DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Time and Space Discretization for One Dimensional PDEs with Application to Climate and Weather Simulations

Linus Hein

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Time and Space Discretization for One Dimensional PDEs with Application to Climate and Weather Simulations

Zeitliche und räumliche Diskretisierung für eindimensionale PDGLs mit Anwendung in Klima- und Wettersimulationen

Author:	Linus Hein
Supervisor:	Prof. Martin Schulz
Advisor:	Dr. Martin Schreiber
Submission Date:	16.09.2019

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 09.09.2019 Linus Hein

Acknowledgments

I want to thank my advisor, Dr. Martin Schreiber, for giving me helpful pointers whenever I thought I was at a dead end. I also want to thank my supervisor, Prof. Dr. Schulz, for allowing me to pursue this subject, even though it is only tangentially related to his main area of research.
Finally, I want to thank my family and friends who have always supported me throughout my studies.

Abstract

Weather and climate models based on the Navier Stokes Equations often employ the hydrostatic assumption whereby one assumes the atmosphere to be in vertical equilibrium between gravity and vertical air pressure. As this assumption affects the simulated physical effects, in this thesis, we relax the hydrostatic assumption, and specifically study the vertical dimension. To this end, we present a tested, modular and flexible prototyping implementation in Python 3 which provides complete access to integration methods and discretization.

Contents

1. Introduction

1.1. Motivation

Some of the great advancements in society that have been enabled by the advent of computers were achieved in the fields of *meteorology* and *climatology*. For example, *meteorology*, or the accurate simulation of short-term weather has become of central economic importance to many industries including agriculture [1], airlines [2], and tourism [3]. *Climatology*, or the study of long-term weather trends has become the key scientific driver of the debate and political action surrounding climate change [4]. Evidently, both of these fields are of central societal importance, and the quality of the underlying scientific models is crucial.

In studying climate and meteorology models, a lot of focus is put on simulations on a large spatial and temporal scale. As one main aim of such models is to make predictions of the real world, the models must simulate time significantly faster than time elapses in the real world. However, simulating the entire planet with all of its small details and physical effects with extreme precision is impossible. Instead, such models often make simplifying assumptions and reduce the spatial and temporal resolution at which to simulate, in order to make computation of the models feasible.

For weather and climate prediction, a lot of these simplifications affect the vertical structure of the atmosphere, i.e. it is simulated with less detail than the horizontal structure. This is justified by reasoning that physical effects in the horizontal dimension are a lot more significant than some short-term effects in the vertical dimension, and should thus be allocated more computational resources. While this line of reasoning enables simulations that are good enough to make many predictions in the real world, it neglects the study of the aforementioned short-term effects in the vertical dimension of the atmosphere. One of the disregarded physical effects, for example, is sound which is essentially a pressure wave propagating through the atmosphere.

For this reason it may be interesting to study just that: the degree to which the omission of short-term effects of the vertical dimension of the atmosphere affect mid- to long-term predictions. Of course, this necessitates accurate simulation of the vertical dimension of the atmosphere, the basics of which we will outline in this thesis. Based on this, we will then showcase an implementation of a simulation of the vertical dimension of the weather system. This is intended as a foundation for others to build upon in

order to look into the effects of omitting short-term vertical effects.

1.2. Methodology

In order to simulate both weather or climate we employ so-called Partial Differential Equations (PDEs). These are mathematical equations which describe the evolution of a system in both time and space, taking only the current state of the weather system as an input. This paradigm harks back to Newton and Leibniz, and many of the fundamental models of today's science are expressed as PDEs. This includes Schrödinger's equation of quantum mechanics, Maxwell's equations of electromagnetics, and the Navier-Stokes equations of fluid dynamics. For meteorology the Navier-Stokes equations (NSE) are of central importance.

However, PDEs only describe in which "direction" a system will evolve in the future, and do not provide any predictions on their own. Thus, they must either be solved analytically[1] (for simple cases), or numerically. Generally, the latter option of numerical approximation on computers is less accurate, and is performed utilizing one of many numerical techniques from the literature. This is the only option for complex systems such as the weather.

In this thesis we use the Runge-Kutta methods which predict a system's evolution by repeatedly simulating small time steps. Each of these small time steps is made by changing the state of the system in the "direction" dictated by PDE.

In order to solve PDEs numerically, in this thesis we adhere to the following four steps:

1. *Modeling*: Finding a system of PDEs that describe a given physical system.

2. *Approximation*: simplifying the system of PDEs by making assumptions about the system. For example, in the small-angle approximation of a pendulum system it is assumed that $\sin(\theta) = \theta$, where θ is an angle close to zero.

3. *Space Discretization*: Inevitably, when simulating a continuous variable in space, such as temperature, it needs to be sampled at some points which means it is discretized. The choice of the location of these points is important and must be considered for every scalar and vector field in the PDE.

4. *Time Discretization/Integration*: Finally, a numerical method for approximating how the system will change, given the simplified PDE and the values of the variables at the sampling locations, must be chosen.

[1] i.e. a closed-form mathematical formula without differential operators is derived that describes the complete evolution of a system over time and space given its initial state

Each of these four steps introduces trade-offs which need to be weighed against one another: In designing a numerical PDE solver (a) accuracy, (b) computation time, and perhaps (c) ease of use and adaptability must be considered. For example recalling Section 1.1, often some accuracy in the vertical dimension is sacrificed for reduced computation time. Generally, these optimizations are domain-specific to the problem at hand.

1.3. Outline

In this thesis we study a highly stylized model of the atmosphere. In particular we put the main focus on the spatial variation in the vertical dimension, i.e. perpendicular to the surface of the Earth.

We will relax the simplifying assumption of *hydrostatic equilibrium* often made in Step 2 above. It posits that the force of gravity balances out the vertical pressure gradient force that results from decreasing air pressure at higher altitudes [5]. In other words, the atmosphere is neither sucked up into the vacuum of space, nor collapsed down to the surface of the planet. While this assumption is evidently sensible on average, it does not take into account local variations that may be of interest (e.g. sound waves). We describe this, together with a short introduction to the NSE and some other simplifications, in Chapter 2, accounting for Step 1 and Step 2 above. Thereafter, in Chapter 3 we describe both spatial and temporal discretization for the NSE which completes Step 3 and Step 4. This concludes the description of the theoretical basis for this stylized model of the atmosphere.

We outline the implementation of the flexible prototyping tool[2] in Chapter 4. Its goal is to permit exploration of the trade-offs between accuracy and computational time, and comparisons between different decisions made in Steps 3 and 4. For this reason we designed the tool with the design principle of modularity. This makes the switch between different design choices as simple as replacing a single component of the software for another. We wrote this tool with the hope that it may be prove useful for other students and researchers in the future. The code comprises approx. 2500 lines of code and is freely available at the Github repository `https://github.com/Linus-H/Vertical-Integration`.

In Chapter 5 we first thoroughly unit-test the implementation of the toolbox to exclude it as a source of errors. Then, utilizing the toolbox, we study two different implementations of the simplified NSE. Finally, in Chapter 6 we summarize the work and propose possible extensions for further research.

[2] written in Python 3

2. Navier Stokes Equations for Atmospheric Simulation

In this chapter we outline the mathematical foundation for the remaining chapters. To this end, in Section 2.1 we define the basic variables and notation. Then, in Section 2.2 we present a quite general form of the Navier Stokes Equations (NSE). They can be used to model the atmosphere which concludes Step 1 (*modeling*).

In Section 2.3.1, we introduce three non-altering rearrangements for the purpose of streamlining computer implementations. Thereafter, in Section 2.3.2 we show two simplifications which do alter the numerical results, including the important hydrostatic assumption. Subsequently, in Section 2.4 we familiarize the reader with a non-altering and generally nonlinear coordinate transformation of the vertical dimension with the purpose of obtaining a better-suited discretization. Then, in Section 2.5 we arrive at the final formulation of the NSE in the non-hydrostatic case. This concludes Step 2 (*approximation*). Specifically, this non-hydrostatic model is studied in following chapters.

In Section 2.6 we present some identities which are helpful for verifying simulation models, as certain properties must be preserved. Last, in Section 2.7 we comment on the boundary conditions that are an integral part of any system of PDEs.

2.1. Variables and Notation

Before one can understand the system of equations that constitute the NSE, the symbols appearing within them must be defined. In the following table we introduce the symbols, their colloquial names, their type, and their SI-unit.

Symbol	Name	Type	SI-unit
t	time	variable	s
$\mathbf{V}_3$	wind speed	vector field	$\frac{m}{s}$
T	temperature	scalar field	K
ρ	density	scalar field	$\frac{kg}{m^3}$
q	specific humidity	scalar field	$\frac{g}{kg}$
p	pressure	scalar field	$\frac{N}{m^2}$
$\mathbf{\Omega}$	angular velocity of Earth	constant vector	$\frac{m}{s}$
$\Phi = gz$	geopotential comprising effects of gravity and centrifugal force	scalar field	$\frac{J}{kg}$
$R \approx 287.0 \frac{J}{kg \cdot K}$	specific gas constant for dry air	scalar constant	$\frac{J}{kg \cdot K}$
$C_p \approx 1000 \frac{J}{kg \cdot K}$	specific heat at constant pressure for dry air	scalar constant	$\frac{J}{kg \cdot K}$
$\mathbf{F}$	source/sink term for momentum	scalar field	$\frac{m}{s^2}$
$\mathbf{Q}$	source/sink term for heat	scalar field	$\frac{J}{mol \cdot K}$
$\mathbf{M}$	source/sink term for specific humidity	scalar field	$\frac{g}{s \cdot kg}$

We calculated the value of R by dividing the ideal gas constant $8.31 \frac{J}{mol \cdot K}$ [6] by the molar mass of dry air $28.96 \frac{g}{mol}$ [7]. The value for specific heat C_p can be found in [8].

Now the only components missing in order to write down the NSE are the differential operators. In the following table A stands in for an arbitrary scalar field. The definition of ∇ is used in later sections and is included for completeness' sake.

Operator Symbol	Definition	Operator Name
$\frac{DA}{Dt}$	$\frac{\partial A}{\partial t} + \mathbf{V}_3 \cdot \nabla_3 A$	material derivative
∇_3	$\begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix}$	3-dimensional nabla-operator
∇	$\begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix}$	2-dimensional nabla-operator

In the following, full standalone versions of the NSE are marked by a box surrounding the equations.

2.2. Navier Stokes Equations

Employing these definitions a general form of the equations can be written as follows [5]:

$$\text{Momentum Equation} \qquad \frac{D\mathbf{V}_3}{Dt} = -2\boldsymbol{\Omega} \times \mathbf{V}_3 - \frac{1}{\rho}\nabla_3 p - \nabla_3 \Phi + \mathbf{F} \tag{2.1}$$

$$\text{Thermodynamic Equation} \qquad \frac{DT}{Dt} = \frac{R}{C_p}\frac{T}{p}\frac{Dp}{Dt} + \frac{\mathbf{Q}}{C_p} \tag{2.2}$$

$$\text{Continuity Equation} \qquad \frac{D\rho}{Dt} = -\rho \nabla_3 \cdot \mathbf{V}_3 \tag{2.3}$$

$$\text{Water Wapor Equation} \qquad \frac{Dq}{Dt} = \mathbf{M} \tag{2.4}$$

$$\text{Equation of State} \qquad p = \rho R T \tag{2.5}$$

Interpretation of the Terms

The material derivatives in the above equations account for the fact that the physical properties of a fluid parcel need to "travel" along with that parcel through space. Imagine, for example, a space filled with a gas which moves to the right at a constant speed. If there happens to be a region of higher density within that gas, this high-density region must move right along with the gas.

Next, we interpret the right-hand sides of the above equations, starting with the Momentum Equation 2.1 [5].

- $-2\mathbf{\Omega} \times \mathbf{V}_3$: this term describes the Coriolis effect which needs to be modeled because the frame of reference is the rotating earth.
- $-\frac{1}{\rho}\nabla_3 p$: this pressure force term describes how air flows from regions of high pressure to regions of low pressure.
- $-\nabla_3 \Phi$: this term describes how air flows from high potential to low potential, i.e. in the direction of gravity.

For more accuracy once could add a term of the form $\frac{\mu}{\rho}\nabla_3^2 \mathbf{V}_3$ accounting for the viscosity [9]. However, dynamic viscosity $\mu \approx 1.7 \cdot 10^{-5} \frac{kg}{sm}$ [10] is very small, so the term can often be ignored.

In the Thermodynamic Equation 2.2, there is only a single term $\frac{R}{C_p}\frac{T}{p}\frac{Dp}{Dt}$ describing how temperature changes with pressure. This relationship dictates the the thermodynamic properties of air.To be more accurate a term of the shape $\frac{\alpha R}{C_p}\nabla_3^2 T$ could be added, to account for thermal conduction. However, for these equations this term can be neglected, because the thermal diffusivity of air $\alpha \approx 2 \cdot 10^{-5} \frac{m^2}{s}$ [10] is sufficiently small.

Lastly, interpreting the Continuity Equation 2.3, it can be helpful to expand the material derivative, in order to highlight how the two similar terms can describe different effects.

$$\begin{aligned} \frac{D\rho}{Dt} &= -\rho \nabla_3 \cdot \mathbf{V}_3 \\ \frac{\partial \rho}{\partial t} + \mathbf{V}_3 \cdot \nabla_3 \rho &= -\rho \nabla_3 \cdot \mathbf{V}_3 \\ \frac{\partial \rho}{\partial t} &= -\mathbf{V}_3 \cdot \nabla_3 \rho - \rho \nabla_3 \cdot \mathbf{V}_3 \end{aligned}$$

The first term accounts for how density travels along with the flow. The second term describes how the density of a gas is reduced if the flow is divergent, i.e. if more material flows away from a point in space than flows in, the amount of material in that point reduces.

In the following, the source/sink terms $\mathbf{F}$, $\mathbf{Q}$ and $\mathbf{M}$ are set to zero in order to study a simplified closed system. Also, the Water Vapor Equation 2.4 is ignored because it becomes trivial when M is set to zero.

Categories of Equations and Variables

The equations can be split into two categories, depending on their function [5]. The first category comprises any equations describing the evolution of a variable over time. Such equations are called *prognostic*, and can be identified by their containing either a material derivative $\frac{D}{Dt}$ or a partial time derivative $\frac{\partial}{\partial t}$. These are the equations which need to be numerically approximated in order to simulate the system.

The second category consists of all remaining equations. They are called *diagnostic*. They do not contain time derivatives, and only describe static relationships between variables at any point in time. In the case of the "raw" NSE, only the Equation of State 2.5 is *diagnostic*, with the remaining equations being *prognostic*.

Conversely, a variable is called *prognostic* if its evolution is described by some *prognostic* equation. Otherwise it is called *diagnostic*.

2.3. Rearrangements and Simplifications

All of the rearrangements and simplifications in this section are based on the book "Fundamentals of Numerical Weather Prediction" by Jean Coiffier [5] unless other sources are cited.

Having written down and interpreted the NSE, in this section we perform the second step of *approximating* them. This is done for multiple reasons, all of which can be boiled down to computational efficiency.

When implementing the original NSE directly, two issues arise. First, the raw equations contain some redundancy, e.g. both ρ and p appear in the equations, even though a linear relationship between the two exists (Equation of State 2.5). Performing this conversion between the two variables during simulation is inefficient. Eliminating of this kind of inefficiency by transforming the equations does not affect the result of simulations. Such rearrangements are called *non-altering*.

Second, depending on what atmospheric effects are of interest, some terms in the NSE are negligible. Computing them would yet again be inefficient. However, simplifying the NSE by ignoring these terms does alter the result of simulations. For this reason such simplifications are called *altering*.

In this section we first make some *non-altering* rearrangements and thereafter introduce some *altering* simplifications. The former have the purpose of condensing the NSE into a more compact and efficient format. The latter only aim to make computation more efficient.

2.3.1. Non-Altering Rearrangements

The main goal of these rearrangements is rewriting the NSE in a form more conducive to applying simplifications later on. Sometimes the rearrangements also have the nice side effect of making computation more efficient. These rearrangements do not change the number of prognostic equations. They can, however, change the number of diagnostic variables, and the type (diagnostic, prognostic) of a variable.

Replacing Occurrences of ρ by p

Having the NSE contain both ρ and p is redundant. In the NSE the only diagnostic equation containing ρ is the Continuity Equation 2.3, as ρ in the Momentum Equation 2.1 can simply be replaced with 2.5. Using the Equation of State 2.5 and the Thermodynamic Equation 2.2, yields:

$$\begin{aligned}\frac{D\rho}{Dt} &\overset{(2.5)}{=} \frac{D\frac{p}{RT}}{Dt} = \frac{1}{RT}\frac{Dp}{Dt} - \frac{p}{RT^2}\frac{DT}{Dt} \\ &\overset{(2.2)}{=} \frac{1}{RT}\frac{Dp}{Dt} - \frac{1}{C_pT}\frac{Dp}{Dt} = \frac{1}{T}\frac{C_p - R}{RC_p}\frac{Dp}{Dt}\end{aligned}$$

Now, inserting this into the Continuity Equation 2.3, we get:

$$\begin{aligned}\frac{1}{T}\frac{C_p - R}{RC_p}\frac{Dp}{Dt} &= \frac{D\rho}{Dt} = -\rho\nabla_3 \cdot \mathbf{V}_3 \\ \frac{1}{T}\frac{C_p - R}{RC_p}\frac{Dp}{Dt} &\overset{(2.5)}{=} -\frac{p}{RT}\nabla_3 \cdot \mathbf{V}_3 \\ \frac{Dp}{Dt} &= -\frac{p}{1 - \frac{R}{C_p}}\nabla_3 \cdot \mathbf{V}_3\end{aligned}$$

This turns ρ into a purely diagnostic variable, and elevates p to the status of a prognostic variable.

Splitting into Vertical and Horizontal Parts

For the second rearrangement one must look at $\Phi = gz$. The only occurrence of Φ is in the Momentum Equation 2.1 in the form of $-\nabla_3\Phi = \begin{pmatrix}0 & 0 & -g\end{pmatrix}^T$. This implies that Φ only affects the vertical component of $\mathbf{V}_3$. For this reason we split $\mathbf{V}_3$ (and thus the NSE) into its vertical and horizontal components. To this end, we define two new variables $\mathbf{V} \in \mathbb{R}^2$ and $w \in \mathbb{R}$, representing horizontal and vertical winds respectively. Additionally, $\boldsymbol{k}$ is the vertical unit vector (always perpendicular to the surface of the Earth), and $f = 2\Omega\sin\phi$ is the Coriolis parameter. Assuming gravity

only to act vertically and being constant across the atmosphere, we write $-\nabla_3\Phi = -g\boldsymbol{k}$. Utilizing these definitions to replace $\mathbf{V}_3$, we arrive at the following equation system:

$$\begin{aligned}
\frac{D\mathbf{V}}{Dt} &= -f\boldsymbol{k}\times\mathbf{V} - \frac{RT}{p}\nabla p \\
\frac{Dw}{Dt} &= -\frac{RT}{p}\frac{\partial p}{\partial z} - g \\
\frac{DT}{Dt} &= \frac{R}{C_p}\frac{T}{p}\frac{Dp}{Dt} \\
\frac{Dp}{Dt} &= -\frac{p}{1-\frac{R}{C_p}}\left(\nabla\cdot\mathbf{V} + \frac{\partial w}{\partial z}\right)
\end{aligned}$$

Using $\ln p$ as a Prognostic Variable

Looking at the NSE after replacing ρ by p and splitting the equations into vertical and horizontal parts, p always occurs in the patterns $\frac{1}{p}\nabla p$, $\frac{1}{p}\frac{Dp}{Dt}$, or $\frac{1}{p}\frac{\partial p}{\partial t}$. In order to further decrease computational intensity, it would be desirable not to calculate $\frac{1}{p}$. This leads to a common transformation of the NSE, namely the replacement of p by $\ln p$, making $\ln p$ a prognostic and p a diagnostic variable. One can observe the effect of this transformation by taking the derivative of $\ln p$, w.r.t. some variable ξ, and applying the chain rule:

$$\frac{d\ln p}{d\xi} = \frac{1}{p}\frac{dp}{d\xi}$$

Note that the right side of this equation is the pattern just observed. Applying the above identity to the NSE results in the following set of equations:

$$\begin{aligned}
\frac{D\mathbf{V}}{Dt} &= -f\boldsymbol{k}\times\mathbf{V} - RT\nabla\ln p \\
\frac{Dw}{Dt} &= -RT\frac{\partial\ln p}{\partial z} - g \\
\frac{DT}{Dt} &= \frac{RT}{C_p}\frac{D\ln p}{Dt} \\
\frac{D\ln p}{Dt} &= -\frac{1}{1-\frac{R}{C_p}}\left(\nabla\cdot\mathbf{V} + \frac{\partial w}{\partial z}\right)
\end{aligned}$$

2.3.2. Altering Simplifications

In the previous section we transformed the NSE into a more practical form, without affecting the result of (theoretical) simulations. In this section we further simplify the

NSE by making certain assumptions about the prognostic variables. All of the following simplifications result in one less prognostic equation and thus one less prognostic variable. Having one less prognostic equation usually reduces computational efforts. However, the simulation results are altered by applying these simplifications.

Hydrostatic Assumption

The hydrostatic assumption is still relevant in current weather prediction methods [11]. However, it is neither implemented nor further studied in this thesis. For these reasons we only briefly outline it in this section.

When simulating a region of the atmosphere, often the width of the region considered is a lot larger than its height. Hence, horizontal effects become dominant and [5],[12] claim that it is sufficient to approximate the vertical dimension. To this end, it is commonly assumed that the force of gravity $-\rho g$ and the vertical of the pressure gradient force $-\frac{\partial p}{\partial z}$ directly cancel each other out, i.e. their sum equals zero [5]. This results in the hydrostatic assumption:

$$\frac{\partial p}{\partial z} = -\rho g \tag{2.6}$$

As gravity and pressure gradient force are the only effects affecting the evolution of vertical wind speed w, this we get

$$\frac{Dw}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial z} - g = 0.$$

This removes the Vertical Momentum Equation from the ranks of the prognostic equations, replacing it with the diagnostic hydrostatic assumption. The resulting set of equations is:

$$\begin{aligned}
\frac{\partial p}{\partial z} &= -\rho g \\
\frac{D\mathbf{V}}{Dt} &= -f\boldsymbol{k} \times \mathbf{V} - \frac{RT}{p}\nabla p \\
\frac{DT}{Dt} &= \frac{R}{C_p}\frac{T}{p}\frac{Dp}{Dt} \\
\frac{Dp}{Dt} &= -\frac{p}{1-\frac{R}{C_p}}\left(\nabla \cdot \mathbf{V} + \frac{\partial w}{\partial z}\right)
\end{aligned}$$

Due to copyright, please refer to
http://wxguys.ssec.wisc.edu/wp-content/uploads/2019/03/WeatherModel.png
for the visualization.

Figure 2.1.: Visualization of vertical column from [13]

Ignoring Horizontal Winds

In this thesis, $\mathbf{V}$ as a prognostic variable is not of interest. Instead, we want to focus on the vertical part of the NSE. Thus, one straightforward simplification to remove $\mathbf{V}$ from the equations is the assumption $\mathbf{V} = \frac{D\mathbf{V}}{Dt} = 0$. This turns the Horizontal Momentum Equation into a diagnostic equation stating that the horizontal gradient of pressure is zero. Starting from the equation system derived in Section 2.3.1 as a baseline, the non-hydrostatic equation system resulting from this assumption is:

$$
\begin{aligned}
\nabla p &= 0 \\
\frac{Dw}{Dt} &= -\frac{RT}{p}\frac{\partial p}{\partial z} - g \\
\frac{DT}{Dt} &= \frac{R}{C_p}\frac{T}{p}\frac{Dp}{Dt} \\
\frac{Dp}{Dt} &= -\frac{p}{1-\frac{R}{C_p}}\frac{\partial w}{\partial z} \\
p &= \rho RT
\end{aligned}
$$

From the lack of horizontal gradients in this equation system, we can deduce that there is no interaction between any columns of the atmosphere. By column of the atmosphere we mean a space described by a rectangular base shape that expands from the surface of the planet to the top of the atmosphere, as can be seen in Fig. 2.1. Because all columns must be the same, knowing about one column is equivalent to knowing about the entire atmosphere. For this reason it is sufficient to view a single column in isolation, when assuming $\mathbf{V} = \frac{D\mathbf{V}}{Dt} = 0$.

2.4. Alternative Vertical Coordinate

In the second equation system of Section 2.3.2, we can only specify the vertical boundaries by fixing the vertical lower and upper limit. In other words, we must manually choose the relative heights to sea level of the lower bound z_{bottom} and the upper bound z_{top}. In order to simulate a column (Fig. 2.1) of the atmosphere, it would be desirable

for z_{bottom} to be located at the planetary surface, and z_{top} at the top of the atmosphere. However, the surface is not flat, meaning the value of z_{bottom} is dependent on location. Also, there is no clear end to the atmosphere, meaning there is no obvious single value z_{top} at which to stop simulating.

Considering these problems, it would be advantageous to measure height by a scale other than meters relative to sea level[1]. As is commonly done in the literature [14], we instead define a placeholder variable s for measuring height which may in general be a nonlinear and continuously differentiable transformation of the height z. In this way, any possible scale for height can be plugged in with ease.

Yet again the interested reader can find all formulas in this section in the book "Fundamentals of Numerical Weather Prediction" [5].

Identities to get from z to s

Switching from z to s as a vertical coordinate affects all differential operators in the NSE.We begin by rewriting the horizontal derivative w.r.t. x or y. In the NSE one always implicitly assumes that derivatives w.r.t. x and y are calculated at constant height z. This can be denoted by a subscript $\left(\frac{\partial A}{\partial x}\right)_z$ (where A is a placeholder for any scalar or vector field). We are interested in $\left(\frac{\partial A}{\partial x}\right)_s$, i.e. the partial derivative of A w.r.t. x at constant s. According to [14] one can write this as (for y analogously):

$$\left(\frac{\partial A}{\partial x}\right)_s = \left(\frac{\partial A}{\partial x}\right)_z + \frac{\partial A}{\partial z}\left(\frac{\partial z}{\partial s}\right)_s$$

To interpret this term, imagine the two-dimensional surface comprising all points in space with the property $s = s_0$. As s is a measurement of height, for every vertical column specified by (x, y) there is exactly one point with the property $s = s_0$.

One can interpret $\left(\frac{\partial A}{\partial x}\right)_s$ as the change in the value of A when moving in the direction of x and changing height z in order stay on the surface $s = s_0$. Infinitesimally speaking, we can decompose this change in value of A into two parts. First, the change of A due to moving in the direction of x at a constant height z: $\left(\frac{\partial A}{\partial x}\right)_z$. Second, the change in A due to having to move vertically in order to stay on the surface $s = s_0$. The magnitude of this term depends on both how much vertical movement (change in z) was necessary $\left(\frac{\partial z}{\partial s}\right)_s$, and on how much A changes with z: $\frac{\partial A}{\partial z}$. Using this identity we define a new

[1] in the hydrostatic system, for example, pressure can be a vertical coordinate [14]

horizontal derivative operator ∇_s:

$$\nabla_s A = \nabla_z A + \frac{\partial s}{\partial z}(\nabla_s z)\frac{\partial A}{\partial s} \tag{2.7}$$

Exploiting Leibniz's notation, we write the replacement for vertical spatial derivatives as follows:

$$\frac{\partial A}{\partial z} = \frac{\partial s}{\partial z}\frac{\partial A}{\partial s} \tag{2.8}$$

Finally, as the material derivative was not dependent on the vertical coordinate system, according to [14] we can write (with $\dot{s} = \frac{ds}{dt}$ being a generalized version of vertical wind speed, i.e. the distance ds along the height measurement s a fluid parcel covers in a time dt):

$$\frac{D}{Dt} = \left(\frac{\partial}{\partial t}\right)_s + \mathbf{V}\cdot\nabla_s + \dot{s}\frac{\partial}{\partial s} \tag{2.9}$$

Identity to get from $\frac{\partial}{\partial z}$ to $\frac{\partial}{\partial s}$

There is an unknown term $\frac{\partial s}{\partial z}$ in Eq. 2.8. To find an identity for this term we define a new variable: hydrostatic pressure π. When not making the hydrostatic assumption, this step may seem a little paradoxical, though even in that case, the definitions hold true.

Hydrostatic pressure π has the properties:

$$\begin{aligned}
\frac{\partial \pi}{\partial z} &= -\rho g = -\frac{p}{RT}g \\
\pi(z) &= \int_\infty^z \rho g dz' \\
\Rightarrow \frac{\partial s}{\partial z} &= \frac{\partial s}{\partial \pi}\frac{\partial \pi}{\partial z} = -g\rho\left(\frac{\partial \pi}{\partial s}\right)^{-1} = -g\frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1}
\end{aligned} \tag{2.10}$$

Now introducing $\frac{\partial \pi}{\partial s}$ as a new prognostic variable, Eq. 2.8 becomes:

$$\begin{aligned}
\frac{\partial A}{\partial z} &= \frac{\partial s}{\partial z}\frac{\partial A}{\partial s} \\
&\overset{2.10}{=} -g\frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial A}{\partial s}
\end{aligned} \tag{2.11}$$

New Prognostic Equations

Now that $\frac{\partial \pi}{\partial s}$ is a new prognostic variable, it needs its own prognostic equation which we obtain by starting from the Continuity Equation 2.3. Utilizing the above identities, it can be shown that the following equations hold [5].

$$\begin{aligned}
\text{original equation:} \quad & \frac{d}{dt}\left(\ln\rho\right) + \nabla_z \cdot \mathbf{V} + \frac{\partial w}{\partial z} = 0 \\
& \frac{d}{dt}\left(\ln\left(\rho\frac{\partial z}{\partial s}\right)\right) + \nabla_s \cdot \mathbf{V} + \frac{\partial \dot{s}}{\partial s} = 0 \\
& \frac{\partial}{\partial t}\left(\rho\frac{\partial z}{\partial s}\right) = -\nabla_s \cdot \left(\rho\frac{\partial z}{\partial s}\mathbf{V}\right) - \frac{\partial}{\partial s}\left(\rho\frac{\partial z}{\partial s}\dot{s}\right)
\end{aligned}$$

Utilizing, $\rho\frac{\partial z}{\partial s} \overset{(2.10)}{=} -\frac{1}{g}\frac{\partial \pi}{\partial s}$, we finally get a diagnostic equation for $\frac{\partial \pi}{\partial s}$.

$$\frac{\partial}{\partial t}\left(\frac{\partial \pi}{\partial s}\right) = -\nabla_s \cdot \left(\frac{\partial \pi}{\partial s}\mathbf{V}\right) - \frac{\partial}{\partial s}\left(\frac{\partial \pi}{\partial s}\dot{s}\right)$$

Integrating this equation from s_{top} to s also yields a diagnostic equation for $\dot{s}$:

$$\dot{s}\frac{\partial \pi}{\partial s} = -\int_{s_{top}}^{s} \nabla_s \cdot \left(\frac{\partial \pi}{\partial s}\mathbf{V}\right) ds' + \frac{\partial \pi}{\partial \pi_{\text{bottom}}} \int_{s_{top}}^{s_{bottom}} \nabla_s \cdot \left(\frac{\partial \pi}{\partial s}\mathbf{V}\right) ds$$

2.5. Non-Hydrostatic Navier Stokes Equations

In this section we apply the alternative coordinate system from Section 2.4 to the non-hydrostatic NSE arrived at in Section 2.3.2. We then use this to derive the equation system which is employed in the remainder of this thesis.

Utilizing the identities from Section 2.4, the Thermodynamic Equation 2.2 becomes:

$$\begin{aligned}
\frac{DT}{Dt} &= \frac{R}{C_p}\frac{T}{p}\frac{Dp}{Dt} \\
\frac{\partial T}{\partial t} &\overset{(2.9)}{=} -\mathbf{V} \cdot \nabla_s T - \dot{s}\frac{\partial T}{\partial s} + \frac{RT}{C_p}\frac{D\ln p}{Dt}
\end{aligned}$$

Adopting $\frac{\partial \pi}{\partial s}$ as a new prognostic variable, the Vertical Momentum Equation can be

written as:

$$\begin{aligned}\frac{Dw}{Dt} &= -\frac{RT}{p}\frac{\partial p}{\partial z} - g \\ &\overset{(2.11)}{=} g\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial p}{\partial s} - g \\ &= -g\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right)\end{aligned}$$

In a similar fashion, the equation for pressure becomes (by exploiting $\Phi = zg$ in the last line):

$$\begin{aligned}\frac{D\ln p}{Dt} &= -\frac{1}{1-\frac{R}{C_p}}\left(\nabla_z \cdot \mathbf{V} + \frac{\partial w}{\partial z}\right) \\ &\overset{(2.7\&2.8)}{=} -\frac{1}{1-\frac{R}{C_p}}\left(\nabla_s \cdot \mathbf{V} - \frac{\partial s}{\partial z}(\nabla_s z)\cdot\frac{\partial \mathbf{V}}{\partial s} + \frac{\partial s}{\partial z}\frac{\partial w}{\partial s}\right) \\ &\overset{(2.10)}{=} -\frac{1}{1-\frac{R}{C_p}}\left(\nabla_s \cdot \mathbf{V} + \frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1}(\nabla_s \Phi)\cdot\frac{\partial \mathbf{V}}{\partial s} - g\frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s}\right)\end{aligned}$$

Now making the assumption that horizontal winds are zero, i.e. $\mathbf{V} = 0$, the above relation necessitates that $\dot{s} = 0$. Setting $\dot{s} = 0$ and $\mathbf{V} = 0$ in all of the equations results in the following equation system which is employed in the remainder of this thesis and thus marked by a double box:

$$\boxed{\begin{aligned}\frac{\partial w}{\partial t} &= -g\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right) \\ \frac{\partial \ln p}{\partial t} &= \frac{g}{1-\frac{R}{C_p}}\frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s} \\ \frac{\partial T}{\partial t} &= \frac{RT}{C_p}\frac{\partial \ln p}{\partial t} \\ \frac{\partial}{\partial t}\left(\frac{\partial \pi}{\partial s}\right) &= 0\end{aligned}}$$

Note that for $\mathbf{V} = \dot{s} = 0$, the material derivative and partial time derivative are the same:

$$\frac{D}{Dt} = \left(\frac{\partial}{\partial t}\right)_s + \mathbf{V}\cdot\nabla_s + \dot{s}\frac{\partial}{\partial s} = \left(\frac{\partial}{\partial t}\right)_s$$

2.6. Useful Identities

2.6.1. Calculating z from s

Having rewritten the NSE in generic s-coordinates, in this section we define the s-coordinate that we are going to use. One way of defining the vertical coordinate, according to Laprise [15] is implicitly, by defining two functions f and h (with some reference pressure π_0, and hydrostatic pressure at the bottom π_{bottom}):

$$\pi(s) = f(s)\pi_0 + h(s)\pi_{bottom}$$

Working with this equation, and integrating Eq. 2.10 with respect to s yields:

$$z(s) = z_{bottom} + \frac{R}{g}\int_s^{s_{bottom}} \frac{T}{p}\left(\pi_0\frac{\partial f}{\partial s} + \pi_{bottom}\frac{\partial h}{\partial s}\right) ds' \tag{2.12}$$

Without loss of generality we assume that $s = 0$ holds at the top of the system and that $s = 1$ is true at the bottom, from which it follows that:

$$\pi(1) = \pi_{bottom} \Rightarrow f(1) = 0, h(1) = 1$$

Also, assuming pressure at the top of the atmosphere to be zero:

$$\pi(0) = 0 \Rightarrow f(0) = 0, h(0) = 0$$

Other than these restrictions, we can choose f and g arbitrarily, as long as $\frac{\partial \pi}{\partial s} = \left(\pi_0\frac{\partial f}{\partial s} + \pi_{bottom}\frac{\partial h}{\partial s}\right)$ is strictly positive for $s \in [0;1]$. This ensures that $z(s)$ is monotonically decreasing, making substitutions of z by s in integrals valid.

2.6.2. Conservation Properties

To verify implementations of these formulas later, it is useful to find quantities which must stay constant over time. Two such quantities are mass and energy. One measure of mass in a given column of the atmosphere (see Fig. 2.1)[2] is given by the following integral:

$$\int_{z_{bottom}}^{z_{top}} \rho(z)dz$$

[2]space described by a rectangular base shape that expands from the surface of the planet to the top of the atmosphere

Exploiting Eq. 2.10 to substitute z by s yields:

$$\int_{s_{top}}^{s_{bottom}} \rho(s)\frac{RT}{pg}\left(\frac{\partial\pi}{\partial s}\right)ds = \int_{s_{top}}^{s_{bottom}} \frac{p}{RT}\frac{RT}{pg}\left(\frac{\partial\pi}{\partial s}\right)ds = \int_{s_{top}}^{s_{bottom}} \frac{1}{g}\left(\frac{\partial\pi}{\partial s}\right)ds \tag{2.13}$$

From the equation system we know that both $\left(\frac{\partial\pi}{\partial s}\right)$ and g are constant over time. This leads to the conclusion that the mass in the observed domain is constant over time in this equation system.

The second constant, energy, can be split up into three separate components ([16] page 45): internal energy $\rho C_v T$, kinetic energy $\frac{\rho}{2}w^2$, and geopotential energy $\rho g z$. The expression for energy density E (i.e. energy per volume) then becomes:

$$E = (C_v T + \frac{1}{2}w^2 + gz)\rho$$

Integrating this over the entire simulated column and yet again exploiting Eq. 2.10 yields energy per area E':

$$\begin{aligned} E' &= \int_{z_{bottom}}^{z_{top}} (C_v T + \frac{1}{2}w^2 + gz)\frac{p}{RT}dz \\ &= \int_{s_{top}}^{s_{bottom}} (C_v T + \frac{1}{2}w^2 + gz(s))\frac{p}{RT}\frac{RT}{pg}\left(\frac{\partial\pi}{\partial s}\right)ds \\ &= \int_{s_{top}}^{s_{bottom}} \frac{1}{g}(C_v T + \frac{1}{2}w^2 + gz(s))\left(\frac{\partial\pi}{\partial s}\right)ds \end{aligned} \tag{2.14}$$

Assuming s_{top} and s_{bottom} are constant, we have shown that the change of energy per area over time can be written as follows (the full derivation can be found in Appendix A):

$$\frac{\partial E'}{\partial t} = w(s_{bottom})p(s_{bottom}) - w(s_{top})p(s_{top}) \tag{2.15}$$

This equation states that the change of energy in a column is proportional to the amount of air ($\sim p$) flowing over the boundaries ($\sim w$). However, as the mass in the observed domain is constant, the mass of air flowing over the boundaries still needs to be contained by the domain. Therefore the domain must grow, i.e. either s_{top} or s_{bottom} must change. However, this change of s_{top} or s_{bottom} would break the premise for Eq. 2.15. In order to prevent this from happening, i.e. to enforce the premise, the right hand side must be kept at zero which in turn calls for restrictions of w and p at the boundaries.

2.7. Boundary Conditions

From the above discussion, it follows that some restrictions must be put in place for the values of the variables at the boundaries of the observed domain, in order to avoid errors.

2.7.1. Vertical Wind

Considering the lower boundary which is placed on the planet's surface, it makes sense to set vertical wind to $w_{bottom} = 0$. No air can move into or out of the ground.

The choice of w at the upper boundary is less constrained. On the one hand, it can make sense to set $w_{top} = 0$. This, together with the choice of $w_{bottom} = 0$, sets the equation for change in energy Eq. 2.15 is zero, ensuring energy conservation. On the other hand, setting $w_{top} = 0$ corresponds to a wall at the top of the atmosphere[3]. Of course this is not true to reality.

The other option is to let w_{top} change freely according to its evolution equation which depends on $\frac{\partial p}{\partial s}|_{s_{top}}$ which itself depends on the boundary condition of p.

2.7.2. Pressure and Density

In this section p and ρ are used interchangeably, as they are connected by the ideal gas law $p = \rho RT$. At the lower boundary there is no restriction to the density, so it is described by its evolution equation. We can infer this by looking at the equation for change in energy Eq. 2.15. The previous section set $w_{bottom} = 0$, so p_{bottom} has no influence on energy conservation anymore.

As discussed above, the upper boundary of p is connected to the upper boundary of w. In case w_{top} evolves according to its diagnostic equation, $\frac{\partial p}{\partial s}|_{s_{top}}$ must be chosen. The most natural formulation is to assume that p is continued smoothly over the upper boundary, and then calculating $\frac{\partial p}{\partial s}|_{s_{top}}$ from that. However, this would violate the conservation of energy[4]. The other option is to postulate $p_{top} = 0$ which is generally not smooth, and results in discontinuous/inaccurate values of $\frac{\partial p}{\partial s}|_{s_{top}}$.

For this reason (and for simplicity's sake) in the remainder of this thesis we make the (physically inaccurate) assumption of $w_{top} = 0$, as this allows for p_{top} to change freely without violating energy conservation.

As mentioned above, setting $w_{top} = 0$ is physically inaccurate. Some more sophisticated models, e.g. [17],[18], employ sponge-layers to deal with this problem.

[3] This is analogous to the reasoning for setting $w_{bottom} = 0$ to simulate a hard wall at the bottom.

[4] or at least our definition of energy

2.7.3. Temperature

For temperature T no restrictions need to be put in place for the boundary conditions, because T affects neither energy conservation nor mass conservation. This means that T is also described by its usual evolution equation at the boundaries.

3. Discretization

Having discussed the NSE and the different kinds of rearrangements and simplifications, we completed the first two steps of *modeling* and *approximating*. Next, we address the problem of simulating the NSE on computers with finite memory and computational capacity. To this end, we must discretize all the continuous spatial and temporal terms in the NSE, in order to make the memory and time required for computation finite. These discretizations account for Step 3 (*Space Discretization*) and Step 4 (*Time Discretization*) of solving a PDE numerically.

The meaning of discretization can be seen by looking at one of the equations to be discretized:

$$\frac{\partial w}{\partial t} = -g\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right)$$

First, spatial derivatives and other spatial differential operators like $\frac{\partial p}{\partial s}$ must be discretized, i.e. they must be approximated. This must be done using a finite number of samples of p around the point where the derivative is to be calculated.

Second, w and p must be discretized themselves. In the real world these are continuous in space. However, when computing, we can only represent them with a finite number of samples. Usually, every variable in a differential equation is assigned to a grid, and at every node of that grid a measurement of the variable is stored.

Third, we must discretize the time variable in $\frac{\partial w}{\partial t}$. To be more specific, we need to reverse the time derivative, i.e. integrate it, in order to gain the real value of w from the diagnostic equation for $\frac{\partial w}{\partial t}$.

In summary, we assume the state of the system at time t to be represented by a number of variables stored in the nodes of a grid. Next we evaluate the value of the differential equation at those grid nodes. We approximate any spatial derivative required for this through its respective discrete spatial derivative. For these discrete spacial derivatives we only make use of the variable values at the grid points. To approximate the state of the system at some other discrete time $t + \Delta t$, we then input the value of the differential equation to an integrator which reverses the time derivative.

Overview

In Section 3.1 we address the discretization of differential operators. In Section 3.2 we discuss the important issue of discretization of prognostic variables onto a regular grid. In Section 3.3 we present the integration operators which we implemented for this thesis.

3.1. Discretization of Differential Operators

In this section we present two methods for approximating spatial derivatives using only samples at discrete spatial locations. The first method is called finite differences and approximates the derivative at a location by only looking at samples close to that location. While not always the most accurate, this method makes no assumptions[1] about the function it is taking the derivative of.

The second method (sometimes called spectral) takes a detour to the frequency domain to approximate the derivative. While this method is highly accurate when applicable, it assumes the boundary conditions to be periodic. This is not the case for the vertical dimension of weather simulation, hence we cannot employ this method for simulating the vertical dimension of the NSE. We present it nonetheless, because it plays a central role for numerical weather prediction in the horizontal dimension.

3.1.1. Finite Differences

In order to derive the commonly known (e.g. refer to [19]) finite difference operators, we must introduce the Taylor Series. It approximates a function around a given point with polynomials and spatial derivatives as follows.

Let $f : \mathbb{R} \times \mathbb{R}^q \to \mathbb{R}$ be a function which we want to develop along its first argument x. All other inputs v are held constant. Employing the Landau notation, and with $\mathcal{O}(\Delta x^{n+1})$ being the error of the approximation, the following holds:

$$f(x + \Delta x, v) = \sum_{k=0}^{n} \frac{1}{k!} \cdot \frac{\partial^k f}{\partial x^k}(x, v) \cdot \Delta x^k + \mathcal{O}(\Delta x^{n+1}) \tag{3.1}$$

Assuming an equidistant grid, i.e. the values of $f(x + k\Delta x, v)$ for $k \in [-l, u] \cap \mathbb{Z}$ are known, this yields a linear equation system of size $l + u + 1 - 1 = l + u$ (-1 because $k = 0$ yields no information). In the following, we omit the second argument v in the equations.

[1] except for sufficient differentiability

$$f(x - l\Delta x) = f(x) + \frac{1}{1!} \cdot \frac{\partial f}{\partial x}(x) \cdot (-l\Delta x)^1 + \frac{1}{2!} \cdot \frac{\partial^2 f}{\partial x^2}(x) \cdot (-l\Delta x)^2 + ... + \mathcal{O}((\Delta x)^{l+u+1})$$

$$...$$

$$f(x) = f(x) + \frac{1}{1!} \cdot \frac{\partial f}{\partial x}(x) \cdot (0)^1 + \frac{1}{2!} \cdot \frac{\partial^2 f}{\partial x^2}(x) \cdot (0)^2 + ... + \mathcal{O}((0)^{l+u+1}) = f(x)$$

$$...$$

$$f(x + u\Delta x) = f(x) + \frac{1}{1!} \cdot \frac{\partial f}{\partial x}(x) \cdot (u\Delta x)^1 + \frac{1}{2!} \cdot \frac{\partial^2 f}{\partial x^2}(x) \cdot (u\Delta x)^2 + ... + \mathcal{O}((\Delta x)^{l+u+1})$$

The equation system consists of $l + u$ unknowns, i.e. $\frac{\partial^k f}{\partial x^k}(x, v)$, and $l + u$ equations. The error of the approximation made by solving for one of the unknowns $\frac{\partial^k f}{\partial x^k}(x, v)$ is $\mathcal{O}((\Delta x)^{l+u+1-k})$ ($-k$ comes from the fact that the equation must be divided by Δx^k before being solved). In this thesis we mostly apply the first derivative. To this end, we mostly use the following four configurations which are also visualized in Fig. 3.1:

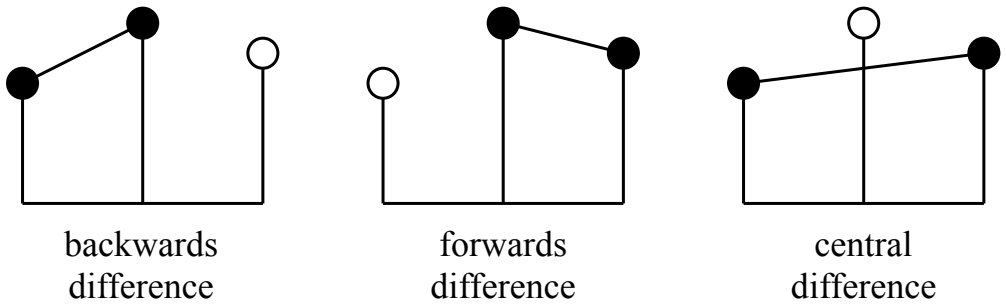

Figure 3.1.: Visualization of forwards ($l = 0, u = 1$), backwards ($l = 1, u = 0$), and central differences ($l = 1, u = 1$)

$$l = 0, u = 1 : \frac{\partial f}{\partial x}(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} + \mathcal{O}((\Delta x)^1)$$

$$l = 1, u = 0 : \frac{\partial f}{\partial x}(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x} + \mathcal{O}((\Delta x)^1)$$

$$l = 1, u = 1 : \frac{\partial f}{\partial x}(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + \mathcal{O}((\Delta x)^2)$$

$$l = 2, u = 2 : \frac{\partial f}{\partial x}(x) = \frac{f(x - 2\Delta x) - 8f(x - \Delta x) + 8f(x + \Delta x) - f(x + 2\Delta x)}{12\Delta x} + \mathcal{O}((\Delta x)^4)$$

3.1.2. Spectral Methods for Periodic Boundary Conditions

While finite differences are quite intuitive in their nature, the derivative of a function can also be calculated by spectral methods. For the method presented here, this requires periodic boundary conditions and entails the usage of the Fourier Transform. According to Johnson [20], the intuition behind this can be seen by assuming that any function on a domain $[0; L]$ can be written as a series:

$$y(x) = \sum_{k=-\infty}^{\infty} Y_k \exp\left(\frac{2\pi i}{L} kx\right)$$

Given that all Y_k are known, the derivative $\frac{dy}{dx}$ can be written as a new series:

$$\frac{dy}{dx}(x) = \sum_{k=-\infty}^{\infty} \left(\frac{2\pi i}{L} kY_k\right) \exp\left(\frac{2\pi i}{L} kx\right)$$

In this example the coefficients Y_k are the Fourier Transform of $y(x)$, and the coefficients $(\frac{2\pi i}{L} kY_k)$ are the Fourier Transform of the derivative $\frac{dy}{dx}(x)$. As there is clearly a linear relationship between the Fourier coefficients of the function and the coefficients of its derivative, the derivative of any function can be calculated in three steps: First, calculating the Fourier coefficients of a function. Second, modifying the calculated Fourier coefficients appropriately. Third, calculating the derivative by applying the inverse Fourier Transform on the the modified Fourier coefficients, i.e. by calculating $\frac{dy}{dx}(x)$ from $(\frac{2\pi i}{L} kY_k)$.

Given the fact that computers are not dealing with continuous functions, but only with discrete samples thereof, we must apply the discrete Fourier Transform. It is commonly calculated by means of the Fast Fourier Transform (FFT). The derivation of this method can be found in [20].

The main advantage of this spectral method is its precision as long as the spatial sampling rate is sufficiently high[2], as will be seen in the numerical testing done in Section 5.1.1.

However, the downsides to this method are twofold. First, the method requires boundary conditions to be periodic in order to work, because the FFT assumes periodic boundary conditions. Second, computing the FFT takes $\mathcal{O}(n \log n)$ which is significantly slower than the $\mathcal{O}(n)$ required for finite difference operators.

We introduced the spectral methods nontheless, because they are common when simulating the horizontal dimension of the weather system on a planetary scale [5],[21],[22].

[2] In signal-processing terms the sample frequency must exceed the Nyquist frequency, and in this case the accuracy is only limited by numerical effects, such as machine precision

This is possible because in the horizontal dimension a spherical planet has periodic boundary conditions. That is, if one were to always travel exactly eastward endlessly, one would end up where one started periodically. In other words, the endless journey eastward is periodic and so are the boundary conditions of the horizontal system.

3.2. Grid Discretizations

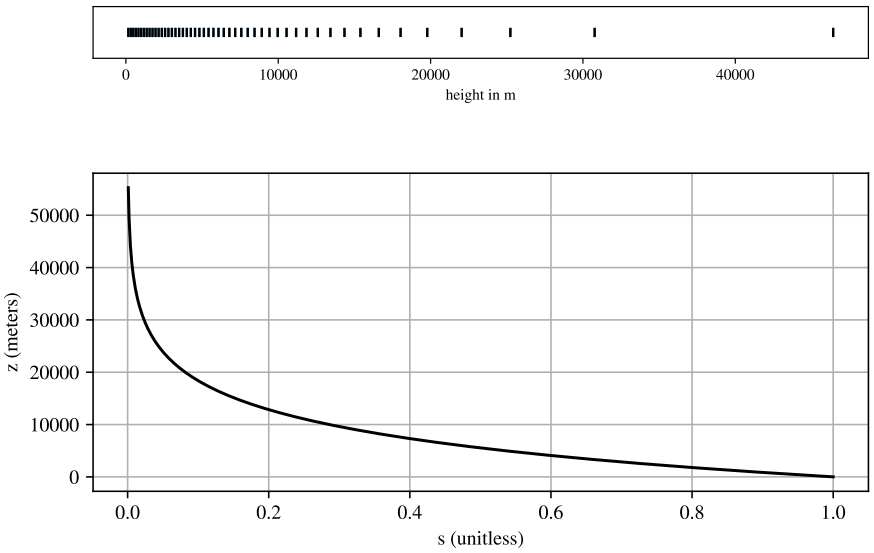

Figure 3.2.: Transformation of s-coordinates to z-coordinates; both diagrams assume that $\pi(s) = s \cdot 1atm$, and $T = 273K$, and use Eq. 2.12 to translate an equidistant s-grid to a non-equidistant z-grid. The first diagram shows where each grid point is located in z-coordinates with 50 grid points. The second diagram shows the relationship between s-coordinates and z-coordinates.

When implementing the non-hydrostatic version of the NSE discussed in Section 2.5, there are three prognostic variables to be considered: Vertical wind speed w, pressure $\ln p$, and temperature T. Each of them varies across the atmosphere and must thus be sampled at different points across space. The question is where to place these points for each variable.

Generally speaking, every variable gets its own arbitrary set of points at which to sample. In theory, the set of points could also vary over time.

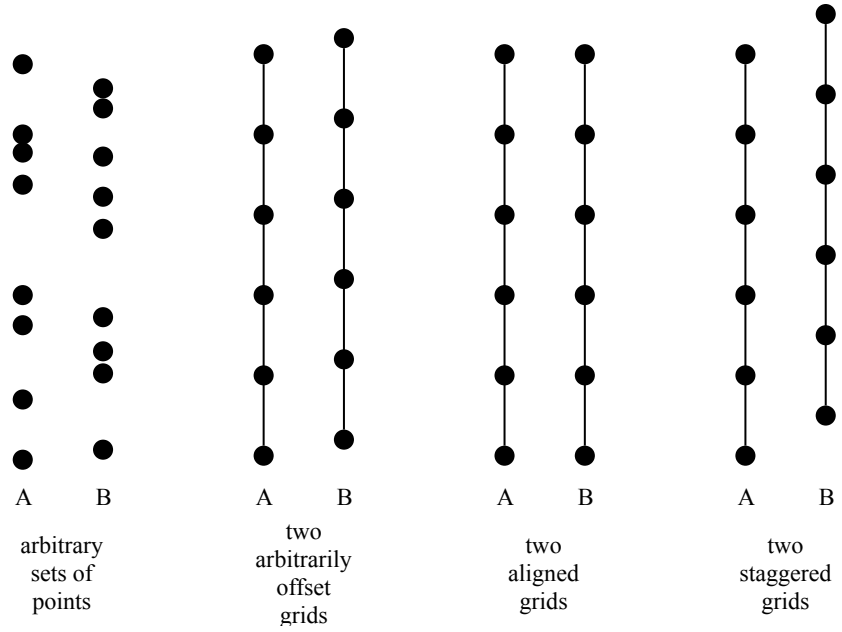

Figure 3.3.: Different variants of discretization

To simplify calculation, we apply grids to bring order to the sets of points. When a variable is sampled along a grid, it is sampled at every node of that grid. In order to utilize the discrete differential operators from Section 3.1, the grids are chosen to be equidistant, i.e. the distance between two consecutive grid points (the mesh size) is always the same. It is also assumed that the mesh size is the same for all three variables.

Note that in this context, the distance between grid points does not necessarily correspond to distance (measured in meters) in the real world. This is due to the alternative coordinate system from Section 2.5 which transforms equidistant grid points in the s-coordinate-system to non-equidistant grid points in the z-coordinate system, depending on how the function $\pi(s)$ is defined. This effect can be observed in Fig. 3.2.

Having now established that all three variables ($\ln p$, T, and w) are sampled along grids of equal mesh size, one remaining question is their relative placement. In other words, the offset between two grids for two variables is a free parameter. In case the offset is zero, we call the two grids aligned. In case the offset is half the distance between two grid points, the two grids are called staggered. To distinguish the two grids, we call one aligned, and the other offset. This is illustrated in Fig. 3.3.

For the NSE there are two prevailing staggered grid systems [23]: The Lorenz grid, and the Charney-Phillips grid which are shown in Fig. 3.4. For the Lorenz grid, vertical wind w is placed on the aligned grid, whereas pressure $\ln p$ and temperature T are on

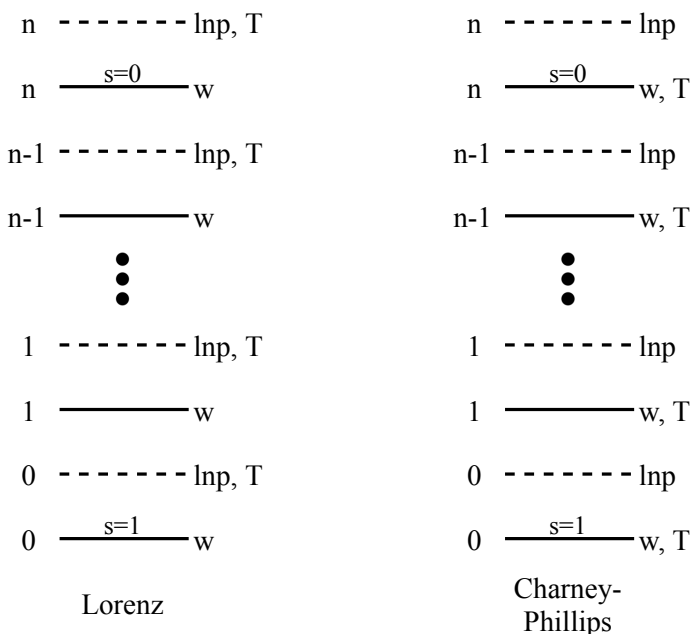

Figure 3.4.: Variable distribution for Lorenz grid and Charney-Phillips grid

the offset grid.

For the Charney-Phillips grid, both vertical wind w and temperature T are placed on the aligned grid, and only pressure $\ln p$ is on the offset grid. The reason for always placing w on the aligned grid, and pressure $\ln p$ on the offset grid, is to enforce the boundary conditions discussed in Section 2.7. By placing wind w on the aligned grid, we can force w_{top} and w_{bottom} to be zero, since there are sample points both at the top and the bottom.

In contrast, with pressure $\ln p$ it is advantageous not to have a sampling point at the upper boundary of the domain, as we may want to postulate pressure at the upper boundary to be zero, because the atmosphere is supposed to end there, i.e. $p_{top} = 0$ which would mean $\ln p = -\infty$. To avoid dealing with non-computable numbers, pressure $\ln p$ is placed on the offset grid.

We can place the remaining variable temperature T on either the offset or the aligned grid, because it not affected by our boundary conditions.

3.3. Discretization of Time and Types of Integrators

Before we give examples of integrators, we must first define them. An integrator is an algorithm that, given a starting condition $x(t_0) = x_0$, can solve a differential equation of the form $\frac{dx}{dt} = f(x,t)$, where x is the state vector and t is time. Its goal is to generate a trace for $x(t)$ over time, after the starting time $t_0 < t$.

Runge-Kutta Methods

One of the simpler classes of integrators consists of the explicit Runge-Kutta methods. These methods begin with the initial value x_0 and then create the trace $x(t)$ by making small time-steps Δt starting at t_0 and modifying the state through addition: $x(t+h) = x(t) + RK(f,x,t,h)$.

The following derivation closely follows the derivation in [24]. Other derivations such as in [25] are also possible.

All Runge-Kutta methods aim to approximate the Taylor series expansion of $x(t)$ with respect to t, i.e.

$$\begin{aligned} x(t+h) &= x(t) + \sum_{k=1}^{n} \frac{h^k}{k!}\frac{d^k x}{dt^k} + \mathcal{O}\left(h^{n+1}\frac{d^{n+1}x}{dt^{n+1}}\right) \\ &= x(t) + \sum_{k=0}^{n-1} \frac{h^{k+1}}{(k+1)!}\frac{d^k f(x(t),t)}{dt^k} + \mathcal{O}\left(h^{n+1}\frac{d^n f}{dt^n}\right) \end{aligned}$$

Exploiting $\frac{df(x(t),t)}{dt} = \frac{\partial f(x(t),t)}{\partial x}\frac{dx}{dt} + \frac{\partial f(x(t),t)}{\partial t} = f\frac{\partial f}{\partial x} + \frac{\partial f}{\partial t}$ this becomes.

$$x(t+h) = x(t) + \sum_{k=0}^{n-1} \frac{h^{k+1}}{(k+1)!}\left(\frac{\partial f}{\partial t} + f(x(t),t)\frac{\partial f}{\partial x}\right)^k f(x(t),t) + \mathcal{O}\left(h^{n+1}\frac{d^n f}{dt^n}\right)$$

The simplest Runge Kutta method is the Explicit Euler or RK1 scheme which can be derived by writing down the Taylor expansion:

$$\begin{aligned} x(t+h) &= x(t) + h \cdot \frac{dx}{dt} + \mathcal{O}(h^2) \\ &= x(t) + h \cdot f(x,t) + \mathcal{O}(h^2) \end{aligned}$$

RK1 has a local truncation error of $\mathcal{O}(h^2)$, and a total accumulated error of $\mathcal{O}(h)$.

RK2 uses more than one evaluation of f in order to do one step:

$$\begin{aligned}
k_1 &= f(x(t), t) \\
k_2 &= f(x(t) + \frac{h}{2}k_1, t + \frac{h}{2}) \\
&= f(x(t) + \frac{h}{2}f(x(t), t), t + \frac{h}{2}) \\
&= f(x(t), t) + h\left(\frac{\partial f}{\partial t} + f(x(t), t)\frac{\partial f}{\partial x}\right) f(x(t), t) + \mathcal{O}(h^2) \\
&= f(x(t), t) + h\frac{df}{dt} + \mathcal{O}(h^2) \\
x(t) + \frac{h}{2}(k_1 + k_2) &= x(t) + \frac{h}{2}\left(f(x(t), t) + f(x(t), t) + h\frac{df}{dt} + \mathcal{O}(h^2)\right) \\
&= x(t) + hf(x(t), t) + \frac{h^2}{2}\frac{df}{dt} + \mathcal{O}(h^3) \\
&= x(t) + h\frac{dx}{dt} + \frac{h^2}{2}\frac{d^2x}{dt^2} + \mathcal{O}(h^3) \\
\Rightarrow x(t + h) &= x(t) + \frac{h}{2}(k_1 + k_2) + \mathcal{O}(h^3)
\end{aligned}$$

RK2 has a local truncation error of $\mathcal{O}(h^3)$, and a total accumulated error of $\mathcal{O}(h^2)$.

In a similar fashion it can be shown that RK4 is:

$$\begin{aligned}
k_1 &= f(x(t), t) \\
k_2 &= f\left(x(t) + \frac{h}{2}k_1, t + \frac{h}{2}\right) \\
k_3 &= f\left(x(t) + \frac{h}{2}k_2, t + \frac{h}{2}\right) \\
k_4 &= f(x(t) + hk_3, t + h) \\
x(t + h) &= x(t) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

RK4 has a local truncation error of $\mathcal{O}(h^5)$, and a total accumulated error of $\mathcal{O}(h^4)$.

4. Implementation

In this chapter we describe the architecture of the custom framework implemented for this thesis. It is written in Python 3 with aid of the numpy library. The aim of the framework is to test the effects of the modifications and simplifications made in Chapter 2, and the effects of discretization described in Chapter 3. To this end we implemented the non-hydrostatic NSE from Section 2.5, and everything discussed in Chapter 3 within the framework.

4.1. Design

We created the framework keeping the design principle of modularity in mind. In this way, we can test and verify every component of the framework individually. It also becomes easier to fulfill the goal of the framework, namely to test different configurations of simplifications and discretizations. Modularizing these simplifications and discretizations makes it easy to switch between them.

We separated the functionality of the framework into two components. The first component generates data through simulations or analytic solutions. This component is called Generator. The second component can then evaluate the generated data by utilizing its error tracking and visualization[1] capabilities. This component is called Evaluator.

Generator

Focusing on the data-generating component, we split its functionality up as follows:

- differential operators, modeled by functions. These functions take in a vector representing a scalar or vector field (represented by numpy arrays), perform their respective operation on it, and return the result of the operation.
- storing the system state in instances of the class `State`. This class wraps a numpy array storing the state variables, and the names of the all axes and variables.

[1] using matplotlib

- differential equations, modeled by classes inheriting from the abstract class `TimeDerivative`. When called and given an instance of `State`, objects of the `TimeDerivative` class calculate the time derivative at the current state. They do so by applying the prognostic equations of their respective differential equation.
- integrators, modeled by classes inheriting from the abstract class `Integrator`. During instantiation they receive an initial state and a differential equation (in the form of a `TimeDerivative` instance) which they will integrate using a specified step size. Every time an instance of `Integrator` is called, it integrates its differential equation by one step, advances its internal time by one step, and outputs the modified `State` instance.
- analytic solutions to differential equations, modeled by classes inheriting from the abstract class `Solution`. As these solutions are analytic, any time t can be specified, and an instance of `Solution` outputs the state of the system modeled by its differential equation. In order to be more similar to `Integrator` classes, they also contain an internal timer which can be advanced by calling the instance of `Solution` without specifying a time t.

With this separation into classes we fulfill the objective of modularity. For one, we can replace one implementation of `Integrator` for another without affecting other components of the program. The same goes for different implementations of the same differential operator. Second, for a given differential equation, it is possible to implement multiple test scenarios or analytic solutions, each represented by their own `Solution` class. Last, it is possible to change the differential equation being simulated by changing the `TimeDerivative` class.

As another design step taken for better readability, any class which repeatedly outputs new data was implemented as an iterator. This includes all classes inheriting from `Integrator` and `Solution`, as both generate new output for every iteration. Iterators make code containing repeated data-generation more compact and readable.

Evaluator

The evaluator component builds on the premise that there is a solution (coming from a `Solution` object) and a calculated result (coming from an `Integrator` object). Both calculated results and solutions are a discrete series of system states over time. Within the program we represent the result by a discrete series of `State` objects at monotonically increasing timestamps. Of course there may be some differences between the accurate solution and the calculated result which we subsequently call errors.

We split the functionality of the component as follows:

- the `ErrorTracker` class which can store a series of errors, along with labels (e.g. time, or spatial resolution). Example: To store errors over time, every time step, an instance of `ErrorTracker` is given a timestamp as a label, the calculated result, and the solution at that timestamp. From this it calculates the error (utilizing a norm of the programmers choice) and stores it along with the timestamp.
- the `ErrorIntegrator` class which also calculates the error whenever it is given a calculated result and a solution. It then adds it to its memorized total error, instead of storing it individually.
- the `WindowManager` class displays both instances of `State` and of `ErrorTracker` visually. Errors can be displayed both on a logarithmic and on an ordinary scale.

  For displaying `State` objects, the class also provides some functionality to apply a custom transformation to the data, before displaying it.

  Example: When simulating, $\ln p$ can be a prognostic variable. In order to display p instead of $\ln p$, we need to transform it through exponentiation first.

4.1.1. Operators

The smallest units of the framework are the operators. We implemented two categories of operators: averaging operators, and the differential operators discussed in Section 3.1.

The averaging operators follow the naming scheme

```
[operation abbreviation]\_e[error order]
```

where the error order indicates how many neighboring variables are taken into account when calculating the local average. When considering a high spatial resolution, using more neighboring variables makes the estimation more accurate.
Example: Averaging samples of a function f on an equidistant grid of mesh size Δx by averaging each grid point with the following grid point.

```
f_average = avg_forward_e1(f, delta_x)
```

The differential operators follow the naming scheme

```
[operation abbreviation]\_n[operator order]\_e[error order]
```

where the `operator order` can either be `1` for the first order derivative, or `2` for the

second order derivative (also known as the laplacian). The error order `e` is the exponent of h in the error term $\mathcal{O}(h^e)$ (see Section 3.1).
Example: calculating the derivative $\frac{df}{dx}$ of some function f w.r.t. x on an equidistant grid with mesh size Δx

```
df_dt = diff_n1_e2(f, delta_x)
```

4.1.2. Class Structure and Information Flow

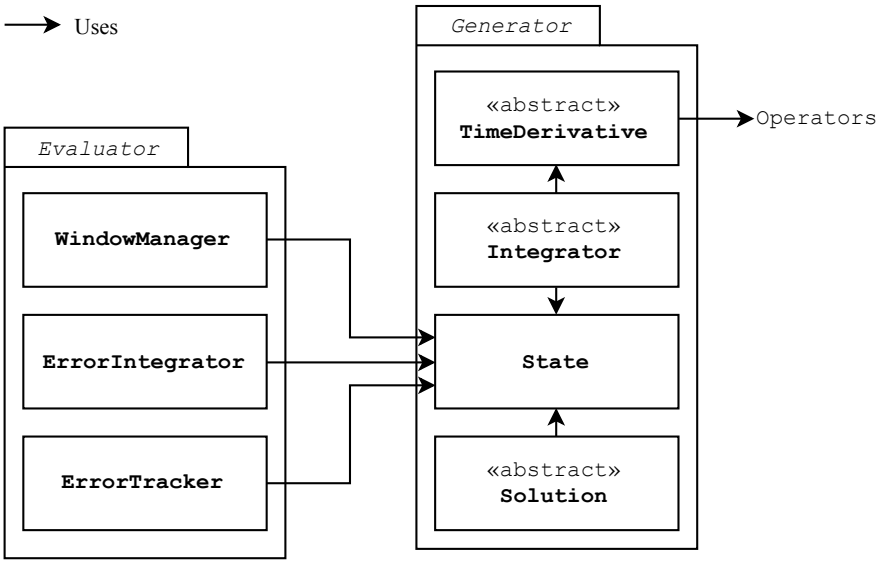

Figure 4.1.: UML-Style-Diagram

The interaction between the different types of classes is visualized in Fig. 4.1. All communication between classes is done by passing references to `State` objects. As `State` objects can take up a lot of space in memory, passing them by reference is better than passing copies. For the same reason, whenever possible, `State` objects are reused to avoid overcrowding memory.

During execution, both the `Integrator` and the `Solution` instance contain one `State` object each. Whenever we request new information from them, they generate it, and return a reference to their internal `State` object. Now, the program that requested the information can process the `State` objects. To this end the Evaluator components can be employed.

4.1.3. Usage of the Framework

In this section we demonstrate the framework on the example of the wave equation with periodic boundary conditions[2] in the following form:

$$\frac{du}{dt} = \frac{dv}{dx}$$
$$\frac{dv}{dt} = c^2 \frac{du}{dx}$$

We wrote multiple pre-implementations of the abstract class `Integrator`, namely implementations of RK1 (explicit Euler), RK2 (explicit Heun), RK4. All of them can be found in the folder `./integrators/`.

The remaining abstract classes are `TimeDerivative` and `Solution`. Both of them are dependent on the differential equation to be simulated, and on the discretization of space. In other words: different spatial discretizations/grids require new implementations of both `TimeDerivative` and `Solution`. For this reason we create separate folders[3] for each differential equation to be analyzed. Within each folder an appropriate implementation of `TimeDerivative` and `Solution` can be found.

In the example of the wave equation, we can implement the `TimeDerivative` class as follows:

```
class PeriodicWaveTimeDerivative(TimeDerivative):
  def __init__(self, delta_x, c):
    self.delta_x = delta_x
    self.c = c

  def __call__(self, state_vars, t):
    # extract the variables from the state variables
    u = state_vars[0]
    v = state_vars[1]

    # calculate the time-derivatives
    du_dt = diff_n1_e4(v, self.delta_x)
    dv_dt = self.c * self.c * diff_n1_e4(u, self.delta_x)

    # transform result into canonical format
    return np.stack((du_dt, dv_dt), axis=-1).transpose()
```

[2] i.e. $u(x) = u(x + a)$ if a is the periodicity
[3] within the `./cases/` folder

We implement the `Solution` class with d'Alembert's solution which dictates that for a given starting condition

$$u(x, t = 0) = f(x)$$
$$v(x, t = 0) = 0$$

the solution can be written as

$$u(x,t) = \frac{1}{2}(f(x+ct) + f(x-ct))$$
$$v(x,t) = \frac{c}{2}(f(x+ct) - f(x-ct)).$$

The resulting implementation is:

```
class WaveEqSolution(Solution):
  def __init__(self, num_grid_points, dt, domain_size, c, f):
    super().__init__(0, dt)
    # store all variables necessary for calculating the solution
    self.c = c
    self.f = f
    # create the grid along which to sample u and v
    self.x = np.tile(np.linspace(0, domain_size, num_grid_points + 1)[:-1],
                     (2, 1))
    # create an instance of State-class
    self.state = utils.State(num_vars=2, dim_vars=num_grid_points,
                      axes=self.x, names=[("x", "u"), ("x", "v")])

  def solution(self, t): # according to D'Alembert
    # u = state_vars[0] and v = state_vars[1]
    state_vars = self.state

    # calculate the result
    state_vars[0] = 0.5 * (self.f(self.x[0] + t * self.c)
                           + self.f(self.x[0] - t * self.c))
    state_vars[1] = self.c * 0.5 * (self.f(self.x[0] + t * self.c)
                                    - self.f(self.x[0] - t * self.c))

    return self.state
```

To put everything together, now, the modus operandi is as follows: First, we must define an initial state of the system. If we want to compare the calculated result against

an analytic solution, the initial state must be gained from an instance of `Solution`. This can be done by first creating an instance of `Solution`, and then asking it for the system state at time 0. Otherwise we can directly create an instance of `State` and set its initial conditions manually.

In the example there is a solution so the starting condition can be gained as follows:

```
# creating an instance of WaveEqSolution
solver = WaveEqSolution(num_grid_points, dt, domain_size, c, f)
initial_state = solver.solution(t=0)
```

Next, we create an instance of `TimeDerivative`.

```
# calculate mesh size
delta_x = domain_size / num_grid_points
# creating an instance of PeriodicWaveTimeDerivative
time_derivative = PeriodicWaveTimeDerivative(delta_x, c)
```

Both this instance and the instance of `State` containing the starting conditions of the system are necessary to create an instance of `Integrator`.

```
# creating an instance of RungeKutta.Explicit
integrator = RungeKutta.Explicit(initial_state, time_derivative,
                                 t0 = 0 , delta_t = dt)
```

Now, depending on the aim of the simulation, we can instantiate the appropriate classes from the Evaluator components which concludes the setup.
As both `Integrator` and `Solution` are implemented as iterators, we can simulate the system with a simple `for` loop. As an iterator, `Integrator` returns the calculated result, and `Solution` returns the analytic result. In this way the programmer has access to the accurate result and the analytic result within the `for` loop. Within it, one can now perform any necessary further operations.

```
for int_state, sol_state in zip(integrator, solver):
  # do operations on the states
```

For some of the common operations a programmer might want to do within the `for` loop, we pre-implemented a function performing the entire setup in the `run_utils.py` file.

Another helpful pre-implementation is that of the numerical reference solution. One can employ it whenever no analytic solution is known, but still needs a reference solution. In this case, a reference solution is created by simulating the differential equation with RK4 and very small time-steps. To avoid re-calculation, we cache the result of this time-intensive simulation.

4.2. Testing

While it is not possible to prove mathematically that the implementation is without fault[4], the next best thing is exhaustive testing. For this framework, testing entails checking the outputs of the components against a reference solution which we must find separately. This reference solution is usually analytic in nature and must derived through manual calculation.

We took the approach of testing the framework bottom-up, i.e. first, we tested the smallest possible components in isolation (unit tests). Then, we tested components building only on tested sub-components, and so on. In the following sections we describe the tests for the Generator components.

4.2.1. Operators

In the case of this framework the smallest components are the operators. We tested them on operations simple enough that an analytic expression exists. Then we compared the result of the numerical operator against this analytic expression. For the averaging operators this step was sufficient.

In order to further verify the derivative operators, the order at which the error converges was also checked, i.e. after changing the spatial resolution by a factor of a, an operator of the n-th error order was expected to reduce its error by a factor of a^n.

4.2.2. Integrators

In order to test the integrators separately from the differential operators, we started with single-variable differential equations. Having only one variable and no spatial dimension means we did not need to utilize any spatial derivative operators. For testing purposes we implemented four such differential equations for which analytic solutions exist as subclasses of `TimeDerivative` and `Solution`. Having isolated the Integrator as the component to be tested, it was then run at different time resolutions. For Runge Kutta methods of order n an increase in resolution by a factor of a was expected to reduce the error by a factor of a^n.

One should always keep in mind that this method of testing for convergence does not always work. For example, for exponential integrators this method would fail, as exponential integrators are either accurate down to machine precision or wholly inaccurate.

[4] we would first have to perform the Herculean task of formally verifying that Python 3 itself is without fault

4.2.3. Differential Equations

Having verified the implementation of both operators and integrators, the only components left to verify are the implementations of the differential equations. These are represented by `TimeDerivative` implementations. The process for this is similar to the previous sections: First, we find some analytic solution. The breadth of this analytic solution can vary in its universality, from analytic solutions describing the evolution of the system from any given starting condition[5], to analytic solutions just describing the stationary solution[6] of the system. We then implement this solution as a `Solution` class.

Thereafter, we give the initial state of the system to an integrator (usually RK4). This integrator in turn calls the `TimeDerivative` implementation to be tested in order to run the simulation. From this we gain a simulation result independent of the analytic solution.

At the end of this simulation we have both an analytic solution and a calculated simulation which we can compare. If the results are coinciding down to numerical precision, the implementation of `TimeDerivative` is not disproved.

Especially for linear PDEs to further verify the solution, one can also vary the resolution with which RK4 is run. If a change in resolution by a factor of a translates into a reduction in error by a factor of a^4, this is a good indicator that the solution described by `TimeDerivative` converges towards the analytic solution.

4.2.4. Example

Now that we have detailed the implementation architecture, we explain the implementation of the non-hydrostatic NSE in the form of a `TimeDerivative` implementation. More specifically an implementation of the Lorenz grid is shown.

[5] which make numeric solutions redundant

[6] A stationary solution is a solution for which the state of the system does not change.

First, we define the variables which appear in the implementation:

Variable Name	Type	Meaning
`delta_s`	scalar	mesh size/distance between s-grid points
`s`	vector	vector containing the s-locations of all grid nodes of the aligned grid.
`dpi_ds`	function	this represents $\frac{\partial \pi}{\partial s}$
`state_vars`	2d-array/ list of vectors	contains the state-variables. $\ln p$=`state_vars[0]` T=`state_vars[1]` w=`state_vars[2]`
`t`	scalar	contains the elapsed simulation time (is not used in this implementation).
`lnp`	vector	contains all samples of $\ln p$
`p`	vector	contains all samples of p
`T`	vector	contains all samples of T
`w`	vector	contains all samples of w
`dlnp_dt`	vector	contains all samples of $\frac{\partial \ln p}{\partial t}$
`dT_dt`	vector	contains all samples of $\frac{\partial T}{\partial t}$
`dw_dt`	vector	contains all samples of $\frac{\partial w}{\partial t}$

The only two operators we will utilize are the following:

```
diff_s_align_n1_e2(f_offset, delta_s)
diff_s_offset_n1_e2(f_aligned, delta_s)
```

Both functions approximate the derivative through central differences with a twist. Normally, when central differences are calculated, if the input is on an aligned grid, the output is also located on an aligned grid. This means the derivative at location $f(s)$ is approximated from values at $f(s - \Delta s)$ and at $f(s + \Delta s)$. In contrast, `diff_s_offset_n1_e2` takes its input on an aligned grid, and outputs it on an offset grid. That is, the derivative at location $f(s)$ is approximated from the values at $f(s - \frac{\Delta s}{2})$ and $f(s + \frac{\Delta s}{2})$. Vice versa, `diff_s_offset_n1_e2` takes its input on an offset grid, and outputs it on an aligned grid.

With these definitions, implementation of the non-hydrostatic NSE is reasonably simple. When reading the code, note the close relation to the mathematical notation which can be seen by writing them down side by side:

Evolution of Vertical Wind Speed:

$$\frac{\partial w}{\partial t} = -g\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right)$$

```
dw_dt =
- const.g * (1 - diff_s_align_n1_e2(p, self.delta_s) / self.dpi_ds(self.s))
```

Evolution of Pressure

$$\frac{\partial \ln p}{\partial t} = \frac{g}{1 - \frac{R}{C_p}} \frac{p}{RT} \frac{\partial w}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}$$

```
dlnp_dt = (const.g / (1 - const.R / const.C_p))
          * (p / (const.R * T))
          * diff_s_offset_n1_e2(w, self.delta_s)
          / self.dpi_ds(self.s + self.delta_s / 2)
```

Evolution of Temperature

$$\frac{\partial T}{\partial t} = \frac{RT}{C_p} \frac{\partial \ln p}{\partial t}$$

```
dT_dt = (const.R * T / const.C_p) * dlnp_dt
```

```
class LorenzTimeDerivative(TimeDerivative):
  def __init__(self, delta_s, s, dpi_ds):
    # store the variables necessary for computation
    self.delta_s = delta_s
    self.dpi_ds = dpi_ds
    self.s = s

  def __call__(self, state_vars, t):
    # extract the state variables from the system state
    lnp = state_vars[0]
    p = np.exp(lnp)
    T = state_vars[1]
    w = state_vars[2]

    # prognostic equation for pressure (offset grid)
    dlnp_dt = (const.g / (1 - const.R / const.C_p)) \
              * (p / (const.R * T)) \
              * diff_s_offset_n1_e2(w, self.delta_s) \
              / self.dpi_ds(self.s + self.delta_s / 2)

    # prognostic equation for temperature (offset grid)
    dT_dt = (const.R / const.C_p) * T * dlnp_dt

    # prognostic equation for vertical wind (aligned grid)
    dw_dt = - const.g \
    * (1 - diff_s_align_n1_e2(p, self.delta_s) / self.dpi_ds(self.s))

    # boundary conditions
      # index 0 <=> s=0 <=> top of atmosphere
      # index -1 <=> s=1 <=> bottom of atmosphere
    # fix pressure to 0 (ln(0)=-inf) above atmosphere
    dlnp_dt[0] = 0
    # fix temperature outside atmosphere to be same as at top of atmosphere
    dT_dt[0] = dT_dt[1]
    # set wind at top and bottom to stay constant at zero
    dw_dt[0] = 0
    dw_dt[-1] = 0

    # transform result into canonical format
    return np.stack((dlnp_dt, dT_dt, dw_dt), axis=-1).transpose()
```

5. Numerical Studies

Having discussed how the NSE were simplified, discretized, and implemented, in this section we test and validate the implementation. To this end we discuss the results of the unit tests from Section 4.2 by examining the numerical errors made by the differential operators and integrators. Thereafter we analyze the implementations of the non-hydrostatic NSE discussed in Section 2.5.

5.1. Validation by Analyzing Numerical Errors

5.1.1. Differential Operators

To test the differential operators, the function $f(x) = \sin(20\pi(x - e))$ with derivative $\frac{df}{dx} = 20\pi\cos(20\pi(x - e))$ was used on the domain $[0; 1]$. The offset of e was included in order to avoid random perfect results. For example, if maxima, minima and zero crossings happen to be located exactly at the sample points, even a simple central difference would result in no error. With an offset of an irrational number, such as e, this is improbable.

To test the different methods of calculating the derivative numerically, we sampled the domain at resolutions ranging from 5 to $20 \cdot 10^7$ samples which were spread equidistantly across the domain. We first look at the implementation of the finite differences. If their implementation is correct, we expect the error $[\texttt{finite\_difference(x)} - \frac{df}{dx}]$ to drop with increasing resolution. More specifically for a finite difference operator of order n, when increasing the number of samples by a factor of a we expect a decrease in error by a factor of a^n. The positive result of the test can be seen in Fig. 5.1.

We can also see that above a certain spatial resolution, the error increases again which is due to numerical errors. The root of these errors is the division of a very small number in the denominator by the small number Δx which is inversely proportional to the resolution. This division of small numbers is not well-supported by floating point numbers, and for that reason errors increase above a certain spatial resolution.

Concerning the spectral derivative, we would expect that the error drops down to machine precision as soon as the Nyquist sampling rate is reached. The Nyquist sampling rate is twice the highest frequency occurring in the signal whose derivative is to be calculated. In this case the highest frequency is 10, because the argument of sin

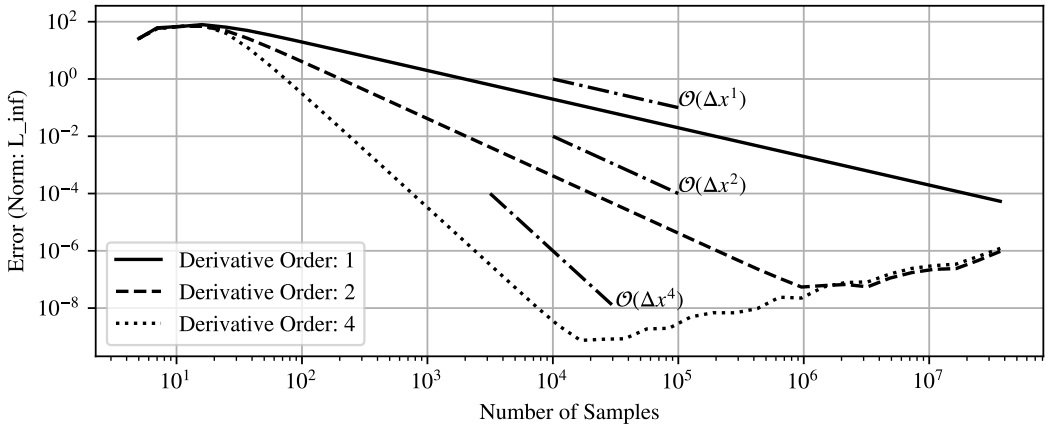

Figure 5.1.: Plot of error of finite difference operators over number of samples; the lines are parallel to the expected error order lines, meaning the test was successful.

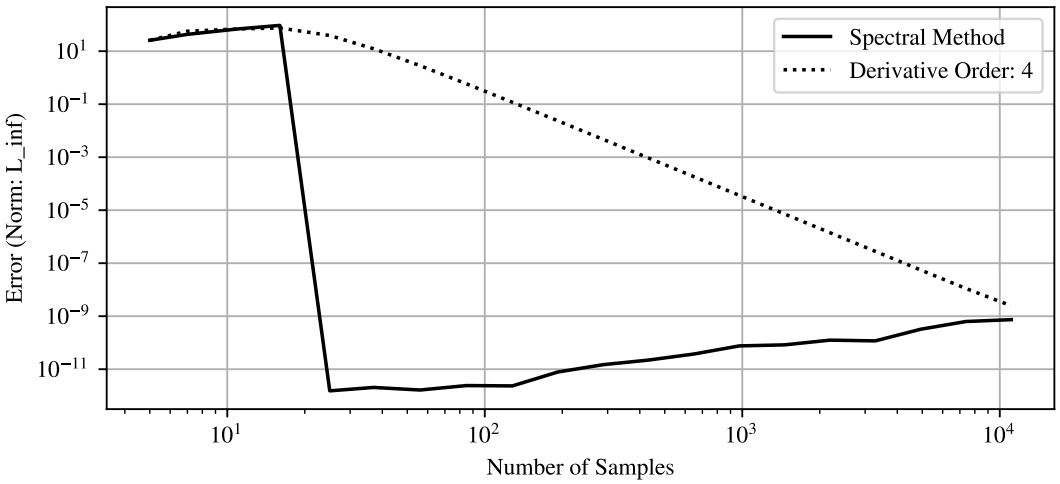

Figure 5.2.: Plot of error of spectral derivative operator over number of samples; as expected the error drops significantly as soon as the Nyquist sampling rate (20 samples) is reached.

was $2\pi \cdot 10$, meaning the Nyquist sampling rate is 20. The fact that this theoretical result can be observed in practice (Fig. 5.2), is a strong indication that the implementation is correct.

5.1.2. Integrators

Runge-Kutta-Integrators

As described in Section 4.2, we tested the integrators by comparing the results to analytic solutions of differential equations. Specifically, we first tested against three types of differential equations, whose solutions are known analytically. The first one was time-invariant and had only one variable, i.e. no spatial resolution. The second one introduced time-variance, while the third one added spatial resolution. In each of the tests, using a Runge-Kutta integrator of order n we expected that decreasing the time step size by a factor of a would decrease the error by a factor of a^n.

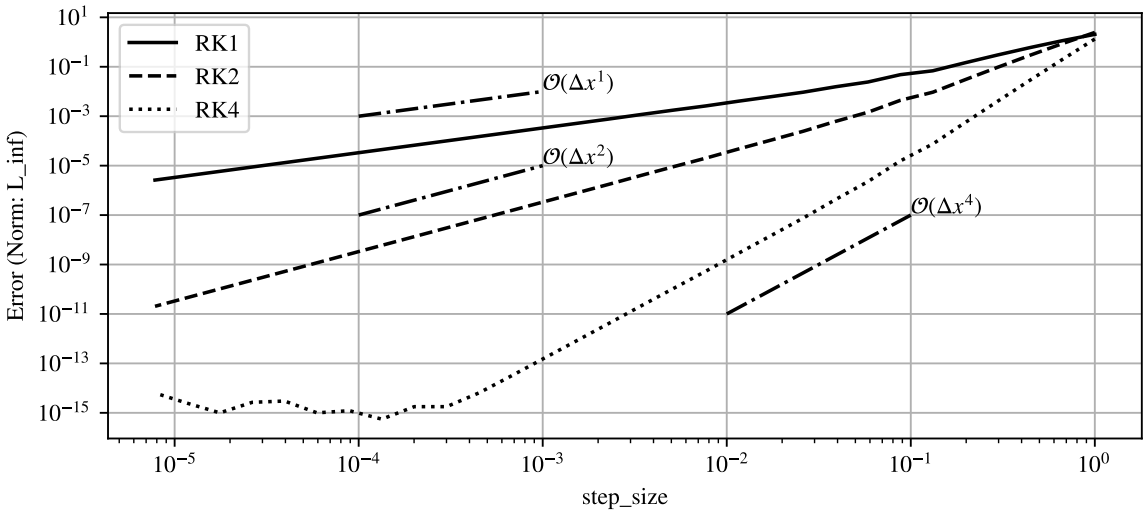

Figure 5.3.: Error of RK-methods on time-invariant, single variable differential equation

Time-Invariant, Single Variable The differential equation is $\frac{dx}{dt} = -3x$ which has the solution $x(t) = x_0 e^{-3t}$. The result of the experiment can be seen in Fig. 5.3.

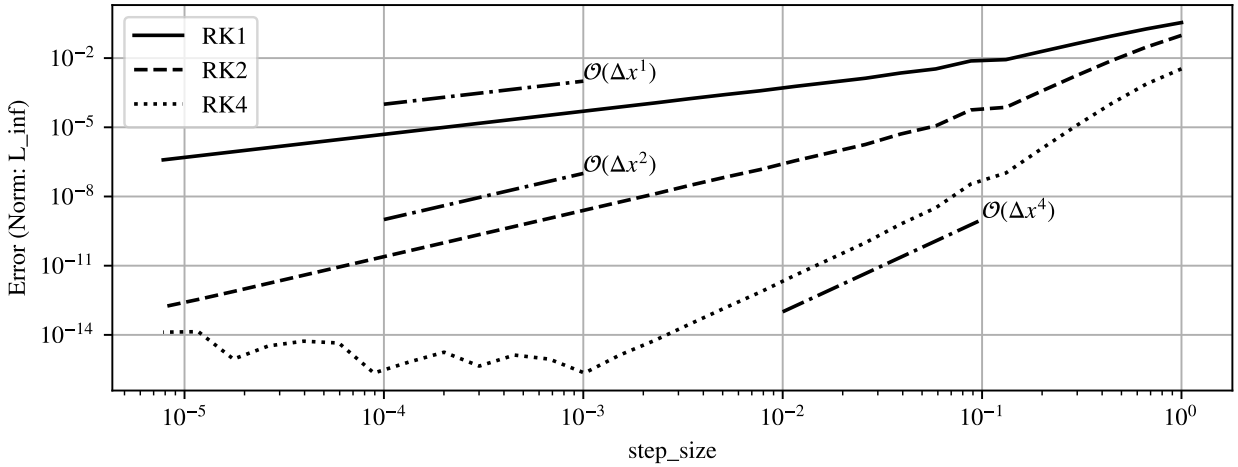

Figure 5.4.: Error of RK-methods on time-variant, single variable differential equation

Time-Variant, Single Variable The differential equation is $\frac{dx}{dt} = \frac{t}{t^2+1}$ which has the solution $x(t) = x_0 + 0.5\ln(t^2+1)$. The result of the experiment can be seen in Fig. 5.4.

Time-Invariant, Multiple Spatially Distributed Variables The differential equation is the wave equation with periodic boundary conditions. With c being the wave propagation speed, the equation system can be written as follows:

$$\frac{du}{dt} = \frac{dv}{dx}$$
$$\frac{dv}{dt} = c^2\frac{du}{dx}$$

If $u(x, t = 0) = f(x)$ (with f being periodic) describes the initial state of u, then according to d'Alembert the solution is:

$$u(x,t) = \frac{1}{2}(f(x+ct) + f(x-ct))$$
$$\Rightarrow v(x) = \int \frac{du}{dt}dx = \frac{c}{2}(f(x+ct) - f(x-ct))$$

The result of the experiment can be seen in Fig. 5.5.

Interestingly, spatial resolution and temporal resolution are intertwined. If a very high spatial resolution (a lot of samples) is chosen, one also must choose a high temporal resolution (small time steps). This can be seen when plotting a heatmap of the error

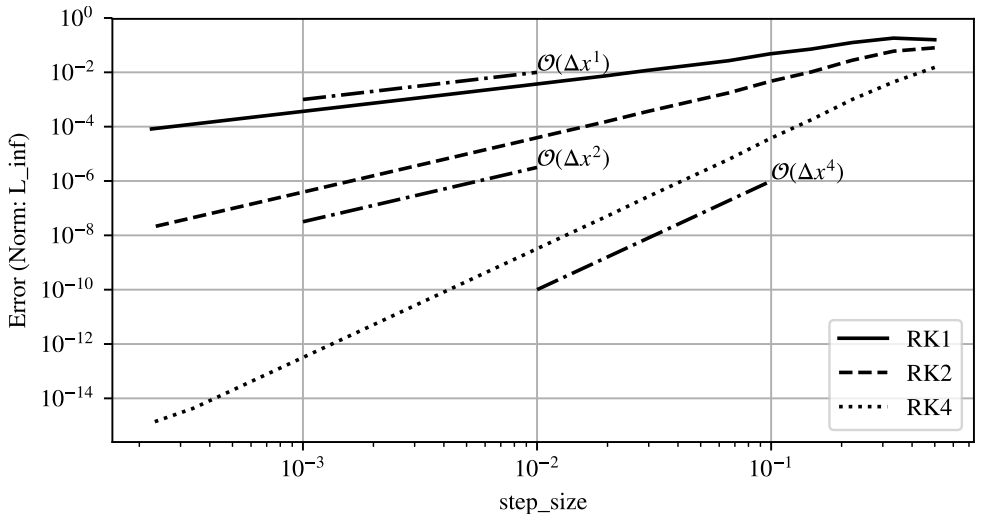

Figure 5.5.: Error of RK-methods on periodic wave equation

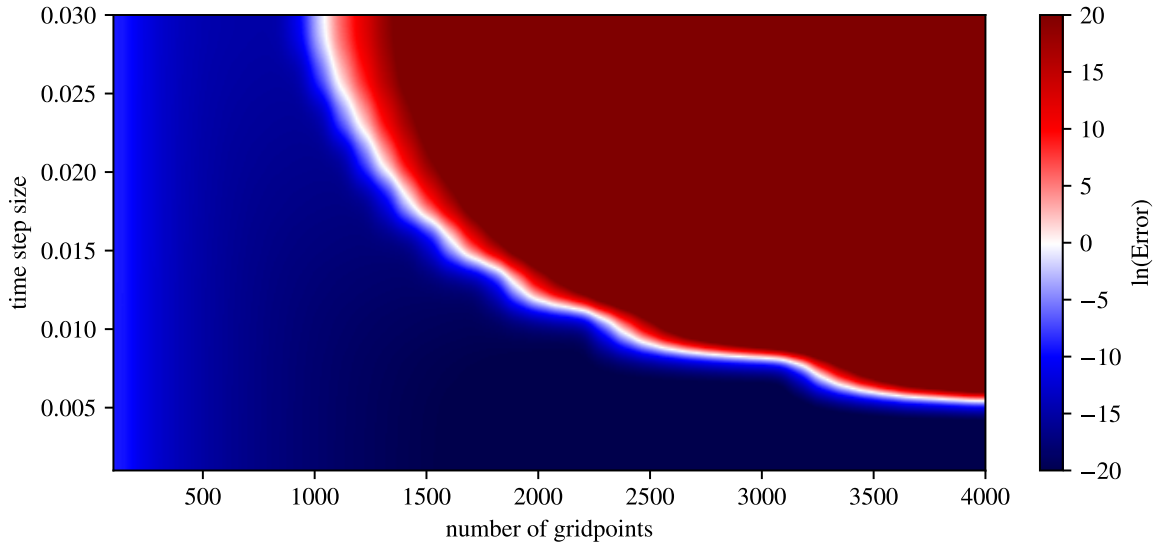

Figure 5.6.: Heatmap of ln(Error) for wave equation over different spatial and temporal resolutions; the white stripe represents the area where the log-error is 0, i.e. the actual error is 1. In the red and the blue areas the log-error is positive (up to 150) and negative (down to -20), respectively.

made with the RK4 integrator over different spatial and temporal resolutions. This phenomenon can be seen in Fig. 5.6. Roughly speaking, in order to keep the error from diverging, an inverse proportion between spatial resolution and time step size should be kept. For example, when doubling the spatial resolution, one should at least halve the time step size, in order to stay stable.

5.2. Study of Errors in Implementation of NSE

In this section we analyze the two implementations of the NSE. As no single standardized benchmarking system for testing the vertical dimension of the non-hydrostatic NSE in isolation exists, complete verification of the implementation is not possible. Instead, we considered four tests:

- comparison with a stationary solution
- comparison of stationary solution with real world measurements
- energy conservation
- comparison of the two implementations to one another

Due to the nature of the non-hydrostatic NSE, we cannot test for mass conservation, because mass is always preserved according to Eq. 2.13.

5.2.1. Comparison with Stationary Solution

We first test the implementation by cross-checking it with the stationary solution. This means that the system state must stay constant in case the starting condition is a stationary solution of the non-hydrostatic NSE. To find a stationary solution for the non-hydrostatic NSE, we set all time derivatives to zero:

$$\frac{\partial w}{\partial t} = 0 = -g\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right) \tag{5.1}$$

$$\frac{\partial \ln p}{\partial t} = 0 = \frac{g}{1 - \frac{R}{C_p}} \frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1} \frac{\partial w}{\partial s} \tag{5.2}$$

$$\frac{\partial T}{\partial t} = 0 = \frac{RT}{C_p}\frac{\partial \ln p}{\partial t} \tag{5.3}$$

The first equation Eq. 5.1 yields the following identity for pressure:

$$\frac{\partial p}{\partial s} = \left(\frac{\partial \pi}{\partial s}\right)$$
$$\Rightarrow p(s) = \pi(s) + p_0$$

The second equation Eq. 5.2 requires one of the elements in the denominator to equal zero. As neither g, p, nor $\left(\frac{\partial \pi}{\partial s}\right)^{-1}$ can be zero, the only possible option is that $\frac{\partial w}{\partial s} = 0$, i.e. $w(s) = w_0 \forall s$. Due to the boundary conditions, at the bottom of the atmosphere $w(s = 1) = 0$, so $w(s) = w_0 \forall s$.

The last equation Eq. 5.3 yields no information, because the right hand side is already zero, due to $\frac{\partial \ln p}{\partial t} = 0$. This means T can be chosen freely, e.g. $T = 273K$. Using this stationary solution to define the initial state of the system, the integrator is expected not to change the system state. This is reflected by both implementations as can be seen in Fig. 5.7, where the magnitude of the error is never greater than 10^{-8}.

5.2.2. Comparison with Real World Measurements

Another method to test the implementation is a comparison with measurements from the real world. First, we can compare the distribution of density $\rho = \frac{p}{RT}$ in the stationary case to real world measurements of the atmosphere made by NASA [26]. For the simulation we assumed that pressure at the bottom of the system was $\pi_{bottom} =$ 1atm = 101325Pa, and $T = 273K$. Additionally, it was assumed that $\pi(s) = s \cdot \pi_{bottom}$ In Fig. 5.8 we see that the calculated result closely mirrors reality.

The second comparison with real world measurements is the speed of sound. The speed of sound is essentially the speed at which a region of increased pressure travels, i.e. the wave speed of the system. To measure wave speed an appropriate starting condition containing a region of increased pressure must be defined. Then the location of this bump must be measured over time. From the trace of time and location of the bump, its speed can be calculated by taking the derivative of location over time.

The result of such an experiment can be seen in Fig. 5.9, namely that the speed of sound in the simulation ranges between $331\frac{m}{s}$ and $341\frac{m}{s}$. This is close to the actual speed of sound at sea level around $331\frac{m}{s}$ according to [27].

For the experiment, the starting condition was chosen as follows: $T = 273K$, $w = 0$. For p we began with the stationary solution as a baseline, i.e. $p(s) = \pi(s)$. Then we added a Gaussian bell curve of magnitude $1\frac{N}{m^2}$ with its maximum at the bottom $s = 1$ onto this configuration. Inserting the Gaussian bump at any location other than the boundary of the domain would result in two regions of increased pressure traveling

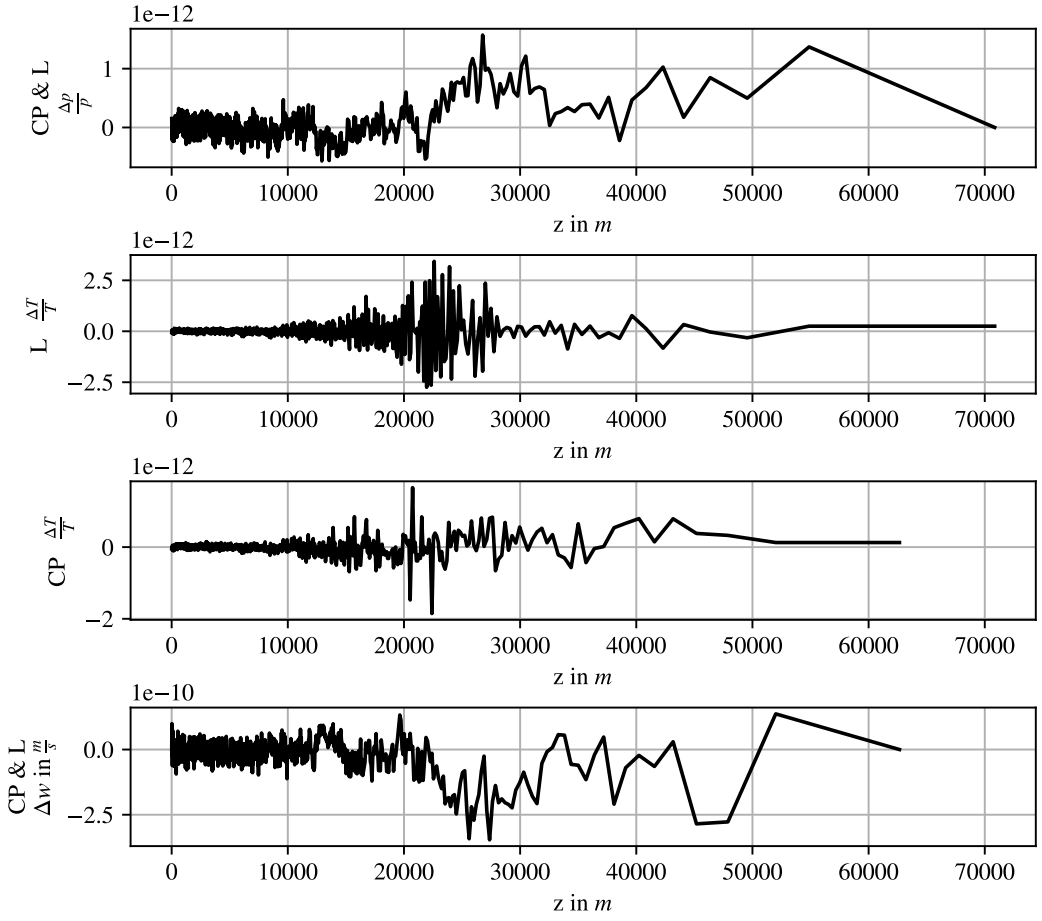

Figure 5.7.: Relative Error (deviation from stationary solution) with RK4, time-step-size 2.5ms, and 1000 grid points, after 240s of simulation time on Lorenz grid (L) and Charney-Phillips grid (CP); the plots for $\frac{\Delta \ln p}{p}$ and Δw for both simulations were essentially identical, so both were only plotted once.

across the atmosphere. This would make measurements within the simulation more difficult.

Note that this is one of the key differences between simulation with the hydrostatic NSE versus with the non-hydrostatic NSE. According to Coiffier, usage of the hydrostatic NSE "eliminates sound waves because of the hydrostatic relation which has a *filtering* effect on them" [5].

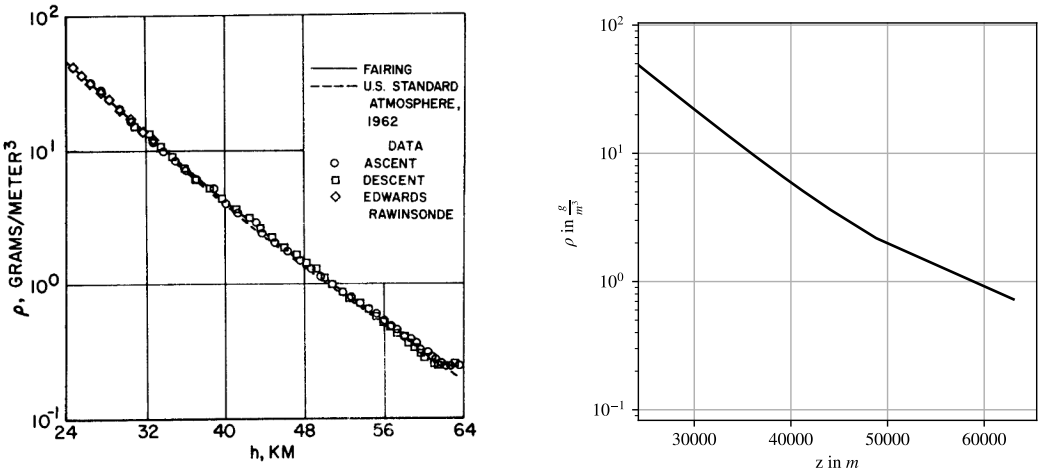

Figure 5.8.: Comparison of real world measurements from NASA [26] with simulated results.

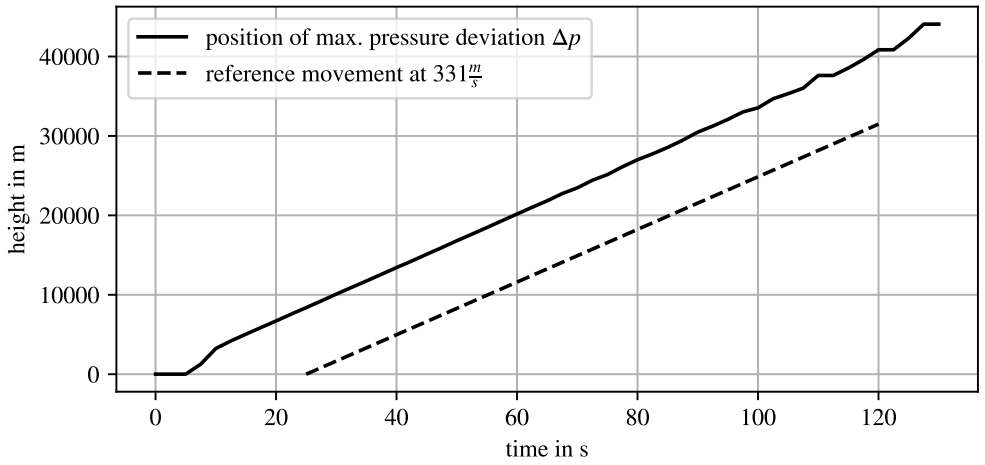

Figure 5.9.: Graph of location over time of (a) a region of increased pressure; (b) a reference object moving at the speed of sound ($331\frac{m}{s}$); the simulation was run with 1000 grid points and a time step size of 2.5ms on a Lorenz grid. Simulations on a Charney-Phillips grid yield similar results.

5.2.3. Energy Conservation

The third way to test the implementation is testing the hypothesis of energy conservation. As the choice of boundary conditions dictates that the energy in the system must stay constant, every simulation of the non-hydrostatic NSE should have the same property. That is, every system state produced by the simulation should have the same energy as the starting condition.

To this end we employ the formula for calculating the total energy in the system per area 2.14:

$$E' = \int_{s_{top}}^{s_{bottom}} \frac{1}{g}(C_v T + \frac{1}{2}w^2 + gz(s)) \left(\frac{\partial \pi}{\partial s}\right) ds$$

When assuming energy to stay constant over time, it makes sense to take the energy contained in the system at the beginning as a baseline. In this way, by subtracting the baseline energy from the energy calculated at some later time during simulation, we can track the error over time.

Of course, in order for it to be possible for an error to appear, the system must change over time, i.e. not be stationary. Other than that any starting condition may be chosen. For the following simulations, the starting conditions were as follows: $T = 273K$, $w = 0$. For p the stationary solution was taken as a baseline, i.e. $p(s) = \pi(s) = s \cdot 101325 \frac{N}{m^2}$. Then a Gaussian bell curve of magnitude $0.1 \frac{N}{m^2}$ with its maximum at $z = 35km$ was added onto this configuration.

The results of this for both the Lorenz grid and the Charney-Phillips grid can be seen in Fig. 5.10. The baseline energy for both grids was nearly identical (down to the fifteenth digit) at $E'_0 = 2.814 \cdot 10^9 J$.

In this example the Lorenz grid leads to lower errors in energy conservation, than usage of the Charney-Phillips grid. This is in line with the design goal of energy conservation that Lorenz had in mind when designing his grid [28].

To explain the larger error when employing the Charney-Phillips grid, one must look at the composition of energy. Recall that energy in the atmospheric system can be split up into three components: internal energy, kinetic energy, and geopotential energy.

When plotting the individual deviations for each component, as is done in Fig. 5.11, we see that the kinetic energy deviation is rather negligible at around $3 \cdot 10^{-3} J$. Instead the increased energy deviation for the Charney-Phillips grid must originate from an asymmetry between the deviation of internal and geopotential energy. It appears that this asymmetry is inherently rooted in the averaging operation necessary to calculate the time-derivative for temperature T numerically.

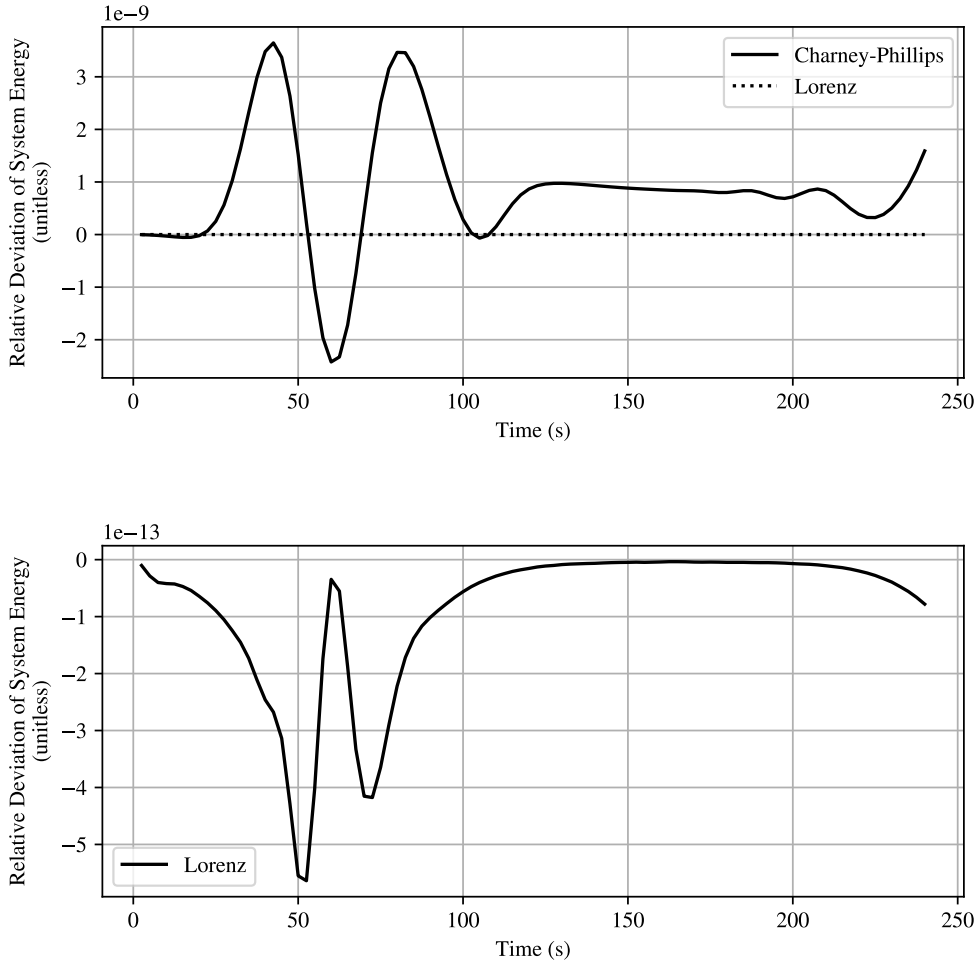

Figure 5.10.: Relative energy deviation $\frac{E'(t)-E'_0}{E'_0}$ over time for Lorenz Grid and Charney-Phillips grid; both simulations were run with 1000 grid points and a time step size of $2.5ms$.

Looking further at the averaging operation, we recall the prognostic equation for T:

$$\begin{aligned}\frac{\partial T}{\partial t} &= \frac{RT}{C_p}\frac{\partial \ln p}{\partial t} \\ &= \frac{gp}{C_p - R}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s}\end{aligned}$$

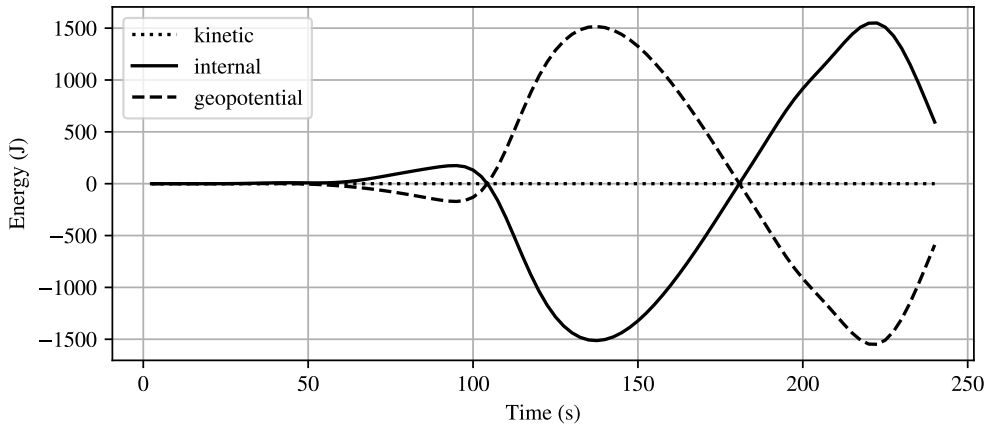

Figure 5.11.: Deviation of each component of energy over time for Charney-Phillips grid; the simulation was run with 1000 grid points and a time step size of $2.5ms$.

We see that T either depends on $\frac{\partial \ln p}{\partial t}$, or on p (when substituting in the prognostic equation for $\frac{\partial \ln p}{\partial t}$). For the Charney-Phillips grid, both of these variables are sampled on the offset grid, meaning that in order to calculate $\frac{\partial T}{\partial t}$, an averaging operator must be applied. Taking an average is not especially accurate. This explanation supports the interpretation that there is no implementation error.

In light of this, when conservation of energy is essential, the Lorenz grid should be preferred when simulating the non-hydrostatic NSE.

5.2.4. Comparison of the two Implementations

Finally, we tested the two implementations by comparing the simulation outputs after a certain amount of elapsed time. For purposes of this test, we initialized both implementations with identical starting conditions. Just as in the previous section, any starting condition can be chosen. For simplicity's sake, we employ the same starting condition as in the previous Section 5.2.3.

Using said starting condition, we ran both simulations for $240s$ of simulation time, with a spatial resolution of 1000 grid points, and a temporal resolution of $2.5ms$. The result of the two simulations on the different grids can be seen in Fig. 5.12.

The similar appearance of the separate simulation results for pressure deviation Δp and vertical wind speed w is no optical illusion. It can also be verified numerically:

When taking the difference between the two simulation results, the largest absolute difference[1] for Δp is $8.71 \cdot 10^{-11}$, and the largest absolute difference for w is $9.32 \cdot 10^{-9}$. We did not determine the difference for T, because the two T-grids are offset to one another by definition of the Charney-Phillips and Lorenz grids, so any comparison would require inaccurate averaging operations.

The minuscule difference of Δp and w between implementations reaffirms the assumption made in Section 5.2.3 that the error in energy conservation for the Charney-Phillips grid must stem from inaccuracies in T.

In summary, both implementations pass all tests.

[1] Also known as the L_∞-norm

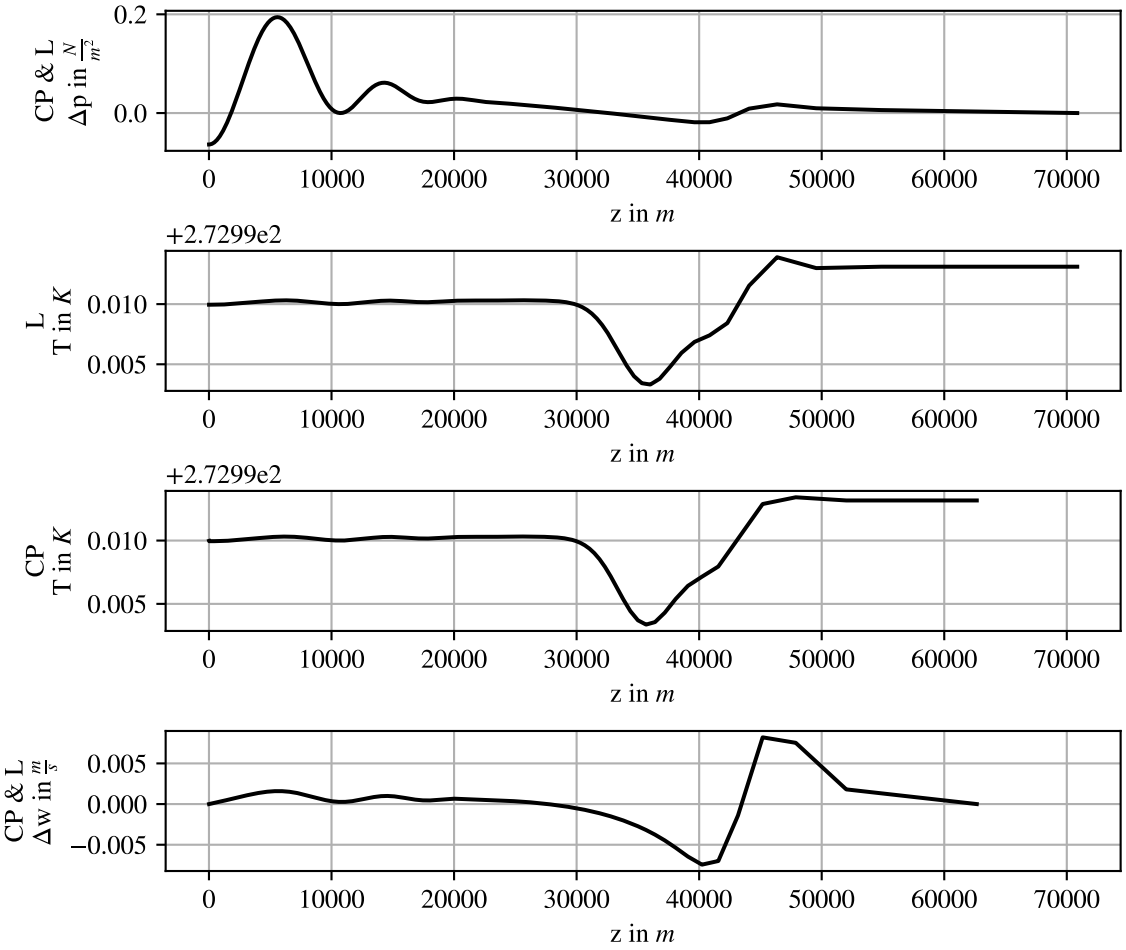

Figure 5.12.: Simulation results for Charney-Phillips grid (CP), and the Lorenz grid (L); the plots for $\ln p$ and Δw for both simulations were nearly identical, so both were only plotted once. Both simulations were run for 240s with 1000 grid points and a time step size of 2.5ms.

6. Conclusion

In this chapter we summarize the thesis and suggest some possible areas of future work based on the Python tool. The code is freely available for this purpose as a Github repository at `https://github.com/Linus-H/Vertical-Integration`.

6.1. Summary

In this thesis we first gave a basic introduction to the Navier Stokes Equations for numerical weather prediction. Having defined the most general equation system, we then rearranged and simplified this system, by assuming that horizontal wind $\mathbf{V}$ is zero. Thereafter, we substituted the variable z by a new vertical coordinate s which leads to a new equation system with some advantageous properties. For example, the substitution leads to a high spatial resolution in the lower areas of the atmosphere in contrast to the low spatial resolution at the top of the atmosphere. On top of this, this substitution allows for the upper bound for the domain to be found dynamically instead of setting it at the beginning of the simulation manually. We employed the resulting equation system in the later sections.

Having established the version of the NSE to be simulated in Chapter 2, we explained the applied methodology for spatial and temporal discretization in Chapter 3. Then we outlined the modular framework of the software in Chapter 4, and we gave an example of an implementation at the end of the chapter. Finally, we described the tests conducted on the implementation and their results in Chapter 5.

6.2. Future Work

There are multiple directions in which the the work done in this thesis can be expanded. The first category consists of more experiments that could be conducted on the non-hydrostatic equation system.

- Observing the effects of varying temporal and spatial resolution; the effects would be expected to be quite significant, as the non-hydrostatic equation system is non-linear.

- Experimenting with other time integrations methods/other integrators, to see whether they yield significantly different results from the Runge-Kutta-methods primarily used in this thesis.
- Measuring the wave-speeds/speed of sound of waves at different wave-lengths.

The second category of possible expansion are comparative in nature:

- compare the simulation results of the non-hydrostatic NSE to simulation results from implementations with the hydrostatic NSE.
- compare the non-hydrostatic implementation to real-world measurements, by adding the remaining two dimensions of the horizontal to the implementation.

Lastly, it would be interesting to define standardized benchmarks for the non-hydrostatic NSE. So far, such benchmarks are mainly adopted for systems employing the hydrostatic NSE [29], making standardized comparisons between different implementations of then non-hydrostatic NSE difficult.

A. Appendix

Derivation: Change of Energy per Area over Time

For this derivation we use the non-hydrostatic NSE derived in Section 2.5. We also assume that s_{top} and s_{bottom} are constant.

$$
\begin{aligned}
\frac{\partial E'}{\partial t} &= \frac{\partial}{\partial t} \int_{s_{top}}^{s_{bottom}} \frac{1}{g}(C_v T + \frac{1}{2}w^2 + gz(s)) \left(\frac{\partial \pi}{\partial s}\right) ds \\
&= \int_{s_{top}}^{s_{bottom}} \frac{1}{g} \left(C_v \frac{\partial T}{\partial t} + \frac{1}{2}\frac{\partial w^2}{\partial t} + g\frac{\partial z(s)}{\partial t}\right) \left(\frac{\partial \pi}{\partial s}\right) ds \\
&\quad + \int_{s_{top}}^{s_{bottom}} \frac{1}{g}(C_v T + \frac{1}{2}w^2 + gz(s)) \left(\frac{\partial}{\partial t}\frac{\partial \pi}{\partial s}\right) ds \\
&= \int_{s_{top}}^{s_{bottom}} \frac{1}{g} \left(C_v \frac{\partial T}{\partial t} + w\frac{\partial w}{\partial t} + g\frac{\partial z(s)}{\partial t}\right) \left(\frac{\partial \pi}{\partial s}\right) ds \\
&= \int_{s_{top}}^{s_{bottom}} \frac{1}{g} \left(\frac{C_v RT}{C_p}\frac{\partial \ln p}{\partial t} - gw\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right) + g\frac{\partial}{\partial t}\left(\frac{R}{g}\int_s^{s_{bottom}} \frac{T}{p}\left(\frac{\partial \pi}{\partial s}\right) ds'\right)\right) \\
&\qquad \cdot \left(\frac{\partial \pi}{\partial s}\right) ds
\end{aligned}
$$

In order to enhance readability, the three terms are now transformed separately, starting with the first term (with $C_v = C_p - R$).

$$
\begin{aligned}
\frac{C_v RT}{C_p}\frac{\partial \ln p}{\partial t} &= \frac{(C_p - R)RT}{C_p}\frac{g}{1 - \frac{R}{C_p}}\frac{p}{RT}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s} \\
&= gp\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s}
\end{aligned}
$$

For now, the second term can stay as is. We reshape the third term as follows (assuming $w(s_{bottom}) = 0$):

$$\begin{aligned}
g\frac{\partial}{\partial t}\left(\frac{R}{g}\int_s^{s_{bottom}} \frac{T}{p}\left(\frac{\partial \pi}{\partial s}\right) ds'\right) &= R\int_s^{s_{bottom}} \left(\frac{\partial \pi}{\partial s}\right)\frac{\partial}{\partial t}\left(\frac{T}{p}\right) ds' + R\int_s^{s_{bottom}} \frac{T}{p}\left(\frac{\partial}{\partial t}\frac{\partial \pi}{\partial s}\right) ds' \\
&= R\int_s^{s_{bottom}} \left(\frac{\partial \pi}{\partial s}\right)\frac{\partial}{\partial t}\left(\frac{T}{p}\right) ds' \\
&= R\int_s^{s_{bottom}} \left(\frac{\partial \pi}{\partial s}\right)\left(-\frac{g}{R}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s}\right) ds' \\
&= -g\int_s^{s_{bottom}} \left(\frac{\partial w}{\partial s}\right) ds' \\
&= -g\left(w(s_{bottom}) - w(s)\right) \\
&= gw(s)
\end{aligned}$$

With these reshaped terms, E' can be written as:

$$\begin{aligned}
\frac{\partial E'}{\partial t} &= \int_{s_{top}}^{s_{bottom}} \frac{1}{g}\left(gp\left(\frac{\partial \pi}{\partial s}\right)^{-1}\frac{\partial w}{\partial s} - gw\left(1 - \frac{\partial p}{\partial s}\left(\frac{\partial \pi}{\partial s}\right)^{-1}\right) + gw\right)\left(\frac{\partial \pi}{\partial s}\right) ds \\
&= \int_{s_{top}}^{s_{bottom}} \left(p\frac{\partial w}{\partial s} + w\frac{\partial p}{\partial s}\right) ds \\
&= \int_{s_{top}}^{s_{bottom}} \left(\frac{\partial (wp)}{\partial s}\right) ds = w(s_{bottom})p(s_{bottom}) - w(s_{top})p(s_{top})
\end{aligned}$$

Bibliography

[1] R Gommes, A Challinor, H Das, MA Dawod, L Mariani, B Tychon, R Krüger, U Otte, RER Vega, and W Trampf. Guide to Agricultural Meteorological Practices. *World Meteorological Organization*, (134), 2010.

[2] Robert Sharman and Todd Lane. *Aviation Turbulence*. Springer, 2016.

[3] Susanne Becken. The importance of climate and weather for tourism: literature review. 01 2010.

[4] Mark Carey. Climate and history: a critical review of historical climatology and climate change historiography. *Wiley Interdisciplinary Reviews: Climate Change*, 3(3):233–249, 2012.

[5] Jean Coiffier. *Fundamentals of numerical weather prediction*. Cambridge University Press, 2011.

[6] Peter J Mohr, Barry N Taylor, and David B Newell. CODATA recommended values of the fundamental physical constants: 2006. *Journal of Physical and Chemical Reference Data*, 80(3):633–1284, 2008.

[7] Klaus Langeheinecke, Peter Jany, and Gerd Thieleke. *Thermodynamik für Ingenieure*. Springer, 1993.

[8] Charles W Beckett and Joseph Hilsenrath. Tables of thermal properties of gases, comprising tables of thermodynamic and transport properties of air, argon, carbon dioxide, carbon monoxide, hydrogen, nitrogen, oxygen and steam. 1955.

[9] Marco Cabral. Navier Stokes and Thermodynamics V0.93. January 2001.

[10] Yunus A Cengel. *Fluid mechanics*. Tata McGraw-Hill Education, 2010.

[11] Jihyeon Jang and Song-You Hong. Comparison of simulated precipitation over East Asia in two regional models with hydrostatic and nonhydrostatic dynamical cores. *Monthly Weather Review*, 144(10):3579–3590, 2016.

[12] Dale R Durran. *Numerical methods for fluid dynamics: With applications to geophysics*, volume 32. Springer Science & Business Media, 2010.

[13] Mark B. Carpenter and Christopher M. Keane. *The Geoscience Handbook: AGI Data Sheets, Fifth Edition*. American Geosciences Institute, 2016.

[14] Akira Kasahara. Various vertical coordinate systems used for numerical weather prediction. *Monthly Weather Review*, 102(7):509–522, 1974.

[15] René Laprise. The Euler equations of motion with hydrostatic pressure as an independent variable. *Monthly weather review*, 120(1):197–207, 1992.

[16] Geoffrey K Vallis. *Atmospheric and oceanic fluid dynamics*. Cambridge University Press, 2017.

[17] Sajal K Kar and Richard P Turco. Formulation of a lateral sponge layer for limited-area shallow-water models and an extension for the vertically stratified case. *Monthly weather review*, 123(5):1542–1559, 1995.

[18] J-J Baik, Y-H Kim, J-J Kim, and J-Y Han. Effects of boundary-layer stability on urban heat island-induced circulation. *Theoretical and Applied Climatology*, 89(1-2):73–81, 2007.

[19] Gordon D Smith and Gordon D Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

[20] Steven G Johnson. Notes on FFT-based differentiation. *MIT Applied Mathematics*, (April), 2011.

[21] Xiong-Shan Chen. Use of the aliased spectral model in numerical weather prediction. *Monthly weather review*, 125(11):2998–3007, 1997.

[22] Frederick G Shuman. History of numerical weather prediction at the National Meteorological Center. *Weather and Forecasting*, 4(3):286–296, 1989.

[23] D Holdaway, J Thuburn, and N Wood. Comparison of Lorenz and Charney–Phillips vertical discretisations for dynamics–boundary layer coupling. Part I: Steady states. *Quarterly Journal of the Royal Meteorological Society*, 139(673):1073–1086, 2013.

[24] Ling-Hsiao Lyu. Appendix C. Derivation of the Numerical Integration Formulae. lecture, August 2016.

[25] Endre Süli and David F Mayers. *An introduction to numerical analysis*. Cambridge university press, 2003.

[26] Terry J Larson and EJ Montoya. Stratosphere and mesosphere density-height profiles obtained with the x-15 airplane. 1963.

[27] Howard C Hardy, David Telfair, and WH Pielemeier. The velocity of sound in air. *The Journal of the Acoustical Society of America*, 13(3):226–233, 1942.

[28] Edward N Lorenz. Energy and numerical weather prediction. *Tellus*, 12(4):364–373, 1960.

[29] David L Williamson, John B Drake, James J Hack, Rüdiger Jakob, and Paul N Swarztrauber. A standard test set for numerical approximations to the shallow water equations in spherical geometry. *Journal of Computational Physics*, 102(1):211–224, 1992.