School of Computation, Information and Technology
Chair of Scientific Computing
Technical University of Munich

Convergence of numerical methods for ordinary differential equations of second order

Interdisciplinary Project Mathematics

Examiner: Prof. Dr. Felix Dietrich

Anastasiya Liatsetskaya

April 14, 2024

Contents

1 Introduction

Partial Differential Equations can be used to model different phenomena from different application domains, for example, Physics, Chemistry, Meteorology and other. As described in [6] the difference between an ODE and PDE is that ODE contain functions of one independent variable and their derivatives on that variable. In contrast PDE can have more than one independent variables and derivatives of functions on these variables. Which makes this class of models more expressive and more difficult to solve. Already for ODEs there are examples when there is no analytical form for the solution of the equation or equation system. For PDE the situation gets more complicated as there are examples when it is not known if a solution exists for an initial values problem, for example in Navier-Stokes equations. If an analytical solution is not known one can apply a numerical algorithm to try to approximate the solution.
In this project report a specific PDE problem is considered namely the second order wave equation defined as

$$\frac{\partial u(x,t)}{\partial t^2} = \Lambda u(x,t)\ x \in \Omega, t > 0,$$

where Λ denotes the Laplacian operator, with boundary conditions

$$u(x,t) = g(x,t)\ x \in \partial\Omega, t > 0.$$

In this report we will use spectral methods to solve the PDE with basis functions being randomly sampled with help of the algorithm Sampling Weights of Neural Networks developed by Prof. Dr. Dietrich and the research team [4]. Then one needs to determine the weights $a_k(t)$ of the basis functions.
As shown in the Chapter 2 weights will be the solution of a linear ODE of second order $Ax''(t) = Bx(t)$. However, this approach might generate four challenges: 1) if many components are chosen in the spectral method, then the matrix A of the linear ODE system will become large 2) the matrix A might be very ill-conditioned 3) the matrix A is not sparse 4) the matrix A might not be square
The goal of the project is to investigate how the linear ODE can be solved, this includes stability and convergence considerations. Also since the considered ODE methods will result in linear systems of equations, the question is investigated, how a potentially large, ill-conditioned and dense linear system of equations can be solved.

2 Numerical Methods for PDEs

In this chapter numerical methods for Partial Differential Equation are introduced. In section 2.1 the spectral method is desribed. We discuss its application to the second order wave equation. The spectral method is not the only method for approximating the solution of a PDE. In the sections 2.2 and 2.3 two other methods Finite Difference and Finite Element methods will be introduced. However, these two methods are not the only alternatives available to the spectral method. One further alternative is the Finite Volume method described in [15].

2.1 Spectral method

The ansatz of the Galerkin spectral approach is to represent the solution of the PDE as a weighted sum of basis functions:

$$u(x,t) = \sum_{k=1}^{\infty} a_k(t)\psi_k(x) \tag{1}$$

where ψ_k are basis functions and $a_k(t)$ are weights. The basis functions are functions with global support. One major advantage of the spectral method is its convergence properties. As shown in [6] for analytical functions $u \in C^w(\Omega)$ the error shows exponentially fast decay depending on the number of components used N. The authors in [6] note that for functions with strongly localised phenomena the spectral method might be inefficient.

This approach separates spatial and temporal part of the PDE solution $u(x,t)$ since the coefficients only depend on time and basis functions only depend on space. The sum contains infinitely many components. One can obtain an approximation to (1) by considering a finite number of components N:

$$u(x,t) \approx \sum_{k=1}^{N} a_k(t)\psi_k(x).$$

There might be a tradeoff when choosing the number N. On one hand the more components are considered, the more accurate the approximation might be. On the other hand bigger number of components might lead to higher computational effort. One method of choosing the number N is described in [6]. The method is based on an error estimator for the approximation error

$$\epsilon_N = ||u_N - u||.$$

The error estimator is built by using multiple spectral method approximations each computed on its own ansatz space with its own number of components N'. The method relies upon the assumption that the more components N' one uses the more accurate is the solution. If three approximations computed with the numbers of components N, N' and N" are used and an error tolerance TOL is given, then in [6] the following strategy of selecting the index N^* is proposed:

$$N^* = N + (N' - N)\log\frac{\text{TOL}}{\epsilon_N}/log\frac{\epsilon'_N}{\epsilon_N}$$

with error estimators $\epsilon_N = ||u_N - u_{N"}||$ and $\epsilon_{N'} = ||u_{N'} - u_{N"}||$. The authors in [6] suggest if $N^* < N$" then the solution N" can be taken. Otherwise one can take the approximations N', N" and N^* and repeat the procedure.
If the basis functions are known, the next step is to determine the coefficients $a_k(t)$. One important consideration here is that the coefficients are not vectors, but rather functions. One way to determine the coefficients is to insert the approximate solution into the original PDE leading to:

$$\sum_{k=1}^{N} a''_k(t) \cdot \psi_k(x) = \Lambda \sum_{k=1}^{N} a_k(t)\psi_k(x) = \sum_{k=1}^{N} a_k(t)\Lambda\psi_k(x). \tag{2}$$

Then the obtained equation needs to be solved for the coefficients $a_k(t)$. The idea described in [6] is to multiply both sides of the equations by a basis function $\psi_k(x)$ and integrate over the domain Ω. Since the coefficients $a_k(t)$ do not depend on space they can be brought outside the integral leading to

$$\sum_{k=1}^{N} a''_k(t) \int_\Omega \psi_i\psi_j dx = \sum_{k=1}^{N} a_k(t) \int_\Omega \Lambda\psi_i\psi_j dx$$

when multiplied with ψ_j from both sides. If the basis functions ψ_i, psi_k have a global support, then the integrals $< \psi_i, \psi_j >= \int_\Omega \psi_i\psi_j dx$ and $\int_\Omega \Lambda\psi_i\psi_j dx$ might not be zero in general.
By multiplying both sides of the system (2) by basis functions one obtains a linear system of ordinary differential equations

$$Ax''(t) = Bx(t) \tag{3}$$

with $a_{ij} =< \psi_i, \psi_j >$ and $b_{ij} = \int_\Omega \Lambda\psi_i\psi_j dx$. The initial values for the ODE can be derived by looking at the boundary conditions $< u(x,0), \psi_i(x) >$ as $a_i(0)$.
The system (3) is the second order system of ODEs, however, there exists a transformation described in [5] which allows to transform it to the first order system. The drawback of this method is the increased dimensionality of the system, since additional variables will be introduced.
The matrix A is symmetric. Its entries are composed of values $< \psi_i, \psi_j >$. These entries are fixed and can be precomputed if the basis functions are fixed. The basis functions $tanh(w_k x + b_k)$ have a global support, as a result the matrix A might be dense. In general [6] the computation of $< \psi_i, \psi_j >$ will require

to compute an integral. There might not be an analytical expression for the integral value. As a result, as discussed in [6] numerical quadrature formulas can be applied to approximate the solution.
Matrix condition number with the respect to inversion is defined as

$$\mathbf{k}(A) = ||A||||A^{-1}||.$$

Condition number is not a property of numerical algorithm, but rather a property of the problem to solve. In case of the matrix inversion problem high condition number means that it is difficult to compute the matrix inverse as the final result is very sensitive towards noise and computational roundoffs which occur by the finite precision arithmetics used by modern day computers. If the matrix A has a low condition number, then the equation (3) can be written in the explicit ODE form:$x'(t) = f(x,t)$. For large condition number the equation (3) is treated as an implicit linear ODE.
If the L-2 norm is chosen for the condition number, then according to [2] there exists a connection between the L-2 norm of a matrix and its singular values. Namely for the largest singular value $\sigma_{max} = ||A||_2$ and for the smallest singular value $\frac{1}{\sigma_{min}} = ||A^{-1}||_2$. Then a matrix will have a large condition number if $\sigma_{max} >> \sigma_{min}$. There exists a connection between singular values and eigenvalues of a matrix. For symmetric and Hermitian matrices the eigenvalues and singular values coincide. In the general case one can use the fact, that AA^T is symmetric for a real valued matrix A and σ_i^2 correspond to the eigenvalues of this matrix. In order to approximate the condition number the power iteration method based can be used. It is an iterative algorithm based on Rayleigh quotient as described in [2]:

$$r(A, z_k) = \frac{z_k^T A z_k}{z_k^T z_k}$$

which allows to approximate the largest eigenvalue of the matrix A. There exists an inverse power iteration which allows to compute the smallest eigenvalue.

2.2 Finite difference methods

One idea when trying to approximate a PDE could be to approximate the derivatives appearing in the equation. One way to approximate the first derivative as shown in [15] is to look at its Taylor series expansion and to take the terms which do not involve derivatives as an approximation:

$$\frac{du}{dx}(x) \approx \frac{u(x+h) - u(x)}{h}$$

[15] also shows how a Taylor expansion can be used to provide information about the approximations error:

$$u(x+h) = u(x) + h\frac{du}{dx} + \frac{h^2}{2}\frac{d^2u}{dx^2} + \frac{h^3}{6}\frac{d^3u}{dx^3} + \frac{h^4}{24}\frac{d^4u}{dx^4}(\epsilon)$$

with $\epsilon \in (x, x+h)$. As a result the error is of order $O(h)$
A similar approach can be applied to the case $u(x-h)$. The obtained approximations for $u(x+h)$ and $u(x-h)$ can be combined to produce a fourth order approximation to the second derivative:

$$\frac{d^2u(x)}{dx^2} = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} - \frac{h^2}{12}\frac{d^4u(\epsilon)}{dx^4}$$

with $\epsilon \in (x-h, x+h)$. This formula is also known as the centered difference approximation. As shown in [15] multiple different difference scheme are possible for Laplacian operator. They also involve different number of points, an example for 5 and 8 point stencils can be found in [15].
One important property of the described approximation schemes is that the introduced schemes will lead to a linear system off equations. No derivatives will be involved anymore in the equation. The approach described in [15] uses a mesh grid, these are the points where the PDE solutions will be approximated. The stencils can lead in case of large PDE systems to large sparse matrices, which are diagonally dominant. As a result, despite having a large system of linear equations it can still be solved efficiently. One particular method described in [15] is Fast Poisson solvers with run time $O(Nlog(N))$ with N being the size of the matrix.

2.3 Finite Element method

Another idea of how to approximate a solution of PDE is to look at the weak formulation of the PDE:

$$\text{Find } u \in V_h \text{ such that } a(u,v) = f(u,v) \forall v \in V_h$$

where $a(u,v) = \int_\Omega \nabla u \cdot \nabla v dx$ and $f(u,v) = \int_\Omega f v dx$. In the formula V_h denotes the space of candidate functions. Among them will be thought for the approximation of the PDE solution. The question is how to select such space V_h. As noted in [15] the weak formulation poses requirements on potential candidates, namely that the gradient of functions in V_h should exist. Further, boundary conditions can also be taken into account.
Both Spectral method and Finite element start from the weak formulation of the problem and use a finite dimensional ansatz space of candidate functions V_h. A major difference between the two methods is that Spectral method relies upon global basis functions, whereas Finite element uses local basis functions.
One interesting choice of the function space V_h is presented in [15]. The functions presented in the book of Saad are piecewise linear on the triangulated domain and continuous. Moreover each function ψ_i is assigned to a specific mesh point x_i:

$$\psi_i(x_j) = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{if } x_i \neq x_j \end{cases} \tag{4}$$

Such construction allows to define functions ψ_i uniquely and these functions form a basis of the space $V_h = \{\psi | \psi_\Omega \text{ continuous }, \psi_\Gamma = 0, \psi_{K_i} \text{ linear } \forall j\}$. As a result each function $\psi \in V_h$ can be expressed as $\psi(x) = \sum_{i=1}^n \epsilon_i \psi_i(x)$ with n being the number of the nodes in the space triangulation.
Then the task is how to determine the coefficients ϵ_i defining n degrees of freedom. This can be done by considering the requirement posed by the weak formulation of the PDE. Such approach will lead to a linear system of equations $A\epsilon = b$ where $a_{ij} = a(\psi_i, \psi_j)$ and $b_i = (f, \psi_i)$. The choice of basis functions from (4) leads to a sparse matrix A, since as noted in [15] an entry $a(\psi_i, \psi_j)$ is nonzero only if nodes i and j are vertices of the same triangle.

3 Sampling weights of deep neural networks

In this chapter the generation of the basis functions $\psi_k(x)$ for the spectral method is described.
Often weights for Neural Networks determined via an optimisation procedure. For a specified loss- function one seeks values of weights which maximise or minimise the function [3]. Often this is done via Gradient descent. This is an iterative algorithm and many steps might be required until the loss will become sufficiently small. Additionally Deep Neural Networks with many layers might contain large number of parameters. As a result, the back propagation done at each iteration step might become very expensive. One advantage of the loss-function optimisation approach is that it also allows to optimise the parameters with respect to the goal the model wants to achieve and it allows to incorporate knowledge of available data points in the choice of the model's weights. Optimisation is not the only way how the weights can be generated. One idea could be to randomly sample weights for the Neural Networks. As mentioned in [4] the weights can be sampled from a distribution where no information about available data points is contained, for example Normal distribution, and as authors describe it can lead to Neural Networks which need large number of neurons to achieve the desired performance. An alternative could be to incorporate information about the data points in the distribution of the Neural Network parameters.
The idea proposed by authors in [4] is to construct a Neural Network layer by layer. In order to determine the weights and bias of layer l two data points x_1 and x_2 as well as their corresponding function values $f(x_1)$, $f(x_2)$ coming from an available data set corresponding to the problem are used. The authors define one to one correspondence between the data points and parameters of the l-th layer. Namely the weights and bias corresponding to a neuron i are defined as

$$w_{l,i} = s_1 \frac{x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}}{||x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}||^2}, \; b_{l,i} = <w_{l,i}, x^{(1)}_{l-1,i}> + s_2$$

where $x^{(1)}_{l-1,i}$ denotes the output of the Neural Network constructed by layers 1 up to $l-1$ applied to the first data point. The data points $x^{(1)}$ and $x^{(2)}$ should not coincide. The constants s_1 and s_2 are chosen such that for the ReLU activation function $\psi(x^{(1)}) = 0$ and $\psi(x^{(2)}) = 1$ and for hyperbolic tanh activation function $\psi(x^{(1)}) = 0.5$ and $\psi(x^{(2)}) = -0.5$. As a result, for the ReLU activation function there values $f(x^{(1)})$ and $f(x^{(2)})$ are linearly interpolated for the regression task. In case of classification the tanh function will build a border between the values of the two points if they belong to different classes.
The next question could be on how to select the pairs of points $x^{(1)}$ and $x^{(2)}$. As the authors in [4] describe, the points which are spatially close, but differ significantly in their output $f(x)$ should get a higher probability to be sampled. The authors define the unnormalised density for the sampling at layer l and given the previous layers with already selected weights and biases as

$$q^{\epsilon}_l(x^1_0, x^2_0 | \{W_j, b_j\}_{j=1}^{l-1}) = \begin{cases} \frac{||f(x^{(2)}_0) - f(x^{(1)}_0)||_Y}{\max\{||x^{(2)}_{l-1} - x^{(1)}_{l-1}||_{X_{l-1}}, \epsilon\}} & \text{if } x^{(1)}_{l-1} \neq x^{(2)}_{l-1} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $\epsilon = 0$ if $l = 1$ and $\epsilon > 0$ otherwise. The function f is assumed to be Lipschitz-continuous and $Y = f(X)$. Then if the density constant is defined as $C_l = \int_{X \times X} q^{\epsilon}_l d\lambda$ where λ is the Lebesgue measure. Then the resulting density is defined as $\frac{q^{\epsilon}_l}{C_l}$ if $C_l > 0$, otherwise the distribution is uniform over the domain. As a result, each layer of the Neural Network might have a distribution which differs from the distributions at all the other layers.
The next question could be which functions can be approximated by a Neural Network constructed via the described procedure. The authors in [4] derive the error bounds for the functions from the Barron space defined as

$$B = \{f : f(x) = \int_{\Omega} w_2 \psi(<w_1, x> - b) d\mu(b, w_1, w_2) \text{ and } ||f||_B < \infty\}$$

then as derived in [4] for any probability measure π and $\epsilon > 0$ there exists a sampled Neural Network Ψ with $N_1 \in \mathcal{N}_{>0}$ neurons and one hidden layer such that

$$||f - \Psi||_2^2 = \int_X |f(x) - \Psi(x)|^2 d\pi(x) < \frac{(3+\epsilon)||f||_B^2}{N_1}.$$

The authors also show that any Neural Network with one hidden layer can be approximated by a Deep Neural Network with any $l \in \mathcal{N}_{>0}$ layers constructed with the weight sampling algorithm. The detailed proof can be found in [4].
With the algorithm for selecting weights of Neural Networks described the question then is how it can be applied for the Partial Differential equations. In case of the spectral method, how the Neural Networks can be used to generate the basis functions $\psi_k(x)$. One idea could be to randomly select points in the input space X and then to construct a one layer Neural Network. Such construction would not use the density defined in the formula (5) since it requires to know the function values $f(x)$. The formula for the weights and biases, however, does not require the function values and can be constructed with the points sampled from the space X. By using such approach the basis functions ψ_k can be constructed. For each pair of sampled points one obtains one basis function.
The described approach of using sampled Neural Networks for constructing the basis functions of the spectral method is not the only way the sampled Neural Networks can be used to approximate solutionf of a PDE. The authors in [4] use sampled Neural Networks to construct a map predicting a solution of Burgers equation at time 1 given the initial condition at time 0. Further details can be found in [4].

4 Numerical Methods for ODEs

Recall that the solution when aplying the spectral method to the PDE is of the form $\sum_{i=1}^{N} a_i(t)\psi_i(x)$ The equation for coefficients $a(t)$ is a system of ordinary differential equations. More specifically this is an instance of linear initial value problem. There exists an analytical solution of this problem, namely

$a(t) = exp(A^{-1}Bt)$. However, in practice this solution might be infeasible to compute due to high cost of computing matrix exponential. As a result, numerical approximations can be used to form the approximations to the solution $u(x,t)$.
If a numerical approximation is used, one important question to consider is whether this approximation is close to the true solution, whether the behaviour of the approximate solution is qualitatively similar to the solution. The second question is how an approximate solution will look like, will it be discretised or not.
In this project two families of methods for numerical solutions of ordinary differential equations will be considered linear implicit Runge- Kutta methods and BDF methods. Both families will build a mesh function $u_\Delta(t) \approx u(t) \forall t \in \Delta$ where Δ corresponds to the sequence of times t_i where the solution will be approximated.
Then the question is what is the error a numerical method makes on the mesh Δ and how does this error behave especially if the mesh is large and the method must make many steps to approximate all points in Δ. Another important point to consider is since the numerical approximation might not be exact, how do accumulated computation errors influence the approximation of the value $u(t_i)$.
According to [5] a mesh function is convergent on the time interval $[t_0, T]$ to the true function $x \in C([t_0, T],^d)$ when the discretization error $\|\epsilon_\Delta\|_\infty = \max_{t\in\Delta}|x(t) - x_\Delta(t)|$ converges to 0 as the maximal step width τ_Δ in mesh goes to zero. Order p of the convergence gives information on how fast does the convergence occur: a mesh function converges with order $p > 0$ if

$$\|\epsilon_\Delta\|_\infty = O(\tau_\Delta^p) \text{ for } \tau_\Delta \to 0$$

where the consistency error $\epsilon(x,t,\tau)$ is the difference between the actual solution and computed numerical approximation of the ODE at time $t + \tau$ starting at the value x and time t. A numerical algorithm algorithm defining approximate solutions of the ODE on the mesh has a consistency order p if the consistency error $\epsilon(t,x,\tau) = O(\tau^{p+1})$ for $\tau \to 0$ across the domain Ω
The result from convergence gives statement that there exists a small step size τ_Δ, such that when the step size goes to zero the convergence can be observed. However, it does not give information on how small this step size τ_Δ should be. This question is important for practical applications since for very small step sizes the computation time and load might be too high. Some ODE problems as described in [5] might require very small step sizes τ_Δ so that the convergence of discrete condition number might be observed. This type of problems are referred to as stiff problems.
An ODE is called stable in the Lyapunov sense if for every $\epsilon > 0$ there exists a $\sigma > 0$ such that the solution curve for the perturbed initial values of the maximal magnitude σ $x \in B_\sigma(x_0)$ are contained within a pipe with radius ϵ and do not leave it for all $t > t_0$. If the problem is unstable with small errors in initial values at t_0 leading to very different results at time t compared to the unperturbed case and the exactness of initial values cannot be guaranteed, which is the case in many practical applications, then one cannot expect a numerical algorithm to compute a close solution to the unperturbed case starting with inexact initial values.
In case of one step methods for initial value problems as shown in [5] for linear ODEs $x' = Cx$ the stability of the problem can be inherited by a discrete phase flow $\Psi^\tau = R(\tau C)$, where R describes a rational approximation to the exponential function, if

$$\max_{\alpha\in\sigma(C)} |R(\tau\alpha)| < 1$$

with $\sigma(C)$ describing the set of eigenvalues of C. τ in the formula describes the step size of the algorithm. Ψ^τ is assumed to be applied recursively. This leads to a set

$$\mathcal{S} = \{z \in \mathcal{C} : |R(z)| \leq 1\}$$

One additional requirement is posed for the eigenvalues leading to $|R(\tau\lambda)| = 1$. Namely such eigenvalues should have index 1. For a given linear problem it is necessary for all eigenvalues λ of the matrix C, that $\tau\lambda \in \mathcal{S}$ for the computed mesh function to be stable. If the region $\mathcal{S}$ is a compact set, then there is a restriction for which step sizes τ the discrete approximation will be stable.
The form of stability region depends on the form of the function R. The authors in [5] distinguish multiple stability definitions.

- A-stability: $\mathbb{C}_- \subset \mathcal{S}$. No restriction of the step size, works for all stable linear systems independent of the eigenvalues of C
- L-stability: A stability and additionally $\lim_{z\to\infty} R(z) = 0$.
- A(α) stability: $\mathcal{S}_\alpha = \{z \in \mathbb{C} : |arg(-z)| \leq \alpha\}$ for $\alpha \in [0, \pi/2]$ No restriction of the step size, but there are restrictions on the allowed eigenvalues

For linear multisteps methods the stability region is formed as

$$\mathcal{S} = \{z \in \mathbb{C} : \rho_z(E)X = 0\} \text{ is a stable difference equation}$$

with ρ being characteristic polynomial of a linear multistep method: $\rho(E)x_\tau = \tau\sigma(E)f_\tau$. Similarly to the one-step case the authors in [5] define the stability definitions for multistep methods as

- A-stable: $\mathbb{C}_- \subset \mathcal{S}$
- $A(\alpha)$-stable: $\mathcal{S}_\alpha \subset \mathcal{S}$ for $\alpha \in [0, \pi/2]$

According to [5] for linear multistep methods two components are required for the algorithm to converge to true solution on the mesh- consistency and stability.

4.1 Linear- implicit Runge- Kutta methods

One family of methods which can be applied to stiff problems is linearly- implicit one step methods. One step methods mean that the algorithm on the step t_{i+1} will use the result u_{t_i}computed on the step t_i to form the solution value $u_{t_{i+1}}$. In order to inherit the stability of the original problem these method build rational approximations of the exponential function in the linear case and not a polynomial as it is the case for explicit Runge- Kutta methods. According to [5] one significant advantage of linearly- implicit one step methods is that the algorithm will not require to perform Newton-Iteration, but rather to solve one linear system of equations in each step.
The idea of the method as described in [5] is to write the discrete evolution operator as $\Psi^\tau x = x + \tau\sum_{j=1}^{s} b_j k_j$. The differential equation $u(t)' = f(u)$ can be brought into the form $u(t)' = Ju(t) + (f(u) - Jx(t))$ with $J = Df(u)$. And then only the first part $Ju(t)$ is implicitly discretised. On the second part an explicit Runge-Kutta method is applied. Then a component k_i can be determined as

$$k_i = J(x + \tau\sum_{j=1}^{i}\beta_{ij}k_j) + (f(x + \tau\sum_{j=1}^{i-1}\alpha_{ij}k_j) - J(x + \tau\sum_{j=1}^{i-1}\alpha_{ij}k_j)) \text{ for } i \in 1..s$$

As a result for each component k_i a linear system of equations must be solved:

$$(I - \tau\beta_{ii}J)k_i = \tau\sum_{j=1}^{i-1}(\beta_{ij} - \alpha_{ij})Jk_j + f(x + \tau\sum_{j=1}^{i-1}\alpha_{ij}k_j)$$

this equation is based on the explicit formulation of the initial value problem:$x' = f(x)$. If the matrix A from the equation (3) is difficult to invert, then one can use formulation suggested in [16] and multiply both sides with the matrix A. This procedure will give the following formulation of the linear system of equations:

$$(A - \tau\beta_{ii}B)k_i = \tau\sum_{j=1}^{i-1}(\beta_{ij} - \alpha_{ij})Bk_j + B(x + \tau\sum_{j=1}^{i-1}\alpha_{ij}k_j)$$

The condition number of matrix $A - \tau\beta_{ii}B$ might be different from the condition number of the matrix A. Despite A being ill conditioned, $A - \tau\beta_{ii}B$ might be well conditioned and can be inverted.
A linear- implicit Runge- Kutta method requires to determine the weights α_{ij}, β_{ij} and b_i. As shown in [5] the idea of rooted trees from J.C. Butcher with light modifications can also be used for linear-implicit methods to determine the coefficients, such that an order of convergence p can be achieved.

4.2 BDF methods

One step methods use the last computed value $x(t_i)$ to compute the mesh value $x(t_{i+1})$. They do not use any further computed values of the mesh Δ, no further history. The idea of k-multistep method is to use already computed values $x(t_{i-k+1}), ..., x(t_i)$ to build the value $x(t_{i+1})$. According to [5] one major benefit of the BDF family of methods is that it requires to solve only one nonlinear system of equations per algorithm step in general case compared to s for a linear- implicit Runge- Kutta method.
As discussed in [5] A-stable linear multiple step method has consistency order $p \leq 2$. As a result, if eigenvalues of the problem lie on the imaginary axis a multi-step method cannot be applied, if consistency order of 2 is not sufficient for acceptable approximation quality. If eigenvalues do not lie on the imaginary axis $A(\alpha)$- stable method can be used. However, as described in [5] if the eigenvalues are close to $\pi/2$, then the constants of the error decay will explode for high consistency order. In both cases using a one step method might give better approximations of the solution on the mesh $x(t)$. [8] has shown existence of $A(\alpha)$ stable k-multistep methods with consistency order $p = k$ for any $k > 0$.
BDF family are algorithms designed for stiff problems. This is a $A(\alpha)$ stable family. One additional property of BDF methods is that they inherit the requirement $R(\infty) < 0$ for the stability function R. The general form of the k-step BDF method is

$$\alpha_k x_\tau(t_{j+k}) + ... + \alpha_0 x_\tau(t_j) = \tau f_\tau(t_{j+k}) \tag{6}$$

This is an instance of linear k-step methods. According to [5] for the number of steps k there exists exactly one method of maximal consistency order $p = k$ and which also suffices the form (6).
However, as shown in [9] a k-step BDF method is stable only of the number of steps $k \leq 6$. Additionally there are cases shown in the book [5] where the angle α gets very small the higher the consistency order of a BDF method and respectively the number of considered steps k are.
There exists a modification of BDF methods for DAE as shown in [5] which in case of the ODE (3) will not require to invert the matrix A. In case of linear differential algebraic problem with index $\nu \leq 1$ an ODE can be formulated as $Bx' = f(x), x(0) = x_0$ where the matrix B is assumed to be singular. As shown in [5] one can represent k-step BDF method as

$$q_k'(t_{i+1}) = f(t_{i+1}, x_\Delta(t_{i+1}))$$

where q_k is a polynomial which interpolates k computed mesh values before $i+1$: $q_k(t_{i-j}) = x_\Delta(t_{i-j})$ for $i = 0, ..., k$. Then one can rewrite the equation as

$$B(q_k'(t_{j+1}) - \mu_k(t_{j+1})(x_\Delta(t_{j+1}) - x^0)) - f(x_\Delta(t_{j+1})), x^0 = q_k(t_{j+1})$$

with $\mu_k(t_{j+1}) = \frac{1}{t_{j+1}-t_j} + ... + \frac{1}{t_{j+1}-t_{j-k+1}}$. In case of the equation (3) the formula above will be transformed to the equation:

$$A(q_k'(t_{j+1}) - \mu_k(t_{j+1})(x_\Delta(t_{j+1}) - x^0)) - Ax_\Delta(t_{j+1}) = 0, x^0 = q_k(t_{j+1})$$

this system can be solved with the Newtons Method or since it is a linear system with respect to $x_\Delta(t_{j+1})$ with a solver for linear systems of equations. The resulting matrix similarly to the case with a linear- implicit Runge- Kutta method might be well conditioned despite the matrix A having a high condition number.

4.3 Note on linear DAE systems

As described in [11] a general form of a linear differential-algebraic system can be phrased as

$$\Sigma : \begin{cases} Ez'(t) = Fz(t) + Gu(t), Ez(0) = Ez_0 \\ y(t) = Hz(t) \end{cases} \tag{7}$$

If an i-th row of the matrix E is not a zero vector,then the i-th row of the system (7) is a differential equation, otherwise it is an algebraic equation. In general the matrix E is assumed to be singular and as a result one cannot transform differential equations in (7) to the explicit form $x'(t) = Ax(t) + Bu(t)$ by using the inverse of E. As a result the problem defined in (3) with ill conditioned matrix A can be considered as a linear DAE. There might appear a question whether there are solution methods designed for DAE and assuming that the matrix E from the system (7) is not invertible.
According to [17] there exists an analytical solution to the DAE $Ax'(t) + Bx(t) = f(t)$ assuming that the matrix pencil $A + \lambda B$ is regular. However, this solution relies upon Jordan canonical form which is difficult to compute numerically. Both [11] and [14] suggest bringing DAE to index one form and then to apply standard integrators for ODEs like BDF.
According to [11] a regular DAE system can be brought to a form resembling an ODE and additionally there will be conditions which a potential solution should fulfill. As a result the set of solutions to the ODE system can be larger than the set of the solutions to the respective DAE. If the system (3) has a unique solution, then according to [11] the index of the system is 0 and no index reduction algorithm needs to be applied. In that case a method for ODEs can be directly applied to the system.

5 Linear system of equations

Both linear- implicit Runge- Kutta and BDF methods require to solve systems of linear equations in order to compute approximations $x(t_i)$. A notable challenge is that in an obtained linear system $\hat{A}x = b$ the matrix $\hat{A}$ can be ill conditioned, not sparse and large. As described in [1] using a direct solver, for example LU, does not scale well with problem size in terms of required memory and computing time. More specific matrix factorisation techniques like LU, QR and Cholesky factorisation have the run time is $O(n^3)$. A second problem might be, that according to [1] direct methods are often based on some form of Gaussian elimination which for ill conditioned problems might lead to very poor quality of computed solution. This leads to a question on how a large scale ill-conditioned dense linear system can be solved.
A similar question is posed in [10] where the authors explore large linear systems resulting from applying MQ-RBFs to elliptic PDEs. The authors propose multiple methods which can also be used in combination with one another. Some of the proposed techniques are:

- Use preconditioners. Goal: construct a linear system with lower condition number
- Domain decomposition. Goal: divide domain into smaller chunks and obtain smaller matrices on the subdomains which potentially are easier to invert.
- Cutoff radius. Goal: introduce cutoff radius to the basis functions and sparcify the matrix $\hat{A}$.
- Change parameters of the basis functions. Goal: obtain better conditioned matrix $\hat{A}$ with more distinct rows.

The authors note regarding the cutoff radius that its introduction makes the basis functions no more global and it might affect convergence of the method. Therefore, the proposed solution is to pose the tradeoff between the accuracy and sparsity as an optimisation problem. This approach however is not further considered in this paper since the $tanh$ functions are designed to have global support.
The choice of the parameters of the basis functions is made by the Neural network and it is not a data agnostic choice.
As a result two methods of preconditioners and domain decomposition are investigated in this paper.

5.1 Preconditioner

As the authors from [1] and [7] if a system of linear equations is ill-conditioned, then it poses problems for both iterative and direct solvers. If a linear system of equations has a large condition number, then an iterative method might fail to converge or will take large number of steps until convergence. For direct

methods which are often based on some form of Gaussian elimination roundoff errors produced by finite precision arithmetics might result in very large deviation of the computed result from the the true solution effectively making it not useful. One idea could be to use a preconditioner on the system of linear equarions. As described in [1] the idea of preconditioners is to transform the system of linear equations $\hat{A}x = b$ to $M^{-1}\hat{A}x = M^{-1}b$. In that case the matrix M^{-1} is called preconditioner. The authors also discuss a tradeoff which arises when building a preconditioner. On one hand the resulting system should be not difficult to solve. On the other hand the construction of a precondition should not be expensive to construct and use. The last criterion is especially important if not just one linear system must be solved, but many as it is the case for BDF and Rosenbrock methods for each step of the algorithms.
As described in [1] there are two approaches by constructing a preconditioner. Namely a preconditioner can be problem-specific or purely algebraic. The authors note that problem-specific preconditioners can achieve a very good performance. Such preconditioners might use the information about the original PDE, its domain and boundary condition, also the discretisation might be taken into account. A weakness of such approach might be as described in [1] that such preconditioners might be very sensitive regarding the details of the problem and the details of the problem might be difficult to obtain and to use. Algebraic preconditioners, on the other hand, use on the information contained in the linear system itself. As a result, they can be applied to a broader class of problems compared to the problem-specific approaches. The drawback is that for a specific problem the perfomance of an algebraic preconditioner can be worse compared to a problem specific one.
In the following text multiple preconditioners are introduced ranging from purely algebraic to preconditioners which use problem specific information. Finally, techniques of combining preconditioners are introduced.

5.1.1 Jacobi and Gauss-Seidel preconditioners

Jacobi and Gauss-Seidel preconditioners have the advantage that they are easy to construct. A Jacobi preconditioner is defined as the diagonal of the matrix $\hat{A}$: $P = D$. The Gauss-Seidel preconditioner is defined as $P = D + L$ where D is the diagonal and L is the lower diagonal part of $\hat{A}$.

5.1.2 Deflation

As described in [7] in case of iterative methods the convergence might be very slow for systems with large condition number. Also there exists a connection between condition numbers and eigenvalues. As a result one idea could be to construct a preconditioner P which clusters together the eigenvalues of the matrix $\hat{A}$. This idea was used in the deflation projector P by Nicolaides [13]:

$$P\hat{A}x = Pb$$

with $P, \hat{A} \in \mathcal{C}^{NxN}$. The deflation projector P is defined as

$$P = I - \hat{A}Z(Z^H\hat{A}Z)^{-1}Z^H,\ Z \in \mathcal{C}^{Nxm}$$

with Z having rank m and $m << N$. If $\hat{A}$ is a diagonalisable matrix and $v_1, ..., v_n$ are eigenvectors corresponding to the eigenvalues $\lambda_1, .., \lambda_n$, then setting Z as $Z = [v_1, ..., v_n]$ will result in the same spectrum $\sigma(P\hat{A})$ as in case for the matrix $\hat{A}$ except for the eigenvalues $\lambda_1, ..., \lambda_n$ which will be shifted to zero. A construction of such preconditioner requires to have eigenvectors of the matrix $\hat{A}$ which might not be easy to obtain. In such case one idea could be to try to approximate the deflation subspace matrix Z.
One strategy of approximating Z is described in [7]. If $\{\lambda_1, ..., \lambda_k\}$ is a set of eigenvalues to be deflated and this set is enclosed by a circle Γ with center at the origin and radius r, then the matrix Z can be written as:

$$Z = \frac{1}{2\pi\sqrt{-1}} \oint_\Gamma (zI - \hat{A})^{-1} Y dz$$

with Y being a random matrix with columns randomly selected from the space Z_k spanned by the eigenvectors corresponding to the eigenvalues $\lambda_1, ..., \lambda_k$. Gaussian quadrature with q points can be used to approximate the integral leading to

$$Z \approx \sum_{i=1}^{q} w_i (z_i I - \hat{A})^{-1} Y$$

with w_i being quadrature weights and z_i being the points of Gaussian quadrature. As noted in [7] such construction will require solving q shifted linear systems $(z_i I - \hat{A})X = Y$.

5.1.3 Domain decomposition

Domain decomposition techniques rely on decomposing the domain into multiple subdomains. Subproblems are solved on the subdomains and then combined to a global solution. As described in [12] the domains can be overlapping or non- overlapping. The stength of the domain decomposition is that the global problem is divided into smaller ones which allows to use parallelisation and might lead to smaller and better conditioned matrices. The method also allows to pay more attention to local phenomena.
One approach to domain decomposition is the Schwarz framework belonging to a class of approaches designed for overlapping subdomains Ω_1^* and Ω_2^*. This is an iterative method. According to [12] for two subdomains the k-th iteration of the algorithm is defined as 4 steps: The first step is to solve the system for w_1^{k+1}

$$\begin{cases} Lw_1^{k+1} = f_1 & \text{in } \Omega_1^* \\ w_1^{k+1} = v^k & \text{on } B^1 \\ w_1^{k+1} = g & \text{on } B_{[1]} \end{cases} \tag{8}$$

then $v^{k+1/2}$ is set to

$$v^{(k+1/2)} = \begin{cases} w_1^{k+1} & \text{on } \Omega_1^* \\ v^k & \text{on } \Omega \setminus \Omega_1^* \end{cases} \tag{9}$$

The next step of the algorithm is to solve for w_2^{k+1} the system

$$\begin{cases} Lw_2^{k+1} = f_2 & \text{in } \Omega_2^* \\ w_2^{k+1} = g & \text{on } B_{[2]} \\ w_2^{k+1} = v^{k+1/2} & \text{on } B^2 \end{cases} \tag{10}$$

eventually v^{k+1} is defined as

$$v^{k+1} = \begin{cases} w_2^{k+1} & \text{on } \Omega_2^* \\ v^{k+1/2} & \text{on } \Omega \setminus \Omega_2^* \end{cases} \tag{11}$$

$Lw_2^{k+1} = f_2$ and Lw_1^{k+1} denote the original PDE equation. This equation is solved on the smaller subdomains. $B_{[i]}$ describe the domain boundaries. Each subdomain some boundaries of which coincide with the domain boundaries needs to take into consideration the defined boundaries conditions of the original PDE. B^i are the boundaries between subdomains.
One difficulty of the domain decomposition approach is how to put together the solutions computed on different subdomains into one solution. Often properties of the solution like global smoothness are desired. That means, that at the boundaries of subdomains the computed solutions need to be connected. One additional difficulty described in [12] is that only neighboring subdomains directly transfer the information to one another, which means that globally the information propagation might be rather slow.
There exists as shown in [12] a matrix formulation for the Schwarz algorithm allowing to treat it as a form of preconditioning. On the subdomains spectral method together with Sampling weights of deep neural networks can be used. In that case the spectral method will be used to approximate local solutions and then the global solution will be assembled from the local ones. Connecting different solutions might be

difficult in that case.

5.1.4 Multigrid Methods

According to [19] and [18] there are algebraic and geometric multigrid methods.
Geometric multigrid does not use subdomains to approximate the solution of a PDE, but rather it uses coarser grids. The support of the coarse grid is global and covers the original domain. As described in [18] the advantage of multigrid methods is that they might remove smooth error components which otherwise might need many iterations of iterative linear systems solvers to disappear. As the authors in [18] describe, going to coarser grid might make low frequency error components to become high frequency. High frequency errors can then be removed in the coarse space and that information will be transferred back to the fine grid space to update the current solution vector of the linear system. As a result, fewer steps of the iterative solver might be needed.
Two key components of the geometric multigrid procedure are the matrix transferring the information from fine to coarse space, the restriction matrix R, and the matrix transferring the information from the coarse to the fine grid, the interpolation matrix I. Given the restriction and interpolation matrices one can represent the original matrix $\hat{A}_h$ in the coarse grid as $\hat{A}_{2h} = R\hat{A}I$. Then one can define a two-grid V cycle [18] as follows: .

Step 1: Iterate on the fine grid $\hat{A}_h u = b$
Step 2: Compute the residual $r_h = b_h - \hat{A}_h u_h$ and transfer it to the coarse grid by $r_{2h} = Rr_h$
Step 3: Iterate the system $\hat{A}_{2h} E_{2h} = r_{2h}$
Step 4: Transfer E_{2h} to the fine grid space by applying $E_h = IE_{2h}$ and update current solution u by setting $u := u + E_h$
Step 5: Iterate $\hat{A}_h u = b$ starting from the updated solution u

Algorithm 1: Multigrid procedure

Multiple V-cycles can be combined together as shown in [18]. One way to define the matrix R described in [18] is to build a weighted average among the neighbors of a grid node and the node itself. Then this value replaces the node and its neighbors in the coarse grid. It is a similar idea to pooling in CNNs. The authors also suggest to use interpolation to reconstruct missing fine grid values from the coarse grid. Based on this idea the matrix I can be constructed. One potential difficulty of applying geometrical multigrid to the Spectral method with Sampling weights of neural networks is how to define a grid. The linear ODE matrix from the system (3) consists of the elements $a_{ij} =< \psi_i, \psi_j >$ and each basis function ψ_i needs two points sampled from the domain to be defined.
An alternative to the geometric multigrid is an algebraic multigrid which does not assume an existence of the geometric grid. As described in [18] three components from the geometric multigrid need to be formulated for the linear system $\hat{A}x = b$: smooth vectors, connected nodes and coarse subset of nodes. The authors suggest defining smooth vectors as the vectors for which the norms of u and $\hat{A}u$ are similar. Connected nodes are selected based on the entries of the matrix $\hat{A}$. Namely, if the entry $\hat{a}_{ij}$ has significant magnitude, then the nodes i and j are thought as connected, neighboring, nodes. Coarse subset of nodes can be defined with help of the heuristics: if the entry $\hat{a}_{ij}$ is large, then the node i or node j should be in the coarse set C, but not both since the nodes are neighbors. If a node i is not in C, then its neighbor j should be in C unless j itself has neighbors which are in C.

5.1.5 Combined preconditioners

In this chapter methods are describd which allow to combine existing preconditioners into a new one, a multi-level preconditioner. According to [20] two SPD preconditioners C_1 and C_2 can be additively combined into an SPD preconditioner P_{a2}:

$$P_{a2} = C_1 + C_2$$

This formula can be generalised for more SPD preconditioners C_1, C_2, C_3 ... C_n. Also multiplicative combination of the preconditioners is possible. For two SPD preconditioners C_1 and C_2 the multiplicative operator can be defined as:

$$P_{m2} = C_1 + C_2 - C_2\hat{A}C_1$$

Similarly to the additive case the multiplicative scheme can be generalised for more SPD preconditioners C_1,...,C_n. For the case of three matrices the combination will lead to

$$P_{m3} = C_1 + C_2 + C_3 - C_2\hat{A}C_1 - C_3\hat{A}C_2 - C_3\hat{A}C_1 + C_3\hat{A}C_2\hat{A}C_1$$

Further the authors note that for a linear system $\mathcal{P}\mathcal{A}x = b$ where $\mathcal{P}$ is a traditional preconditioner, where the examples of traditional preconditioners among others are according to [20] diagonal scaling methods, approximate inverse and incomplete Cholesky preconditioners, the matrix $\mathcal{A}$ can further be a combination of the linear system matrix $\hat{A}$ and a preconditioner P. According to the authors the use of the second preconditioner $\mathcal{P}$ might improve the converge of iterative solvers.
The authors in [20] describe multiple examples, how the techniques described above can be combined with deflation, multigrid and domain decomposition methods. One interesting property described by the authors is that from the algebraic point of view the two-level preconditioned conjugated gradient methods resulting from the fields of deflation, multigrid and domain decomposition show similarity and in some cases even equivalence.

6 Conclusion

In the project numerical solution of the second order wave equation via Galerkin spectral method with one layer Neural Networks as basis functions is investigated. The basis functions have a global support. The Neural Networks have hyperbolic $tanh$ activation function. Weights and biases are determined with help of the Sampling weights for Neural Networks algorithm defined in [4]. The weights $a_k(t)$ lead to a system of linear ordinary differential equations. Two families of methods for stiff ODEs are introduced. Both families will eventually lead to linear systems of equations which need to be solved in order to numerically compute the solution at the next time step. Since the matrix $\hat{A}$ in the obtained linear system $\hat{A}x = b$ might be ill-conditioned, preconditioners are introduced which might transform the linear system in a system which is easier to solve.
Since the linear ODE problem might be stiff the two families introduced are designed for stiff ODEs. The choice between linear- implicit Runge-Kutta methods and BDF methods might pose a tradeoff. On one hand, linear- implicit Runge- Kutta methods can achieve higher convergence order compared to BDF methods which are restricted to $k \leq 6$ to ensure stability [5]. But on the other hand, linear- implicit Runge- Kutta might have a higher computational cost.
The choice of preconditioner or a combination of preconditioners might also lead to a tradeoff between quality of the preconditioner and the its construction cost. The authors in [20] look at this tradeoff for different preconditioners. The authors note, that traditional preconditioners, an example of which is the Jacobi preconditioner, might be cheap to construct, however, they might lead to a poorer performance compared to deflation or multigrid two-level preconditioners. The authors suggest using a traditional preconidtioner as one of the building blocks for a two-level preconidtioner. For example, combining it with deflation. According to the authors deflation preconditioner is expected to show better performance in context of iterative solvers compared to abstract balancing and additive coarse- grid correction methods. However, this would require to solve accurately the second level correction which might be expensive. Additionally, the approximation of the matrix Z consisting of eigenvectors as shown in [7] might lead to multiple systems of linear equations to be solved. As a result, it might be expensive in practice to use the deflation preconditioner.
The authors in [19] investigate performance of an algebraic multigrid preconditioners. According to the authors an algebraic multigrid preconidtioner might achieve good performance, especially compared to one-level preconditioners like ILU, by trying to reduce both short range and long range error components. However, as the authors describe before applying the preconditioner the coarse grid must be constructed.

The size of the coarse set might have a large impact on computational effort with larger sets potentially resulting in the increased computational time. Ideally the coarse system should not be expensive to solve, however it should contain enough information to improve the convergence rate. This might also lead to a tradeoff when constructing the coarse set of nodes. The algorithm also requires to define an interpolation scheme. Several potential examples are discussed in [19].
As a result, one could try to apply to the system (3) BDF methods together with an algebraic multigrid preconditioner combined with Jacobi or Gauss- Seidel preconditioner. If more expensive computational time can be afforded, then a linear- implicit Runge- Kutta method or a deflation preconditioner combined with Jacobi or Gauss- Seidel preconditioner might lead to improved results.

7 Future work

Potential future work might construct different combinations of preconditioners. As discussed in the Chapter "Combined preconditioners" one can combine different preconditioners together. Then one could investigate the efficiency versus computational cost for the derived preconditioners.
One other interesting direction of future work might be to look at different choices of sampled basis functions for the spectral method. In this project one- layer Neural Networks with $tanh$ activation functions were considered. One other choice could be to construct deeper Neural Networks or to use a different activation function. Another idea might be to use sampled basis functions in combination with local Galerkin methods, for example, with the finite element method. Such methods would require basis functions with local support. As a result, the matrix entries $a_{ij} = \int_\Omega \psi_i \psi_j dx$ might often be zero leading to sparse matrices. Sparse matrices might result in reduced computational costs. However, the one advantage of the spectral method is that it might converge exponentially fast. Also local Galerkin methods might require a high number of basis functions to show good performance.
The distribution from which the points x_i and x_j are sampled when constructing the basis functions might be of further interest. One idea might be to try to use problem specific distributions by sampling the points. The hope is that such distributions might lead to a better choice of the weights and biases and, consequently, to a better choice of basis functions for the spectral method.

References

[1] Michele Benzi. “Preconditioning techniques for large linear systems: a survey”. In: *Journal of computational Physics* 182.2 (2002), pp. 418–477.

[2] Per Niklas Benzler Waaler. “Strategies for computing the condition number of a matrix”. In: *Bachelor’s Theses in Mathematical Sciences* (2017).

[3] Christopher M Bishop. “Pattern recognition and machine learning”. In: *Springer google schola* 2 (2006), pp. 645–678.

[4] Erik L Bolager et al. “Sampling weights of deep neural networks”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[5] Peter Deuflhard and Folkmar A Bornemann. “Numerische Mathematik. II”. In: (1994).

[6] Peter Deuflhard and Martin Weiser. *Numerische Mathematik 3: Adaptive Lösung partieller Differentialgleichungen*. Walter de Gruyter GmbH & Co KG, 2020.

[7] Craig C Douglas, Long Lee, and Man-Chung Yeung. “On solving ill conditioned linear systems”. In: *Procedia Computer Science* 80 (2016), pp. 941–950.

[8] RD Grigorieff and J Schroll. “On a (α)-stable methods with high order of consistency”. In: *Computing* 20 (1978), pp. 343–350.

[9] Ernst Hairer and Gerhard Wanner. “On the instability of the BDF formulas”. In: *SIAM journal on numerical analysis* 20.6 (1983), pp. 1206–1209.

[10] EJ Kansa and YC Hon. “Circumventing the ill-conditioning problem with multiquadric radial basis functions: applications to elliptic partial differential equations”. In: *Computers & Mathematics with applications* 39.7-8 (2000), pp. 123–137.

[11] Jan Lunze. “Eigenschaften von linearen DAE-Systemen”. In: *at-Automatisierungstechnik* 64.2 (2016), pp. 81–95.

[12] Tarek PA Mathew. *Domain decomposition methods for the numerical solution of partial differential equations*. Springer, 2008.

[13] Roy A Nicolaides. “Deflation of conjugate gradients with applications to boundary value problems”. In: *SIAM Journal on Numerical Analysis* 24.2 (1987), pp. 355–365.

[14] Martin Otter and Hilding Elmqvist. “Transformation of differential algebraic array equations to index one form”. In: *Proceedings of the 12th International Modelica Conference*. Linköping University Electronic Press. 2017.

[15] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[16] Lawrence F Shampine and Mark W Reichelt. “The matlab ode suite”. In: *SIAM journal on scientific computing* 18.1 (1997), pp. 1–22.

[17] Mazi Shirvani and Joseph WH So. “Solutions of linear differential algebraic equations”. In: *SIAM review* 40.2 (1998), pp. 344–346.

[18] Gilbert Strang. *Multigrid Methods*. 2006. URL: https://math.mit.edu/classes/18.086/2006/am63.pdf.

[19] Klaus Stüben. “A review of algebraic multigrid”. In: *Numerical Analysis: Historical Developments in the 20th Century* (2001), pp. 331–359.

[20] Jok Man Tang et al. “Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods”. In: *Journal of scientific computing* 39 (2009), pp. 340–370.