}$$

$$\begin{aligned} \frac{\partial^2 u_{apx}(\mathbf{x})}{\partial x_j x_k} = \sum_{i=0}^{N} & -4\omega_i (c_{ij} - x_j)(c_{ik} - x_k) e^{-\gamma_i r_i^2} \\ & \left[(f_i^2 - \gamma_i^2) sin(f_i r_i^2 - \varphi_i) + 2 f_i \gamma_i cos(f_i r_i^2 - \varphi_i) \right] \end{aligned} \tag{30}$$

4. Experimental Design

This section 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 [20].

4.1. Testbed

The testbed is a collection of multiple two-dimensional scalar PDEs that are analytically solved such that $u_{ext}(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$ is a solution to the underlying PDE. The equations used here are a mixture of different testbeds. Specifically, the equations PDE 2 and PDE 3 are picked from the testbed in [4], [25] and [17]. The more complex equations are taken from the National Institute of Standards and Technology (NIST) website [15] 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 Abbreviation	Description
PDE 0A: Gauss Kernel	superposition of 5 GaK
PDE 0B: Gauss Sine Kernel	superposition of 3 GSK
PDE 1: Polynomial 2D	polynomial of order 20
PDE 2: Chaquet PDE 1	slightly curved e-function
PDE 3: Chaquet PDE 3	polynomial of order 2
PDE 4: Sine Bump 2D	homogeneous sine on unit square
PDE 5: Arctan Circular Wave Front	arctan with steep gradient
PDE 6: Peak 2D	sharp Gauss function with steep gradient
PDE 7: Boundary Line Singularity	root function defined on $x \in \mathbb{R}^+$
PDE 8: Interior Point Singularity	root function with singularity at $[0.5, 0.5]$
PDE 9: Arctan Wave Front Homogeneous Boundary Conditions 2D	arctan oscillation on diagonal of unit square

Table 2: Table of the testbed PDEs used in this paper.

4.2. Software Architecture

To simplify the preparation, execution and evaluation of the experiments, a comprehensive software architecture is defined. A simplified class diagram, without any methods and attributes is shown in figure 3 below. The architecture is organised in four main segments: Optimisation Algorithm (OptAlgo), Kernels, Testbed and Post Processing (post_proc)

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. Although the fitness function is the criterion that is optimised, it is not applicable as an objective quality measure. As [4] 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

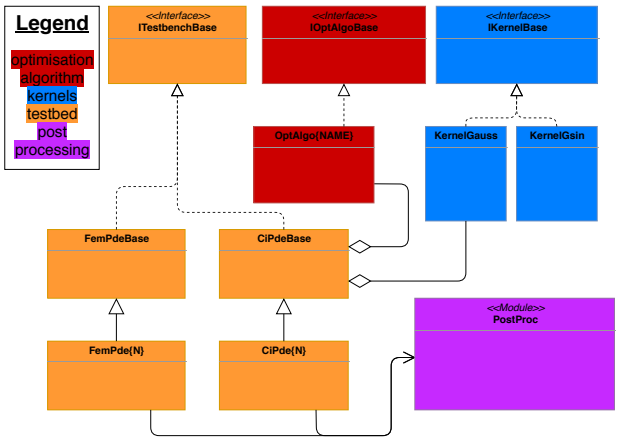


Figure 3: This reduced Unified Modeling Language (UML) class diagram gives an overview of the testbed design and its interfaces and classes.

Thus, [4] define a new quality measurement based on the Root-Mean-Square Error (RMSE) over the collocation points as seen in equation (31).

$$RMSE^2 = \frac{1}{(n_C + n_B)} \left[\sum_{i=1, \mathbf{x}_i \in C}^{n_C} ||u(\mathbf{x}_i) - u_{ext}(\mathbf{x}_i)||^2 + \sum_{j=1, \mathbf{x}_j \in B}^{n_B} ||u(\mathbf{x}_j) - u_{ext}(\mathbf{x}_j)||^2 \right] \tag{31}$$

This quality criterion has three inherent issues. At first, it firmly depends on the number of collocation points used. Further, only the quality on the collocation points is measured 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.

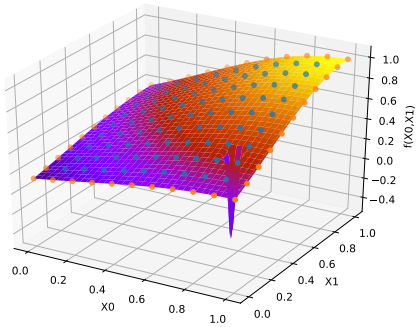


Figure 4: Aliasing Error: sharp inaccuracies in between collocation points

Finally, the FEM method doesn't use collocation points, so this quality measurement could not be calculated. This leads to the quality measurement formulation used in this thesis: the L2 norm defined for functions as denoted in equation (32). This measures the distance between the analytical solution and the numerical approximation.

$$||u_{ext} - u_{apx}|| = \sqrt{\int_\Omega (u_{ext}(\mathbf{x}) - u_{apx}(\mathbf{x}))^2 dx} \tag{32}$$

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 taking the areas between these points into account.
To validate the results, a statistical significance test is performed. The test function is based on the Wilcoxon test at a significance level of $\alpha = 0.05$. 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.

4.4. Baseline: NGSolve

NGSolve [20] is a state of the art FEM solver that is in part developed and maintained by numerous well-known institutes such as the Vienna University of Technology, the University of Göttingen and the Portland State University.

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 (3), the terms $\vec{b} = 0$ and $c = 0$, thus these parts vanish. The matrix A is either $\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$ or with a negative sign $\left[\begin{smallmatrix} -1 & 0 \\ 0 & -1 \end{smallmatrix}\right]$ as only the non-mixed second order derivatives occur in the equations. This results in the weak form as presented in equation (33), where f is the inhomogeneous part of the PDE.

$$\int_\Omega \nabla u^T \nabla v dV = \int_\Omega f v dV \tag{33}$$

To enhance the performance of NGSolve, static condensation is turned on. Further, a multigrid preconditioner is used. All problems are approximated by second order polynomials. The automatic mesh refinement is performed until a maximum of $5 \cdot 10^4$ DOF is reached. To properly interpret the results, 20 replications for each problem are performed. This is only needed for time and memory usage, the solution itself is not a random variable. To further reduce memory usage, 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. 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 3 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 evaluate to zero $\frac{\partial u}{\partial y} = 0$ and thus vanishes. This does not effect the memory usage, since all $5 \cdot 10^4$ DOF must be created to terminate.

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.

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 3: These are the results obtained by the FEM solver in terms of distance to the analytical solution.

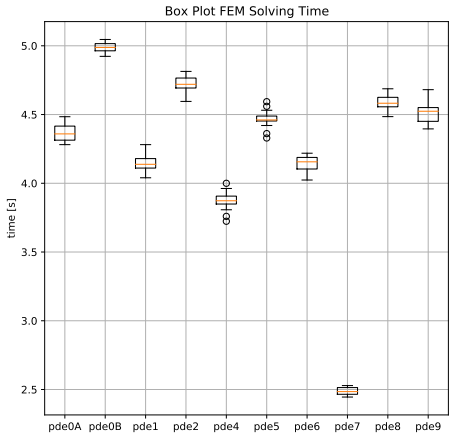


Figure 5: Boxplot: time (in seconds) needed to solve the testbed PDE (without PDE3)

One possible explanation for that is the fundamental structure of solution. 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 supports this hypothesis, that less DOF take less time and memory to solve the PDE and still end up with a better quality.

4.5. 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 is used. Similarly, the reported Computational Intelligence (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 tests. If not stated otherwise, the following parameters from table 4 are used in the subsequent experiments.

The FEM solver terminates after it surpasses $5 \cdot 10^4$ DOF. The whole number of collocation points ($n_B + n_C$) is the equivalent for the CI solver. Typically, fewer collocation points are used, since the evaluation time of the fitness function scales poorly with more points. [4] use 100 equally

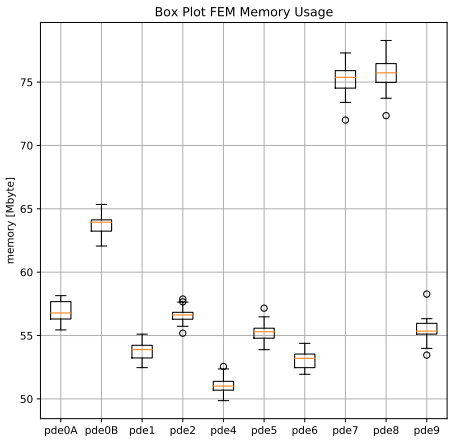


Figure 6: Boxplot: memory (in Mbyte) needed to solve the testbed PDE (without PDE3)

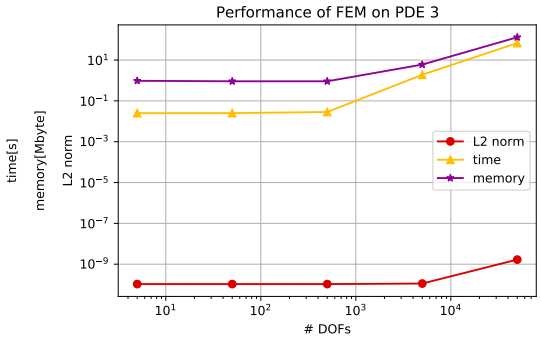


Figure 7: PDE 3 performance metrics at different DOF budgets.

spaced collocation points over the domain to solve the PDE. Here, 121 points are created. Figure 9 shows the values of ξ and φ. It further describes how the weighting factor emphasises the areas closer to the boundary. Typically, DE uses larger population sizes. [13] describe an empirical study on choosing this parameter. They discuss the trade-off between premature convergence and computational effort. The results suggest that population sizes of $2 \cdot dim$ are more likely to stagnate, but also converge faster. Since time is a critical resource for the CI-solver, this population size is chosen. To account for the statistical influence, every setup is reevaluated by 20 replications where each replication starts with independent initial guesses. The initialisation is done by a standard normal distribution. This means that every value in the $\mathbf{p_{apx}}$ vector is drawn from a normal distribution. On the contrary, [4] 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.

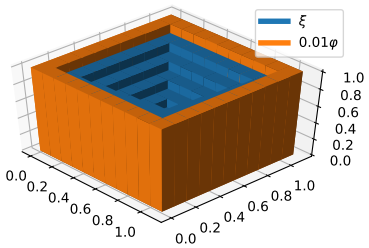
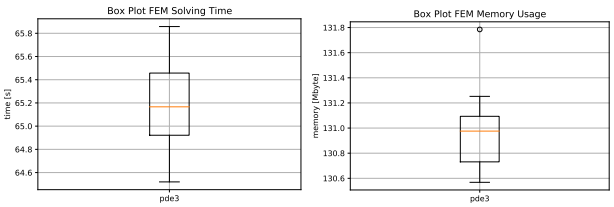


Figure 9: Weighting factor φ and ξ on every collocation point



(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.

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

Table 4: These predefined parameters are used for the following numerical experiments.

4.6. 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 any time or memory usage comparisons must be performed between experiments on the same machine. In the experiments it is separately denoted which comparisons are allowed. The following table 5 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. Due to the Python incompatibilities, NGSolve can only be installed on machine 1. Thus, any time or memory usage comparisons between the CI solver and the FEM solver must be conducted on machine 1.

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 5: Comparison of the machines used for the experiments.

5. Experiment 0: Serial Memetic JADE

This section describes the results obtained by the most basic adaption of JADE for solving PDEs. The algorithm used here builds on the concepts described in [4]. Aside from substituting some parameters, the only main difference is the usage of JADE instead of a CMA-ES.

5.1. Hypotheses

A memetic algorithm was first mentioned in [16]. In essence, it is a hybridisation of a population-based evolutionary algorithm and a deterministic direct 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 $2 \cdot dim$ #FE for the DS. This experiment provides a first insight into the

Algorithmus 1: Pseudocode of memetic JADE

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

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.

5.2. Experiment Setup

The standard parameters from table 4 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.

5.3. Results

Table 6 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. [25] also use these two PDEs, but their paper did not provide any usable error metric. The following table 7 lists the smallest L2 norm reached

Paper	Parameter	RMSE PDE 2	RMSE PDE 3
[4]	4 kernel max #FE=10^6 50 replications	$(1.75 \pm 1.14)10^{-4}$	$(1.09 \pm 0.846)10^{-5}$
[3]	10 harmonics max #FE = $G \cdot \lambda$ = $1.2 \cdot 10^6$ 10 replications	$(6.37 \pm 0.733)10^{-3}$	$(5.90 \pm 0.799)10^{-3}$
[22]	50 max tree length 12 generations 20 replications	$(6.9 \pm 8.3)10^{-4}$	N/A
[17]	unknowns: N/A #FE=$5 \cdot 10^5$ replications: N/A	7.25610^{-4}	9.48910^{-6}
serial memetic JADE	5 kernel max #FE = 10^6 20 replications	$(2.98 \pm 1.55)10^{-2}$	$(3.82 \pm 1.94)10^{-2}$

Table 6: This table compares the results obtained with the serial memetic JADE to the numerical results obtained by similar work in literature.

after 10^4 #FE and 10^6 #FE, respectively. Remarkable is that the results on PDE 5 get significantly worse when more #FE are used. The following two images 10 and 11 show the time and memory usage for solving the testbed with 10^4 #FE in relation to the FEM solver. The images

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

Table 7: L2 norm reached with serial JADE at 10^4 #FE and 10^6 #FE

highlight the varying complexity of the testbed PDEs. Although these results are not obtained for 10^6 #FE, they provide insight on how the solver scales with more #FE.

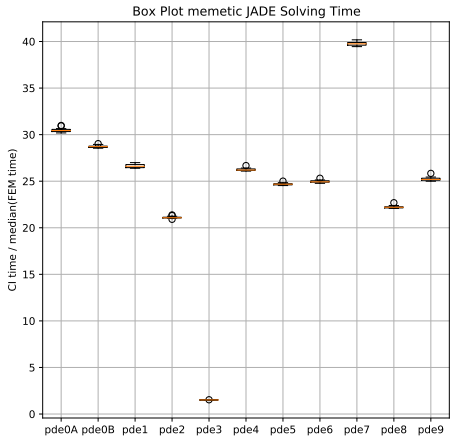


Figure 10: Relative solving time results of memetic JADE after 10^4 #FE.

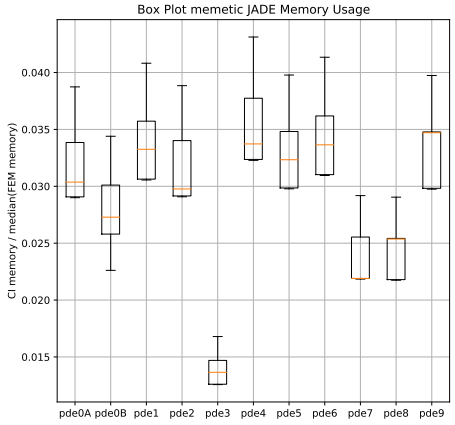


Figure 11: Relative memory usage results of memetic JADE after 10^4 #FE.

5.4. Discussion

5.4.1. Comparison to Literature

The RMSE reached on the PDE 2 and 3, are clearly not as good as the results obtained in previous papers, especially the results presented in [4]. There might be a few reasons for that:

- As stated in the previous table 4, 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 PDE.

- Since PDE 0A is defined as a combination of 5 GaK, at least that many kernels must be provided to the solver. This setup results in a greater search dimension, as compared in table 6. In general, more kernels result in a better solution, which is confirmed by experiments in [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 [13], the population size should not be lower than that. The combination of larger problem dimension and smaller population size could influence the convergence to the worse.
- Compared to [4], the #FE-budget for the local DS search is smaller. Thus, the algorithm puts more emphasis on the exploration. However, due to the internal parameter-adaption JADE can perform both - exploration and exploitation. Preliminary experiments have shown that the direct search converges fast with very little progress after more than 100 #FE.
- JADE might simply be not as well suited for the problem as e.g. a CMA-ES.

5.4.2. Solving Time/Memory Usage

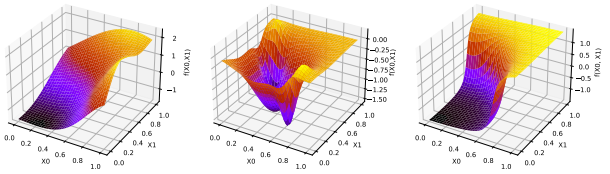
The results from table 7 show that the CI solver can not nearly compete with the results obtained by the FEM solver from table 3. However, more interesting is the comparison of time and memory usage. In the current implementation, the population and the corresponding function values as well as the F 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 uses less memory on all problems (figure 11).
The CI solver takes between 3 and 40 times as long as the FEM solver (figure 10) to perform 10^4 #FE. Since the solving time scales linearly, 10^6 #FE take about 10 times longer. 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.

5.4.3. PDE 0A

The purpose of this PDE is to show that the solver would converge globally towards the analytical solution, if it can be represented by a finite number of kernels. However, his can not be confirmed with the current implementation. While more function evaluations do tend to generate better results, 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. 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.

5.4.4. PDE 5

A fascinating property of PDE 5 is observed: more function evaluations result in a significantly worse solution quality. This property can even be concluded from a visual perspective. A comparison between the best solution after 10^4 #FE and the best solution after 10^6 #FE is shown in figure 12. The solution after 10^4 #FE describes the global structure more accurately. The distributions of the



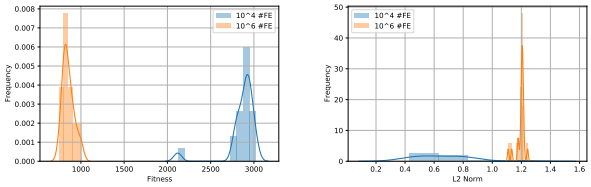
(a) best solution 10^4 #FE
L2 norm: 0.4283
FE value: 2980.96

(b) best solution 10^6 #FE
L2 norm: 1.1095
FE value: 1007.44

(c) analytical solution to testbed PDE 5.

Figure 12: Comparison of best solution after 10^4 and 10^6 #FE.

achieved fitness value after 10^4 and 10^6 #FE is shown in figure 13a. 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 13b. 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. This phenomenon can be explained by the



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

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

Figure 13: Histograms of the fitness and the L2 norm reached with 10^4 #FE and 10^6 #FE on PDE 5.

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. The monotonically decreasing fitness value at every generation of an exemplary individual within the population is plotted in figure 14. 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 L2 norm and the fitness at every generation of one individual from the population is plotted in figure 14.

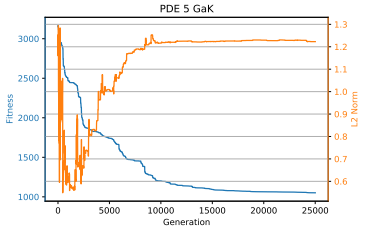


Figure 14: Fitness value and L2 Norm of one individual at every generation on PDE 5.

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.

6. Experiment 1: Parallel Population JADE

With the serial algorithm, 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.

6.1. Hypotheses

The parallel JADE is very similar to the serial JADE 1. The main differences are that 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 [12]. 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. This section discusses the question if the explained distinctions actually influence the results. Further, the time-benefits of this strategy are examined.

6.2. Experiment Setup

Similar to the experiment 0 before, the standard parameters from table 4 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 is used to proof statistical significance. Additionally, a new parameter is introduced: the number of allocated parallel processes. Typically, this parameter is set to the number of available processors, but this also depends on the general workload of the machine. On machine 1, 5 processes are implemented, while machine 2 is capable of handling 30 processes. With other machines, it might be possible to introduce even more processes, potentially increasing the speed-up even further.

6.3. Results

The two images 15 and 16 display a boxplot of the time- and memory usage. To ensure comparability with the results obtained for the serial JADE, these images represent the results at 10^4 #FE on machine 1. Table 8 shows the mean and the median L2 norm reached on the testbed by both the serial and the parallel algorithm to identify the

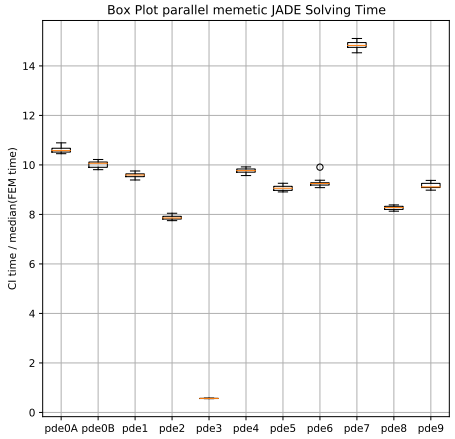


Figure 15: Relative solving time results of parallel memetic JADE after 10^4 #FE.

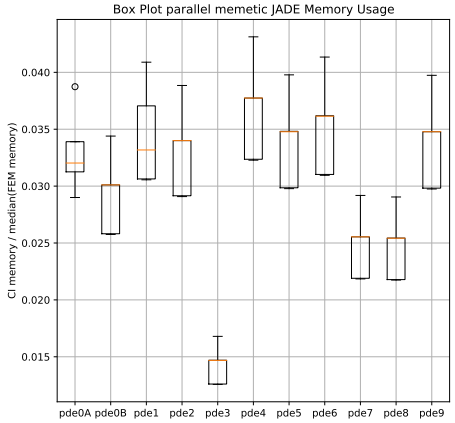


Figure 16: Relative memory usage results of parallel memetic JADE after 10^4 #FE.

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

Algorithm	serial JADE		parallel JADE		
stat	mean	median	mean	median	Wilcoxon Test
PDE 0A	0.6596 ± 0.5510	0.9285	0.6939 ± 0.6635	0.9243	unsig. undecided
PDE 0B	0.2027 ± 0.1302	0.1516	0.2809 ± 0.3071	0.2035	unsig. worse
PDE 1	0.0149 ± 0.0049	0.0151	0.0239 ± 0.0467	0.0146	unsig. undecided
PDE 2	0.0257 ± 0.0140	0.0224	0.0300 ± 0.0157	0.0255	unsig. worse
PDE 3	0.0328 ± 0.0169	0.0285	0.0371 ± 0.0206	0.0295	unsig. worse
PDE 4	0.0378 ± 0.0083	0.0352	0.0505 ± 0.0121	0.0481	sig. worse
PDE 5	1.1968 ± 0.0286	1.2056	1.2030 ± 0.0465	1.2053	unsig. undecided
PDE 6	0.4135 ± 1.2133	0.0018	0.5814 ± 1.3550	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 8: This table compares the results obtained by the parallel and the serial JADE. All results are obtained at 10^6 #FE.

6.4. Discussion

6.4.1. Memory Usage

As the boxplot 16 shows, the memory usage is roughly at the same level, somewhere between 1.5 to 4.0 percent of FEM solver. The memory usage is very similar to the serial experiment. This is expected, since memory-wise no integral changes have been performed.

6.4.2. Solving Time

The main purpose of the parallelisation is to cut down the solving time and accelerate all further experiments. To

that extent, the population is evaluated in parallel. As the solving time boxplot in figure 15 shows, this goal was achieved. The execution time after 10^4 #FE now ranks between the 8 and 15-fold of the FEM solver. Remarkable is that on PDE 3, the CI solver is even faster than NGSolve and takes only about $0.56 \cdot 65.2$ s ≈ 36.5 s. Of course, to compete with the solution quality, more #FE are needed, which increases the execution time. Image 17 shows the calculated mean Speed-Up (sp) over 20 replications, that is accomplished by substituting the serial JADE with the parallel version. The sp is calculated by dividing the serial solving time with the parallel solving time. The 95% confidence interval is shown in yellow. Plot 17 shows that the

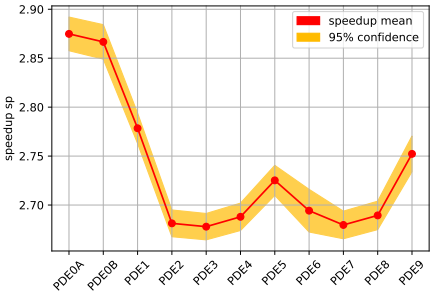
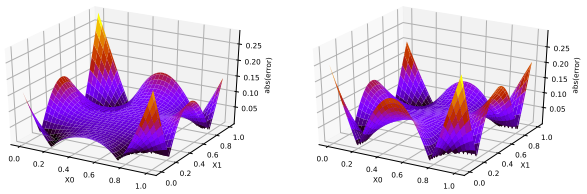


Figure 17: Mean speed-up of parallel JADE with 95% confidence interval.

speed-up varies by testbed function. This can be directly linked to the number of operations needed to construct the fitness function, which is primarily determined by 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 ($time(fit\ evaln) \uparrow \rightarrow sp \uparrow$). Larger fitness functions are needed by the testbed PDEs 0A, 0B, 1, 5, and 9 where the RHS are constructed of more terms.

6.4.3. PDE 4

Table 8 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 18. The plot suggests that the underlying structure of the error is similar in both cases. Only on the boundary $x_1 = 0, x_0 \in [0, 1]$, the serial JADE generates a visually smaller error. In order



(a) Absolute error, serial JADE
L2 norm: 0.0608
FE value: 0.4632

(b) Absolute error, parallel JADE
L2 norm: 0.0733
FE value: 0.6976

Figure 18: Absolute error of worst solution on PDE 4 by parallel and serial memetic JADE at 10^6 #FE

to make the results easier to interpret, the histogram from figure 19 shows the L2 norm data from table 8. The best solutions of both algorithms are at a similar quality level. The parallel sample introduces more results with a worse quality. 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. 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.

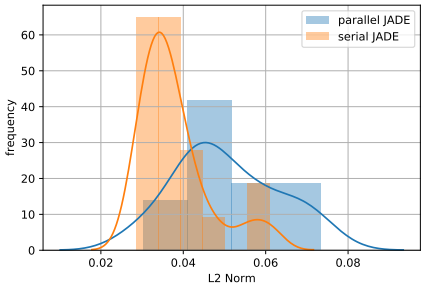


Figure 19: Histogram of the L2 norm data obtained by the serial and the parallel JADE on PDE 4 from table 8.

7. Experiment 2: Adaptive Number of Kernels

Although the parallel algorithm is effectively faster, the quality of the achieved solution is still not good enough. A common inaccuracy, especially with the testbed PDE 0A is that not all Gauss "bumps" are represented in the approximation.

7.1. Hypotheses

The idea tested here is an adaptive scheme for the number of kernels used. This new concept requires a convergence based halting criterion in the JADE algorithm. The algorithm is extended by a so called "state detector". Ideally, the state detector 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 as 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 function value is allowed to change over dT generations. The new paJADE is wrapped into the memetic framework. A flowchart of the process in shown in figure 20. 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. 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 approximation quality, the number of kernels must be increased. If the function value could not be decreased, a restart around the previous best population is performed.

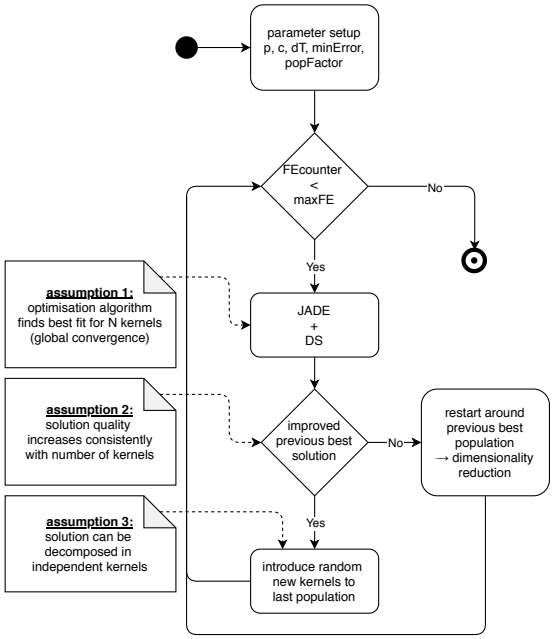


Figure 20: Flowchart of the adaptive kernel scheme.

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. 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. 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 PDE 2 the solution can not simply be decomposed into independent kernels. 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.

7.2. Experiment Setup

Again, as in the experiments before, machine 1 runs at 10^4 #FE and machine 2 performs 10^6 #FE. The 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 two new parameters dT and minError must be set. The minError is again set to 0. The delay time dT is set to 100. This choice is rather arbitrary and depending on the PDE, different values might be more successful. However, this property is not analysed in the current experiment.

7.3. Results

The table 9 shows the L2 norm data obtained by the adaptive JADE and compares them against the results from the parallel JADE. The Wilcoxon test indicates mixed results. The adaptive kernel scheme works fine on the PDE 0A, but it also produces significantly worse results on the problems PDE 2, 3, 4 and 7.

Algorithm	parallel JADE 10^6 #FE		adaptive JADE 10^6 #FE		
stat	mean	median	mean	median	Wilcoxon Test
PDE 0A	0.6939 ± 0.6635	0.9243	9.694E-16 ± 1.486E-16	9.255E-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 9: Comparison of the achieved L2 norm by the pJADE and the paJADE at 10^6 #FE.

7.4. Discussion

7.4.1. 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. 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 3.

7.4.2. Significantly Worse Quality

The Wilcoxon significance test of table 9 shows that the adaptive scheme is worse for the PDEs 2, 3, 4 and 7. 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 21 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. It seems that JADE exploits some areas

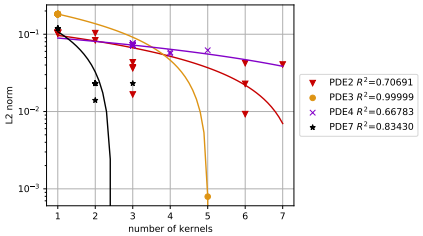


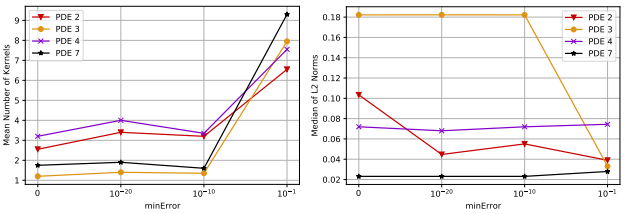
Figure 21: Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels.

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. 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 become better. As expected, the average number of kernels in the solution gets increased. This is confirmed by the plot in figure 22a. Figure 22b 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 10. Although



(a) Plot of the mean number of kernels against the *minError*. **(b)** Plot of the median L2 norm against the *minError*.

Figure 22: 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		
stat	mean	median	mean	median	Wilcoxon Test
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 10: Statistical comparison of the achieved L2 norm by paJADE with $minError = 0$ and $minError = 10^{-1}$ after 10^6 #FE.

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 23. Compared to the same plot at $minError = 0$, the coefficient of determination R^2 is smaller, indicating a poor correlation and a greater spread.

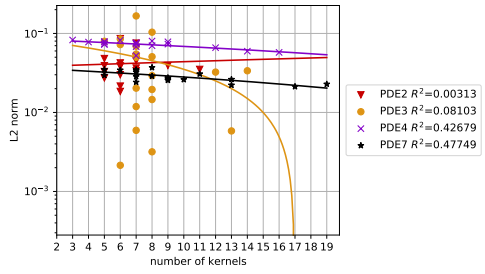
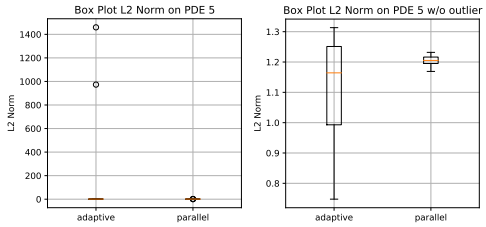


Figure 23: Semi-logarithmic plot of the correlation between the L2 norm and the number of kernels. The results are produced with a $minError = 10^-1$ after 10^6 #FE.

7.4.3. PDE 5

The results presented in table 9 show 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 24a. The same data is shown with and without the outlier. In general, it



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

Figure 24: Boxplot of solution quality on PDE 5 at 10^6 #FE with and without outliers.

can be said that the adaptive scheme exhibits a greater spread in the quality of the solution.

8. Experiment 3: Gauss-Sinus Kernel

Up until now, the results on PDE 5 were always considerably worse than on all other testbed functions. The idea discussed in this section is the usage of a new kernel. Theoretically, the Gauss Sine kernel (GSK) should be able to solve the testbed PDEs 0A and 0B exactly.

8.1. Hypotheses

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. This experiment tries to answer the question if the new GSK kernel type can effectively improve the results on PDE 5.

8.2. Experiment Setup

Machine 2 runs the experiment at the full computational budget of 10^6 #FE. 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 change. 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 parameters for the experiments are taken from table 4.

8.3. Results

The following table shows the statistical test. The comparison between the GSK and the GaK on the memetic parallel JADE is shown in table 11. The comparison is done with a budget of 10^6 #FE.

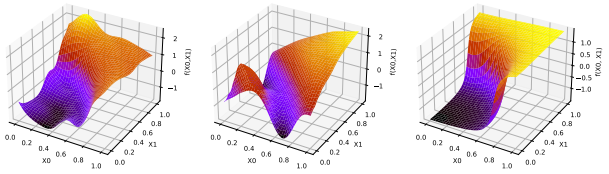
Algorithm	parallel JADE GaK 10^6 #FE		parallel JADE GSK 10^6 #FE		
stat	mean	median	mean	median	Wilcoxon Test
PDE 0A	0.6939 ± 0.6635	0.9243	0.8106 ± 0.7929	0.6765	unsig. undecided
PDE 0B	0.2809 ± 0.3071	0.2035	0.0667 ± 0.0470	0.0614	sig. better
PDE 1	0.0239 ± 0.0467	0.0146	0.1665 ± 0.1015	0.1952	sig.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

Table 11: Statistical comparison of the the parallel JADE using the GaK and the GSK.

8.4. Discussion

8.4.1. PDE 5

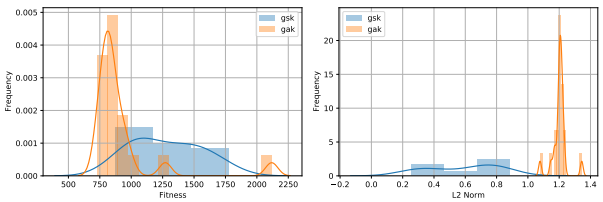
The hypothesis of the GSK is that it significantly increases the approximation quality of PDE 5. Further, it should overcome the phenomenon where the L2 norm increases with more #FE. As table 11 shows, the results are indeed significantly better. Again, this can be confirmed from a visual perspective by looking at the 3D plot of the approximation. Figure 25 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. The following histograms in figure



(a) best run GSK L2 norm: 0.252574 **(b)** worst run GSK L2 norm: 0.883869 **(c)** solution PDE 5.

Figure 25: Comparison of the best and the worst result generated by memetic pJADE with GSK after 10^6 #FE.

26b 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" are smaller than the same statistical indicators of the "GaK-data". The histogram of the fitness values is shown in figure 26a. Contrary, the fitness values of the GaK are smaller than the fitness values of the GSK. Because the fitness function changes with the usage of the GSK, the numerical values can not be compared directly. Similar to the plot in section



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

Figure 26: Histograms of GSK and GaK L2 norm and fitness value on PDE 5 after 10^6 #FE.

5.4.4, figure 27 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. This again shows that a good fitness value does not necessarily

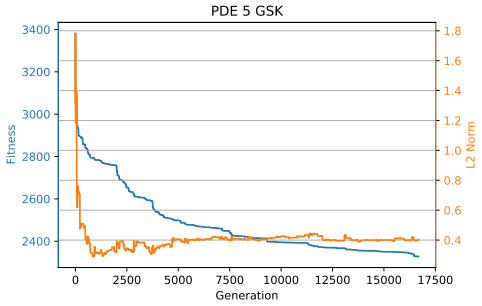


Figure 27: Fitness value and L2 Norm of an exemplary individual at every generation on PDE 5 using a GSK.

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

9. Limitations

9.1. Testbed

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 [4], the testbed falls short of Ordinary Differential Equation (ODE) and systems of differential equations. However, the results from the experiment 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.

9.2. 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 deviation 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. Similarly, the number of collocation points on the boundary n_B could be increased. This would shift the relative importance of the fitness function to the boundary, however it would also increase the computational effort.

9.3. 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. The ERD plot from figure 28 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 an algorithm reaches these $4 \cdot 11 = 44$ target values. Remarkable is the virtually non-existing performance difference between the serial and the parallel memetic JADE. 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 $\left(\frac{1}{11}\right)$ 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.

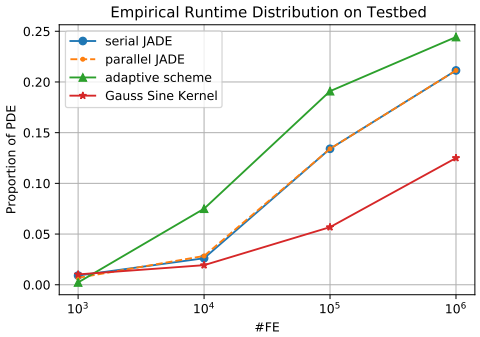


Figure 28: Empirical Runtime Distribution of all algorithms on the 11 testbed PDEs at target values $5 \cdot 10^{-2}$, $1 \cdot 10^{-2}$, $5 \cdot 10^{-3}$ and $1 \cdot 10^{-3}$.

10. Universial 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 ([18], [7]). In particular, [18] 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.

$$gsk(\mathbf{x}) = \omega e^{-\gamma||\mathbf{x}-\mathbf{c}||^2} sin(f||\mathbf{x}-\mathbf{c}||^2 - \varphi) \tag{34}$$

To prove that the universal approximation theorem is also applicable to the GSK, it must comply with the conditions placed by [18]. At first, a kernel, in this case the $gsk(\mathbf{x})$, must be continuous and bounded. This already restricts $\gamma > 0$. However, [4] found that not placing any limits on the parameters results in a better performance. Thus, this constraint is not implemented in the current version of the algorithm. Secondly, the integral over the whole domain of the kernel $K(\mathbf{x})$ must not be 0. Thus, it needs to be shown that

$$\int_{\mathbf{x}=-\infty}^{\infty} gsk(\mathbf{x})\ d\mathbf{x} \neq 0 \tag{35}$$

Intuitively, the next restriction on $\omega \neq 0$ is found.

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 coordinate transformation. This results in

$$\lim_{t\to\infty} \int_{r=0}^{t} \int_{\theta=0}^{2\pi} e^{-\gamma(r^2)} sin(fr^2 - \varphi) r\ dr\ d\theta \tag{36}$$

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. The integral can be further simplified by substituting $r^2 = u$. The resulting expression can be solved with "integration by part" $\int f(u)\frac{g(u)}{du} = f(u)g(u) - \int g(u)\frac{f(u)}{du}$. The formula has to be applied twice. The same integral is retrieved. Thus, the equation can be rearranged and solved for the integral. Considering the constant factors and the integral limits gives

$$\lim_{t\to\infty} \frac{\pi e^{-\gamma r^2}(-\gamma sin(fr^2 - \varphi) - f cos(fr^2 - \varphi))}{\gamma^2 + f^2} + C \Bigg|_{r=0}^{t}. \tag{37}$$

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

$$\frac{-\pi(\gamma sin(\varphi) + f cos(\varphi))}{\gamma^2 + f^2} \neq 0. \tag{38}$$

This places more constraints on the parameter of the GSK as seen in the equation (39) below.

$$\begin{aligned} &\gamma > 0 \\ &\omega \neq 0 \\ &\gamma sin(\varphi) + f cos(\varphi) \neq 0 \end{aligned} \tag{39}$$

These restrictions on the parameters could be enforced during the optimisation process. They could further be used to limit the search dimension. Thus, other optimisation algorithms that are good at handling constraints must be used.

11. 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

$$\hat{\mathbf{p}}_{\mathbf{apx}} = \Big[\underbrace{[\hat{\mathbf{p}}_{K_0}]}_{\text{kernel 0}}, \cdots \underbrace{[\hat{\mathbf{p}}_{K_i}]}_{\text{kernel i}}, \cdots \underbrace{[\hat{\mathbf{p}}_{K_N}]}_{\text{kernel N}}\Big]^T \tag{40}$$

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 $\hat{u}_{apx}(\mathbf{x})$.

$$\hat{u}_{apx}(\mathbf{x}) = \sum_{i=0}^{N} K(\hat{\mathbf{p}}_{K_i}, \mathbf{x}) \tag{41}$$

Since the order of the summation is irrelevant, any kernel-wise permutation describes an optimal solution $\hat{\mathbf{p}}_{\mathbf{apx}}$. Thus, the fitness function $F(u_{apx}(\mathbf{x}))$ has at least $N!$ number of local optima and all of them share the same function value. 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 $r = ||\hat{\mathbf{p}}_{\mathbf{apx}}||$. The following 3D plot in figure 29 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 $\hat{\mathbf{p}}_{\mathbf{apx}} = [2, 1, -1]^T$ describes an optimal solution that consists of 3 kernels. Any permutation of these three coordinates is itself a perfect fit. 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.

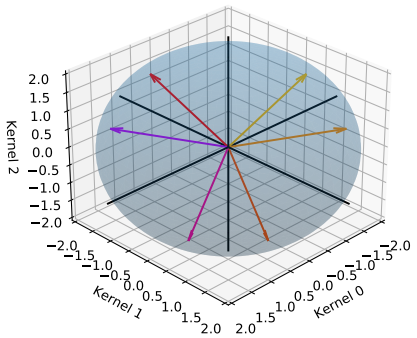


Figure 29: Distribution of exemplary optima on the fitness function in 3D space.

The symmetry is a fundamental property of the sum of RBF. Thus, it could even be applied in other fields that use this kind of representation. One of these could be non-linear function approximation.
Obviously, the order of summation is not only true for the optimum, but also for every other point in between. Thus, the entire fitness function exhibits this radial symmetry. More work investigating the structure of the fitness function must be done. With more knowledge about the features of this function, it might be possible to design algorithms specifically to that problem.

12. Conclusion

Using JADE could not replicate the objectively good performance observed with other related work. The reason for this 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 other optimisation algorithms. To speed up the optimisation process, the parallel JADE is implemented. Thus, the already present parallel hardware structure can be used. The parallel JADE works as expected, however it does not reduce the computational effort. Since nowadays, single-core machines do not exist any more, a parallel structure should be the standard for all newly created software. 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. 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. However, experiments have shown that this might not always be the case. 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 of the results for all other PDEs 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 into 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 investigated. This could lead to the design of an algorithm that is explicitly tailored to this optimisation problem. It might even be a starting point for further theoretical advances.

References

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

[2] David S. Broomhead and David Lowe: "Multivariable Functional Interpolation and Adaptive Networks". In: *Complex Systems* (1988).

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

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

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

[6] R. J. Guyan: "Reduction of stiffness and mass matrices". In: *AIAAJ Aeronautics / Astronautics* (1965), p. 380. DOI: 10.2514/3.2874.

[7] Thomas Hangelbroek and Amos Ron: "Nonlinear approximation using Gaussian kernels". en. In: *Journal of Functional Analysis* 259.1 (July 2010), pp. 203–219. ISSN: 0022-1236. DOI: 10.1016/j.jfa.2010.02.001. URL: http://www.sciencedirect.com/science/article/pii/S0022123610000467 (visited on 02/28/2020).

[8] Nikolaus Hansen; Sibylle D. Müller, and Petros Koumoutsakos: "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)". In: *Evolutionary Computation* 11.1 (Mar. 2003). Publisher: MIT Press, pp. 1–18. ISSN: 1063-6560. DOI: 10.1162/106365603321828970. URL: https://doi.org/10.1162/106365603321828970 (visited on 08/15/2020).

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

[10] Daniel Howard and Simon C. Roberts: "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., July 2001, pp. 34–41. ISBN: 978-1-55860-774-3. (Visited on 02/27/2020).

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

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

[13] Mallipeddi and Suganthan: "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. June 2008, pp. 3663–3670. DOI: 10.1109/CEC.2008.4631294.

[14] Nikos E Mastorakis: "Unstable Ordinary Differential Equations: Solution via Genetic Algorithms and the method of Nelder-Mead". en. In: Elounda, Greece, Aug. 2006, 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).

[15] William F. Mitchell: *NIST AMR Benchmarks*. en. Mar. 2018. URL: https://math.nist.gov/amr-benchmark/index.html (visited on 02/21/2020).

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

[17] Natee Panagant and Sujin Bureerat: "Solving Partial Differential Equations Using a New Differential Evolution Algorithm". en. In: *Mathematical Problems in Engineering* 2014.2014 (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).

[18] J. Park and I. W. Sandberg: "Universal Approximation Using Radial-Basis-Function Networks". In: *Neural Computation* 3.2 (June 1991). Publisher: MIT Press, pp. 246–257. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.2.246. URL: https://doi.org/10.1162/neco.1991.3.2.246 (visited on 08/11/2020).

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

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

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

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

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

[24] Suganthan: *Suganthan/CEC2019*. original-date: 2019-07-01T11:46:13Z. Jan. 2020. URL: https://github.com/P-N-Suganthan/CEC2019 (visited on 03/20/2020).

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

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