Computer Science

Mathematics

Towards a Compiler for Partial Differential Equations for Analog Computers

Dirck van den Ende

Supervisors:
Henning Basold (LIACS) & Arjen Doelman (MI)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

Mathematical Institute (MI)
www.universiteitleiden.nl/en/science/mathematics

05/07/2023

Abstract

Analog computers can be a fast and energy-efficient way of simulating ordinary differential equations. Simulating partial differential equations requires more work. One method is approximate all but one of the dimensions in the PDE with a grid. This yields a large system of ODEs of the form $\dot{x} = f(t, x)$. Analog computers can solve systems of this form up to a given size, depending on the number of components available in the analog computer. To be able to solve larger systems of ODEs, the system $\dot{x} = f(t, x)$ can be split up into smaller groups of equations, which may depend on each other. To prevent having to solve all smaller systems at once, an iterative method is used. When a value from a different group is needed, the value from the previous iteration is used.

In this thesis, a PDE-to-ODE compiler (PTOC) is introduced, which automatically converts systems of PDEs into iterative systems of ODEs. Furthermore, it is proven that the iterative method described above converges locally to the solution of the original system of ODEs. When f has the additional property that it is Lipschitz continuous, the iterative method converges globally to a solution. Lastly, a heuristic for dividing the system of ODEs into groups is introduced, which aims to reduce the amount of data that needs to be stored by the analog computer. The application of these techniques is implemented in the PTOC tool.

Contents

Chapter 1

Introduction

In recent years there has been a renewed interest in analog computation. With the introduction of electronic analog computing, which we will focus on in this thesis, devices have gotten smaller and more energy-efficient at simulating ordinary differential equations than their digital counterparts. Recent analog computers have the ability to be reconfigured digitally, allowing a broader class of problems to be solved. An example of such a "hybrid computer" is the Anadigm AN120E04 Reconfigurable FPAA, which can be controlled and reconfigured with electrical signals, sent from another device [4]. The efficiency with which analog computers can solve ODEs raises the question if they can also be used to simulate partial differential equations.

It has been proven that a general-purpose analog computer (GPAC), which is a mathematical abstraction of an analog computer, cannot solve all PDEs. In particular, it is shown that the Dirichlet problem for Laplace's equation on a disc cannot be solved using a GPAC [18]. For this reason, a new mathematical concept called the extended analog computer was introduced. This is an abstract machine that can solve arbitrary PDEs with very few limitations. One problem with this abstract machine is that it may be too general, and therefore not physically realizable [17].

To still be able to simulate PDEs using an analog computer, a discretization of the spatial dimensions using a grid can be used to approximate the PDE with a large system of ODEs [21]. For example, the one-dimensional heat equation $\partial_t u = \partial_{xx} u$ can be approximated with a collection of functions $u_0, \ldots, u_n$, which each solve an ODE $\dot{u}_i = u_{i-1} - 2u_i + u_{i+1}$ for $i \in \{1, \ldots, n-1\}$, where u_0 and u_n are given boundary conditions. Here n corresponds with the precision of the approximation, a higher value of n means a finer approximation of the spatial dimension. In general a system of PDEs of the form

$$\frac{\partial u_i}{\partial t} = F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \ldots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}\right) \tag{1.1}$$

where the derivatives on the right are only dependent on the spatial dimensions and not on time, can be approximated with a system of ODEs of the form $\dot{x} = f(t, x)$.

Because the resulting system of ODEs can be large, the equations may have to be split up into groups. The reason for this is the limited amount of components that an analog computer can use at once. An iterative method was introduced by Bekey and Ung [21] where we start with an initial guess, and use the guess from the previous iteration when a value is needed from an equation that is outside the current group. This iterative method is described by equations of the form $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$. The iterative method should eventually converge to a solution of $\dot{x} = f(t, x, x)$.

1.1 Problems

Existing compilers for ODEs targeting analog computers, like Legno [2], can simulate multi-dimensional systems of the form $\dot{x} = f(t, x)$. In the case of the heat equation $\partial_t u = \partial_{xx} u$, an ODE of this form is produced when discretization is applied to the spatial dimension. This is not the case, however, for some other PDEs, like the wave equation $\partial_{tt} u = \partial_{xx} u + \partial_{yy} u$, which features a higher-order time derivative. In this thesis, the different possible forms of input PDEs are explored.

The iterative method that is introduced when splitting the output system of ODEs into groups, may not converge to the true solution of the system of ODEs. In particular, it is useful to know if the iterative method converges if the function f is Lipschitz continuous or continuously differentiable. A function $f : \Omega \to \mathbb{R}^m$ is defined to be Lipschitz continuous if there exists some Lipschitz constant $L \geq 0$ such that for all $x, y \in \Omega$, it holds that $||f(x) - f(y)|| \leq L||x - y||$. This is stronger than the requirement of continuous differentiability when the domain Ω is not compact. This distinction is of particular interest as the domain Ω is often $\mathbb{R}^n$, where n is the number of grid points used in the approximation of the PDE.

The division of the system of ODEs into groups also introduces a new problem for performance, namely that the values from the previous iteration need to be stored. Only the values of the grid cells at the boundary of the grid groups need to be stored between iterations, since others are not referenced by grid cells from other groups. For this reason, Bekey and Ung used grid groups of a certain shape to reduce the number of values that need to be stored between iterations [21].

1.2 Contributions

In this thesis, the theoretical aspects of the conversion from a system of PDEs to a system of ODEs and the iterative method are discussed. Proofs are provided for circumstances under which the iterative method always converges (locally) to a solution. In particular, the iterative method converges locally on some time interval $[0, T] \subset [0, \tau]$ when f is continuously differentiable, and globally on the interval of definition $[0, \tau]$ when f is (globally) Lipschitz continuous. A PDE-to-ODE compiler (PTOC), was developed to implement the theoretical findings for converting a system of PDEs to a system of ODEs. PTOC aims to build on the concept of the Legno compiler by introducing automatic conversion from a system of PDEs to an iterative system of ODEs in a format similar to that used in Legno. PTOC automatically converts a system of PDEs of the form

$$\frac{\partial^k u_i}{\partial t^k} = F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \ldots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}\right)$$

to a system of PDEs of the form (1.1), with a single time derivative on the left. This system of PDEs is then approximated with a system of ODEs. PTOC automatically determines a grid division using a heuristic that aims to minimize the number of cells that need to be stored between iterations. It then generates the resulting iterative ODEs, for a given amount of iterations. PTOC also includes a tool that can simulate the generated systems of ODEs and visualize the result. The PTOC tool is available at [9].

This thesis aims to bring together the theoretical and practical aspects of simulating PDEs on analog computers. The PTOC tool should serve as an entry point for a compiler for PDEs targeting analog computers.

1.3 Thesis overview

In Chapter 2, it is shown that PDEs like the wave equation can be converted to a system of PDEs in which each PDE is of the form (1.1). In Chapter 3, several heuristics are discussed that attempt to minimize the number of values that need to be stored between iterations. The most important heuristic is the optimal hyperrectangle heuristic, which determines a hyperrectangle that maximizes the ratio between the volume without and with the boundary included. In Chapter 4, the iterative method is formalized and its local convergence to a solution of the original system of ODEs is shown when f is continuously differentiable. For f globally Lipschitz continuous, it is shown that this convergence is not just local, but global. The effectiveness of the iterative method for estimating a solution of an ODE is also discussed. In Chapter 5, the usage and implementation of PTOC are discussed.

1.4 Preliminaries

For this thesis, we assume the reader has basic knowledge of solving ordinary differential equations, continuous dynamics, and the Picard-Lindelöf theorem. For more information on these topics, see [20]. It is expected that users of the PTOC tool have a basic understanding of the use of the command-line interface, and have access to a machine running Linux.

Chapter 2

Forms of the input PDE

An analog computer can generally solve systems of ODEs that are of the form $\dot{x} = f(t, x)$. Therefore, when approximating a PDE with a system of ODEs, it is useful to have a PDE of a specific form, such that the resulting ODE will be of the form $\dot{x} = f(t, x)$.

Also, by the Picard-Lindelöf Theorem [20, Theorem 2.2], there exists a unique solution of this system in the neighborhood of an initial condition if f is locally Lipschitz continuous. This can be beneficial to avoid inconsistent results from the analog computer. If the Picard-Lindelöf theorem cannot be applied, an ODE may have multiple solutions. An example of this is $\dot{x} = \sqrt{x}$, which is not locally Lipschitz continuous at $x = 0$. This equation has two solutions for the initial condition $x(0) = 0$, namely $x(t) = 0$ and $x(t) = \frac{1}{4}t^2$.

In this chapter, the different possible forms for the input system of PDEs are discussed. This theory is applied in PTOC by first converting the input PDE to a standard form, and then approximating it with an ODE.

2.1 The standard form

The form of the systems of PDEs that will be approximated with an ODE of the form $\dot{x} = f(t, x)$ is

$$\frac{\partial u_i}{\partial t} = F_i \left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \ldots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}} \right), \tag{2.1}$$

with $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_{\geq 0}$ multi-indices over $x = (x_1, \ldots, x_n)$. The conversion process is described in more detail in Chapters 4 and 5. We consider the temporal and spatial dimensions to be given explicitly, meaning that a single spatial derivative on the left means the equation is not considered to be in the correct form. When a PDE represents a physical process, this property is often considered implicit. We will call a PDE of the above form to be a PDE in standard form.

Example 2.1. The heat equation is in standard form:

$$\frac{\partial u}{\partial t} = \Delta u =: \frac{\partial^2 u}{\partial x_1^2} + \cdots + \frac{\partial^2 u}{\partial x_n^2}.$$

Example 2.2. The wave equation is not in standard form:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \Delta u.$$

However, the wave equation can be rewritten to be in standard form by introducing an auxiliary variable, as we will show in the following section.

2.2 Conversion to standard form

Differential equations like the wave equation can be converted to a system of PDEs that is in standard form by splitting the equation up into multiple parts. To formalize this, we introduce the concept of a PDE being reducible to another.

Definition 2.3. A system of PDEs (1) is defined to be **reducible** to another system of PDEs (2) with the same dimensions if for the set of solutions $S_1 \subset \mathrm{Map}(\mathbb{R}^k, \mathbb{R}^m)$ of (1) and $S_2 \subset \mathrm{Map}(\mathbb{R}^k, \mathbb{R}^n)$ of (2) with $m \leq n$, there exists a projection $\rho : \mathbb{R}^n \to \mathbb{R}^m$ (a linear map such that $\rho \circ i = \mathrm{id}_{\mathbb{R}^m}$, where $i : \mathbb{R}^m \to \mathbb{R}^n$ is the inclusion map) such that the map

$$\begin{aligned} \rho_* : S_2 &\to S_1, \\ x_2 &\mapsto \rho \circ x_2. \end{aligned}$$

is well-defined and surjective, i.e. for any solution x_2 of (2), $\rho \circ x_2$ is a solution of (1), and for any solution x_1 of (1) there exists a solution x_2 of (2) with $x_1 = \rho \circ x_2$.

What this definition means is that when looking at a solution of a PDE with fewer equations (1), it can be extended to a solution of a PDE with more equations (2). This definition ensures that any solution of PDE (2) can be directly translated to a solution of PDE (1), which is a property that can be used by PTOC to convert PDEs to a standard form. The following example shows how higher-order derivatives can be eliminated using this definition.

Example 2.4. The Dirichlet equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0,$$

with certain conditions on u, is reducible to the system of PDEs

$$\begin{aligned} \frac{\partial v}{\partial x} + \frac{\partial w}{\partial y} &= 0, \\ \frac{\partial u}{\partial x} &= v, \\ \frac{\partial u}{\partial y} &= w. \end{aligned}$$

with the same conditions on u, and no conditions on v and w. If (u, v, w) is a solution to the above system, then by substitution of $\partial_x v$ and $\partial_y w$ for $\partial_{xx} u$ and $\partial_{yy} u$ respectively, u is a solution of the

Dirichlet equation. Also, if u is a solution of the Dirichlet equation, then taking $v = \partial_x u$ and $w = \partial_y u$ gives a solution to the above system.

Note that the conditions given on u may also be replaced with conditions on v and w. For example, when a specific expression is given for $\partial_x u$ on (part of) the boundary of the domain $\partial\Omega$, it could be replaced with an expression for v on $\partial\Omega$, since $v = \partial_x u$. In some cases, this will be automated by PTOC, which is discussed in Section 2.3.

It is more beneficial to eliminate higher-order time derivatives than higher-order spatial derivatives, since we want to rewrite the PDE to the form (2.1). We formalize this in the following definition, where we combine the definitions of the standard form and reducibility.

Definition 2.5. A system of PDEs is **reducible to standard form** if it is reducible to a system of PDEs where each PDE is of the form

$$\frac{\partial u_i}{\partial t} = F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \ldots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}\right),$$

where the derivatives on the right do not depend on t.

This definition will ensure that the PDE we try to solve can be converted into a PDE that is in standard form, which in turn results in an ODE of the form $\dot{x} = f(t, x)$. The following lemma yields a broad class of PDEs that can be reduced to standard form.

Lemma 2.6. *Let $F_i : \mathbb{R}^{1+N+n+m} \to \mathbb{R}^m$ be functions such that for any $t \in \mathbb{R}$, $x \in \mathbb{R}^N$ and $(y_1, \ldots, y_n, x_1, \ldots, x_{m-1}) \in \mathbb{R}^{n+m-1}$, the map $f_i[t, x, y_1, \ldots, y_n, x_1, \ldots, x_{m-1}] : \mathbb{R} \to \mathbb{R}$ defined by $f_i[t, x, y_1, \ldots, y_n, x_1, \ldots, x_{m-1}](a) = F_i(t, x, y_1, \ldots, y_n, x_1, \ldots, x_{m_1}, a)$ is bijective. Then the system of PDEs*

$$F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \ldots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}, \frac{\partial u_i}{\partial t}, \ldots, \frac{\partial^m u_i}{\partial t^m}\right) = 0$$

is reducible to standard form.

Proof. Consider the system PDEs

$$\begin{aligned}
\frac{\partial u_{i,m-1}}{\partial t} &= f_i\left[t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \ldots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}, u_{i,1}, \ldots, u_{i,m-1}\right]^{-1}(0), \\
\frac{\partial u_i}{\partial t} &= u_{i,1}, \\
\frac{\partial u_{i,1}}{\partial t} &= u_{i,2}, \\
\vdots \quad & \quad \vdots \\
\frac{\partial u_{i,m-2}}{\partial t} &= u_{i,m-1}.
\end{aligned}$$

Suppose $(u_i, u_{i,1}, \ldots, u_{i,m-1})$ is a solution of the above system. Then we can substitute back to get $u_{i,k} = \partial_t^k u_i$. The first equation then gives back the original PDE, which means u_i is a solution to the original PDE. Likewise if u_i is a solution to the original system, writing $u_{i,k} = \partial_t^k u_i$ for all $k \in \{1, \ldots, m-1\}$ gives that $(u_i, u_{i,1}, \ldots, u_{i,m-1})$ is a solution of the above system. □

Example 2.7. The wave equation from Example 2.2 is reducible to standard form. We have

$$F\left(t, x, \frac{\partial^2 u}{\partial x_1^2}, \dots, \frac{\partial^2 u}{\partial x_n^2}, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial t^2}\right) = \frac{\partial^2 u}{\partial t^2} - c^2 \Delta u.$$

This expression is bijective in $\partial_{tt} u$. Hence by Lemma 2.6 the wave equation is reducible to standard form.

Remark 2.8. By Lemma 2.6, any system of PDEs of the form

$$\frac{\partial^m u_i}{\partial t^m} = F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \dots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}, \frac{\partial u_i}{\partial t}, \dots, \frac{\partial^{m-1} u_i}{\partial t^{m-1}}\right), \tag{2.2}$$

is reducible to standard form. Each PDE can be rewritten to

$$\begin{aligned}
\frac{\partial u_{i,m-1}}{\partial t} &= F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \dots, \frac{\partial^{\alpha_n} u_{j_n}}{\partial x^{\alpha_n}}, u_{i,1}, \dots, u_{i,m-1}\right), \\
\frac{\partial u_i}{\partial t} &= u_{i,1}, \\
\frac{\partial u_{i,1}}{\partial t} &= u_{i,2}, \\
\vdots \quad & \quad \vdots \\
\frac{\partial u_{i,m-2}}{\partial t} &= u_{i,m-1}.
\end{aligned} \tag{2.3}$$

This remark covers both examples discussed before. There are examples of PDEs that are not reducible to standard form, as the following example shows.

Example 2.9. The following PDE is not reducible to standard form:

$$\frac{\partial u}{\partial t}\frac{\partial u}{\partial x} = 1.$$

As the function F is not invertible when $\partial_x u = 0$, Lemma 2.6 cannot be applied. The only way this can be rewritten is by taking $\partial_t u = 1/\partial_x u$. However, then the solutions of the form $u(t, x) = (\tilde{u}(t), 0)$ are not generated by this new PDE.

We can conclude that not all PDEs are reducible to standard form. However, many PDEs used in practice, like the heat equation and wave equation, are reducible to standard form.

2.3 Boundary and initial conditions

In the context of compilers, when converting a PDE to the standard form, the initial and boundary conditions also have to be transformed. We will look at systems of the form (2.2), which are rewritten to (2.3). For any boundary condition $(\partial_x^\alpha u)|_{\partial\Omega}(t, x) = g(t, x)$, we have a boundary condition on u_i:

$$\left.\left(\frac{\partial^\alpha u_i}{\partial x^\alpha}\right)\right|_{\partial\Omega} = \left.\left(\frac{\partial^\alpha \left(\frac{\partial^i u}{\partial t^i}\right)}{\partial x^\alpha}\right)\right|_{\partial\Omega} = \frac{\partial^i}{\partial t^i}\left.\left(\frac{\partial^\alpha u}{\partial x^\alpha}\right)\right|_{\partial\Omega} = \frac{\partial^i g}{\partial t^i}$$

If g is given by the user, then this derivative can be determined at compile time. In PTOC boundary conditions are implemented differently, where the user does not give the values of the spatial derivatives at the border. Instead, the user gives a single function that represents the values of u anywhere outside the domain. Then the PDE compiler samples from this function if a value from outside the domain is needed for an approximation of a spatial derivative. In this case, the same method can still be used to determine the time derivatives. This way the spatial derivatives at the boundary are given implicitly by this boundary function. In the future, this feature should be changed such that specific Neumann boundary conditions can be given, instead of a general function. These boundary conditions could then be used to determine the values of grid cells outside the domain. Note that any partial spatial derivatives of at most order two per variable, so $\partial_x^\alpha u$ with $\alpha \in \{0, 1, 2\}^n$, can be approximated at the entire domain without any additional boundary conditions, other than the value of u on the boundary. In this thesis, the main focus will be on PDEs of this type.

When a PDE with a higher-order time derivative is given, extra initial conditions need to be given by the user. Specifically, if a PDE of the form (2.2) is given by the user, then initial conditions for $u, \partial_t u, \dots, \partial_t^{m-1} u$ need to be given. These will then be the initial conditions for $u, u_1, \dots, u_{m-1}$ respectively.

Chapter 3

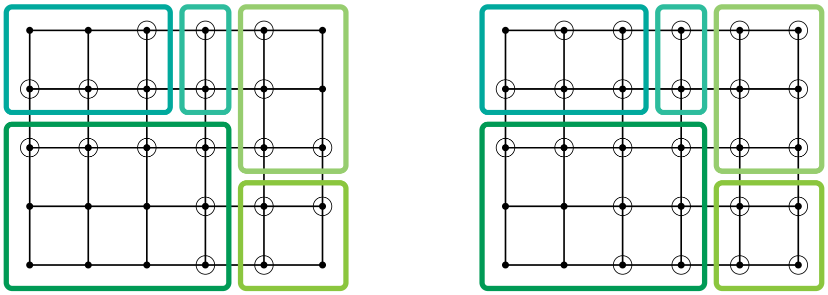
Division of grid groups

When generating a system of ODEs that approximates a system of PDEs, the number of equations that need to be simulated simultaneously can be very large. This can cause a problem when the number of components inside an analog computer is limited. For example, the Anadigm AN120E04 Reconfigurable FPAA has four configurable analog blocks (CABs), which each have limitations on the configurations that can be applied to them [4].

This problem is solved by dividing the grid into groups. The grid cells in each group will be simulated simultaneously. With this, a new problem arises. Namely, to determine the values at certain grid cells, values from grid cells from other groups are needed. This is solved by working with iterations and improving on an initial guess of the solution of the ODE. This is described in more detail in Chapter 4.

When dividing the grid into groups, with some cells depending on values (from a previous iteration) from other groups, the values of cells need to be stored between iterations. Not all cell values need to be stored, however. For example, if each cell only needs to use the values from the eight adjacent cells in a 2D grid, then only the boundary cells need to be stored. This is illustrated in Figure 3.1a. A dependency like this is present when simulating the 2D heat equation. The range of dependencies can also be more than one cell and can differ between dimensions. This is illustrated for a range of $(2, 1)$ in Figure 3.1b. With range we mean the width of the border of each group in each dimension, where the border consists of all cells for which the values need to be stored between iterations.

A question that arises is the following: What is a good way to divide an n-dimensional grid into groups of a maximum size, such that the number of cell values that need to be stored between iterations, is minimized? A problem that is related to this is the k-cut problem, which asks for the minimum total edge weight that needs to be removed in an arbitrary weighted graph to create k separated graph components. Solving this problem is NP-hard in general [8]. In this chapter, several heuristics will be discussed to divide the grid. Note that this problem does not take into account that there are other physical limitations, that may require the groups to have a certain shape or size.



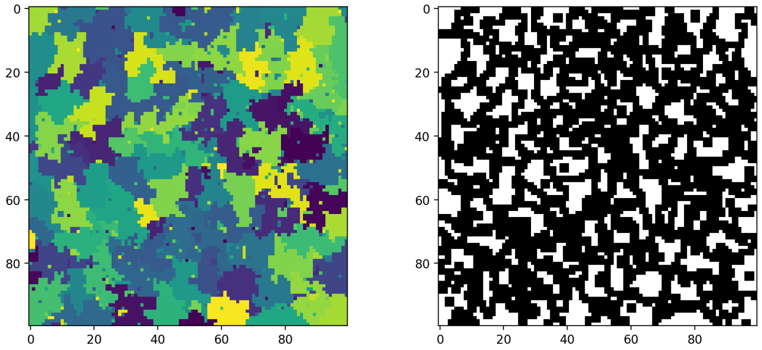
(a) Range $(1, 1)$ (b) Range $(2, 1)$

Figure 3.1: Example of the division of a 2D grid, where the grid cells for which the values need to be stored are marked.

3.1 Random base heuristic

As a base heuristic, a randomized algorithm is used. The algorithm repeatedly picks two random neighboring cells and tries to join their groups. If the cells are already in the same group nothing happens. If the groups cannot be joined because the resulting group would be too large, a failure counter is incremented. When the failure counter reaches some predetermined value, the algorithm halts.

Figure 3.2 shows a visualization of the group division of the random algorithm, as well as the border cells whose values have to be stored between iterations.

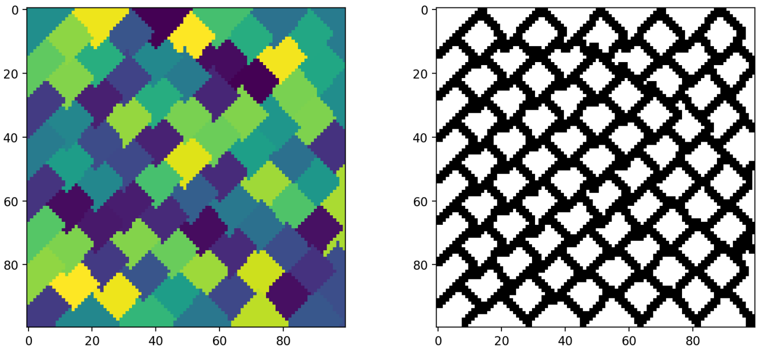


(a) Groups (b) Borders (black)

Figure 3.2: Visualization of a possible group division for the random base heuristic on a 100×100 grid with a maximum group size of 120 and range $(1, 1)$. Interior cells: 33.0%.

3.2 Spread heuristic

An improvement on the random heuristic is to instead spread in all directions evenly until the maximum group size is reached. This is implemented by using a Breadth-First-Search starting at the first node that has not been added to a group yet. The algorithm only spreads to cells that have not been added to a group yet. Figure 3.3 shows a visualization of this heuristic.



(a) Groups

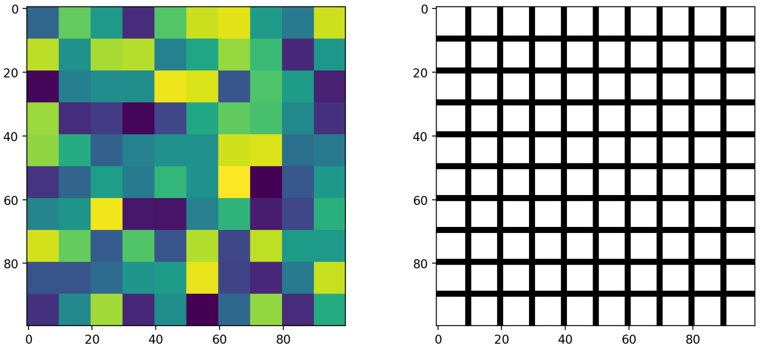
(b) Borders (black)

Figure 3.3: Visualization of a possible group division for the spread heuristic on a 100 × 100 grid with a maximum group size of 120 and range $(1, 1)$. Interior cells: 55.5%.

3.3 Hypercube heuristic

This heuristic divides the grid into hypercubes that are of maximal size, without exceeding the group size limit. At the edge of the grid, hypercubes are cut off if the dimensions of the hypercube do not divide the dimensions of the grid. Figure 3.4 shows a visualization of the hypercube heuristic in two dimensions. It is clear that the number of shared components is much smaller than for the random heuristic.

Note that this heuristic does not take into account that multiple dimensions can have different border widths, which can impact the performance of the heuristic. This is not shown in Figure 3.4, since the range used is $(1, 1)$, which means the dependency range is symmetric between dimensions.



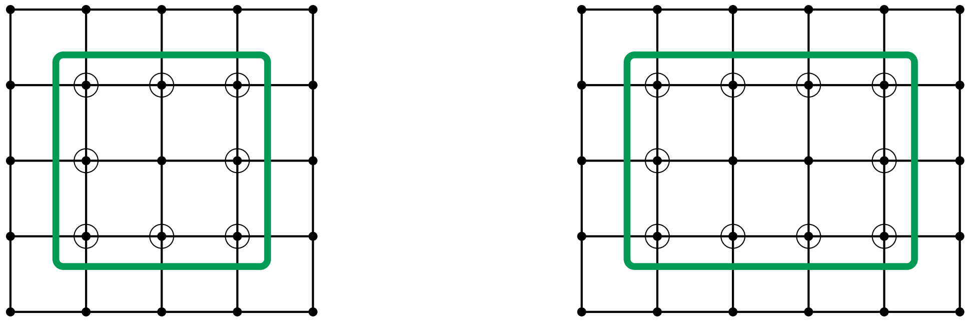
(a) Groups

(b) Borders (black)

Figure 3.4: Visualization of a possible group division for the hypercube heuristic on a 100 × 100 grid with a maximum group size of 120 and range $(1, 1)$. Interior cells: 67.2%.

3.4 Optimal hyperrectangle heuristic

To account for a different range in different dimensions, we introduce a heuristic that divides the grid into hyperrectangles instead of squares. Given some N, the hyperrectangle with a volume of at most N cells can be determined, such that the ratio between interior and total cells is maximized. This in turn minimizes the number of values that need to be stored between iterations in a single group. In this section, an algorithm will be introduced that can determine the dimensions of such a hyperrectangle in polynomial time. Note that the optimal hyperrectangle is not always a hypercube, even when the range in each dimension is equal. This is illustrated in Figure 3.5. The ratio between the interior and total cells is $\frac{1}{9}$ for the largest square and $\frac{1}{6}$ for the optimal rectangle.



(a) Largest square

(b) Optimal rectangle

Figure 3.5: The largest square and optimal rectangle, given a maximum group size of $N = 12$ and range $(1, 1)$.

Given the range in each dimension $(r_1, \ldots, r_n)$ in each dimension, define $d_i := 2r_i$. This is the decrease in width in each dimension when comparing the complete hyperrectangle and the interior cells, which also form a hyperrectangle. Given $d_1, \ldots, d_n$ and the maximum group size N, the problem is to find the following:

$$M(\{d_1, \ldots, d_n\}, N) := \max \left\{ \prod_{k=1}^{n} \frac{x_k - d_k}{x_k} : (x_1, \ldots, x_n) \in \mathbb{Z}^n_{\geq 0},\ \prod_{k=1}^{n} x_k \leq N,\ x_k \geq d_k \right\}.$$

In addition, the values of $x_1, \ldots, x_n$ that correspond with this maximum need to be known. These values are the dimensions of the hyperrectangle. Dynamic programming can be used to determine this value. This method can also be used to retrieve the values of $x_1, \ldots, x_n$ that maximize the above expression. The expression can be rewritten to

$$M(\{d_1, \ldots, d_n\}, N) = \begin{cases} \max_{d_1 \leq x \leq N} (x - d_1) M\left(\{d_2, \ldots, d_n\}, \left\lfloor \frac{N}{x} \right\rfloor\right), & \text{if } N \geq d_1, \\ 0, & \text{if } N < d_1. \end{cases}$$

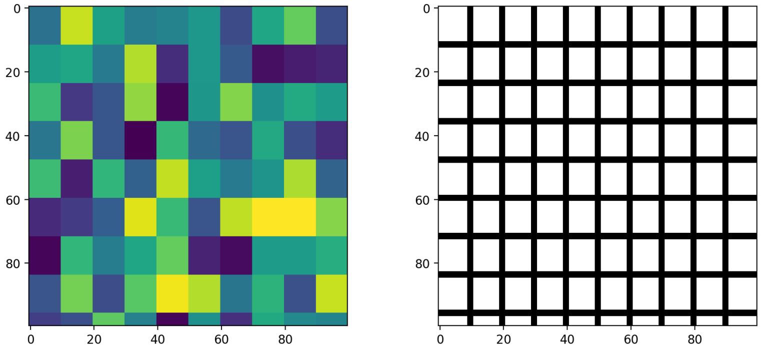
Using dynamic programming by iterating over the dimensions gives a complexity of $\mathcal{O}(nN^2)$. The full algorithm is formulated in Algorithm 3.1.

The rectangle is then tiled across the grid, starting in one corner. If the grid does not divide one dimension of the rectangle perfectly, the rectangle is cut off at the edge of the grid. Figure 3.6 shows a visualization of the division of groups of the optimal hyperrectangle heuristic. The dimensions of the rectangles differ slightly from those of the maximum square.

PTOC uses the optimal hyperrectangle heuristic to divide a grid into groups. Details of the implementation are discussed in Chapter 5.

Algorithm 3.1 Calculating the maximum inner to outer volume ratio of a hyperrectangle

```
$M_i \leftarrow 1, \quad$ for $i \in \{0, \dots, N\}$
for $D \in \{1, \dots, n\}$ do
    for $i \in \{0, \dots, d_D\}$ do
        $M'_i \leftarrow 0$
    end for
    for $i \in \{d_D + 1, \dots, N\}$ do
        $M'_i \leftarrow \max_{x \in \{d_D+1, \dots, i\}} (x - d_D) M_{\lfloor i/x \rfloor} / x$
    end for
    $M \leftarrow M'$
end for
```



(a) Groups

(b) Borders (black)

Figure 3.6: Visualization of possible group division for the optimal hyperrectangle heuristic on a 100×100 grid with a maximum group size of 120 and range $(1, 1)$. Interior cells: 68.9%.

3.5 Results

Figure 3.7 shows the ratio between the number of border cells and the total grid size for the different heuristics, with a range of $(1, 1)$. The optimal rectangle heuristic performs the best for almost all group sizes. However, the hypercube heuristic performs very similarly for a high maximum group size, as the optimal rectangle for a large group size is close to a square as well. For lower maximum group sizes it can be very beneficial to use the optimal rectangle heuristic over the hypercube heuristic.

When considering a different range in different dimensions, the hypercube heuristic performs worse, as it does not account for the extra border width in one of the dimensions. Figure 3.8 shows the difference in performance between the heuristics when considering a different range in different dimensions. The optimal rectangle heuristic again performs best. The difference between the two heuristics is largest for small maximum group sizes.

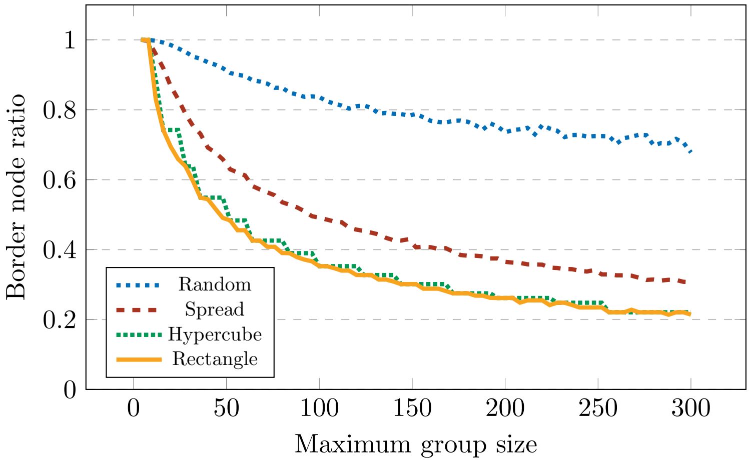


Figure 3.7: Ratio of values that need to be saved between iterations to the total number of values, on a 256 × 256 square grid, with range $(1, 1)$.

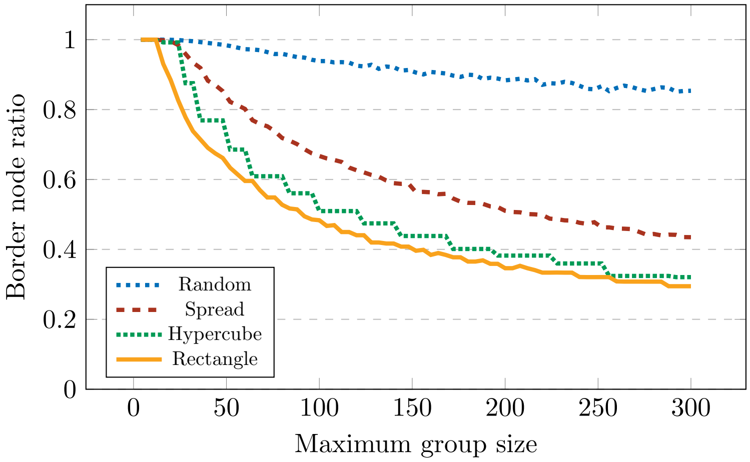


Figure 3.8: Ratio of values that need to be saved between iterations to the total number of values, on a 256 × 256 square grid, with range $(2, 1)$.

When considering a 3D grid, the ratio between the border and total volume will be larger than for 2D grids. For a large cube with sides of length n, the ratio between the surface and volume of the cube will be approximately $\frac{6n^2}{n^3} = \frac{6}{n}$. For a square, this ratio will be approximately $\frac{4n}{n^2} = \frac{4}{n}$. This means that the ratio of border cells to total cells will be larger in a 3D grid.

Figure 3.9 shows the border to total volume ratio for a 3D grid, with a range of $(1, 1, 1)$. The results are similar to those in 2D, where the optimal rectangle heuristic performs best. The difference between the hypercube and optimal rectangle heuristics is more pronounced, as the steps in which the largest cube grows compared to the maximum group size are larger.

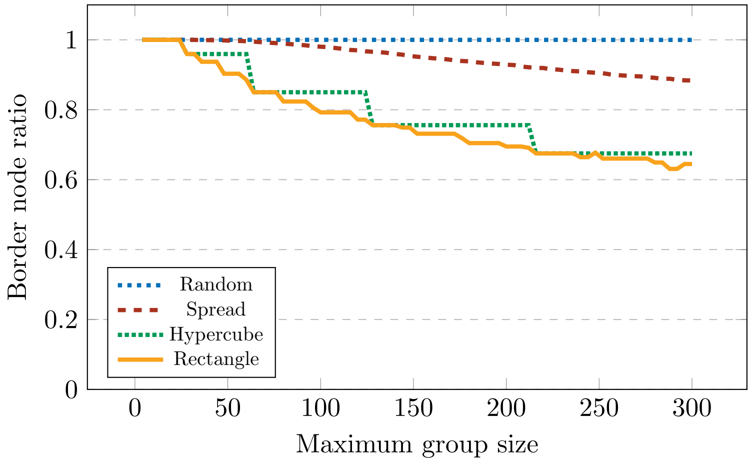


Figure 3.9: Ratio of values that need to be saved between iterations to the total number of values, on a $64 \times 64 \times 64$ cube grid, with range $(1, 1, 1)$.

3.6 Other considerations

An important consideration that has to be made when dividing the ODE in the real world, is the feasibility of the division of the groups, and real-world limitations. Depending on how the analog computer can be configured, it may be more beneficial to choose a different shape for the groups. For example, a square group may help reduce the maximum distance between any two cells in the group, which could reduce the distance electrical signals have to travel inside the computer.

Another consideration is that it may be beneficial to make the grid more dense in some areas, to account for chaotic behaviour in some areas. In this case, the division of groups would have to account for this. This would however be very difficult to automate and is not considered in PTOC.

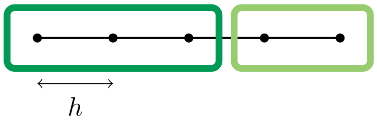
It should also be noted that in this thesis, mostly PDEs with spatial derivatives of at most second order are considered. In this case, the range in each dimension is at most 1, which means the benefit of the optimal hyperrectangle heuristic over the hypercube heuristic may be less pronounced.

Chapter 4

The iterative method and its mathematical justification

As discussed in Chapter 3, to avoid having to use a large number of resources when solving the system of ODEs resulting from the discretization of a PDE, the grid can be split up into groups. Each group of cells then corresponds to a smaller system of ODEs. When values from a different group are needed to determine the solution of the current system of ODEs, a previous guess for the value outside the group is used. This guess improves over time, by using the values from the previous iteration in the current iteration. The following example illustrates how the iterative method uses values from previous and current iterations.

Example 4.1. Take the one-dimensional heat equation $\partial_t u = \partial_{xx} u$, with the following grid and groups:



The two cells at the left and the right are the boundary cells, for which a special boundary condition is given. For this example, we will take $u(t) = 0$ at the boundary. Naming the cells $u_0, \dots, u_4$ will yield the following approximation of the PDE:

$$\begin{aligned}\dot{u}_1 &= \frac{1}{h^2}\left(0 - 2u_1 + u_2\right),\\ \dot{u}_2 &= \frac{1}{h^2}\left(u_1 - 2u_2 + u_3\right),\\ \dot{u}_3 &= \frac{1}{h^2}\left(u_2 - 2u_3 + 0\right).\end{aligned}$$

These equations follow from the approximation $\partial_{xx} u(t,x) \approx (u(t,x-h) - 2u(t,x) + u(t,x+h))/h^2$. When introducing the two groups, in the equations for u_0, u_1, and u_4 nothing changes, since these do not reference any values from different groups. For u_2, the value of u_3 is taken from the previous iteration. For u_3, the value of u_2 is taken from the previous iteration. This results in the following iterative system:

$$\begin{aligned}
\dot{u}_{1,k+1} &= \frac{1}{h^2}\left(0 - 2u_{1,k+1} + u_{2,k+1}\right), &\quad u_{1,0}(0) &= u_1(0),\\
\dot{u}_{2,k+1} &= \frac{1}{h^2}\left(u_{1,k+1} - 2u_{2,k+1} + u_{3,k}\right), &\quad u_{2,0}(0) &= u_2(0),\\
\dot{u}_{3,k+1} &= \frac{1}{h^2}\left(u_{2,k} - 2u_{3,k+1} + 0\right), &\quad u_{3,0}(0) &= u_3(0).
\end{aligned}$$

Note that at iteration k, the top two equations can be solved separately from the bottom one. As an initial guess, we will take constant functions, that are equal to the initial condition given at each grid point.

Using this iterative method, the goal is to approximate a solution of the input PDE. There are two steps where accuracy is lost: First when the PDE is converted to an approximation system of ODEs, and then again when this system is converted to an iterative system. We will focus on the second step since the first step is not unique to analog computers, rather it is a more general problem when simulating PDEs. The second part is unique to analog computers, however, because of their ability to solve (small) systems of ODEs. This is different from a method like Picard iteration, which does not need feedback loops to solve an ODE, and hence an analog computer is not required.

The use of the iterative method to approximate the solution of a system of ODEs is justified in this chapter. It is proven that, if the iterative method converges, it converges to a solution. Furthermore, the iterative method converges to the solution of the system of ODEs on some local time interval $[0, T]$. First, the iterative method is formalized in mathematical terms.

4.1 Formalizing the iterative method

The iterative method can be described as a sequence of systems of ODEs $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$, where each equation in the system corresponds to one grid cell. In addition, there is a restriction that at a given grid cell, values from different groups are taken from the previous iteration, and values from the same group are taken from the current iteration. This is formulated in the following definition.

Definition 4.2. Let $\Omega \subset \mathbb{R}^n$, $\tilde{f} : [0,\tau] \times \Omega \to \mathbb{R}^n$ and $f : [0,\tau] \times \Omega^2 \to \mathbb{R}^n$. Suppose we have the initial condition $x_0 \in \Omega$. The quadruple (Ω, τ, f, x_0) is called a **restricted iterative method** for the equation $\dot{x} = \tilde{f}(t, x)$, if the following hold:

(a) $\tilde{f}(t, y) = f(t, y, y)$ for all $t \in [0, \tau]$ and $y \in \Omega$;

(b) There exist subspaces $\mathbb{R}^{n_i} \subset \mathbb{R}^n$ such that $\mathbb{R}^n = \bigoplus_i \mathbb{R}^{n_i}$, and for all $t \in [0, \tau]$, $y, z \in \Omega$ and i, we have $\rho_i(y), \tilde{\rho}_i(z) \in \Omega$ and

$$\rho_i(f(t, y, z)) = f(t, \rho_i(y), \tilde{\rho}_i(z)),$$

where $\rho_i : \bigoplus_j \mathbb{R}^{n_j} \to \mathbb{R}^{n_i}$ and $\tilde{\rho}_i : \bigoplus_j \mathbb{R}^{n_j} \to \bigoplus_{i \neq j} \mathbb{R}^{n_j}$ are the induced projection maps.

If only (a) holds, this quadruple is called a **(general) iterative method** for $\dot{x} = \tilde{f}(t,x)$. Furthermore, we say that $x : [0,\tau] \to \Omega$ is a **solution of the original system** if $\dot{x} = \tilde{f}(t,x) = f(t,x,x)$ and $x(0) = x_0$. We call the sequence $\{x_k\}_{k\in\mathbb{N}}$ a **solution of the iterative method** if $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$ with $x_{k+1}(0) = x_0(t) = x_0$ for all $k \in \mathbb{N}$ and $t \in [0,\tau]$.

This definition means that we have some system of ODEs $\dot{x} = \tilde{f}(t,x)$, for which the inputs are split up such that $\dot{x} = f(t,x,x)$. By using the values from the current iteration as the second parameter, and the values from the previous iteration as the third, we get the iterative method. In the restricted iterative method the coordinates used from the second and third parameters may not overlap. We will also refer to a sequence of equations $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$ as being a (restricted) iterative method when we mean that f is part of a quadruple (Ω, τ, f, x_0) which is a (restricted) iterative method. Note that the function $\tilde{f}$ can be retrieved from the iterative method directly, by using $\tilde{f}(t,y) = f(t,y,y)$. Hence it is not included in the quadruple in the definition of the iterative method.

Remark 4.3. As covered in Chapter 3, not all values need to necessarily be stored between iterations. The definition of the iterative method could therefore be changed to incorporate these values that do not need to be stored, by changing $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$ to $\dot{x}_{k+1} = f(t, x_{k+1}, \rho(x_k))$, where $\rho : \mathbb{R}^n \to \mathbb{R}^m$ is a map with $m \leq n$. Then f becomes a map $f : [0,\tau] \times \Omega \times \tilde{\Omega} \to \mathbb{R}^n$, where $\tilde{\Omega} := \Omega \cap \mathbb{R}^m$. By taking $m \leq n$, we indicate that there are m values that are stored between iterations. This addition to the definition is not required for the proofs in this chapter, hence it is omitted.

Example 4.4. The equation $\dot{x}_{k+1} = f(t, x_{k+1}, x_k) = x_{k+1}x_k$ is not a restricted iterative method. Since x_{k+1} is referenced on the right side, we would need to have $\rho(y) = y$. However, this would mean that $\tilde{\rho}(z) = 0$, since ρ and $\tilde{\rho}$ cannot map to the same subspace. This yields

$$\rho(f(t,y,z)) = \rho(yz) = yz \neq 0 = f(t,y,0) = f(y, \rho(y), \tilde{\rho}(z)).$$

The equation is a general iterative method of the equation $\dot{x} = x^2$.

Example 4.5. Both iterative systems of equations

$$\begin{aligned} \dot{x}_{k+1} &= x_{k+1}y_k, \\ \dot{y}_{k+1} &= x_k y_{k+1} \end{aligned} \tag{4.1}$$

and

$$\begin{aligned} \dot{x}_{k+1} &= x_{k+1}y_{k+1}, \\ \dot{y}_{k+1} &= x_{k+1}y_{k+1} \end{aligned} \tag{4.2}$$

are restricted iterative methods for the original system $\dot{x} = \dot{y} = xy$. In (4.1) the two equations are in separate groups, as both use values from the previous iteration. In (4.2) the two equations are in the same group, which also makes the iterative method unnecessary in this case.

The iterative method used in PTOC conforms to the definition of a restricted iterative method. However, many things can be proven for the general iterative method. Therefore, in most of the remainder of this chapter, properties of the general iterative method will be proven.

4.1.1 Comparison with Picard iteration

The description of the iterative method is very similar to Picard iteration. Picard iteration only uses values from the previous iteration for approximating the current one, yielding an iterative system of ODEs of the form $\dot{x}_{k+1} = f(t, x_k)$. This transforms the problem into an integral equation $x_{k+1}(t) = x_0 + \int_0^t f(s, x_k(s))\ ds$, which cannot be done in the general iterative method because values from the current iteration are inputs to f. Figure 4.1 shows the relationship between Picard iteration and the two iterative methods.

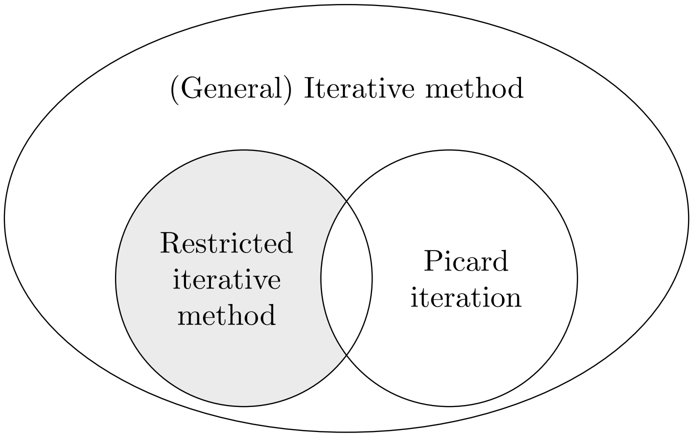


Figure 4.1: A visualization of the relationship between different iterative ODEs.

Any restricted iterative method is also a general iterative method. Likewise, the iterative method is also a more general form of Picard iteration. Below follow several examples of iterative ODEs of different types:

- Restricted iterative method, but not Picard iteration: $\dot{x}_{k+1} = x_{k+1}$;
- Restricted iterative method and Picard iteration: $\dot{x}_{k+1} = y_k$, $\dot{y}_{k+1} = x_k$;
- Picard iteration, but not a restricted iterative method: $\dot{x}_{k+1} = x_k$;
- General iterative method, but not restricted iterative method or Picard iteration: $\dot{x}_{k+1} = x_k x_{k+1}$.

Most systems that are generated by PTOC are a restricted iterative method, but not Picard iteration, as variables from the same group are almost always referenced.

4.2 Convergence to a solution

For Picard iteration, the local convergence of the sequence of functions $\{x_k\}_{k\in\mathbb{N}}$ is already shown in the proof of the Picard-Lindelöf theorem [20, Theorem 2.2]. In this section, a similar statement will be proven for the general iterative method. Also, if the function f is globally Lipschitz continuous, the Picard iteration sequence converges for all $t \geq 0$. This is also the case for the general iterative method. First, we will show that if the iterative method converges to some function, this function will be a solution of the original system of ODEs.

Remark 4.6. The existence of a solution of the system of ODEs or the iterative method does not imply the existence of a solution of the original PDE. As mentioned at the start of this chapter, it may not be generally true that a finer grid will approximate the original PDE better. We will refrain from investigating this further in this thesis.

Lemma 4.7. *Let (Ω, τ, f, x_0) be an iterative method, with f continuous. Let $\{x_k\}_{k\in\mathbb{N}}$ be a solution of the iterative method, with each x_k continuously differentiable. Let $x : [0, \tau] \to \Omega$ be continuously differentiable. Suppose that for all $t \in [0, \tau]$ the following hold:*

$$\begin{aligned} \lim_{k\to\infty} x_k(t) &= x(t), \\ \lim_{k\to\infty} \dot{x}_k(t) &= \dot{x}(t). \end{aligned}$$

Then x is a solution of the original system.

Proof. Let $t \in [0, \tau]$ and $\varepsilon > 0$. Because f is continuous, there exists some $\delta > 0$ such that $||f(t, a, b) - f(t, x(t), x(t))|| < \frac{\varepsilon}{2}$ for all a and b with $||a - x(t)|| + ||b - x(t)|| < \delta$. Because of the convergence of the derivatives, there exists some $k \in \mathbb{N}$ such that

$$||\dot{x}(t) - \dot{x}_{k+1}(t)|| < \frac{\varepsilon}{2}, \quad ||x_{k+1}(t) - x(t)|| + ||x_k(t) - x(t)|| < \delta.$$

This implies that

$$||f(t, x_{k+1}(t), x_k(t)) - f(t, x(t), x(t))|| < \frac{\varepsilon}{2}.$$

Then

$$\begin{aligned} ||\dot{x}(t) - f(t, x(t), x(t))|| &\leq ||\dot{x}(t) - \dot{x}_{k+1}(t)|| + ||\dot{x}_{k+1}(t) - f(t, x(t), x(t))|| \\ &= ||\dot{x}(t) - \dot{x}_{k+1}(t)|| + ||f(t, x_{k+1}(t), x_k(t)) - f(t, x(t), x(t))|| \\ &< \frac{\varepsilon}{2} + \frac{\varepsilon}{2} \\ &= \varepsilon. \end{aligned}$$

This holds for all $\varepsilon > 0$, and hence $\dot{x}(t) = f(t, x(t), x(t))$. □

If instead the sequence $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to x, the condition $\dot{x}_k(t) \to \dot{x}(t)$ follows automatically. This gives the following corollary.

Corollary 4.8. *Let $\{x_k\}_{k\in\mathbb{N}}$ be a solution of an iterative method (Ω, τ, f, x_0), with x_k continuously differentiable and f continuous. If $x : [0, \tau] \to \Omega$ is continuously differentiable such that $x_k \to x$ uniformly, then x is a solution of the original system.*

This corollary justifies the use of the iterative method to approximate a solution. It guarantees that if the iterative method converges, it converges to a solution.

4.2.1 Local convergence

Next, we will prove that, given sufficient conditions, the iterative method always converges close to $t = 0$. This procedure is used in the proof of the Picard-Lindelöf theorem [20, Theorem 2.2], which gives an explicit interval $[-T, T]$ on which the iterations converge to the unique solution of the system. Because of the nature of the analog computer only simulating forward in time, we will instead consider only finite positive time intervals $[0, \tau]$. Theorem 4.10 gives an equivalent of the Picard-Lindelöf theorem for the iterative method. Before giving the theorem, we will introduce some notation that will be used in the remainder of this chapter.

Notation 4.9. When considering a product space $A \times B$, we will define the norm $|| \cdot ||_1$ on $A \times B$ to be $||(a, b)||_1 := ||a||_A + ||b||_B$. In particular, when using the norm $|| \cdot ||_1$ on $[0, \tau] \times \Omega^2$, we use $||(t, x, y)||_1 := |t| + ||x||_\Omega + ||y||_\Omega$. When considering a function $g : A \to B$, we define the norm $||g||_\infty := \sup_{a \in A} ||g(a)||_B$. This norm is always finite if A is compact.

Theorem 4.10. *Let $\Omega \subset \mathbb{R}^n$, $\tau \geq 0$ and $f : [0, \tau] \times \Omega^2 \to \mathbb{R}^n$ Lipschitz continuous with Lipschitz constant L using the norm $|| \cdot ||_1$ on $[0, \tau] \times \Omega^2$. Let $\delta > 0$ and $x_0 \in \Omega$ such that $\overline{B}_\delta(x_0) \subset \Omega$. Define $M := \sup_{x,y \in \overline{B}_\delta(x_0), t \in [0,\tau]} ||f(t, x, y)||$. Then for any $T < \min\{\frac{\delta}{M}, \frac{1}{2L}, \tau\}$ a solution of the iterative method (Ω, T, f, x_0) exists, and any solution of this iterative method converges uniformly to the unique solution x of the original system.*

Proof. The theorem will be proven in three steps: First, it is shown that a solution of the original system exists. Then it is shown that a solution of the iterative method exists. These two are combined to show that the solution of the iterative method converges to the solution of the original system.

<u>A solution of the original system exists and is unique:</u> We first show that a solution x of the original system $\dot{x} = f(t, x, x)$ exists and is unique on the interval $[0, T]$. Consider the function $\tilde{f} : [0, \tau] \times \Omega$ defined by $\tilde{f}(t, x) := f(t, x, x)$. This function is Lipschitz with Lipschitz constant $2L$, using the norm $|| \cdot ||_1$ on $[0, \tau] \times \Omega$:

$$||\tilde{f}(t, x) - \tilde{f}(t, y)|| = ||f(t, x, x) - f(t, y, y)|| \leq L||(t, x, x) - (t, y, y)||_1 = 2L||x - y||.$$

In particular, it is locally Lipschitz. From the Picard-Lindelöf theorem [20, Theorem 2.2], it follows that a solution x of $\dot{x} = \tilde{f}(t, x) = f(t, x, x)$ exists and is unique on $[0, T]$ for all $T < \min\{\frac{\delta}{M}, \tau\}$.

<u>A solution of the iterative method exists:</u> Next, we will show that a solution of the iterative method exists and is well-defined. We will prove this using induction on k. Note that $x_0 \in \Omega$ is a constant, hence the statement is true for the base case of $k = 0$. Now let $k \geq 0$ and suppose that functions $x_0, \dots, x_k : [0, T] \to \Omega$ exist such that $\dot{x}_{i+1} = f(t, x_{i+1}, x_i)$ for all $i \in \{0, \dots, k - 1\}$. We will prove that such a function x_{k+1} also exists.

Consider the function $\tilde{f} : [0, T] \times \overline{B}_\delta(x_0) \to \mathbb{R}^n$ defined by $\tilde{f}(t, x) = f(t, x, x_k(t))$. Note that we have $||\tilde{f}(t, x)|| = ||f(t, x, x_k(t))|| \leq M$ for all $t \in [0, T]$ and $x \in \overline{B}_\delta(x_0)$, because $x_k(t) \in \overline{B}_\delta(x_0)$. The function $\tilde{f}$ is Lipschitz since the domain $[0, T] \times \overline{B}_\delta(x_0)$ is compact, and $\tilde{f}$ is continuously differentiable. Hence, by the Picard-Lindelöf theorem, there exists a unique solution x_{k+1} of $\dot{x}_{k+1} = \tilde{f}(t, x_{k+1}) = f(t, x_{k+1}, x_k)$ on $[0, T]$, since $T < \min\{\frac{\delta}{M}, \tau\}$. This is what we needed to prove. By induction, a solution of the iterative method $\{x_k\}_{k \in \mathbb{N}}$ exists.

The solution of the iterative method converges: Next, we will prove that the sequence $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to the solution x of $\dot{x} = f(t,x,x)$ on $[0,T]$. Subtracting the differential equations $\dot{x} = f(t,x,x)$ and $x_{k+1} = f(t,x_{k+1},x_k)$ gives

$$\dot{x} - \dot{x}_{k+1} = f(t,x,x) - f(t,x_{k+1},x_k).$$

Integrating both sides and using $x_{k+1}(0) = x_k(0) = x_0$ gives

$$\begin{aligned} x(t) - x_{k+1} &= x(0) - x_{k+1}(0) + \int_0^t (f(s,x(s),x(s)) - f(s,x_{k+1}(s),x_k(s)))\, ds \\ &= \int_0^t (f(s,x(s),x(s)) - f(s,x_{k+1}(s),x_k(s)))\, ds. \end{aligned}$$

Taking norms on both sides we get

$$\begin{aligned} ||x - x_{k+1}||_\infty &= \sup_{t\in[0,T]} \left|\left| \int_0^t (f(s,x(s),x(s)) - f(s,x_{k+1}(s),x_k(s)))\, ds \right|\right| \\ &\leq \sup_{t\in[0,T]} \int_0^t ||f(s,x(s),x(s)) - f(s,x_{k+1}(s),x_k(s))||\, ds. \end{aligned}$$

Note that $x(s), x_{k+1}(s), x_k(s) \in \overline{B}_\delta(x_0) \subset \Omega$, and hence the Lipschitz continuity of f can be used:

$$\begin{aligned} ||x - x_{k+1}||_\infty &\leq \sup_{t\in[0,T]} \int_0^t L||(x(s),x(s)) - (x_{k+1}(s),x_k(s))||_1\, ds \\ &\leq \sup_{t\in[0,T]} \int_0^t L(||x(s) - x_{k+1}(s)|| + ||x(s) - x_k(s)||)\, ds \\ &\leq \sup_{t\in[0,T]} \int_0^t L(||x - x_{k+1}||_\infty + ||x - x_k||_\infty)\, ds \\ &\leq LT(||x - x_{k+1}||_\infty + ||x - x_k||_\infty) \\ &= LT||x - x_{k+1}||_\infty + LT||x - x_k||_\infty. \end{aligned}$$

It then follows that

$$(1 - LT)||x - x_{k+1}||_\infty \leq LT||x - x_k||_\infty.$$

Since $T < \frac{1}{2L}$, we have $1 - LT > 0$ and hence

$$||x - x_{k+1}||_\infty \leq \frac{LT}{1 - LT}||x - x_k||_\infty.$$

Also since $T < \frac{1}{2L}$, we have $\frac{1}{LT} > 2$, which implies $\frac{LT}{1-LT} = \frac{1}{\frac{1}{LT}-1} < 1$. This means that

$$\lim_{k\to\infty} ||x - x_k||_\infty \leq \lim_{k\to\infty} \left(\frac{LT}{1 - LT}\right)^k ||x - x_0||_\infty = 0.$$

Therefore the sequence $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to x on the interval $[0,T]$. □

Remark 4.11. As can be seen in the proof of Theorem 4.10, an explicit bound can be given on the difference between the value at a given iteration and the solution value, if an upper bound for the solution x is known:

$$||x - x_k||_\infty \leq \left(\frac{LT}{1 - LT}\right)^k ||x - x_0||_\infty \leq \left(\frac{LT}{1 - LT}\right)^k (||x||_\infty + ||x_0||_\infty).$$

When f is not Lipschitz continuous, but continuously differentiable, the solution of the iterative method $\{x_k\}_{k \in \mathbb{N}}$ still converges to x on some interval $[0, T]$. Taking any compact neighborhood K of x_0 will give a Lipschitz continuous function $f|_{[0,\tau] \times K^2}$.

4.2.2 Order of the difference

Outside the interval $[0, T]$ given by Theorem 4.10 nothing can be said about the actual difference between the solution of the iterative method and the solution of the original system. However, for analytic functions, the order of the difference can be determined. The following lemma will be used to prove this.

Lemma 4.12. *Let $\Omega \subset \mathbb{R}^n$ with $0 \in \Omega$. Let $f : [0, \tau] \times \Omega \to \mathbb{R}^n$ be analytic at 0 and $x, y : [0, \tau] \to \Omega$ also analytic at $t = 0$. Suppose there exists some $k \geq 0$ such that for all $t \in [0, \tau]$, it holds that $x(t) - y(t) = O(t^k)$. Then $f(t, x(t)) - f(t, y(t)) = O(t^k)$.*

Proof. Write $f(t, x_1, \dots, x_n) = (f_1(t, x_1, \dots, x_n), \dots, f_n(t, x_1, \dots, x_{2n}))$. Since f is analytic, for $i = 1, \dots, n$ we have that

$$f_i(t, x_1, \dots, x_n) = \sum_{m=0}^{\infty} c_m t^{\alpha_{m,0}} x_1^{\alpha_{m,1}} \dots x_n^{\alpha_{m,n}},$$

where $\alpha_m \in \mathbb{N}_{\geq 0}^{n+1}$ such that $\{\alpha_m\}_{m \in \mathbb{N}}$ is some sequence over all natural number tuples of length $n + 1$. Now suppose that $f_i(t, x_1(t), \dots, x_n(t))$ and $f_i(t, y_1(t), \dots, y_n(t))$ differ at some index m. Since $x(t) - y(t) = O(t^k)$, the first power in the Taylor series at which $t^{\alpha_{m,0}} x_1(t)^{\alpha_{m,1}} \dots x_n(t)^{\alpha_{m,n}}$ will differ from $t^{\alpha_{m,0}} y_1(t)^{\alpha_{m,1}} \dots y_n(t)^{\alpha_{m,n}}$ is at least t^k. Since $f_i(t, x(t)) - f_i(t, y(t))$ will have the order of the lowest power at which any of the $t^{\alpha_{m,0}} x_1(t)^{\alpha_{m,1}} \dots x_n(t)^{\alpha_{m,n}}$ will differ, we get $f(t, x(t)) - f(t, y(t)) = O(t^k)$. □

Theorem 4.13 builds on this lemma and states that the Taylor series of the solution x of the original system and iteration x_k are the same up to the term t^k. The theorem is a generalization of the same statement, but considering Picard iteration instead of the iterative method.

Theorem 4.13. *Let (Ω, τ, f, x_0) be an iterative method, where $0 \in \Omega$ and f is analytic at 0, with radius of convergence enclosing Ω. Let x be analytic at $t = 0$ and a solution of the original system. Let $\{x_k\}_{k \in \mathbb{N}}$ be a solution of the iterative method, where each x_k is analytic at $t = 0$. Then for all $t \in [0, \tau]$ and $k \in \mathbb{N}$ we have $x(t) - x_k(t) = O(t^{k+1})$.*

Proof. We prove the theorem using induction. For $k = 0$ we have $x(0) = x_0$, and hence the constant term in the Taylor series of x around 0 is x_0. This means $x(t) - x_0 = O(t)$.

Let $k \geq 0$ and suppose that the theorem holds for k. We will prove that the theorem holds for $k+1$ as well. We will use the notation $\alpha(t) = \Theta(t^\ell)$ if ℓ is the lowest integer such that $\alpha(t) = O(t^\ell)$. By the induction hypothesis, there exists a $p \geq k+1$ such that $x(t) - x_k(t) = \Theta(t^p)$. If $x_{k+1} = x$, the statement holds. If $x_{k+1} \neq x$, there exists some $q \geq 0$ such that $x(t) - x_{k+1}(t) = \Theta(t^q)$. Combining equations $\dot{x} = f(t, x, x)$ and $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$ gives

$$\dot{x} - \dot{x}_{k+1} = f(t, x, x) - f(t, x_{k+1}, x_k). \tag{4.3}$$

Note that $(t, x(t), x(t)) - (t, x_{k+1}(t), x_k(t)) = O(t^{\min\{q,k+1\}})$. By Lemma 4.12 we have $f(t, x(t), x(t)) - f(t, x_{k+1}(t), x_k(t)) = O(t^{\min\{q,k+1\}})$.

Because $x(0) = x_{k+1}(0)$, it follows that $q > 0$. This means that taking the derivative will decrease the order of the difference, such that $\dot{x}(t) - \dot{x}_{k+1}(t) = \Theta(t^{q-1})$. Combining this with the order of the right side we get $q - 1 \geq \min\{q, k+1\}$. If $q \leq k+1$, then this would imply $q - 1 \geq q$, which is false. Hence we have that $q > k+1$, which means $x(t) - x_{k+1}(t) = O(t^{k+2})$. □

Theorem 4.13 can be useful to the user to determine the number of iterations that are needed to achieve the desired precision for the output. However, in reality, the precision achieved may be higher because of the use of a combination of the current and the previous iteration values. An example of a system where the order of the difference is larger than indicated in Theorem 4.13 is given below.

Example 4.14. Consider the system

$$\begin{cases} \dot{x} &= -y, & x(0) = 1, \\ \dot{y} &= x, & y(0) = 0. \end{cases}$$

The solution of this system is $(x(t), y(t)) = (\cos t, \sin t)$. When using Picard iteration, in each iteration one extra term will be added to the Taylor expansion of the solution. Hence the number of correct Taylor terms for (x_k, y_k) will be $(1, 1)$, $(2, 2)$, $(3, 3)$, etc.

Now consider the iterative method $\dot{x}_{k+1} = -y_{k+1}, \dot{y}_{k+1} = x_k$. Then the number of correct terms for x_{k+1} will be one higher than for y_{k+1}, and the number of correct terms for y_{k+1} will be one higher than for x_k. This means we get the sequence $(1, 1)$, $(3, 2)$, $(5, 4)$, $(7, 6)$, etc. Hence, in this case, the iterative method will converge to the true solution more quickly. More examples of systems where the actual achieved precision is higher, are investigated further in Section 4.3.

The order of the difference between the solution of the original system and the solution of the iterative method is also not necessarily a good indication of the actual difference between the two values. For example, the functions 1, t, t^2, etc. converge to 0 on the interval $[0, 1)$. However, for $t \geq 1$ this sequence diverges and the accuracy of the approximations gets worse. It is unknown to the author of this thesis, if there exists some sequence like this that is also a solution of an iterative method, such that a solution of the original system exists on a larger interval than the interval of convergence of the iterative method.

4.2.3 Convergence when f is Lipschitz continuous

In Theorem 4.10 the interval on which the iterations converge is dependent on the Lipschitz constant L of f. However, this dependence is not required. In this section, we will show that if f is Lipschitz continuous on some domain Ω, the solution of the iterative method $\{x_k\}_{k\in\mathbb{N}}$ converges to the solution of the original system x as long as $\{x_k\}_{k\in\mathbb{N}}$ and x map $[0,\tau]$ onto a subset of Ω. First, we introduce a variant of Grönwall's inequality, which will be used in the proof of global convergence.

Theorem 4.15. *(Generalized Grönwall's inequality for α non-decreasing, [20, Lemma 2.7]) Let $\alpha : [0,\tau] \to \mathbb{R}$ be non-decreasing, $\psi : [0,\tau] \to \mathbb{R}$, and $\beta : [0,\tau] \to \mathbb{R}_{\geq 0}$. Suppose that for all $t \in [0,\tau]$ the following holds:*

$$\psi(t) \leq \alpha(t) + \int_0^t \beta(s)\psi(s)\ ds.$$

Then for all $t \in [0,\tau]$,

$$\psi(t) \leq \alpha(t) \exp\left(\int_0^t \beta(s)\ ds\right).$$

This inequality will be used to prove the following result, which under certain conditions gives a bound on the difference between two consecutive iterations. The proof is based on a proof of the global existence of a solution for an initial value problem with f globally Lipschitz continuous [20, Corollary 2.6].

Lemma 4.16. *Let (Ω,τ,f,x_0) be an iterative method with f Lipschitz continuous with Lipschitz constant L, when using the norm $||\cdot||_1$ on $[0,\tau]\times\Omega^2$. Let $\{x_k\}_{k\in\mathbb{N}}$ be a solution of this iterative method. Then, there exists some $C > 0$ such that $||x_{k+1}(t) - x_k(t)|| \leq ||x_1 - x_0||_\infty \frac{1}{k!}C^k t^k$ for all $t \in [0,\tau]$ and $k \geq 0$. In fact, we have $C = Le^{L\tau}$.*

Proof. We prove the lemma by induction on k. For $k = 0$ the statement becomes $||x_1(t) - x_0|| \leq ||x_1 - x_0||_\infty$, which is true. Let $k > 0$ and assume the inequality is true for $k-1$. Then

$$\begin{aligned}
||x_{k+1}(t) - x_k(t)|| &= \left|\left|x_{k+1}(0) - x_k(0) + \int_0^t (f(s, x_{k+1}(s), x_k(s)) - f(s, x_k(s), x_{k-1}(s)))\ ds\right|\right| \\
&= \left|\left|\int_0^t (f(s, x_{k+1}(s), x_k(s)) - f(s, x_k(s), x_{k-1}(s)))\ ds\right|\right| \\
&\leq \int_0^t ||f(s, x_{k+1}(s), x_k(s)) - f(s, x_k(s), x_{k-1}(s))||\ ds \\
&\leq \int_0^t L||(x_{k+1}(s) - x_k(s), x_k(s) - x_{k-1}(s))||_1\ ds \\
&= \int_0^t L||x_{k+1}(s) - x_k(s)|| + L||x_k(s) - x_{k-1}(s)||\ ds \\
&= L\int_0^t ||x_k(s) - x_{k-1}(s)||\ ds + L\int_0^t ||x_{k+1}(s) - x_k(s)||\ ds.
\end{aligned}$$

We will now apply Grönwall's inequality with

$$\begin{aligned}
&\alpha(t) = L\int_0^t ||x_k(s) - x_{k-1}(s)||\ ds, \\
&\beta(t) = L, \\
&\psi(t) = ||x_{k+1}(s) - x_k(s)||\ ds.
\end{aligned}$$

Since the norm is positive, α is non-decreasing. Also,

$$\int_0^t \beta(s)\ ds = Lt.$$

This gives

$$\begin{aligned} ||x_{k+1}(t) - x_k(t)|| &\leq \alpha(t) \exp\left(\int_0^t \beta(s)\ ds\right) \\ &= \left(L \int_0^t ||x_k(s) - x_{k-1}(s)||\ ds\right) e^{Lt} \\ &\leq Le^{L\tau} \int_0^t ||x_k(s) - x_{k-1}(s)||\ ds \\ &= C \int_0^t ||x_k(s) - x_{k-1}(s)||\ ds. \end{aligned}$$

Using the induction hypothesis we get

$$\begin{aligned} ||x_{k+1}(t) - x_k(t)|| &\leq C \int_0^t ||x_1 - x_0||_\infty \frac{C^k s^k}{k!}\ ds \\ &= ||x_1 - x_0||_\infty \frac{C^{k+1} t^{k+1}}{(k+1)!}. \end{aligned}$$

By induction, the statement holds for all $k \geq 0$. □

Lemma 4.16 can be used to show that the sequence of iterations actually converges to a solution. This is stated in the following theorem.

Theorem 4.17. *Let (Ω, τ, f, x_0) be an iterative method with f Lipschitz continuous. Let $\{x_k\}_{k\in\mathbb{N}}$ be a solution of this iterative method. Then $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to a solution of the original system x.*

Proof. From Lemma 4.16 we have that $||x_{k+1} - x_k||_\infty \leq C^{k+1}\tau^{k+1}/(k+1)!$. Let $k, k' \geq 0$. Then

$$\begin{aligned} ||x_{k'+k} - x_{k'}||_\infty &\leq ||x_{k'+1} - x_{k'}||_\infty + \cdots + ||x_{k'+k} - x_{k'+k-1}||_\infty \\ &\leq ||x_1 - x_0||_\infty \left(\frac{C^{k'}\tau^{k'}}{k'!} + \cdots + \frac{C^{k'+k}\tau^{k'+k}}{(k'+k)!}\right) \\ &\leq ||x_1 - x_0||_\infty \left(e^{C\tau} - \sum_{\ell=0}^{k'-1} \frac{C^\ell \tau^\ell}{\ell!}\right) \end{aligned}$$

Since the second part converges to 0, for any $\varepsilon > 0$ there exists some $K \geq 0$ such that for all $k_1, k_2 \geq N$, $||x_{k_1} - x_{k_2}||_\infty < \varepsilon$. The sequence is Cauchy, and because the space of continuously differentiable functions is a Banach space under the norm $||\cdot||_\infty$, it converges uniformly. By Corollary 4.8 it converges to a solution. □

Lemma 4.18. *Let (Ω, τ, f, x_0) be an iterative method with $f : [0, \tau] \times \mathbb{R}^{2n} \to \mathbb{R}^n$ continuously differentiale. Let $\{x_k\}_{k\in\mathbb{N}}$ be a solution of this iterative method. Suppose that the sequence $\{x_k\}_{k\in\mathbb{N}}$ is uniformly bounded, i.e.*

$$\sup_{k\in\mathbb{N}} ||x_k||_\infty < \infty.$$

Then the sequence $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to a solution of the original system x.

Proof. Define the set $\tilde{\Omega} := \{y \in \Omega : ||y|| \leq \sup_{k\in\mathbb{N}} ||x_k||_\infty\}$. This set is closed and by the assumption that the sequence x_k is uniformly bounded, also bounded. Hence $\tilde{\Omega}$ is compact. Note that all x_k map to $\tilde{\Omega}$, hence we can rewrite the system with $f|_{[0,\tau]\times\tilde{\Omega}^2}$ and $x_k : [0, \tau] \to \tilde{\Omega}$. Since $[0, \tau] \times \tilde{\Omega}^2$ is compact, $f|_{[0,\tau]\times\tilde{\Omega}^2}$ is Lipschitz continuous. By Theorem 4.17, the sequence $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to a solution on $[0, \tau]$. □

Note that a bound on the sequence $\{x_k\}_{k\in\mathbb{N}}$ is generally not known a priori. Hence when using the iterative method a bound would have to be given by the user. An alternative could be to divide the time interval $[0, \tau]$ into smaller parts.

4.2.4 Dividing the time interval

To keep the bounds on the iterations under control, Theorem 4.10 can be used to split up the interval into smaller parts. The following theorem expands on Theorem 4.10 by removing the dependence of T on the initial condition x_0. This will later allow us to divide the time interval into a finite amount of smaller parts, on which the iterative method will have a solution and will converge.

Lemma 4.19. *Let $\Omega \subset \mathbb{R}^n$ be compact. Let $f : [0, \tau] \times \Omega^2 \to \mathbb{R}^n$ be Lipschitz continuous with Lipschitz constant L. Let $\delta > 0$ and $x_0 \in \Omega$ such that $\overline{B}_\delta(x_0) \subset \Omega$. Define $M := \sup_{x,y\in\Omega,t\in[0,\tau]} ||f(t, x, y)||$. Then for any $T < \min\{\frac{\delta}{M}, \frac{1}{2L}, \tau\}$ a solution $\{x_k\}_{k\in\mathbb{N}}$ of the iterative method (Ω, τ, f, x_0) exists and converges uniformly to the solution of the original system x on $[0, T]$.*

Proof. Let $\delta > 0$ and $x_0 \in \Omega$ such that $\overline{B}_\delta(x_0) \subset \Omega^\circ$. In particular $\overline{B}_\delta(x_0) \subset \Omega$, which gives

$$M = \sup_{x,y\in\Omega,t\in[0,\tau]} ||f(t, x, y)|| \geq \sup_{x,y\in\overline{B}_\delta(x_0),t\in[0,\tau]} ||f(t, x, y)|| =: \tilde{M}.$$

Theorem 4.10 gives that the statement is true for any $\tilde{T} < \min\{\frac{\delta}{\tilde{M}}, \frac{1}{2L}, \tau\}$. By the above inequality we have $\frac{\delta}{M} \leq \frac{\delta}{\tilde{M}}$, hence $T < \min\{\frac{\delta}{\tilde{M}}, \frac{1}{2L}, \tau\}$. This means the statement is true for T. □

We can use Lemma 4.19 to prove that, if a solution exists, using the iterative method over smaller time intervals will result in finding the entire solution. This is formulated in the following theorem.

Theorem 4.20. *Let $f : [0, \tau] \times \Omega^2 \to \mathbb{R}^n$ be continuously differentiable. Suppose that $x : [0, \tau] \to \Omega^\circ$ is a solution of the system*

$$\begin{aligned} \dot{x} &= f(t, x, x), \\ x(0) &= x_0. \end{aligned}$$

Then there exists a $T > 0$, such that for any $\varepsilon > 0$, there exists a $K \geq 0$, such that for any set of functions $x_{m,k} : [mT, \min\{(m+1)T, \tau\}] \to \Omega$ with

$$\dot{x}_{m,k+1} = f(t, x_{m,k+1}, x_{m,k}),$$
$$x_{m,k}(mT) = \begin{cases} x_0, & \textit{if } m = 0, \\ x_{m-1,K}(mT), & \textit{if } m > 0, \end{cases}$$

we have $||x_{m,k}(t) - x(t)|| < \varepsilon$ for all $m \geq 0$ and $k \geq K$.

Proof. Note that x maps $[0, \tau]$ to a compact set, because it is continuous and $[0, \tau]$ is compact. Hence its image is bounded and closed. Because Ω° is open there exists some $\delta_0 > 0$ such that the following holds

$$\tilde{\Omega} := \left\{ y \in \mathbb{R}^n : \inf_{t \in [0,\tau]} ||y - x(t)|| \leq 2\delta_0 \right\} \subset \Omega^\circ \subset \Omega.$$

Without loss of generality assume that $\Omega = \tilde{\Omega}$. Note that f is then Lipschitz continuous on Ω, with some Lipschitz constant L. Also since Ω is compact f is bounded with some bound M. Take $T < \min\{\frac{\delta_0}{M}, \frac{1}{2L}\}$ with $0 < T \leq \tau$.

We claim that for any $m \geq 0$ and $\varepsilon > 0$, there exists some $K_{m,\varepsilon} \geq 0$ such that for any $k \geq K_{m,\varepsilon}$ and $t \in [mT, \min\{(m+1)T, \tau\}]$ we have $||x_{m,k}(t) - x(t)|| < \varepsilon$. From this taking $K = \max\{K_{0,\varepsilon}, \ldots, K_{\lceil \tau/T \rceil, \varepsilon}\}$ yields the statement of the theorem. We use induction to prove this claim.

For $m = 0$ the claim follows directly from Lemma 4.19, since $B_{\delta_0}(x_0) \subset \Omega$.

Let $m > 0$ and assume that the claim is true for $m - 1$. Because f is continuously differentiable, the solution of the system is continuously dependent on the initial condition. This means that there exists some $\delta > 0$ such that the solution on the interval $[mT, \min\{(m+1)T, \tau\}]$ starting from $y_0 \in B_\delta(x(mT))$ will differ from x by at most $\frac{\varepsilon}{2}$. Without loss of generality, we can assume that $\delta < \varepsilon \leq \delta_0$, which means that this solution maps onto Ω. Call this solution $y : [mT, \min\{(m+1)T, \tau\}] \to \Omega$ and $y(0) = y_0$. Note that we have $B_{\delta_0}(y_0) \subset \Omega$.

By the induction hypothesis there exists some $K \geq 0$ such that $||x_{m,k}(mT) - x(mT)|| = ||x_{m-1,k}(mT) - x(mT)|| < \delta_0$ for any $k \geq K$. We denote y to be the solution starting from $x_{m,k}(mT)$. By Lemma 4.19 there exists some $\tilde{K} \geq 0$ such that $||y(t) - x_{m,k}(t)|| < \frac{\varepsilon}{2}$ for $t \in [mT, \min\{(m+1)T, \tau\}]$ and $k \geq \tilde{K}$. Take $K_{m,\varepsilon} = \max\{K, \tilde{K}\}$ We get the following for all $k \geq K_{m,\varepsilon}$ and $t \in [mT, \min\{(m+1)T, \tau\}]$:

$$||x_{m,k}(t) - x(t)|| \leq ||x_{m,k}(t) - y(t)|| + ||y(t) - x(t)|| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

This proves the claim, and therefore the theorem. □

Note that the set of functions $x_{m,k}$ is guaranteed to exist, for sufficiently large K, by Lemma 4.19. For sufficiently large K the values $x_{m,K}(mT)$ will have a difference of at most δ from the true solution.

This method of splitting up the interval is very inefficient, as the value of T can be very small. This can create the need to reconfigure the analog computer many times, as it will be iterating over both the x_k and time intervals $[nT, (n+1)T]$. For this reason, this method of splitting up the time interval into predetermined parts is not used in PTOC. However, the user can still do this manually. If the user enters some time interval $[0, \tau]$, and the iterations only converge on some interval $[0, \tau_*]$, the procedure can be repeated on $[\tau_*, \tau]$. When taking τ_* sufficiently close to the maximal interval on which the iterations converge, Theorem 4.20 guarantees that the procedure can approximate the solution at some time point within a finite amount of time.

4.3 Analysis of the iterative method

In this section, the effectiveness of the iterative method will be analyzed. This will be done in two different settings. One setting where a low-dimensional ODE is split up and does not necessarily represent a restricted iterative method. In the other setting the effectiveness will be investigated for ODEs that are an approximation of a PDE. In this case, the iterative ODE will always be a restricted iterative method.

4.3.1 Analysis of low-dimensional ODEs

First, we look at the effectiveness of the iterative method in a basic one-dimensional example: Solving the ODE $\dot{x} = x^2$ with initial condition $x(0) = 1$. Note that in this case, an analog computer does not have issues determining the solution of this ODE in one iteration, hence this is not a realistic example.

As the iterative method we take $\dot{x}_{k+1} = f(t, x_{k+1}, x_k) = x_k x_{k+1}$. We compare this to Picard iteration, which uses the equation $\dot{x}_{k+1} = f_p(t, x_{k+1}, x_k) = x_k^2$. The solution of the original system is $x(t) = \frac{1}{1-t}$. Note that the iterative method used in this example is not a restricted iterative method.

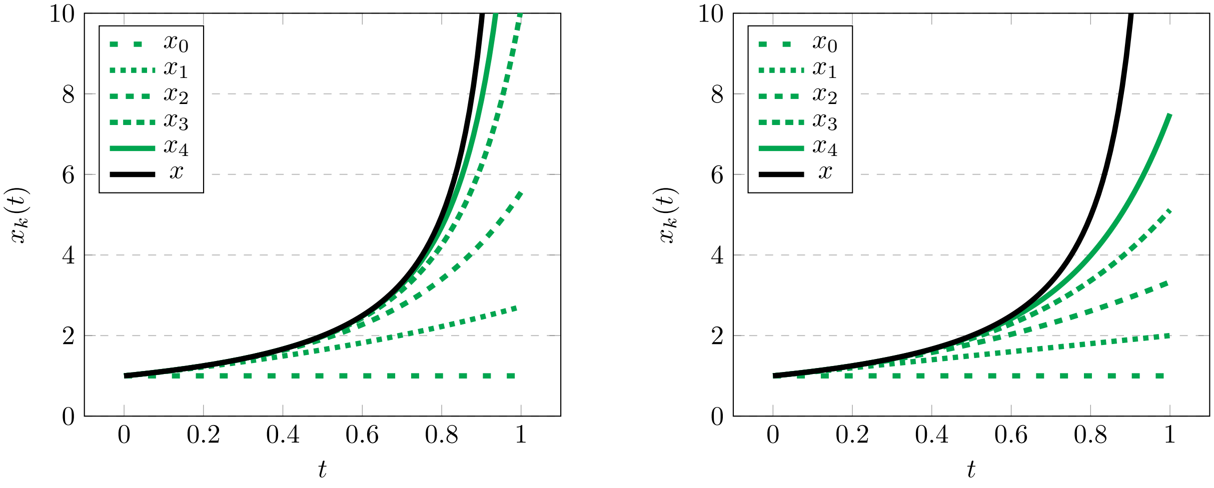
In Figure 4.2a the first five iterations of the iterative method are shown. Comparing this to Figure 4.2b, where Picard iteration is used, it can be seen that the iterative method converges faster to the true solution.

From Figure 4.2, we can conclude that it is likely that both the iterative method and Picard iteration converge to $\frac{1}{1-t}$ on the entire interval of existence $[0, 1)$. This is a larger interval than is guaranteed by the theorems proven in this chapter. Further research could be done to look into the uniform convergence of the iterative method on intervals $[0, \tau]$ as $\tau \uparrow 1$.

Likewise, further research could also be done to look into the convergence of the iterative method on $[0, \tau]$ as $\tau \to \infty$, for systems that have a solution on $[0, \infty)$. Specifically, the difference between the speed of convergence as τ increases could be investigated. In this thesis, we refrain from looking into this further.

Next, we will look at a three-dimensional system where the use of the iterative method does not have a significant effect on the speed of convergence. Consider the system of ODEs

$$\begin{cases} \dot{x} &= & y + z, & \qquad x(0) = 2, \\ \dot{y} &= & z, & \qquad y(0) = 1, \\ \dot{z} &= & y, & \qquad z(0) = 1. \end{cases} \tag{4.4}$$

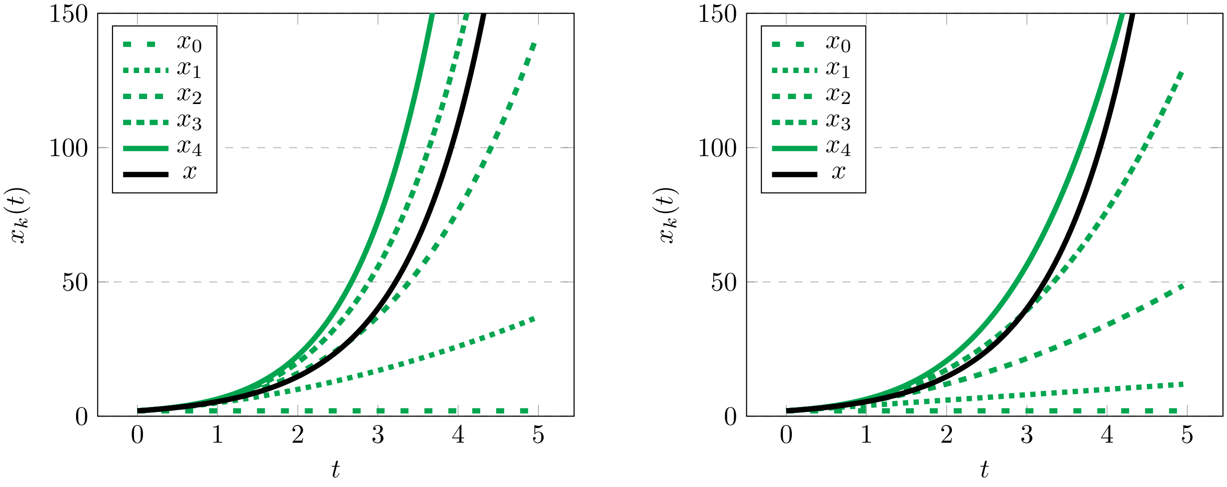


(a) (b)

Figure 4.2: Output of the iterative method for the first 5 iterations, with the solution indicated in black. (a) shows the iterative method $f(t, x_{k+1}, x_k) = x_k x_{k+1}$. (b) shows Picard iteration $f(t, x_{k+1}, x_k) = x_k^2$.

The solution of this system is $(x(t), y(t), z(t)) = (2e^t, e^t, e^t)$. We will consider the restricted iterative method $\dot{x}_{k+1} = y_{k+1} + z_k$, $\dot{y}_{k+1} = z_k$, $\dot{z}_{k+1} = y_k$. Here x and y are in a group, and z is in a separate group. At iteration k, the number of correct Taylor terms for x_k, y_k, and z_k is $k+1$, since the accuracy of y_k and z_k is increased by one term at each iteration. Also, x_k has one more correct term than z_{k-1}.

In Figure 4.3, the graphs of the first 5 iterations of the iterative method and Picard iteration are shown. The iterative method does not converge more quickly than Picard iteration in this case.



(a) (b)

Figure 4.3: The value of x for the first 5 iterations of both the iterative method and Picard iteration of the system (4.4). (a) shows the iterative method. (b) shows Picard iteration.

4.3.2 Analysis of PDEs

We will look at the one-dimensional heat equation $\partial_t u = \partial_{xx} u$. The domain used is $\Omega = (-10, 10)$, with the boundary condition $u|_{\partial\Omega}(t, x) = \sin x$ and initial condition $u|_{\Omega}(0, x) = \sin x$. The time interval on which the PDE is simulated is $[0, 1]$. This PDE is approximated with the system of ODEs

$$\begin{aligned} u_0 &= \sin(-10), \\ \dot{u}_x &= u_{x-1} - 2u_x + u_{x+1}, \qquad 0 < x < n, \\ u_n &= \sin(10). \end{aligned}$$

In this example, a scale of 0.08 units per grid cell is used, which means the number of grid cells is $n = 250$. The number of iterations and the number of grid groups are varied per run. Figure 4.4 shows the output of the program without using grid groups. Because no groups are used, the number of iterations can be set to one.

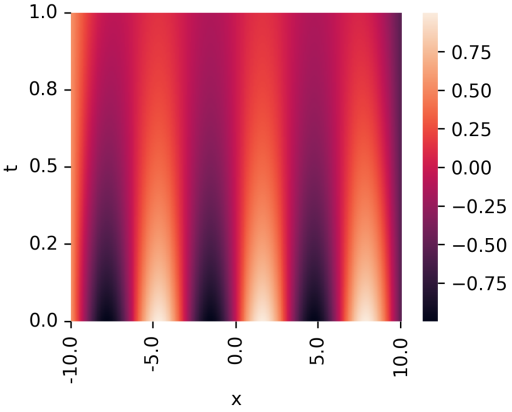
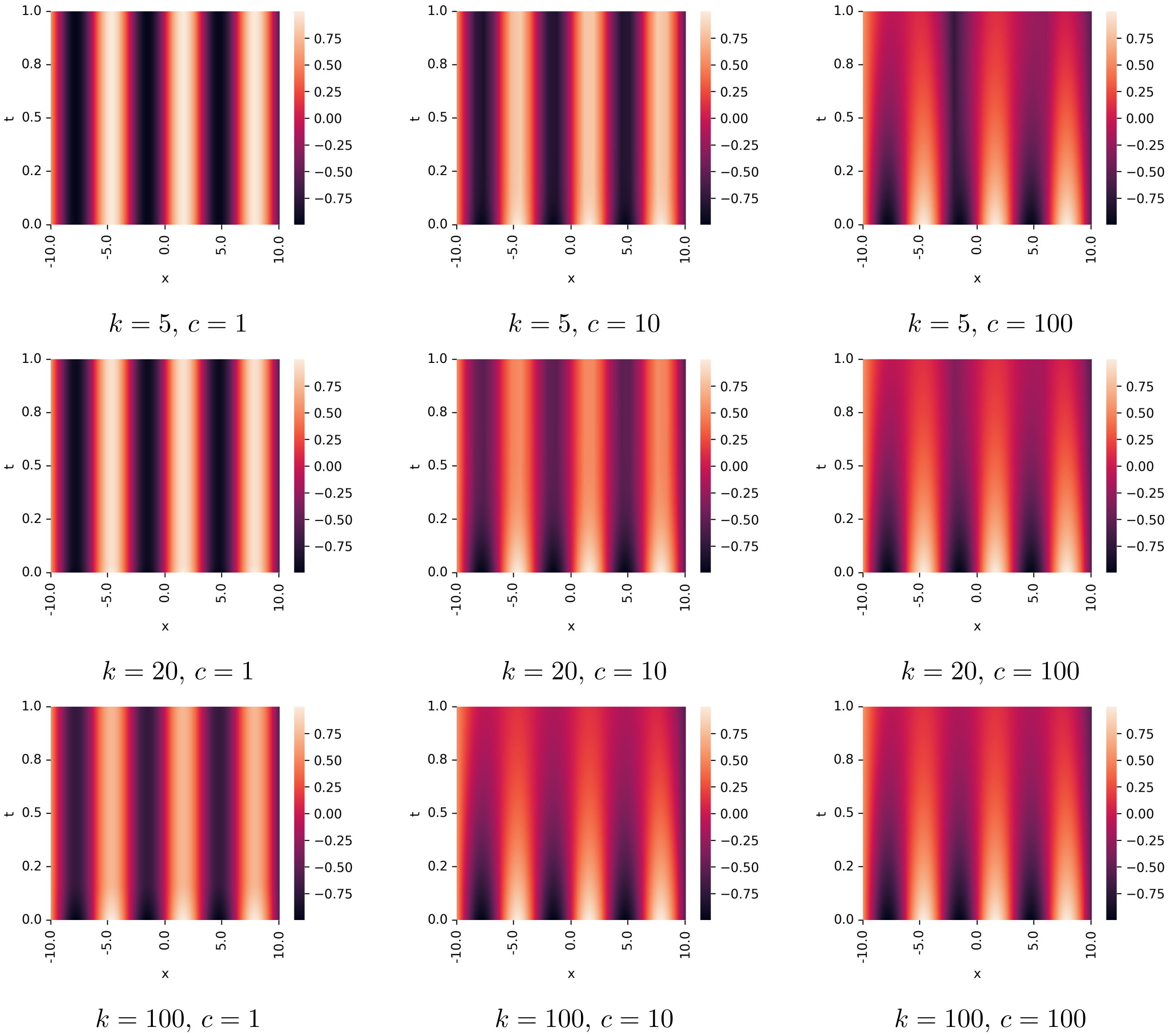


Figure 4.4: The output of PTOC when simulating $\partial_t u = \partial_{xx} u$, without groups.

Figure 4.5 shows the output of the program for varying numbers of iterations and group sizes. It can be seen that the group size has a big impact on the number of iterations it takes to converge to a result that is close to the solution of the large system of ODEs.

For the case $k = 5$, $c = 100$, a darker area can be seen around $x \approx -2$. This is due to a boundary between two groups at this location. As the number of iterations increases, this boundary becomes less pronounced. At these boundaries, the values of the output of the simulation are further from the values of the true solution shown in Figure 4.4.



$k = 5,\ c = 1$ $k = 5,\ c = 10$ $k = 5,\ c = 100$

$k = 20,\ c = 1$ $k = 20,\ c = 10$ $k = 20,\ c = 100$

$k = 100,\ c = 1$ $k = 100,\ c = 10$ $k = 100,\ c = 100$

Figure 4.5: The output of PTOC when simulating $\partial_t u = \partial_{xx} u$, using varying group sizes (c) and iteration counts (k).

Chapter 5

The PTOC tool

The PTOC tool consists of two main parts. The PDE compiler is the main program and is used to convert PDEs in a specific format to ODEs in a similar format. To test the output of this program, another tool is included that can read ODEs from a file and simulate them. There are also several scripts included that can convert the output data of these simulations to visual representations.

5.1 PDE compiler

The PDE compiler is the main program in PTOC. It can convert a PDE in a specific PDE specification format to approximation systems of ODEs. It does this by using an approximation hypergrid, where the grid cell scale is given by the user. The compiler automatically divides the grid into groups based on an abstract component limit given by the user, which ensures that the output systems of ODEs each do not contain more variables than this component limit.

Several arguments can be given to the compiler as command-line arguments, to modify the compiler behavior. The user can give the arguments `--ode` and `--splitter` to run the ODE simulator and ODE splitter respectively. See sections 5.2 and 5.3.5. For the main PDE compiler, the following can be modified using command-line arguments:

- The abstract component limit for the maximum size of grid groups;
- The maximum number of grid cells that can be used in the approximation grid.

Other settings are given in the `.pde` input specification file.

5.1.1 Input PDE format

The input PDE should be in the following format:

```
pde {
    option value;
    option value;
    ...
}
```

Most values have to be expressions, which can contain arithmetic operators, logical operators, special functions such as sin, exp, etc., and derivatives. The following options have to be given by the user:

- **Dimension names:** The names of the dimensions can then be used in other expressions, like the domain definition;
- **Domain:** Defines a space that is the domain of the PDE. The boundary of this domain is automatically determined by the compiler;
- **Domain pivot:** A pivot needs to be given that lies inside the domain. The compiler uses this pivot to generate the connected component of the domain that the pivot is contained in;
- **Grid scale:** The scale of each grid cell, which is the same in each dimension;
- **Equations:** PDE equations with only time derivatives on the left, and only spatial derivatives on the right. Each variable should correspond with exactly one equation;
- **Initial conditions:** For each variable, an initial condition should be given for each time derivative up to (but not including) the time derivate in its equation. For example in the equation $\partial_{tt}u = \partial_{xx}u$, the initial values of u and $\partial_t u$ need to be given;
- **Boundary values:** The boundary values at each point in time should be given for each variable. In this case, the boundary values for the time derivatives should not be given, as these are determined by the PDE compiler. See Section 5.3.2. Note that the boundary does not have to be one cell in width, as the equations can have spatial derivatives of an order higher than two. This means that the range will be more than one in one of the dimensions;
- **Interval:** Should be given for each variable, and all of its derivatives up to (but not including) the time derivative in its equation. This indicates a bound that the solution of the PDE (and resulting ODE approximation) is not expected to exceed. This can be used by an analog computer to determine the scaling of equations;
- **Time:** Simulation time of the PDE;
- **Iteration count:** The number of iterations to generate approximation ODEs for;
- **Emit:** These statements indicate that simulation data from a certain variable should be stored somewhere, for use after the simulation is completed.

Figure 5.1 shows an example of the one-dimensional wave equation $\partial_{tt}u = \partial_{xx}u$ in the PDE specification format. The value of u is the output. Only one iteration is run when simulating this system, which means that it will only get close to a solution if there is no component limit.

```
pde {
    dims [x]; domain x * x < 100; pivot [0]; scale 0.08;
    equation dt(dt(u)) = dx(dx(u));
    init u = sin(x * 3.1415 / 10); init dt(u) = 0;
    boundary u = 0; interval u = [-2, 2]; interval dt(u) = [-2, 2];
    time 20; iterations 1; emit u as u; }
```

Figure 5.1: PDE system specification for the one-dimensional wave equation $\partial_{tt}u = \partial_{xx}u$.

5.1.2 Output ODE format

The ODE specification format is similar to the PDE specification format, with fewer options. The main difference is that variables are now given directly without a time derivative on the left. Instead, an `integ` expression can be used to indicate that a variable is equal to another integrated, with a certain starting value. Figure 5.2 shows an example of the ODE specification format.

```
system {
    var x = integ(5 * x - 3, 1);
    emit x as x; interval x = [-5, 5];
    time 1; }
```

Figure 5.2: ODE system specification for the equation $\dot{x} = 5x - 3, x(0) = 1$.

This example shows the four options that can be given in an ODE specification. Unlike a PDE specification, an ODE specification can contain multiple systems. An emitted variable from a previous system can be referenced in a new system.

5.2 ODE simulation tool

For checking the output of the PDE compiler, an ODE simulation tool is included, which can be used by running the main program with `--ode`. This tool reads an ODE specification file and simulates the systems listed. All data generated from `emit` statements is then output to a CSV file. There are also two scripts included that can be used to visualize this output. One creates graphs of the data. For checking the output of the PDE compiler, a heatmap visualizer is also included. This visualizer sorts the emitted variables that do not start with an underscore in alphabetical order. Then it displays a heatmap of the data with each column representing one variable. Figure 5.3 shows an example of a heatmap visualization for the wave equation.

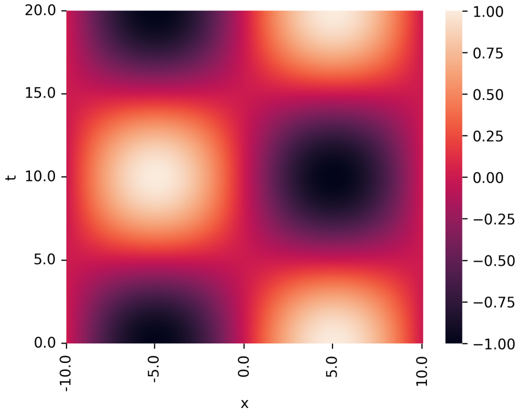


Figure 5.3: A heatmap visualization of a simulation of the wave equation $\partial_{tt}u = \partial_{xx}u$, with domain $(-10, 10)$, grid scale 0.08, initial conditions $u(0; x) = \sin(\pi x/10)$ and $\partial_t u(0; x) = 0$, boundary condition $u(t) = 0$, and time interval $[0, 20]$.

5.3 PDE compiler implementation

This section covers the main stages of the compiler implementation: Parsing the input, transforming the input PDE, generating the grid, generating the ODE, and splitting the ODE. Figure 5.4 shows these different stages.

The compiler and simulation tools are implemented in C++. The graphical tools are written in Python. For the source code and installation, see [9].

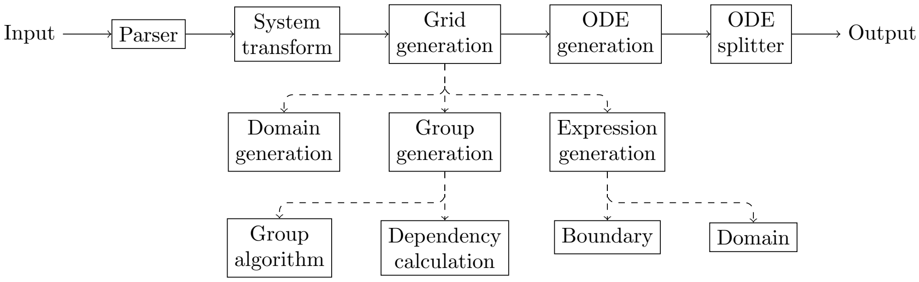


Figure 5.4: Overview of the PTOC pipeline

5.3.1 Parser

The parser converts the input in the PDE specification format to an abstract object containing expressions for all equations, emits, etc. After this object is created, the user input is validated to check for errors or missing entries. Then the system is transformed into a system of PDEs of the form

$$\frac{\partial u_i}{\partial t} = F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \dots, \frac{\partial^{\alpha_m} u_{j_m}}{\partial x^{\alpha_m}}\right).$$

5.3.2 System transform

The input system can have systems of PDEs of the form

$$\frac{\partial^k u_i}{\partial t^k} = F_i\left(t, x, \frac{\partial^{\alpha_1} u_{j_1}}{\partial x^{\alpha_1}}, \dots, \frac{\partial^{\alpha_m} u_{j_m}}{\partial x^{\alpha_m}}\right), \tag{5.1}$$

where the derivatives on the right do not depend on t. As described in Chapter 2, a PDE of the above form with $k = 1$ can be approximated directly by an ODE of the form $\dot{x} = f(t, x)$. When $k > 1$, the system can be transformed to the form (2.3), which is a system of PDEs that have the above form with $k = 1$. By Remark 2.8 the right side of (5.1) could also contain time derivatives. However, this is not implemented in PTOC.

The system transformer rewrites the input system as described in Remark 2.8. The initial and boundary conditions are automatically generated using the method described in Section 2.3. The time derivatives of the boundary conditions are determined symbolically as compile time.

5.3.3 Grid generation

When the system is transformed into a system of the desired form, a hypergrid is generated. Each grid cell holds information about the ODEs that are associated with it. The generation of the grid happens in three stages.

First, the cells that are part of the domain are determined. This is done with the given pivot coordinates. Breadth-First-Search is used to expand the domain from the pivot until the connected component of the domain containing the pivot is determined. After this, the border cells are determined, for which the maximum range in each dimension for the approximations of the derivatives is calculated first.

After this, a hyperrectangular grid that covers this connected component is determined. The cells that are part of the domain are marked. The optimal hyperrectangle heuristic is then used to determine the group division of the grid, taking into account the component limit given by the user. If the component limit is very large, the largest hypercube heuristic is used instead to save computation time. When the grid is divided, the cells that need to be stored between iterations are determined.

After this, the expressions that correspond with the ODEs at each of the grid cells are generated. On the boundary, the dimension names are replaced with the coordinates of the grid cell. Inside the domain, the spatial derivatives present will be replaced with approximations. When generating these approximations the current iteration being generated is taken into account, such that taking values from different grid groups results in taking a value from the previous iteration. After generating the expressions, each cell contains all the required information to generate the combined system of ODEs.

Approximations

The approximations used are based on the symmetric approximations [16]. In PTOC we use a generalization of these approximations for n dimensions and arbitrary orders. For each dimension, the coefficients of the approximation in that dimension are determined. Then the tensor product over all dimensions is taken. The following example illustrates this.

Example 5.1. Consider the PDE $\partial_t u = \partial_{xxy} u$. The approximation of the derivative in the x dimension has coefficients $1/h^2$, $-2/h^2$ and $1/h^2$, as the approximation at a point (x, y) is

$$\frac{\partial^2 u(t,x,y)}{\partial x^2} \approx \frac{u(t,x-h,y) - 2u(t,x,y) + u(t,x+h,y)}{h^2}.$$

We will use the following notation for this approximation:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{h^2} \left\{ \textcircled{1} \text{---} \textcircled{-2} \text{---} \textcircled{1} \right\}$$

In the y dimension we have the approximation

$$\frac{\partial u}{\partial y} \approx \frac{1}{2h} \left\{ \begin{matrix} -1 & 0 & 1 \end{matrix} \right\}$$

Combining these using the tensor product gives

$$\frac{\partial^3 u}{\partial x^2 \partial y} \approx \frac{1}{2h^3} \left\{ \begin{matrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & -2 & 1 \end{matrix} \right\}$$

In the case that the amount of times a derivative is taken is even, the approximation is symmetric. We can determine this approximation with the following formula:

$$\frac{\partial^k u(t,x,y)}{\partial x^k} \approx \frac{1}{h^k} \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} u\left(t, x + h\left(i - \frac{k}{2}\right), y\right).$$

To make the approximation symmetric for k odd, we determine the approximation formula for $k-1$ and subtract it from a version of itself that is shifted by two cells. In this case, an extra factor $\frac{1}{2}$ is needed to compensate for the shift. The formula for this approximation is then

$$\begin{aligned} \frac{\partial^k u(t,x,y)}{\partial x^k} \approx \frac{1}{2h^k} \sum_{i=0}^{k-1} (-1)^{k-1-i} \binom{k-1}{i} \Bigg[& u\left(t, x + h\left(i + 1 - \frac{k-1}{2}\right), y\right) \\ & - u\left(t, x + h\left(i - 1 - \frac{k-1}{2}\right), y\right) \Bigg]. \end{aligned}$$

For the first four derivatives, the approximations are shown below.

$$\frac{\partial u}{\partial x} \approx \frac{1}{2h} \left\{ \begin{matrix} -1 & 0 & 1 \end{matrix} \right\}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{h^2} \left\{ \begin{matrix} 1 & -2 & 1 \end{matrix} \right\}$$

$$\frac{\partial^3 u}{\partial x^3} \approx \frac{1}{2h^3} \left\{ \begin{matrix} -1 & 2 & 0 & -2 & 1 \end{matrix} \right\}$$

$$\frac{\partial^4 u}{\partial x^4} \approx \frac{1}{h^4} \left\{ \begin{matrix} 1 & -4 & 6 & -4 & 1 \end{matrix} \right\}$$

5.3.4 ODE generation

In this stage, all of the information that was generated on the grid is combined. For each iteration, the grid is re-generated. The output ODEs of this stage are not separated by different grid groups, this is done in a separate stage.

5.3.5 ODE splitter

Instead of separating the ODE by the grid groups introduced in the grid generation stage, the output ODE is split up as much as possible. Because of the introduction of the grid groups, it is guaranteed that the size of each ODE after the split is smaller than the component limit. It can however also happen that the ODE is split up further because there are no dependencies between cells.

First, all of the dependencies between single ODEs in the input systems of ODEs are determined. This results in a dependency graph. The strongly connected components of this dependency graph are determined using Kosaraju's algorithm [19]. These components will determine where the ODE is split. After this, the dependencies between strongly connected components are determined. This graph is then topologically sorted. The ODEs are output in the order of the topological ordering. An example of this splitting up of an ODE is given below.

Example 5.2. Consider the ODE specification from Figure 5.6. Figure 5.7 shows the dependency graph between the variables. The variables y and z form a strongly connected component, and this component is dependent on x. This means the output will have two systems and the system containing x will be given first. Figure 5.8 shows the output of the splitter program.

```
system {
    var x = integ(x, 1); var y = integ(x - z, 1); var z = integ(y, 1);
    interval x = [-100, 100]; interval y = [-100, 100];
    interval z = [-100, 100];
    time 5; emit z as z; }
```

Figure 5.6: ODE system specification that can be split up.

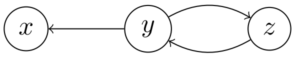


Figure 5.7: Dependency graph of the variables from the system in Figure 5.6.

```
system {
    var x = integ(x, 1); emit x as __v0;
    interval x = [-100, 100]; time 5; }
system {
    var y = integ(__v0 - z, 1); var z = integ(y, 1);
    emit y as __v1; emit z as z; emit z as __v2;
    interval y = [-100, 100]; interval z = [-100, 100];
    time 5; }
```

Figure 5.8: Split specification of the system in Figure 5.6.

Note that the output values of some ODEs need to be stored between iterations because the ODE is split up. This may introduce more values that need to be stored than when using the grid groups directly to split up the grid. However, these values do not need to be stored between iterations. Note that this deleting of values that are no longer necessary is not implemented in the ODE simulator.

Chapter 6

Related Work

Analog circuits have been used to simulate partial differential equations before. However, this research mostly focused on the physical implementation of specific PDEs [12, 14, 16, 21]. In this thesis, a more general approach is provided, which attempts to automate the component-sharing method described by Bekey and Ung [21]. Most of the work on the component-sharing method and simulation of partial differential equations focuses on implementation and speed. This thesis focuses on the correctness of the component-sharing method, as well as a general abstract implementation of a compiler for PDEs on analog computers.

6.1 The extended analog computer

The possibility of simulating PDEs using analog computers was first discussed by Rubel [17]. He concluded that the theoretical model for a general-purpose analog computer (GPAC) is not enough not to be able to simulate PDEs [18]. For this reason, a new concept called the extended analog computer (EAC) was introduced. The EAC was described as a conceptual machine with layers, each layer would be able to produce a broader class of (analytic) functions than the previous layers. This concept was later formalized by considering sets $\mathrm{EAC}_n \subset \mathcal{C}^\omega(\Lambda)$ of computable functions by an EAC in the first n layers [15]. Then $\mathrm{EAC} := \cup_{n\in\mathbb{N}}\mathrm{EAC}_n$ is the set of all functions computable by an EAC.

One of the rules that makes the EAC more general than the GPAC is that it can solve PDEs. Given functions $F_i \in \mathrm{EAC}_{n-1}$, if $f \in \mathcal{C}^\omega(\Lambda)$ is a solution to the set of equations

$$F_i(x, f(x), f^{(\alpha_1)}(x), \ldots, f^{(\alpha_l)}(x)) = 0$$

for all $x \in \Lambda$ with $f^{(\alpha_1)}, \ldots, f^{(\alpha_l)}$ partial derivatives of f, then $f \in \mathrm{EAC}_n$.

Attempts have been made to implement parts of the EAC [14]. However, it is unknown if an EAC can be fully physically implemented. It is not known if the set EAC contains all of $\mathcal{C}^\omega(\Lambda)$, or if there are analytic functions that cannot be produced by an EAC [17]. For this reason, other methods were considered to simulate PDEs.

6.2 The component-sharing method

One such method is to discretize the spatial dimensions but keep the time dimension continuous. This introduces a large system of ODEs, one for each grid cell in the discretized space [21]. An implementation of this discretized method for fourth-order PDEs was realized by Ratier [16]. Because the system of ODEs can become large, an iterative approach was taken by Howe and Hsu [11] to reduce the number of ODEs that needed to be solved in parallel. In this thesis, this iterative method is formalized and several theorems about its convergence are proven. It should be noted that analog computers are a lot more powerful now than when Howe and Hsu published their paper.

In a paper by Bekey and Ung, it is suggested to make the groups of cells that are processed in parallel a certain shape, to reduce the number of grid cell values that need to be stored between iterations [21]. In this thesis, this is further investigated by looking at several heuristics that attempt to minimize the number of border grid cells.

6.3 Compilation for analog computers

Recently, both theoretical and practical research into creating compilers for analog devices has started [1, 2, 3, 5]. Several compilers that targeted simulated analog devices were developed by Archour and Rinard. First came Arco [3], which had fewer capabilities than its successor Jaunt [1]. Later they built on this work by creating a compiler that targeted the HCDCv2 analog device, called Legno [2]. This thesis attempts to build on their work by introducing similar configuration options in the ODE output of the PTOC program, to what the Legno compiler takes as input.

Chapter 7

Conclusions and Further Research

Analog computers can be used to simulate PDEs using a continuous time dimension and a discretization of the spatial dimensions. The main difference with classical digital simulations is that the PDE is first approximated with a system of ODEs. This system is split up into smaller systems that can be simulated by an analog computer. Because values from different groups after this split may be dependent on each other, an iterative method is used. After this, an analog computer can solve these ODEs without the loss of accuracy that floating-point and discrete-time calculations on digital computers have. In this thesis, the theoretical aspects of the conversion from a system of PDEs to a system of ODEs, the iterative method, and the division of groups in the large ODE were discussed. We introduced PTOC, a tool that automatically generates an approximation system of ODEs from a given system of PDEs.

The theory behind which PDEs can be converted to this standard form was discussed. The main result is that PDEs with a higher-order time derivative on the left can be converted to systems of PDEs with a first-order time derivative on the left. This conversion happens automatically in PTOC.

An optimal hyperrectangle heuristic was introduced to divide a grid of points, such that the number of boundary cells is minimized. It was shown that this heuristic performed better than several other heuristics, such as filling the grid with hypercubes. This heuristic was implemented in the PTOC tool, from which an iterative method could be created.

The local and global convergence of this iterative method under different assumptions was covered. The two main theorems proven are: If the function f from the iterative method $\dot{x}_{k+1} = f(t, x_{k+1}, x_k)$ is Lipschitz continuous on some domain Ω, and each x_k maps onto Ω, then the sequence $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to a solution on the interval of definition $[0, \tau]$. Also, if f is continuously differentiable, there exists some $T > 0$ such that $\{x_k\}_{k\in\mathbb{N}}$ converges uniformly to a solution on $[0, T]$. This statement is proven in a similar way to the Picard-Lindelöf theorem.

7.1 Further theoretical research

More research should be done on the conversion from PDEs to the standard form described in Chapter 2. It may even be possible to simulate PDEs that cannot be converted to this standard form, while still taking advantage of analog computations. In this thesis, we specifically considered PDEs where the time dimension is implicit in the physical system the PDE describes. It may be possible to take advantage of one continuous spatial dimension, instead of a continuous time dimension. An example of this could be Laplace's equation $\partial_{xx}u + \partial_{yy}u = 0$, which does not have a time derivative and can therefore not be converted to a PDE in the standard form. A problem with this could be that the domain of such an equation can have a particular shape. Whereas a continuous time dimension would mean that a simulation is run on some fixed interval $[0, \tau]$, converting a spatial to a time dimension would mean this interval would vary based on the position in the domain. On an analog computer, time is simulated as actual time and only flows forward. The shape of the domain could make it difficult to choose a starting position such that everything can be simulated correctly, and the analog computer does not need to simulate backward in time.

The heuristics considered when dividing the grid into groups were all tested on a square grid, which may have a large impact on performance when the domain has a particular shape. Further research could be done to improve the heuristics described to also take advantage of the specific shape of the domain. Also, the input PDE may show chaotic behavior in certain areas of the domain. It may be possible to improve the grid resolution in certain areas, which also has to be taken into account when dividing grid groups since the maximum size of a grid group is now different in different areas of the grid. Choosing areas of the grid that have an increased resolution could be done automatically or by the user. One question that arises from this is if the given PDE can even be approximated using a grid, since a finer grid may not mean the output will be more accurate. In mathematical terms, if $u(t, x)$ is the solution of the input PDE at some point x, and $u_h(t, x)$ is the solution from the approximation system of ODEs at the same point, given the grid scale h, is it true that $\lim_{h \downarrow 0} u_h(t, x) = u(t, x)$? This is a question that can be answered in a more general context of the simulation of PDEs, rather than in the context of analog computers.

An important open question on the iterative method is whether it always converges with the same radius of convergence as the solution of the ODE. For example, in Section 4.3.1 we considered an iterative method that was used to approximate the solution of $\dot{x} = x^2, x(0) = 1$. The solution to this equation is $x(t) = \frac{1}{1-t}$, which exists on $[0, 1)$. It seems that both the iterative method and Picard iteration converge on the interval $[0, 1)$ as well. However, in this thesis, this convergence was only proven on an interval $[0, T]$, for some $T < 1$. The closest result to proving that the solutions converge everywhere is Lemma 4.18, which says that if the sequence of solutions in the iterative method is bounded, it converges uniformly to a solution on an interval $[0, \tau]$. If the iterative method converges to a solution on all intervals $[0, \tau]$, it may be interesting to look into the difference in the speed of convergence as $\tau \to \infty$, and whether the limit of a solution $x(t)$ as $t \to \infty$ can be approximated efficiently. A deeper analysis of the speed of convergence of the iterative method, depending on the size of the groups, could also be done.

In this thesis, the use of the iterative method in the context of simulating PDEs on analog computers is discussed. This method seems to be very inefficient for simulating complex PDEs however. There are other contexts in which large systems of ODEs need to be solved, such as neural ordinary differential equations [7, 22]. The methods described in this thesis could benefit the simulation of these equations for AI applications. One problem with these neural ODEs is that numerical integration significantly impacts the performance of the model [22]. This could be resolved with analog computers.

7.2 Improvements to PTOC

The PTOC tool should serve as an entry point for future compilers that can run on real analog computers. It should be possible to connect PTOC to the Legno [2] analog computer compiler, as the output of PTOC is specifically designed to be close to the input that Lego expects. The output is very general however, hence other compilers may also be connected to PTOC. Several other improvements to the PTOC tool could be made.

One of these improvements is to allow the referencing of lower-order time derivatives on the right side of the PDE. As shown in Chapter 2 this is possible. This would mean a modification to the system transform pipeline stage.

The other configuration options in the input `.pde` file could also be modified. The explicit mention of dimensions could be removed in favor of automatically detecting dimension names from the domain definition. In this case, it may the useful to still give the user an option to give the dimensions explicitly. A heuristic may be introduced that can automatically determine the pivot that is now given by the user. The user would then only need to give the pivot when determining it automatically is impossible, or if the user wants to select a specific pivot. It may also be possible to determine the intervals that the user has to give automatically, by analyzing the input PDE and determining bounds on the variables. It may be possible to use a type system to determine bounds on the values that the solution of the input PDE, or the output ODEs, admit. It may be possible to do this for specific types of PDEs. One could for example consider using logical abstractions to get more information about the given PDE. This could be done using higher-order functions to prove more general statements about specific kinds of PDEs [6, 13]. In addition, sheaves may be used to glue solutions of PDEs together, which may be useful when considering grids with different cell scales [10].

It should be noted that, whenever the input PDE is linear, the output ODE is also linear. It may be more efficient to solve this output ODE using a digital computer, by using the fact that the solution of the equation $\dot{x} = Ax$ with $A \in \mathrm{Mat}(\mathbb{R}, n \times n)$ is the matrix-exponential $x(t) = e^{At}x_0$. A component could be added to PTOC that determines if the input PDE is linear and if so, solves it digitally.

Furthermore, the input of boundary conditions should be changed. Currently, a single boundary function is given for each variable, which is then used as the value of the variable "anywhere outside the domain". However, in many real PDE problems, certain spatial derivatives are given at the boundary of the domain, which can then be used internally to compute the boundary values at grid points that do not border the domain directly. These extra grid points outside the boundary are only required for PDEs with spatial derivatives of an order higher than two.

When generating the grid expressions, as an optimization, automatic simplifications could be added. This would make the simulation of ODEs, both digitally and on analog computers, faster. It would also make the output systems produced more readable for debugging and manual modifications by the user.

The ODE simulator could be improved to simulate the output systems more efficiently. For example, currently, the Euler method is used to simulate the systems of ODEs. However, by using more advanced discrete methods to simulate ODEs the result may be more accurate. Specifically, the method used could be adapted to the type of systems of ODEs that the PTOC tool produces.

Another flaw of the simulator is that all `emit` variables are stored, regardless of whether they are used again in the future. Removing `emit` variables that are not referenced after the current system could improve performance. However, it should be noted that some variables may be wanted by the used as output. Hence this removal of variables should be handled with care. One could consider introducing more complexity to the communication between the output systems of ODEs, such as adding a type system for input and output variables, namespaces for the ODEs, or explicit mentions of which ODEs can use a certain `emit` value, which may improve the readability of the output. However, the current output aims to be able to be easily translated to code an analog computer can read and process, which may be a problem when introducing a more complex structure.

It may also be faster on some analog devices to simulate a system that is as large as possible, instead of many smaller systems. The splitter stage could be modified to allow for combining disjoint systems that together do not exceed the component limit.

PTOC aims to be an entry point to compilers that can compile PDEs to real analog hardware. Together with the theoretical justifications described in this thesis, it could be possible that PDEs can be simulated more efficiently on analog computers in the future, as the accuracy and speed of the hardware increase.

References

[1] Sara Achour and Martin Rinard. "Time Dilation and Contraction for Programmable Analog Devices with Jaunt". In: *SIGPLAN Not.* 53.2 (Mar. 2018), pp. 229–242. ISSN: 0362-1340. DOI: 10.1145/3296957.3173179.

[2] Sara Achour and Martin Rinard. "Noise-Aware Dynamical System Compilation for Analog Devices with Legno". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 149–166. ISBN: 9781450371025. DOI: 10.1145/3373376.3378449.

[3] Sara Achour, Rahul Sarpeshkar, and Martin C. Rinard. "Configuration Synthesis for Programmable Analog Devices with Arco". In: *SIGPLAN Not.* 51.6 (June 2016), pp. 177–193. ISSN: 0362-1340. DOI: 10.1145/2980983.2908116.

[4] Anadigm. *AN120E04 Datasheet Reconfigurable FPAA*. 2003. URL: https://www.anadigm.com/_doc/DS021000-U004.pdf (visited on 06/07/2023).

[5] Ulrich Berger et al. "Programming with Ordinary Differential Equations: Some First Steps Towards a Programming Language". In: *Revolutions and Revelations in Computability*. Vol. 13359. Lecture Notes in Computer Science. Switzerland: Springer International Publishing AG, 2022, pp. 39–51. ISBN: 3031087399.

[6] Flavien Breuvart, Marie Kerjean, and Simon Mirwasser. "Unifying Graded Linear Logic and Differential Operators". In: *8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023)*. Ed. by Marco Gaboardi and Femke van Raamsdonk. Vol. 260. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 21:1–21:21. ISBN: 978-3-95977-277-8. URL: https://drops.dagstuhl.de/opus/volltexte/2023/18005.

[7] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[8] E. Dahlhaus et al. "The Complexity of Multiway Cuts (Extended Abstract)". In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. STOC '92. Victoria, British Columbia, Canada: Association for Computing Machinery, 1992, pp. 241–251. ISBN: 0897915119. DOI: 10.1145/129712.129736.

[9] Dirck van den Ende. *PTOC: PDE to ODE compiler*. Version 1.0. July 2023. URL: https://github.com/dirckvdende/PDE-hybrid-converter.

[10] M.P. Fourman, C.J. Mulvey, and D.S. Scott. *Applications of Sheaves: Proceedings of the Research Symposium on Applications of Sheaf Theory to Logic, Algebra and Analysis, Durham,*

July 9-21, 1977. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1979. ISBN: 9783540348498. DOI: 10.1007/bfb0061811. URL: https://books.google.nl/books?id=owZ8CwAAQBAJ.

[11] Stephen K. T. Hsu and Robert M. Howe. "Preliminary Investigation of a Hybrid Method for Solving Partial Differential Equations". In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: Association for Computing Machinery, 1968, pp. 601–609. ISBN: 9781450378994. DOI: 10.1145/1476589.1476669.

[12] Yipeng Huang et al. "Hybrid Analog-Digital Solution of Nonlinear Partial Differential Equations". In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-50 '17. Cambridge, Massachusetts: Association for Computing Machinery, 2017, pp. 665–678. ISBN: 9781450349529. DOI: 10.1145/3123939.3124550.

[13] Marie Kerjean and Jean-Simon Pacaud Lemay. "Higher-Order Distributions for Differential Linear Logic". In: *Foundations of Software Science and Computation Structures*. Ed. by Mikołaj Bojańczyk and Alex Simpson. Cham: Springer International Publishing, 2019, pp. 330–347. ISBN: 978-3-030-17127-8.

[14] Jonathan W. Mills. "The nature of the Extended Analog Computer". In: *Physica D: Nonlinear Phenomena* 237.9 (2008). Novel Computing Paradigms: Quo Vadis?, pp. 1235–1256. ISSN: 0167-2789. DOI: 10.1016/j.physd.2008.03.041.

[15] Monika Piekarz. "The Extended Analog Computer and Functions Computable in a Digital Sense". In: *Acta Cybern.* 19.4 (Jan. 2010), pp. 749–764. ISSN: 0324-721X.

[16] Nicolas Ratier. "Analog computing of partial differential equations". In: *2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*. 2012, pp. 275–282. DOI: 10.1109/SETIT.2012.6481928.

[17] L.A. Rubel. "The Extended Analog Computer". In: *Advances in Applied Mathematics* 14.1 (1993), pp. 39–50. ISSN: 0196-8858. DOI: 10.1006/aama.1993.1003.

[18] Lee A Rubel. "Some mathematical limitations of the general-purpose analog computer". In: *Advances in Applied Mathematics* 9.1 (1988), pp. 22–34. ISSN: 0196-8858. DOI: 10.1016/0196-8858(88)90004-8.

[19] R. Stephens. *Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C#*. Wiley, 2019, p. 421. ISBN: 9781119575993. URL: https://books.google.nl/books?id=Me6VDwAAQBAJ.

[20] Gerald Teschl. *Ordinary differential equations and dynamical systems*. Vol. 140. Graduate Studies in Mathematics. Providence, R.I: American Mathematical Society, 2012. ISBN: 9780821883280.

[21] Man T. Ung, George A. Bekey, and Michael J. Merritt. "Analysis of Hybrid Computer Interface Errors on the Solution of Partial Differential Equations". In: *Math. Comput. Simul.* 16.3 (July 1974), pp. 47–58. ISSN: 0378-4754. DOI: 10.1016/S0378-4754(74)80031-5.

[22] Aiqing Zhu et al. "On Numerical Integration in Neural Ordinary Differential Equations". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 27527–27547. URL: https://proceedings.mlr.press/v162/zhu22f.html.