Discontinuous Galerkin Methods for Numerical Weather Prediction

DG in a large-eddy simulation

C. Caljouw

TU Delft

Royal Netherlands Meteorological Institute
Ministry of Infrastructure and the Environment

Discontinuous Galerkin Methods for Numerical Weather Prediction

DG in a large-eddy simulation

by

C. Caljouw

in partial fulfilment of the requirements for the degree of
Master of Science in Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Friday November 17th, 2017 at 16:00 PM.

Student number:	4223098	
Project duration:	February 15, 2017 – November 17, 2017	
Daily supervisors:	Dr.ir. D. den Ouden-van der Horst	TU Delft
	Prof.dr. A.P. Siebesma	TU Delft, Royal Netherlands Meteorological Institute
Responsible professor:	Prof.dr.ir. C. Vuik	TU Delft
Other thesis committee members:	Dr.ir. M. Keijzer	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

Royal Netherlands Meteorological Institute
Ministry of Infrastructure and the Environment

Abstract

The coarse grid of numerical weather prediction and climate models requires parametrization models to resolve atmospheric processes that are smaller than the grid size. For parametrization development, these processes are simulated by a high resolution model. At the Royal Netherlands Meteorological Institute, the Dutch Atmospheric Large-Eddy Simulation (DALES) is used [15]. This three-dimensional high resolution model uses advection schemes that are too diffusive when steep gradients are present. In this thesis, an advection scheme based on the Discontinuous Galerkin (DG) method is implemented for DALES.

The DG method is known to be dispersive [22]. To remove those non-physical oscillations, the moment limiter of Krivodonova is used [18]. Krivodonova constructed the limiter for one- and two-dimensions. In this thesis the moment limiter and limiting order are derived for three-dimensions.

DALES is a model based on the finite difference method and uses operational splitting. Therefore, the DG advection scheme needs a mapping from each cell average to all nodal values that are needed for one DG cell, and a mapping back, which we called mapping a and b respectively. Mappings a that are discussed are taking the cell average as value for all nodal points of the DG cell (cell average a), and taking the L_2-projection of the cell average to the continuous finite element space (L_2-projection). This thesis describes mappings b that calculate cell averages of nodal DG values (cell average b) and calculate the cell averages of the tendencies of DG values (cell average of tendency). Using cell average a combined with cell average of tendency, made the DG method as diffusive as the first order upwind scheme. Substituting the cell average a method with the L_2-projection, the DG method became very dispersive, meaning that there was not enough diffusion. At last, cell average b was tested with the L_2-projection. Its numerical results showed that the speed of the advection was slower than the theoretical velocity. Therefore, a method is suggested which does not need mappings. An option could be a supergrid that takes multiple DALES cells as a DG cell.

Keywords- Runge-Kutta discontinuous Galerkin (RKDG), large-eddy simulation, three-dimensional moment limiter.

Preface

This thesis is the last requirement to fulfil the graduation requirements for the degree Master of Science in Applied Mathematics at the Delft University of Technology. The research of this project took place during an internship at the Research and Development department for Weather and Climate Modelling of the Royal Netherlands Meteorological Institute, also known as KNMI. I would like to sincerely thank several people who were involved in this project.

First of all, I would like to thank my supervisors. Dennis den Ouden-van der Horst, thank you so much for taking the time to help me and to read and review my work every single time I asked you to. I also would like to thank Pier Siebesma for giving me the opportunity to do an internship at the KNMI and for introducing and teaching me the beautiful aspects of the atmosphere, especially clouds.

Second of all, I would like to thank my colleagues at the KNMI who, among other things, showed me the world of politics. I had never thought I could learn this much about politics during an internship at a meteorological institute. Also I would like to mention and thank Frederik Jansson and Gijs van den Oord for helping me out with Fortran and Git. Without those two, it would have taken me much more time to implement a method in Fortran.

Moreover, I would like to thank my family and friends for supporting me throughout the project. Especially Matthijs, whether it is about my struggles and victories of this project or the politic viewpoints of my colleagues, you would always hear me out.

Lastly, I want to thank prof.dr.ir. C. Vuik and dr.ir. M. Keijzer for their time and willingness to be a member of my graduation committee.

Last of all, I wish you much pleasure in reading, criticizing and valuing my Master thesis.

Cindy Caljouw
Utrecht, October 2017

Contents

1 Introduction

Mathematically modelling atmospheric phenomena is one of the two main challenges of numerical weather prediction (NWP) and climate models. The most important atmospheric processes are turbulence, convection and cloud formation. However, the coarse grid of the models does not resolve processes that are smaller than the grid size, therefore, these sub-grid processes require parametrizations. For parametrization development, the processes are simulated in a model with a higher resolution. At the Royal Netherlands Meteorological Institute (KNMI), the Dutch Atmospheric Large-Eddy Simulation model, also known as DALES, is used. On top of that, DALES can be used to predict weather on a smaller domain with a higher resolution, for example to provide short-range forecasts for near-surface wind and solar power for the renewable energy sector.

Nevertheless, DALES can still be improved, especially the implemented advection schemes. Each finite difference advection scheme of DALES has its own favourable properties, like computational time or accuracy. However, the implemented high accuracy methods are still too diffusive and/or dispersive when steep gradients in temperature, moisture and momentum are present.

The other main challenge of NWP and climate models is to evaluate these models as accurate and efficiently as possible. This can be done by using all the available computational resources. The fundamentals of a finite difference scheme do not take full advantage of the architecture of the modern-day computers and is thus not the most computational efficient method.

To solve these two problems, the discontinuous Galerkin (DG) method is suggested. DG is an attractive method, because it allows discontinuities, it has a geometric flexibility and it has a high parallel-scalability due to a compact stencil. However, it is known that non-physical oscillations are generated with DG [22]. Therefore, a limiter has to be added to remove these non-physical oscillations.

The goal of this thesis is to implement an advection scheme based on the DG method in DALES without changing the other subroutines. The corresponding research question is:

> Can DG be used as an advection solver in DALES such that:
>
> - the computational time is less than the WENO method, and/or
> - the numerical accuracy is better than the WENO method, while the computational time is not doubled?

1.1. Methodology

Before the DG advection scheme is implemented in DALES, the method is created in multiple stages. In the literature study prior to this work [5], DG is tested for a one-dimensional test case in MATLAB. Thereafter, the advection equation of DALES is tested in two- and three-dimensions in Fortran, in which DALES is implemented. Last but not least, the advection scheme is tested in a stripped version

of DALES such that the computational time is reduced and no other subroutines can influence the results.

1.2. Outline

In Chapter 2, the background of DALES is given by explaining the origin of the model equations. Thereafter, in Chapter 3 the numerical methods of DALES are explained and the shortcomings of the implemented advection schemes are shown. Chapter 4 describes the discontinuous Galerkin method and the three-dimensional moment limiter is derived. On top of that, the numerical results are shown for both DG with and without moment limiter. Thereafter, DG is tested in the stripped version of DALES and compared with the most accurate advection scheme of DALES, the WENO method, in Chapter 5. This thesis is concluded in Chapter 6 with a conclusion, some encountered implementation problems, some remarks and recommendations for further research.

2

Atmospheric modelling

In this chapter some general information is given on the atmosphere. Furthermore, the general equations for atmospheric modelling are shown and explained. Finally, an explanation is given on the adjustments for the governing equations of DALES, its boundary conditions and its prognostic variables.

2.1. The atmosphere

The atmosphere is a layer of multiple gasses, known as air, around the Earth that is kept in place by the Earth's gravity. The atmosphere consists of a number of layers:

- troposphere,
- stratosphere,
- mesosphere,
- thermosphere,
- exosphere.

Each layer has its own properties like composition and temperature profile. In Figure 2.1, the different layers are shown with the typical temperature and air pressure as a function of height.

The many different processes that take place in the atmosphere make up the daily weather that we experience. The most important ones that take place close to the Earth's surface are turbulence, convection and particularly, cloud formation. These three processes are results of small-scale movement of air in the atmosphere. The large scale atmospheric circulation acts on scales larger than the domain used by DALES. These are in general imposed as a large scale forcing in DALES and are not part of the actual dynamics within the model.

2.1.1. Air movement

Air moves as a result of density differences. For the large scale air circulation, pressure differences and the Coriolis force, which is an apparent force resulting from the Earth's rotation, however, for the high resolution model, density differences are more relevant. The density of air depends on temperature and pressure, and because of certain factors, the density can differ locally. In this section, the cause of density differences are explained and thereafter several factors are given.

As the Sun warms up the Earth's surface, water evaporates and the air near the surface warms up. Warm air and moist air are lighter than cold air and dry air, therefore, the warm air parcel with water vapour travels higher into the sky. How far the air parcel rises, depends on the temperature of the surrounding air and the amount of water vapour the air parcel holds.

During the upward motion of the air parcel, the air pressure of this air parcel decreases and as a result the air parcel expands. Due to this expansion the air parcel loses energy and consequently, the air

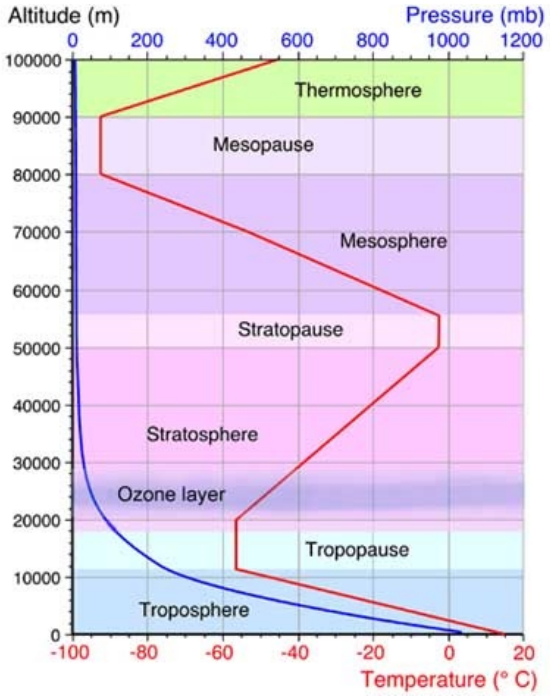


Figure 2.1: Temperature and pressure of the atmospheric layers. Image taken from [9].

parcel cools (see Figure 2.2). At a certain point, the air pressure and the temperature have decreased so much that the air parcel cannot hold the amount of water vapour any more; the air parcel becomes oversaturated. Further cooling leads to condensation, however, the air parcel keeps rising by the release of latent heat during condensation. For this reason, the water droplets can get higher in the sky and can even become ice crystals.

The tiny water droplets and ice crystals are so small and light that they are able to stay up in the air. When there is a visible amount of tiny water droplets, ice crystals or a mixture of both, it forms a cloud in the sky.

Ultimately the Sun is the major reason behind all rising motion. Other factors causing air to move, are [23]:

- Orography: Air parcels are forced to rise because of physical obstacles like mountains.
- Large scale convergence: Streams of air flowing from different directions are forced to rise where they meet.
  - For example frontal systems: When warm air and cold air meet each other, the warm air mass rises over the cold air mass.

Just as the factors of air movement are discussed, the formation of clouds is explained. However, another important ingredient has not been mentioned, namely the particles that are needed for the water vapour to condense onto. These particles are called condensation nuclei. Examples of condensation nuclei are salt, dust and smoke particles. Nevertheless, there are always enough condensation nuclei available to initiate condensation in the case of oversaturation.

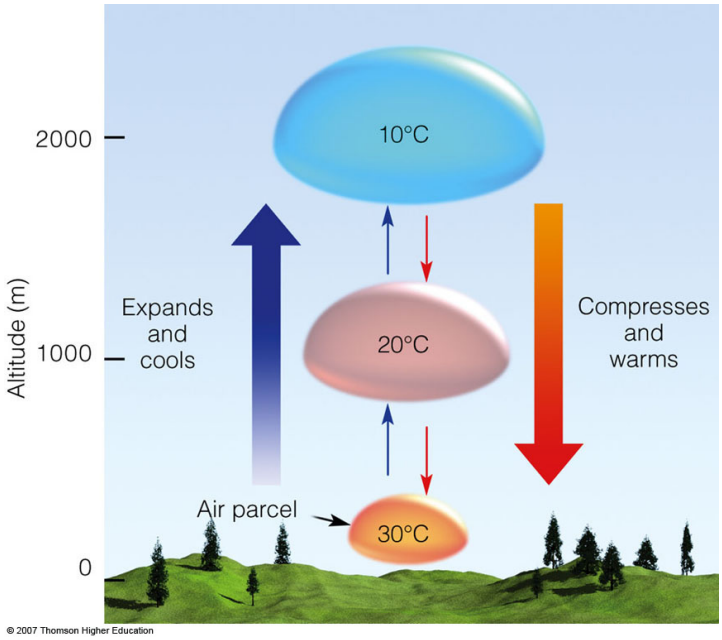


Figure 2.2: Temperature changes of air parcel. Image taken from [29].

2.2. Model equations

The dynamics of the atmosphere can be described by the conservation laws of momentum, mass, energy and moisture. With the four laws, the time evolution of momentum, density, temperature and humidity can be described.

Conservation of momentum

First, for a small package of air, the momentum must be conserved. The *Navier-Stokes equations* describe the differential equations that the motion of air in the atmospheric boundary layer must satisfy:

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) = -\nabla p - 2\rho(\boldsymbol{\Omega} \times \boldsymbol{u}) - \rho \boldsymbol{g}, \tag{2.1}$$

where ρ denotes the density, $\boldsymbol{u}$ the velocity field, p the pressure, $\boldsymbol{\Omega}$ the Coriolis force and $\boldsymbol{g}$ the effective gravity, which is the sum of the true gravity and the centrifugal force. [1] The equation is derived from Newton's second law for motion relative to a rotating coordinate frame and says that the acceleration that follows the relative motion in the rotating frame is equal to the sum of the effective gravity, the pressure gradient and the Coriolis force. More information on the derivation of the equation can be found in [16].

Conservation of mass

Second, conservation of mass must hold, which is known as the *continuity equation*[16]:

$$\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \boldsymbol{u} = 0. \tag{2.2}$$

This means that the local rate of change of density equals the divergence of mass.

Conservation of energy

Third, the energy of the system is conserved. By all means, the change in specific internal energy of the system e is equal to the difference between the heat added to the system dQ and the work done by the system dw, known as the first law of thermodynamics (per unit mass):

$$de = dQ - dw = Tds - pd\alpha. \tag{2.3}$$

[1] $\otimes$ denotes the outer product, $\boldsymbol{u} \otimes \boldsymbol{v} = \boldsymbol{u}\boldsymbol{v}^T$, and $\times$ the cross product, $\boldsymbol{a} \times \boldsymbol{b} = \|\boldsymbol{a}\| \, \|\boldsymbol{b}\| \sin(\theta)$ where θ is the angle between $\boldsymbol{a}$ and $\boldsymbol{b}$ in the plane where the two vectors are contained.

To create the prognostic equations for thermodynamic variables like temperature, the ideal gas law is needed:

$$p = \rho R_d T = \frac{R_d T}{\alpha}, \tag{2.4}$$

where R_d is the gas constant, T the temperature and α the specific volume.

Using $dp\alpha = pd\alpha + \alpha dp$, the ideal gas law (2.4) and the use of the enthalpy h (which is defined as $h = e + p\alpha$), equation (2.3) can be rewritten:

$$dh = c_p dT = Tds + \alpha dp, \tag{2.5}$$

where c_p is the heat capacity at constant pressure for dry air and ds is the change in entropy.

The heating of air changes both the temperature and the pressure of the air parcel. Therefore, the "potential" temperature of the air parcel is defined.

When a dry air parcel is adiabatically moved, it means that the system does not lose or gain heat ($dQ = Tds = 0$). Using this information, equation (2.4) and integrating (2.5) from state $T_1 = T$, $p_1 = p$ to the reference state $T_0 = \theta$, $p_0 = 1000$ hPa, we obtain:

$$\theta = T\left(\frac{p}{p_0}\right)^{-R_d/c_p} \Leftrightarrow T = \theta\Pi, \tag{2.6}$$

in which Π is the exner function given by $\Pi = \left(\frac{p}{p_0}\right)^{R_d/c_p}$.

The potential temperature θ describes what temperature a dry air parcel at a pressure p and temperature T would have if it were compressed or expanded to the standard pressure p_0. Consequently, we have redefined the temperature such that the pressure contribution is removed. Moreover, θ is conserved under dry adiabatic changes.

However, an air parcel can contain water vapour or even little water droplets. For this mixture, another variable must be introduced which is also conserved under the phase transition between liquid water and water vapour.

Conservation of moisture

Since the air is a mixture, the mass of air can be written as the mass of liquid water m_l, water vapour m_v and dry air m_d:

$$m = m_l + m_v + m_d. \tag{2.7}$$

Instead of the masses, the mass fractions q are used in equations.

$$q_c = \frac{m_l}{m}, \quad q_v = \frac{m_v}{m}, \quad q_t = q_v + q_c. \tag{2.8}$$

The total water specific humidity q_t is the sum of the water vapour specific humidity q_v and the cloud liquid water specific humidity q_c. One can understand that q_t is conserved under the phase transitions from liquid water to water vapour and vice versa.

Consequently, a temperature variable can be constructed which is also conserved under the phase transition. During the phase transition latent heat is released, i.e. $dQ = Tds = Ldq_c$ where L is the latent heat of vaporization. Therefore, when using the same steps as for (2.6), we get a temperature conserved under the phase transition:

$$\theta_l = \theta \exp\left(\frac{-L}{c_p T} q_c\right) \approx \theta - \frac{L}{c_p \Pi} q_c. \tag{2.9}$$

The liquid water potential temperature θ_l can be linearly approximated, since q_c is usually very small ($\sim 10^{-3}$, a few grams of liquid water per kilogram of air mixture). More information on the thermodynamics of atmospheric modelling can be found in [16] and [1].

When there is no precipitation or other explicit sources, θ_l and q_t are conserved and for conserved variables $\phi(x, y, z, t)$ it holds that the total derivative $\frac{D\phi}{Dt} = 0$. Writing out the total derivative, we obtain:

$$\begin{aligned} \frac{D\phi}{Dt} &= \frac{\partial \phi}{\partial t} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial \phi}{\partial z}\frac{\partial z}{\partial t}, \\ &= \frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi \qquad\qquad = 0, \end{aligned} \tag{2.10}$$

which is the advection equation. If there is an explicit source S_ϕ, for example precipitation, the equation is given by:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = S_\phi, \tag{2.11}$$

2.3. Governing equations of DALES

DALES is a large-eddy simulation, for this reason the equations are slightly different from equations (2.1), (2.2) and (2.11).

A large-eddy simulation is a numerical model for turbulence. For the simulation of the time evolution of air flow, all different time and length scales affect the flow field, therefore, all scales must be resolved. However, the difference between the largest ($\sim$ 1 km) and smallest scales ($\sim$ 1 mm) of eddies is substantial. LES models reduce the computational cost by parametrizing the smallest length scales.

The eddies that are smaller than the grid size, called the sub-grid scales, are filtered out of the numerical solution. This is done by using a low-pass filter, which can be seen as an averaging of the flow quantities in time and space. The sub-grid scale dynamics are subsequently parametrized by the sub-grid model. For more details on LES models, the book of Berselli et al. [2] can be read.

Moreover, the variables $\varphi \in \{u, v, w, \theta_l, q_t\}$ in DALES are decomposed into a resolved and a subgrid part by averaging the field over the grid box:

$$\varphi = \tilde{\varphi} + \varphi', \tag{2.12}$$

such that the following properties hold:

$$\tilde{\tilde{\varphi}} = \tilde{\varphi} \quad \& \quad \widetilde{\varphi'} = 0. \tag{2.13}$$

The $\tilde{\cdot}$ denotes the quantity that is averaged over the LES filter width and the subfilter-scale fluctuations with respect to the filtered value is denoted with a prime $\cdot'$.

In DALES, the anelastic approximation is used. The anelastic approximation takes the density differences in the vertical direction of the continuity equation into account, while the Boussinesq approximation, used in the previous version DALES 3.2, takes no density variations into account unless the density is multiplied by the gravitational acceleration.

The filtered equations of DALES are derived by Böing and can be found in the appendix of his dissertation [4]. In general, these filtered equations can be stated as:

$$\frac{\partial}{\partial t}\tilde{\varphi} = -\frac{1}{\rho_0(z)}\nabla \cdot (\rho_0(z)\tilde{\varphi}\tilde{\boldsymbol{u}}) + \frac{1}{\rho_0(z)}\nabla \cdot (\rho_0(z)K_h \nabla \tilde{\varphi}) + S_\varphi, \tag{2.14}$$

where K_h is the eddy diffusivity coefficient, $\rho_0(z)$ the time-independent base state density and S_φ is the source term for variable φ. The focus of this thesis is the advection term, therefore, the diffusion and source term is ignored in this thesis, since DALES uses operational splitting. This means that these terms are solved in a routine separate from the advection routine.

2.3.1. Prognostic variables

The three mandatory prognostic variables of DALES are the velocity $\boldsymbol{u}$, the liquid water potential temperature θ_l and the sub-filter scale turbulence kinetic energy e, which is used in the parametrization of the sub-filter scale dynamics. On top of that, the total water specific humidity q_t, the rain water specific humidity q_r, the rain droplet number concentration N_r, and up to 100 passive and reactive scalars can be included. Even though q_t is not obligatory, the humidity is a very important variable and should always be used. The other additional prognostic variables do not have to be calculated unless these are used. Figure 2.3 shows the processes of all variables and how the variables are affected by them.

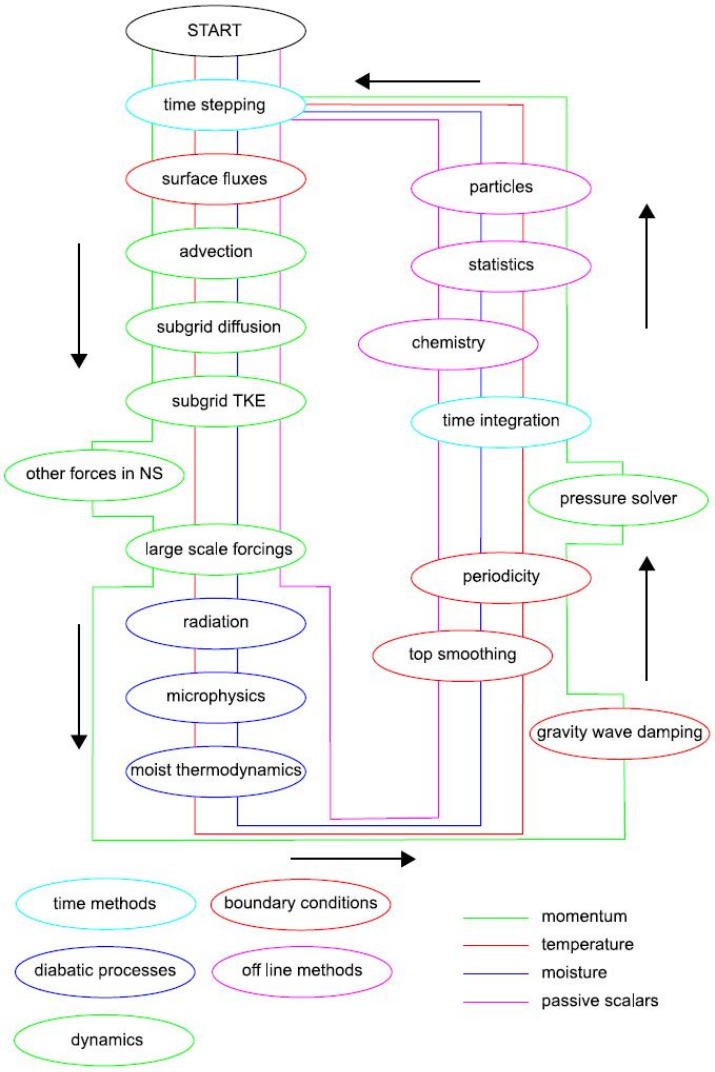


Figure 2.3: Flowchart of DALES. Image taken from [15].

2.3.2. Boundary conditions

Like any other model, boundary conditions are formulated for DALES. As only the advection scheme is important for this thesis, the constraints on the resolved quantities $\tilde{\boldsymbol{u}}$ and $\tilde{\varphi}$ are named explicitly.

For the lateral directions of the domain, periodic boundary conditions in the lateral directions are applied for all fields. For the boundary condition at the top, the following conditions are used:

$$\frac{\partial \tilde{u}}{\partial z} = \frac{\partial \tilde{v}}{\partial z} = 0, \quad \tilde{w} = 0, \quad \frac{\partial \tilde{\varphi}}{\partial z} = \text{constant in time.} \tag{2.15}$$

A sponge layer is used to constrain the sub-grid fluctuations of the velocity and the scalars, such that the oscillations are damped out.

At the bottom of the computational domain ($z = 0$), the boundary conditions become very complex. To be exact, it needs its own model to parametrize the turbulent drag and the exchange between the surface and the atmosphere. Fortunately, for the advection equation the surface has a simple no-slip boundary:

$$\tilde{u} = 0, \quad \tilde{v} = 0, \quad \tilde{w} = 0. \tag{2.16}$$

By definition there are no resolved fluctuations in the vertical direction at the surface, as a result the surface fluxes enter the domain at subfilter-scale. More information on the other processes of DALES can be found in [15].

3

Numerical methods of DALES for the advection equation

The present chapter describes the numerical methods that are important for the advection equation in DALES. First, the spacial discretisation of the variables are specified and thereafter, the time integration method is explained. Last but not least, a short summary of the shortcomings of the tested advection schemes during the literature study prior to this thesis work [5] is given.

3.1. Grid spacing

In DALES, a uniform Cartesian grid is used in the horizontal directions with optional stretching in the z-direction. Normally the horizontal grid sizes Δx and Δy are 100 m and the vertical grid size Δz is around 50 m. On top of that, a staggered grid in space is used, to be exact, the Arakawa C-grid which can be seen in Figure 3.1. An Arakawa C-grid defines the pressure p, the SFS-TKE e and the scalars φ in the centre of the cell, while the velocity components, u, v, and w, are defined at the faces of the cell. As w is given at a different height than the other variables, this level is called the half level, denoted by z_h, and the other variables are at the full level z_f. This is summarized in Table 3.1.

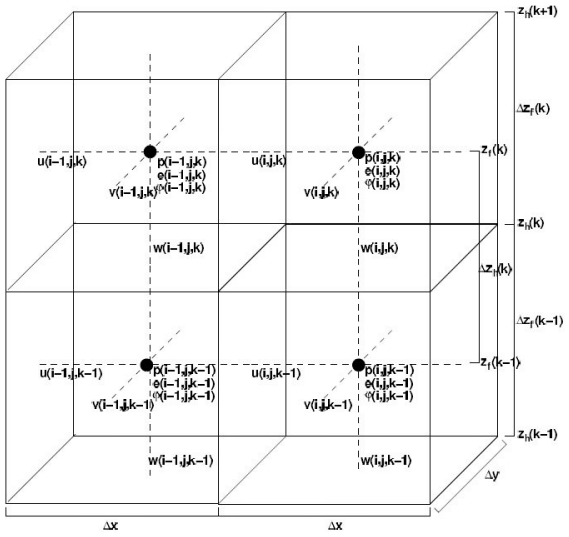


Figure 3.1: Arakawa C-grid. Image taken from [15].

Variable	Position	z level
p, e, φ	$\boldsymbol{x} + \frac{1}{2}(\Delta x, \Delta y, \Delta z)$	z_f
u	$\boldsymbol{x} + \frac{1}{2}(0, \Delta y, \Delta z)$	z_f
v	$\boldsymbol{x} + \frac{1}{2}(\Delta x, 0, \Delta z)$	z_f
w	$\boldsymbol{x} + \frac{1}{2}(\Delta x, \Delta y, 0)$	z_h

Table 3.1: Position of the variables in the Arakawa C-grid.

3.2. Time integration method

In DALES, the partial differential equations (PDEs) are solved in a semi-discrete way. This means that the system is discretized in two stages; first only in space, thereafter in time. By only discretizing in

space, the remaining problem problem exists of only ordinary differential equations (ODEs):

$$\frac{\partial}{\partial t}\phi = g(\phi). \tag{3.1}$$

Henceforth, an ODE solver, in this case a time integration method, can be used to obtain the solution of the PDEs. Solving a PDE system in a semi-discrete way is also known as the method of lines [20].

The time integration method that DALES uses, is the third order Runge-Kutta method (RK3) which is an explicit time integration method. This method calculates the next time step ϕ^{n+1} in three steps as follows:

$$\phi^{*} = \phi^{n} + \frac{\Delta t}{3} g(\phi^{n}), \tag{3.2a}$$

$$\phi^{**} = \phi^{n} + \frac{\Delta t}{2} g(\phi^{*}), \tag{3.2b}$$

$$\phi^{n+1} = \phi^{n} + \Delta t\, g(\phi^{**}), \tag{3.2c}$$

where the asterisks denote the intermediate time steps.

The size of the time step Δt is determined adaptively to keep the numerical solution stable. The two criteria that limit the time steps are the Courant-Friedrichs-Lewy (CFL) condition:

$$\mathrm{CFL} = \max\left(\left|\frac{u\Delta t}{\Delta x}\right| + \left|\frac{v\Delta t}{\Delta y}\right| + \left|\frac{w\Delta t}{\Delta z}\right|\right),$$

and the diffusion number d [38], which is needed for the diffusion terms that arose from the LES-filtering:

$$d = \max\left(\sum_{i=1}^{3} \frac{K_m \Delta t}{\Delta x_i^2}\right),$$

where K_m is the eddy viscosity coefficient.

3.3. Shortcomings of the implemented advection schemes of DALES

In DALES eight advection schemes can be chosen. Each advection scheme has their own favourable properties such as high accuracy or little computation time, however they also have their own cons. In the literature study of this thesis [5], the following four advection schemes are tested:

- First order upwind,
- Second order central difference,
- Fifth order upwind,
- Weighted essentially non-oscillatory (WENO) method.

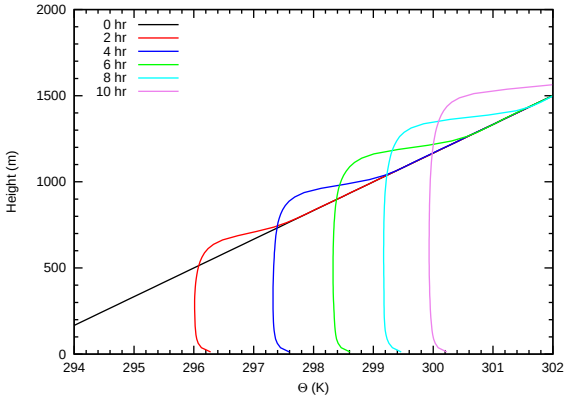
These advection schemes in DALES have been tested with a one-dimensional test case in [5]. Every scheme has its own advantages and disadvantages, but none of them are as flawless as they need to be.

The low order difference methods may have a low computational time, also when implemented for higher dimensional problems. Nonetheless, the first order upwind is overly diffusive and the second order central differencing has too many dispersive errors; the initial condition is not recognizable after a few time steps.

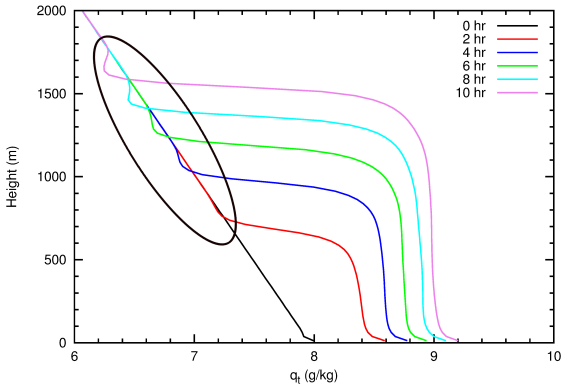
In addition, the high accuracy methods, the fifth order upwind and WENO method, are still diffusive, though they are not overly diffusive like the first order upwind. Moreover, the fifth order upwind is dispersive. This problem is absent when the WENO method is used, but its computational time is much

longer.

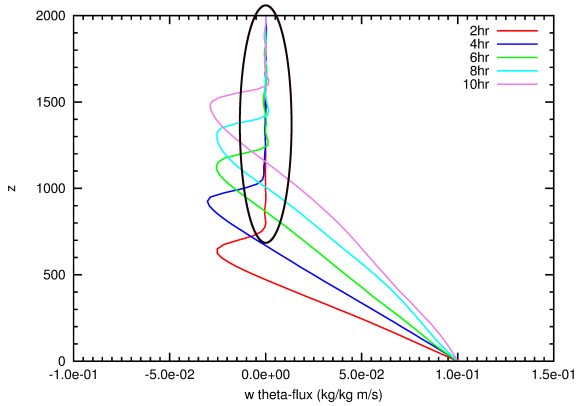
On top of that, the WENO and fifth order upwind method have a time lag making long simulations inaccurate. This time lag also appears in the solutions of the first order upwind and second order central methods. It seems that the speed of the numerical solution is lower than the actual speed u, leading to a time lag that increases with time. More research is needed to find the source of the time lags.



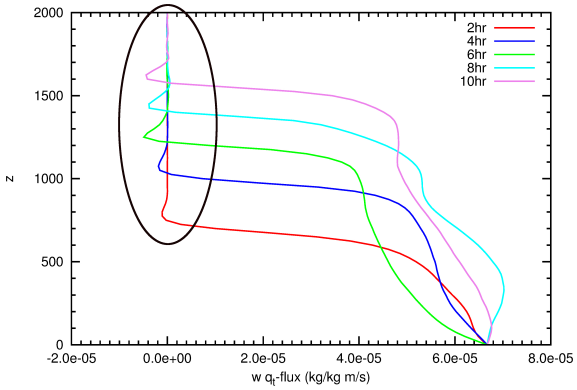
(a) The potential temperature θ_l as a function in height z.



(b) The total humidity q_t as a function in height z.



(c) The flux $w\theta_l$ as a function in height z.



(d) The flux wq_t as a function in height z.

Figure 3.2: The time evolution of the development of a convective boundary layer for the averaged variables in the horizontal directions using second order central difference: the total humidity q_t, the potential temperature θ_l and their corresponding fluxes. Images are provided by A.P. Siebesma.

In Figure 3.2, the problems with the current advection schemes are illustrated. In this figure, the second order central difference method is used to simulate the development of a dry convective boundary layer for humidity and potential temperature. The second order central difference method is known for its dispersion which one can see in Figure 3.2c. In addition, the overshoots in the total humidity and its flux are non-physical, see Figures 3.2b and 3.2d. These problems are an effect of the incorrect approximation of the fluxes.

For example, the flux at position $x_{i+1/2}$ with $\varphi_{i+1} = 1$, $\varphi_i = 0$ and wind speed is $u_{i+1/2} = 1$, is approximated with the second order central difference method:

$$\hat{F}^{2\text{nd}}_{i+1/2} = \frac{u_{i+1/2}}{2}(\varphi_{i+1} + \varphi_i) = \frac{1}{2}(1+0) = \frac{1}{2}. \tag{3.3}$$

In other words, more φ is advected to its neighbour than there is (see Figure 3.3).

These overshoots are not present when an upwind scheme is used which is more diffusive in general, however, when these schemes are used to simulate stratocumulus clouds, the stratocumulus clouds

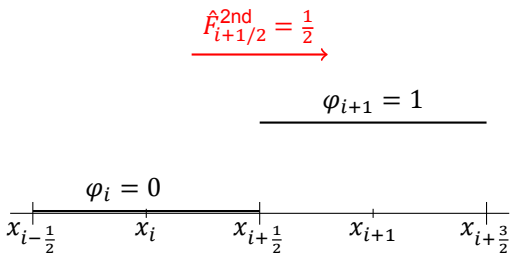


Figure 3.3: Example of incorrect approximation of the flux with the second order central difference method.

break up faster than observed in nature. Due to excessive numerical diffusion of the advection scheme, the mixing between the stratocumulus and the dry and warmer air aloft becomes too strong. Note that the moisture fluxes close to the surface in Figure 3.2d are strongly fluctuating; this is not caused by the numerical inaccuracy advection scheme, but is just a finite size effect due to the fact that the simulation was done a rather small domain.

As the advection scheme is important for the DALES model, an other advection scheme should be implemented such that either the accuracy is better than the implemented advection schemes of DALES or the computational time is less. The best possible outcome of this thesis project is to have an advection scheme that is better in both accuracy and computational time.

4

Discontinuous Galerkin for the advection equation

In this chapter the discontinuous Galerkin (DG) method is explained. First, the differences are shown between DG and the standard methods: finite difference methods, finite volume methods and finite element methods. Then the basics of DG will be explained. Thereafter, DG is created for the two- and three-dimensional advection equation of DALES. Moreover, the moment limiter of Krivodonova is extended to three-dimensions.

4.1. Relations and differences between FDM, FVM, FEM and DG

There are several numerical methods that can be used to solve partial differential equations (PDE), such as:

- the finite difference method (FDM),
- the finite volume method (FVM),
- the finite element method (FEM).

The most popular method is the finite difference method, which DALES is also based on [22]. In this section, some general fundamental differences between the methods are named.

The FDM solves PDEs directly by approximating the derivatives using local Taylor expansions, while FVM solves the differential equations after integration over a control volume. For FEM, the domain is divided in a finite number of non-overlapping elements so that the numerical solution is reconstructed on every element by giving weights to specified basis functions. The weights are found by solving the equations that are obtained by using the weak form of the PDEs.

Both FEM and FDM find the nodal values while FVM gives the cell averages. A disadvantage of FDM is the difficulty that comes into play when having unstructured grids, while this is not a problem for FVM and FEM. Only the FVM has the advantage of guaranteeing mass conservation and allowing discontinuities.

DG is a combination of FEM and FVM. FVM is actually a DG method with constant basis functions and DG is a special case of FEM where discontinuities are allowed (see Figure 4.1). Therefore, DG has good qualities from both methods ([22],[26]):

- discontinuities are allowed,
- unstructured grids can be used,
- conservation of mass,

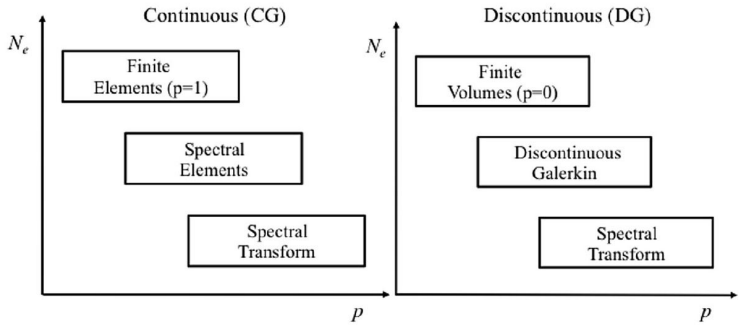


Figure 4.1: Relation between FEM, FVM and DG where N_e is a finite number of non-overlapping elements and p the order of basis functions. (Spectral elements is a finite element method with high order basis functions and spectral transform is also a finite element method with only one element and high order basis functions.) Image taken from [22].

- dynamic h-p refinements (where h is the grid size and p the polynomial order of the basis function),
- high scalability - only communication between elements which share faces.

The negative effect of DG is the non-physical oscillations in the results [22]. However, these can be solved by using a limiter. For these reasons, the discontinuous Galerkin method is chosen to be used in DALES.

4.2. Basics of discontinuous Galerkin

In this section, the basics of DG are explained by solving a simple one-dimensional advection equation. Some numerical results are shown with and without using the moment limiter.

The one-dimensional advection equation in flux form that is used to explain DG, is:

$$\begin{cases} \frac{\partial \varphi}{\partial t} + \frac{\partial f(\varphi)}{\partial x} = 0 & x \in [a,b], t > 0, \\ \varphi(x,0) = \varphi_0(x) & x \in [a,b], \\ \varphi(a,t) = \varphi(b,t) & t > 0, \end{cases} \tag{4.1}$$

where $\varphi(x,t)$ is the quantity of interest and $f(\varphi) = u\varphi$ the given flux function.

First, the domain $\Omega = [a,b]$ is partitioned into K non-overlapping elements $\Omega = \cup_{k=1}^{K} I_k$ with $I_k = [x_{k-1/2}, x_{k+1/2}]$, $k = 1, \dots, K$. Just as in the finite element method, the function $\varphi(x,t)$ is approximated locally on every element I_k:

$$\varphi(x,t) \cong \varphi_h(x,t) = \bigoplus_{k=1}^{K} \varphi_h^k(x,t), \text{ where}$$
$$\varphi_h^k(x,t) \in V_h^N(I_k) = \{v : v \in \mathbb{P}^N(I_k), k = 1, \dots, K\}.$$

Here $\mathbb{P}^N$ is the space of polynomials of degree N and $\oplus$ denotes that φ_h is the direct sum of local polynomial functions φ_h^k.

DG is resolved around the weak formulation of the equation which is obtained by multiplying the differential equation and initial condition with an arbitrary piecewise continuous function η and integrating over element I_k:

$$\begin{cases} \int_{I_k} \left[\frac{\partial \varphi}{\partial t} + \frac{\partial}{\partial x} f(\varphi) \right] \eta \, dx = 0, & \text{(4.2a)} \\ \int_{I_k} \varphi(x,0) \eta \, dx = \int_{I_k} \varphi_0(x) \eta \, dx. & \text{(4.2b)} \end{cases}$$

Using integration by parts, equation (4.2a) can be rewritten:

$$\int_{I_k} \frac{\partial \varphi}{\partial t}\eta - f(\varphi)\frac{\partial \eta}{\partial x}\, dx + \left[f(\varphi_h)\eta\right]_{x_{k-1/2}}^{x_{k+1/2}} = 0. \tag{4.3}$$

As the approximated function is allowed to have discontinuities, there are some ambiguities around the boundaries of the elements as can be seen in Figure 4.2. Therefore, the following notation is used to

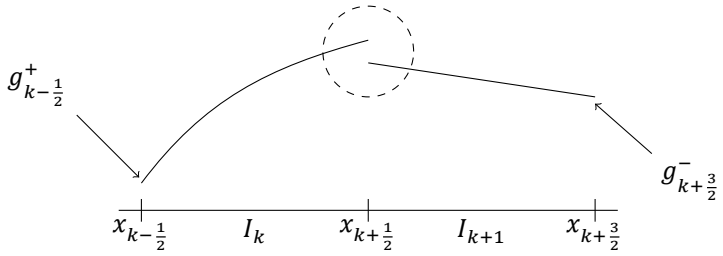


Figure 4.2: Example of a function g which is discontinuous at the element boundaries.

indicate which value is used:

$$\eta^+_{h,k-1/2} = \lim_{x \downarrow x_{k-1/2}} \eta_h(x),$$
$$\eta^-_{h,k+1/2} = \lim_{x \uparrow x_{k+1/2}} \eta_h(x).$$

With the given notation, the third term of (4.3) can be written as:

$$\left[f(\varphi_h)\eta_h(x)\right]_{x_{k-1/2}}^{x_{k+1/2}} = \hat{F}_{k+1/2}\eta^-_{k+1/2} - \hat{F}_{k-1/2}\eta^+_{h,k-1/2},$$

where $\hat{F}_{k\pm1/2}$ denotes the numerical flux value on the boundary $x_{k\pm1/2}$ which can depend on both $x^-_{k\pm1/2}$ and $x^+_{k\pm1/2}$.

There are many choices for the numerical fluxes [20], such as:

- upwind,
- Godunov,
- Lax-Friedrich.

The DG solution is not very sensitive to the choice of numerical fluxes for basis functions with polynomial degree $N \geq 3$. This means that a very simple numerical flux should suffice [22].

Hereafter, η is chosen to be a test function from V_h and φ^k_h is approximated by assigning weights to specified basis functions:

$$\varphi^k_h(x,t) = \sum_{j=0}^{N} \hat{a}^k_j(t)\psi_j(x) = \sum_{j=0}^{N} a^k(x^k_j,t)\ell^k_j(x),\ \forall x \in I_k. \tag{4.4}$$

The first expression $\varphi^k_h(x,t)$ is known as the *modal form* and the second the *nodal form*. Thus, $\hat{a}^k_j(t)$ are the modal coefficients and $a^k(x^k_j,t)$ the nodal coefficients. Basis functions $\psi_j(x)$ are the functions belonging to the modal form and $\ell^k_j(x)$ to the nodal form. With the use of the *Vandermonde matrix* which is defined as a $(k+1)\times(k+1)$ matrix V by $V_{ij} = \psi_j(x_i)$, one can switch between the two forms:

$$V\hat{\boldsymbol{a}}^k = \boldsymbol{a}^k. \tag{4.5}$$

In [5], both forms are applied to the one-dimensional advection equation. Indeed, the two forms gave exactly the same results, meaning that the solution is insensitive to the form. However, each representation has its own favourable properties. The modal form can be handy when going from order $N-1$

to N since only one extra basis function is added. This is not the case for the nodal form, because all basis functions are changed completely. Moreover, most limiters are used on the modal form of the solutions. However, for the nodal representation there is no need to transfer between the spectral and physical space, making it easier to plot and implement the boundary and initial conditions. Also, for the definition of element continuity the nodal form is handier. Therefore, the nodal representation is primarily used in this thesis project.

4.2.1. Nodal discontinuous Galerkin method

For nodal DG, the Lagrange polynomials are chosen as basis function $\ell_j^k(x)$. The Lagrange polynomials based on points x_0, x_1, ..., x_N are defined as:

$$\ell_j(x) = \prod_{i=0, i\neq j}^{N} \frac{x - x_i}{x_j - x_i} \tag{4.6}$$

In this thesis, the Lagrange polynomials are based on the Legendre-Gauss-Lobatto (LGL) points. These points are $N+1$ nodes in the interval $[-1, 1]$ that satisfy:

$$(1 - x^2)\frac{d}{dx}P_N(x) = 0, \tag{4.7}$$

where $P_n(x)$ is the nth Legendre polynomial. In Figure 4.3, the first five Legendre polynomials are shown.

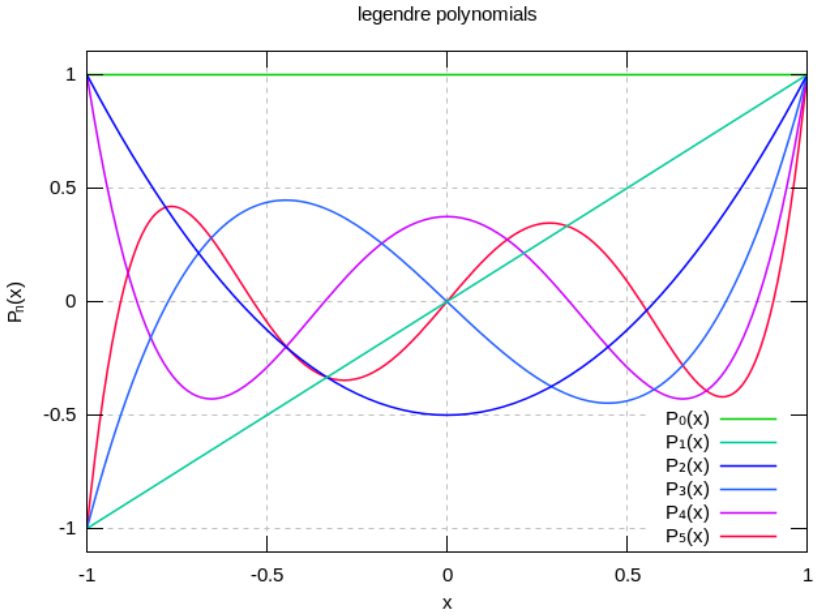
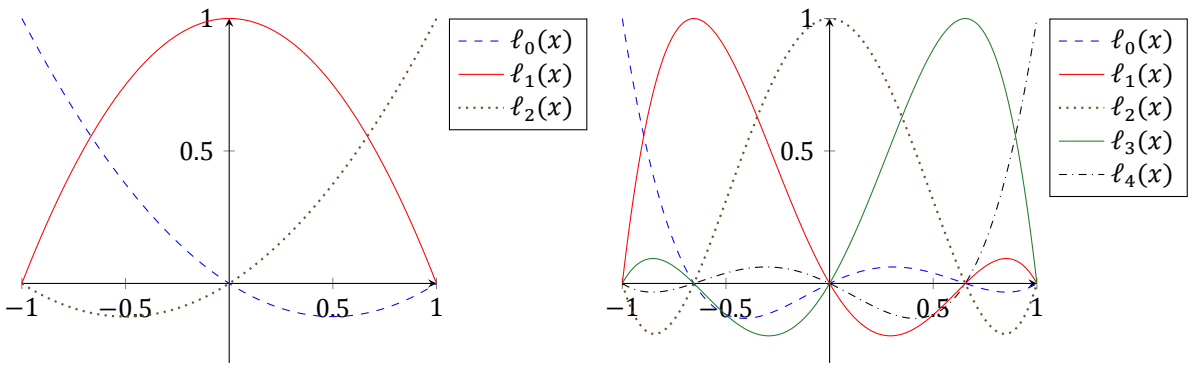


Figure 4.3: The first five Legendre polynomials $P_n(x)$. Image taken from [39].

One can also take $N+1$ equidistant nodes, however, the maxima and minima of the Lagrange polynomials can get out of control when N becomes greater. In [22], the problems of using equidistant nodes are explained more thoroughly. For this reason we chose to use LGL nodes.

All elements $[x_{k-1/2}, x_{k+1/2}]$ are mapped to a reference element with mapping $\xi(x)$. As a result, the basis functions $\ell_j^k(x)$ are the same for every element. In Figure 4.4, the Lagrangian basis function are shown for $N = 2$ and $N = 4$.



(a) $N = 2$. (b) $N = 4$.

Figure 4.4: Lagrangian polynomials $\ell_j(x)$ using $N + 1$ LGL nodes.

After the basis functions are chosen, we can fill in:

$$\forall x \in I_k : \begin{cases} \eta_h(x) = \ell_i(\xi(x)),\ i \in \{0, \dots, N\}, & \text{(4.8a)} \\ \varphi_h^k = \sum_{j=0}^{N} a_h^k(x_j^k, t)\ell_j(\xi(x)), & \text{(4.8b)} \end{cases}$$

into the weak formulation (4.2):

$$\begin{cases} \int_{I_k} \sum_{j=0}^{N} \frac{\partial}{\partial t} a^k(x_j^k, t)\ell_j(\xi(x))\ell_i(\xi(x)) - f(\varphi_h)\frac{\partial \ell_i(\xi(x))}{\partial x}\, dx + \left[f(\varphi_h)\ell_i(\xi(x))\right]_{x_{k-1/2}}^{x_{k+1/2}} = 0, & \text{(4.9a)} \\ \int_{I_k} \sum_{j=0}^{N} a^k(x_j^k, t)\ell_j(\xi(x))\ell_i(\xi(x))\, dx = \int_{I_k} \varphi_0(x)\ell_i(\xi(x))\, dx. & \text{(4.9b)} \end{cases}$$

This must hold for all $i \in \{0, \dots, N\}$.

Before the element matrices can be calculated, the flux function must be known. For this example, the following are chosen as flux function and as simple numerical flux:

$$f(\varphi) = u\varphi \text{ where } u \in \mathbb{R}_{>0}, \quad \text{(4.10a)}$$

$$\hat{F}_{k\pm1/2} = f(\varphi_{h,k\pm1/2}^-), \quad \text{(4.10b)}$$

which is known as the upwind flux.

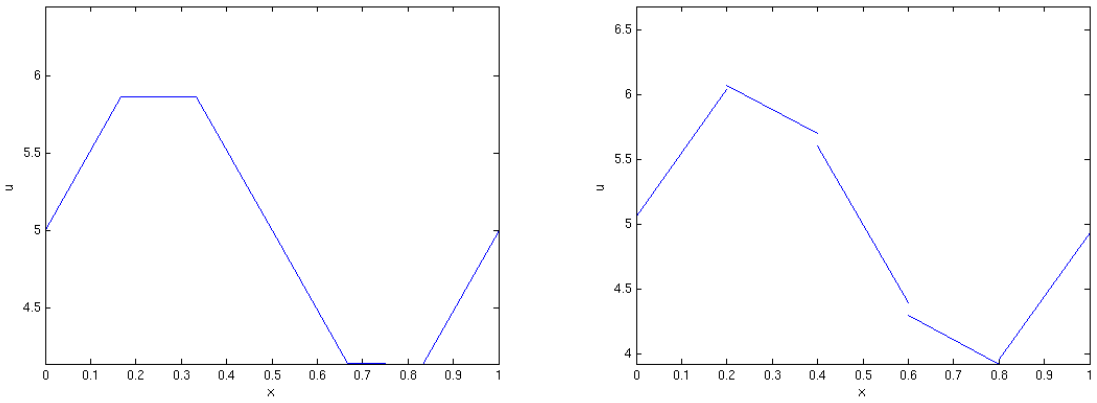
At this instant, all information is known to be able to write the following matrix-vector equation from (4.9a) and (4.9b):

$$\begin{cases} M^k \frac{\partial}{\partial t}\boldsymbol{a}^k - S^k\boldsymbol{a}^k + F_1^k\boldsymbol{a}^k - F_2^k\boldsymbol{a}^{k-1} = 0, & \text{(4.11a)} \\ M^k\boldsymbol{a}^k(0) = \left(\int_{I_k} \varphi_0\ell_i(\xi(x))\, dx\right)_i = \tilde{\boldsymbol{\varphi}}_h^k, & \text{(4.11b)} \end{cases}$$

where $\boldsymbol{a}^k = \begin{pmatrix} a^k(x_1^k, t) \\ a^k(x_2^k, t) \\ \vdots \\ a^k(x_N^k, t) \end{pmatrix}$.

It is interesting to note that $M^k\boldsymbol{a}^k(0) = \tilde{\boldsymbol{\varphi}}_h^k$ has to be solved to get the initial vector $\boldsymbol{a}^k(0)$ instead of just using the nodal values of $\varphi_0(x_i^k), i \in \{0, \dots, N\}$. By solving $M^k\boldsymbol{a}^k(0) = \tilde{\boldsymbol{\varphi}}_h^k$, the initial condition $\boldsymbol{a}^k(0)$

is a projection of the initial condition $\varphi_0(x)$ to the finite element space. The L_2-projection of the initial condition is the function $\varphi_h(x,0)$ which minimizes $\|\varphi_0(x) - \varphi_h(x,0)\|_{L_2}$. An L_2-projection can be done to a continuous and a discontinuous space. In Figure 4.5, the differences between the two spaces are shown. In this thesis, the L_2-projection is done to the discontinuous space.



(a) Continuous space. (b) Disontinuous space.

Figure 4.5: L_2-projection of $\varphi_0(x) = 5 + \sin(2\pi)$ to a finite element space. Image taken from [40].

To test DG, an initial condition is chosen that contains a smooth and a discontinuous part. For this example, we take domain $\Omega = [-5, 5]$ and initial condition (see Figure 4.6):

$$\varphi_0(x) = \begin{cases} \frac{1-\cos(\pi(x-1))}{2}, & x \in [-3,-1], \\ 1, & x \in [1,3], \\ 0, & \text{otherwise.} \end{cases} \tag{4.12}$$

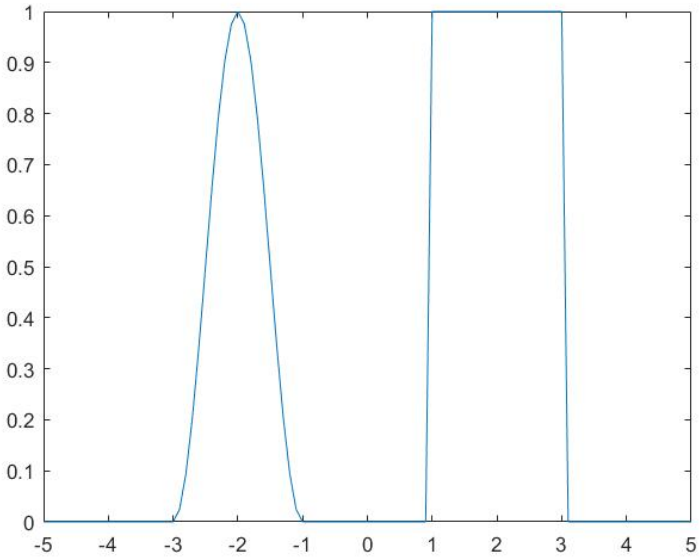


Figure 4.6: Initial condition φ_0.

For example, linear basis functions are used ($N = 1$). All elements are mapped to a reference element, allowing to generalize the element matrices as much as possible. The coordinate transformation

$\xi(x) = \frac{x - x_{k-1/2}}{\Delta x}$ is used in this example to take advantage of a linear coordinate transformation for the integration. In this case the Lagrange polynomials are defined on $[0, 1]$ by:

$$\ell_0(\xi) = 1 - \xi, \qquad \ell_1(\xi) = \xi. \tag{4.13}$$

Hence the element matrices of the matrix-vector form of the problem, see (4.11a), are defined as:

$$M_{ij}^k = \Delta x \int_0^1 \ell_i(\xi)\ell_j(\xi)\, d\xi \Rightarrow M^k = \Delta x \begin{pmatrix} \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} \end{pmatrix}, \tag{4.14a}$$

$$S_{ij}^k = u \int_0^1 \ell_j(\xi)\frac{d}{d\xi}\ell_i(\xi)\, d\xi \Rightarrow S^k = u \begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \tag{4.14b}$$

$$F_1^k = u \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \tag{4.14c}$$

$$F_2^k = u \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}. \tag{4.14d}$$

When for example Lax-Friedrichs flux is chosen as numerical flux, only the element flux matrices are different than the ones above. In Appendix A.1, the flux matrices are derived for that case.

To find the L_2-projection of the initial condition, the initial condition multiplied with the basis function is integrated exactly:

$$\tilde{\varphi}_h^k = \begin{pmatrix} \int_{I_k} \varphi_0 \ell_0(\xi(x))\, dx \\ \int_{I_k} \varphi_0 \ell_1(\xi(x))\, dx \end{pmatrix}, \tag{4.15a}$$

$$\int_{I_k} \varphi_0 \ell_0(\xi(x))\, dx = \begin{cases} \frac{\Delta x}{2}\left[\xi - \frac{1}{2}\xi^2 + \frac{1}{a^2}\cos(a\xi + b) + (\xi - 1)\frac{1}{a}\sin(a\xi + b)\right]_0^1, & I_k \subseteq [-3, -1], \\ \frac{\Delta x}{2}, & I_k \subseteq [1, 3], \\ 0, & \text{otherwise}, \end{cases} \tag{4.15b}$$

$$\int_{I_k} \varphi_0 \ell_1(\xi(x))\, dx = \begin{cases} \frac{\Delta x}{2}\left[\frac{1}{2}\xi^2 - \frac{1}{a^2}\left(a\xi\sin(a\xi + b) + \cos(a\xi + b)\right)\right]_0^1, & I_k \subseteq [-3, -1], \\ \frac{\Delta x}{2}, & I_k \subseteq [1, 3], \\ 0, & \text{otherwise}, \end{cases} \tag{4.15c}$$

where $a = \pi\Delta x$ and $b = \pi(x_{k-1/2} - 1)$. The element matrices and initial condition of modal DG with $N = 1$ can be found in [5].

One can also choose to calculate the integrals inexactly, for example, when N is quite large or the initial condition is given as a vector of values instead a function. Then one can choose to use quadrature rules. The quadrature rule that works well with the LGL nodes x_i is the Lobatto-Gauss-Legendre quadrature:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{N} \omega_i f(x_i) \text{ with } \omega_i = \frac{2}{N(N+1)(P_N(x_i))^2}. \tag{4.16}$$

This rule is exact for all polynomials of order $2N - 1$ or less [17].

The linear coordinate transformation $\xi(x) = \frac{2(x - x_k)}{\Delta x}$ is used in order to have reference element $[-1, 1]$

where also the LGL nodes lay. Then the following element matrices are obtained for (4.11a):

$$M_{ij}^k = \int_{I_k} \ell_i(\xi(x))\ell_j(\xi(x))\,dx = \frac{\Delta x}{2}\int_{-1}^{1} \ell_i(\xi)\ell_j(\xi)\,d\xi \approx \frac{\Delta x}{2}\sum_{p=0}^{N} \omega_p \ell_i(\xi_p)\ell_j(\xi_p) = \frac{\Delta x}{2}\omega_i\delta_{ij}, \quad (4.17a)$$

$$S_{ij}^k = u\int_{I_k} \ell_j(\xi(x))\frac{d\ell_i(\xi(x))}{dx}\,dx = u\int_{-1}^{1} \ell_j(\xi)\frac{d\ell_i(\xi)}{d\xi}\,d\xi \approx u\sum_{p=0}^{N} \omega_p \ell_j(\xi_p)\ell_i'(\xi_p) = u\omega_j\ell_i'(\xi_j), \quad (4.17b)$$

$$F_{1,ij} = f(\varphi_{k+1/2}^-)\ell_i^k(x_{k+1/2}^-) \Rightarrow F_1 = u\begin{pmatrix} 0 & \dots & \dots & 0 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & -1 \end{pmatrix}, \quad (4.17c)$$

$$F_{2,ij} = f(\varphi_{k-1/2}^-)\ell_i^k(x_{k-1/2}^-) \Rightarrow F_2 = u\begin{pmatrix} 0 & \dots & 0 & 1 \\ \vdots & \ddots & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & 0 \end{pmatrix}. \quad (4.17d)$$

The integral for the L_2-projection of the initial condition can also be calculated using the LGL quadrature:

$$\tilde{\varphi}_i^k = \int_{I_k} \varphi_0\ell_i(\xi(x))\,dx \approx \frac{\Delta x}{2}\sum_{p=0}^{N} \varphi_0(\xi_p)\omega_p\ell_i(\xi_p) = \frac{\Delta x}{2}\varphi_0(\xi_i)\omega_i. \quad (4.18)$$

With the element matrices and initial condition, one can solve the problem with DG. In the literature study of this thesis project [5], DG has been tested and indeed we have shown that the method is dispersive. Therefore, a limiter will be introduced.

4.2.2. Moment limiter for one-dimensional problems

A disadvantage of the DG method is the presence of non-physical oscillations in the results, therefore, a limiter is needed. However, with most limiters the solution is reduced to first order accuracy and the advantage of high order methods is lost.

The limiters that are made for DG are defined for the modal form of the DG, thus, the following steps have to be taken:

1. Transform from nodal to modal representation,
2. Apply limiter,
3. Transform back from modal to nodal representation.

These steps can be done on element basis [22], meaning that there is no global assembly operation needed, which can save computational time. On top of that, the transformations are done with the Vandermonde matrix using (4.5).

In this thesis project the moment limiter is used. The choice for the moment limiter is the simple implementation and the underlying idea. This allowed us to create a three-dimensional moment limiter, which will be discussed in Section 4.3.6.

The moment limiter was first developed by Biswas et al. and is a generalization to higher order of the second order minmod limiter of van Leer [3]. Krivodonova generalized the limiter and extended it to two-dimensional problems on tensor product meshes [18].

The moment limiter gradually reduces the order if limiting is needed. It limits the derivative of order j in a given cell using the derivatives of order $j-1$ in the neighbouring cells. The limiter starts by limiting, when needed, the highest orders first. Then the limiting process continues until it is not needed to limit

any more or until all terms are limited. With this strategy the solution has the highest order accuracy possible when limiting is needed. Furthermore, by limiting gradually the limiter avoids artificial limiter-induced steepening, which turns sine waves into square waves.

The moment limiter uses the minmod function which is defined as:

$$m(a_1, \dots, a_N) = \begin{cases} s \min_{1 \le j \le N} |a_j|, & \text{if } \operatorname{sgn}(a_1) = \dots = \operatorname{sgn}(a_N) = s, \\ 0, & \text{otherwise.} \end{cases} \tag{4.19}$$

The moment limiter also uses the modal form of the DG solution $\varphi_h^k(x,t) = \sum_{j=0}^{N} \hat{a}_j^k(t)\psi_j(x)$. The pseudoalgorithm of the Moment limiter can be found in Algorithm 1.

Algorithm 1 Moment Limiter.

for all elements I_k **do**
  Set $j = N$
  while $j = N$ **or** ($\tilde{a}_j^k \neq \hat{a}_j^k$ **and** $j > 1$) **do**
    $\alpha_j = \frac{\sqrt{j-1/2}}{\sqrt{j+1/2}}$
    $\tilde{a}_j^k = m\left(\hat{a}_j^k, \alpha_j(\hat{a}_{j-1}^{k+1} - \hat{a}_{j-1}^k), \alpha_j(\hat{a}_{j-1}^k - \hat{a}_{j-1}^{k-1})\right)$
    $j = j - 1$
  end while
end for

The idea behind this algorithm is that roughly speaking the $\tilde{a}_j^k$ corresponds to the jth derivative of the solution of element k. Thus, this coefficient is compared with the numerical derivative using forward and backward differences. More information can be found in Section 4.3.6 where, among other things, the derivation of the moment limiter is given.

4.2.3. CFL condition for Runge-Kutta discontinuous Galerkin

In Chapter 3, it is explained that DALES uses the third order Runge-Kutta method to integrate in time and the CFL condition to adaptively choose the size of the time step. For the Runge-Kutta discontinuous Galerkin (RKDG) the CFL condition is not restricted by the common 1-limit, but the condition has to ensure L^2-stability:

$$\|u\frac{\Delta t}{\Delta x}\| \le \mathrm{CFL}_{L^2}. \tag{4.20}$$

In [7], Cockburn and Shu give the CFL_{L^2} conditions for different order Runge-Kutta integration methods and polynomial orders. These CFL_{L^2} numbers are given in Table 4.1.

N	0	1	2	3	4	5	6	7	8
$v = 1$	1.000	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
$v = 2$	1.000	0.333	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$
$v = 3$	1.256	0.409	0.209	0.130	0.089	0.066	0.051	0.040	0.033
$v = 4$	1.392	0.464	0.235	0.145	0.100	0.073	0.056	0.045	0.037
$v = 5$	1.608	0.534	0.271	0.167	0.115	0.085	0.065	0.052	0.042

Table 4.1: CFL_{L^2} for RKDG order v and polynomial order N. The $\star$ denotes that the method is unstable when the ratio $\frac{\Delta t}{\Delta x}$ is held constant. Table taken from [7].

4.2.4. Error calculation for discontinuous Galerkin

The numerical error is estimated by measuring the difference between the exact solution $\varphi(x,t) = \varphi_0(x - ut)$ and its approximation $\varphi_h(x,t)$. In the error calculation, instead of the exact solution, the L_2-projected initial condition is compared to the numerical solution after c periods which should exactly be the L_2-projected initial condition. This is chosen, since the L_2-projection is advected, which is the finite element version of the initial condition and not the initial condition, moreover, comparing the L_2-projection simplifies the error calculation. For the initial condition given by (4.12) with computational

domain $\Omega = [-5, 5]$ and velocity $u = 1$, one period is 10 seconds.

A very simple way to obtain the error is by using vector norms. The vector norms that are used in [5] are the ℓ_1-norm, ℓ_2-norm (Euclidean norm) and the ℓ_∞-norm (infinity norm):

$$\left\|\boldsymbol{a}_h(0) - \boldsymbol{a}_h(10c)\right\|_1 = \sum_i \left|a_h(0)_i - a_h(10c)_i\right|, \tag{4.21a}$$

$$\left\|\boldsymbol{a}_h(0) - \boldsymbol{a}_h(10c)\right\|_2 = \sqrt{\sum_i \left(a_h(0)_i - a_h(10c)_i\right)^2}, \tag{4.21b}$$

$$\left\|\boldsymbol{a}_h(0) - \boldsymbol{a}_h(10c)\right\|_\infty = \max_i \left|a_h(0)_i - a_h(10c)_i\right|, \tag{4.21c}$$

where $c \in \mathbb{N}$ is the number of periods.

Since two functions are compared, a function norm should be used. For this reason the difference between the two functions is also calculated with the L_2-norm:

$$\|\varphi - \varphi_h\|_2 = \sqrt{\int_\Omega |\varphi - \varphi_h|^2 \, d\Omega} = \sqrt{\boldsymbol{\alpha}^T M \boldsymbol{\alpha}}, \tag{4.22}$$

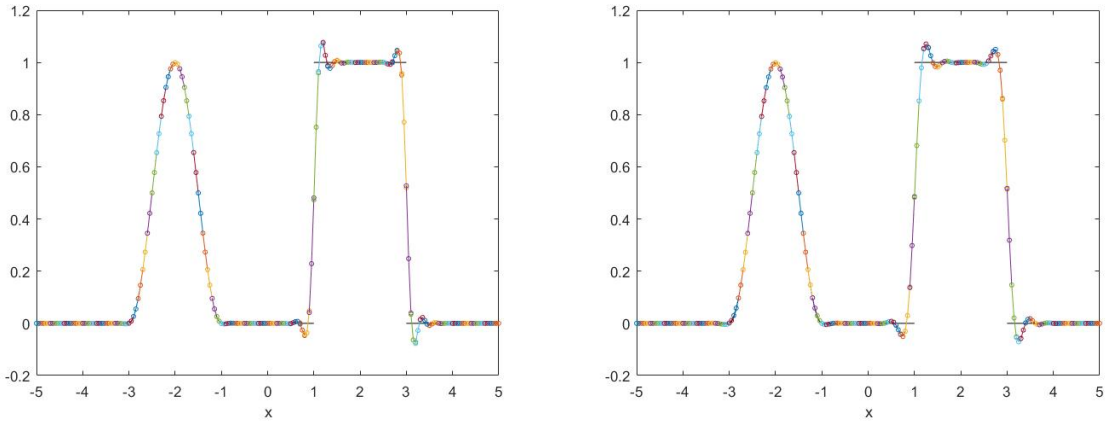
where $\boldsymbol{\alpha}$ is defined as $\alpha_j^k = a_h^k(x_j^k, 0) - a_h^k(x_j^k, 10c)$. The derivation of the L_2-norm can be found in [5].

4.2.5. Numerical results for the one-dimensional problem

In this section, a summary is given of the numerical results for the one-dimensional tests with and without moment limiter of the literature study [5]. First, the numerical results are given for DG without limiter. Moreover, the computation time and convergence of the methods are given. Thereafter, a summary of the results with the moment limiter are shown.

DG without limiter

For all three different polynomial orders $N = 1$, $N = 2$ (see Figure 4.7) and $N = 4$, DG seems to work well, obviously for higher polynomial degree the accuracy is higher. DG is especially good in smooth regions and is also conservative. Moreover, there is no time lag as the tested finite difference methods in [5]. Nevertheless, DG is as diffusive as the fifth order upwind and the WENO method. The only disadvantage of DG is the dispersion error around the discontinuity.



(a) DG with $N = 2$ at $t = 10$.

(b) DG with $N = 2$ at $t = 50$.

Figure 4.7: DG with $N = 2$ using exact integration, $\Delta x = 0.1$ and $\Delta t = 0.95\text{CFL}_{L^2} \frac{\Delta x}{|u|}$.

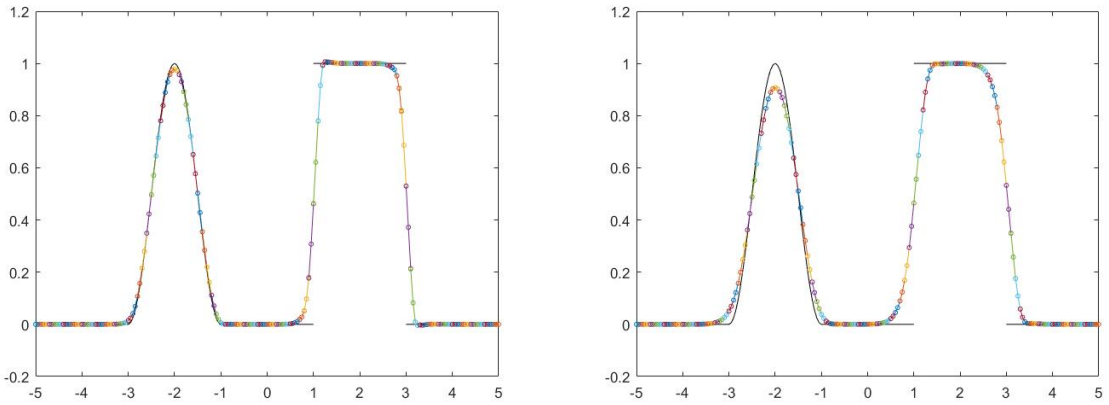
In [5], the computational time of the three different DGs and the tested advection schemes are compared. We have seen that the low order schemes, the first order upwind method and the second order

central difference were the fastest, and the fifth order upwind, the WENO method and DG with $N = 1$ were neither the fastest nor the slowest. DG with $N = 2$ and DG with $N = 4$ were the slowest. In comparison with the WENO method, DG with $N = 2$ was 2.5 times slower and for $N = 4$ even 10 times slower.

The convergence of DG for Cartesian grids is $\mathcal{O}(\Delta x^{N+1})$, which has been proven by LeSaint and Raviart [19]. In [5], this is tested by coupled and decoupled tests, and by smooth and less smooth functions. The coupled tests are done since the advection equation depends on both time and space. However, the time integration method is theoretical of order 3, thus DG with $N = 4$ is tested without the coupling to time. For both coupled and decoupled tests, we have seen that the DG method with basis functions of polynomial order N can indeed superconverge with order $N + 1$. However, when less smooth functions or even discontinuous functions are advected, the superconvergence is not obtained.

DG with moment limiter

In [5], the moment limiter is tested for DG with $N = 1$, $N = 2$ (see Figure 4.8) and $N = 4$. Hence for $N = 1$, only one coefficient can be limited, meaning that when limiting is needed, the second order derivatives are set to zero. For $N = 2$, two coefficients can be limited and for $N = 4$ four coefficients. The moment limiter adds extra diffusion which results in the removal of the dispersion. For longer time simulations, the diffusion has a significant impact on the results. On top of that, at $t = 10$ the remainder of the dispersion is still shown except for $N = 4$. In the tests, we have seen that for $N \geq 2$ the limited results are the same after a few periods. This is probably because the limiter categorizes this function as a smooth function.



(a) at $t = 10$ (b) at $t = 50$

Figure 4.8: Moment limited DG using exact integration with $N = 2$,$\Delta x = 0.1$ and $\Delta t = 0.95\text{CFL}_2 \frac{u}{\Delta x}$.

Clearly, the moment limiter slows down the routine. For $N = 2$ and $N = 4$ the computational time is doubled, but for $N = 1$ it becomes even six times slower. Compared to the WENO method, the moment limited DG with $N = 2$ is six times slower and with $N = 4$ 18 times. However, we suspect that with parallelization the simulations of DG can be computed much faster.

In general, a limiter reduces the order of the method. This is also the case for the moment limiter. For all tests, even the tests with smooth functions the convergence of order $N+1$ is not obtained. However, compared to the WENO method, the total numerical error of the limited DG with $N = 4$ is smaller than of the WENO method.

4.3. Discontinuous Galerkin for DALES

In this section, the DG method is applied on the advection equation of DALES. First, the equation is solved in two-dimensions and later extended to the three-dimensional model. Moreover, the derivation of the moment limiter is shown in one- and three-dimensions.

4.3.1. Advection equation of DALES

The exact equation from DALES that is to be solved is given in Chapter 2 by Equation (2.14). However, as it is told in that chapter, only the advection part of the equation has to be solved by the DG routine. Therefore, the advection equation for the model is given by:

$$\frac{\partial \varphi(\boldsymbol{x},t)}{\partial t} + \frac{1}{\rho(z)}\left(\frac{\partial}{\partial x}(\rho(z)u(\boldsymbol{x},t)\varphi(\boldsymbol{x},t)) + \frac{\partial}{\partial y}(\rho(z)v(\boldsymbol{x},t)\varphi(\boldsymbol{x},t)) + \frac{\partial}{\partial z}(\rho(z)w(\boldsymbol{x},t)\varphi(\boldsymbol{x},t))\right) = 0, \tag{4.23}$$

which can be written in vector form:

$$\frac{\partial \varphi}{\partial t} + \frac{1}{\rho(z)}\nabla \cdot \boldsymbol{f}(\varphi) = 0, \quad \text{with } \boldsymbol{f}(\varphi) = \rho(z)\begin{pmatrix} u(\boldsymbol{x},t) \\ v(\boldsymbol{x},t) \\ w(\boldsymbol{x},t) \end{pmatrix}\varphi. \tag{4.24}$$

Again $\varphi(\boldsymbol{x},t)$ is the quantity of interest, such as the total water specific humidity q_t or the liquid water potential temperature θ_l. Variables u, v and w are still the wind speeds in the x-, y- and z-direction respectively and $\rho(z)$ denotes the density function in height z.

First, the partial differential equation will be solved in the (x,z)-domain. This is done to understand the counting of the elements and their grid points and the definition of the element matrices. Moreover, the numerical results of the two-dimensional problems are still understandable. On top of that, the extension to three-dimensions can be done quite easily, since the density ρ depends on the height z and not on x and y. This will have influence on the derivation of the element matrices. The terms depending on the y-direction are derived equivalent to the terms depending on the x-direction. As a result, the element matrices can be easily extended to three-dimensions.

4.3.2. Two-dimensional problem

First, one needs to know what the computational domain is of DALES and how it is partitioned. Recall that the domain in DALES is split into cuboids, to be precise the Arakawa C-grid (see Section 3.1). For this reason, the two-dimensional computational domain is partitioned into quadrilateral elements. In the x- and z-direction the domain is partitioned into K_x elements and K_z elements respectively. To number the K elements, where $K = K_z K_x$, x-lexicographic counting [32] is used. Figure 4.9 shows a non-equidistant grid with its x-lexicographic counting and in Table 4.2, the counting is indicated more clearly. Every element Ω_k is defined by the tensor product $[x_{k-1/2}, x_{k+1/2}] \times [z_{k-1/2}, z_{k+1/2}]$ and its width and length are denoted by Δx_k and Δz_k respectively (see Figure 4.9).

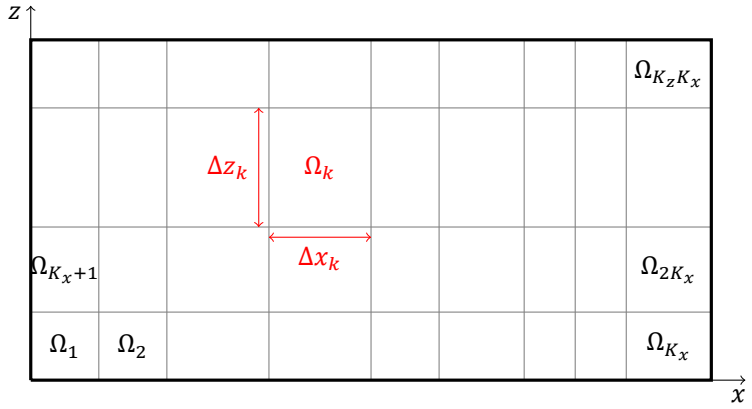


Figure 4.9: Domain partitioned into K elements using x-lexicographic counting.

$(K_z-1)K_x+1$	... ...	$K_z K_x$
$(K_z-2)K_x+1$		$(K_z-1)K_x$
⋮		⋮
K_x+1		$2K_x$
1	... ...	K_x

Table 4.2: x-lexicographic counting of the domain.

The function $\varphi(\boldsymbol{x},t)$ is approximated locally on each element Ω_k as follows:

$$\varphi(\boldsymbol{x},t) \cong \varphi_h(\boldsymbol{x},t) = \bigoplus_{k=1}^{K} \varphi_h^k(\boldsymbol{x},t), \text{ where}$$

$$\varphi_h^k(\boldsymbol{x},t) \in V_h(\Omega_k) = \{v : v \in \mathbb{P}^{N_x}([x_{k-1/2}, x_{k+1/2}]) \times \mathbb{P}^{N_z}([z_{k-1/2}, z_{k+1/2}]), k = 1, \dots, K_z K_x\}.$$

Here $\mathbb{P}^N$ is the space of polynomials of degree N. The polynomial orders in the x-direction and z-direction are denoted by N_x and N_z.

The weak formulation is derived by multiplying the partial differential equation (4.24) with an arbitrary test function η and integrating over element Ω_k:

$$\int_{\Omega_k} \left(\frac{\partial \varphi}{\partial t} + \frac{1}{\rho(z)} \nabla \cdot \boldsymbol{f}(\varphi) \right) \eta \, d\Omega = 0, \tag{4.25}$$

Integrating by parts and using Gauss's theorem, the equation can be rewritten:

$$\int_{\Omega_k} \frac{\partial \varphi}{\partial t} \eta \, d\Omega - \int_{\Omega_k} \boldsymbol{f}(\varphi) \cdot \nabla \frac{\eta}{\rho(z)} \, d\Omega + \int_{\partial\Omega_k} \left[\boldsymbol{f}(\varphi) \frac{\eta}{\rho(z)} \right] \cdot \boldsymbol{n} \, d\Gamma = 0. \tag{4.26}$$

The quantity of interest, φ, is approximated by a weighted sum of basis functions ϕ_j:

$$\varphi^k(\boldsymbol{x},t) = \sum_{j=1}^{N_p} a_h^k(\boldsymbol{x}_j^k, t) \phi_j(\boldsymbol{\xi}(\boldsymbol{x})). \tag{4.27}$$

where $N_p = (N_x + 1)(N_z + 1)$.

Like in the one-dimensional case, Lagrangian basis functions based on the LGL nodes are chosen (see Figure 4.4). These functions are extended to two-dimensions by using a tensor product of the one-dimensional Lagrangians:

$$\phi_j(\xi_x, \xi_z) = \ell_{x,j}(\xi_x) \ell_{z,j}(\xi_z), \text{ with } j \in \{1, \dots, N_p\}. \tag{4.28}$$

In Figure 4.10, some examples of two-dimensional basisfunctions $\phi_j(\xi_x, \xi_z)$ are shown. Hence $\phi_j(\boldsymbol{\xi}(\boldsymbol{x}_i)) = \delta_{ij}$, thus it is always zero in an LGL-node $\boldsymbol{\xi}(\boldsymbol{x_i})$ except if $i = j$.

For the implementation and clarification, the grid points and their corresponding basis functions are counted in an x-lexicographic way (See Table 4.3).

$N_z(N_x+1)+1$	...	...	$(N_x+1)(N_z+1)$
⋮			⋮
$(N_x+1)+1$			$2(N_x+1)$
1	...	...	N_x+1

z ↑, x →

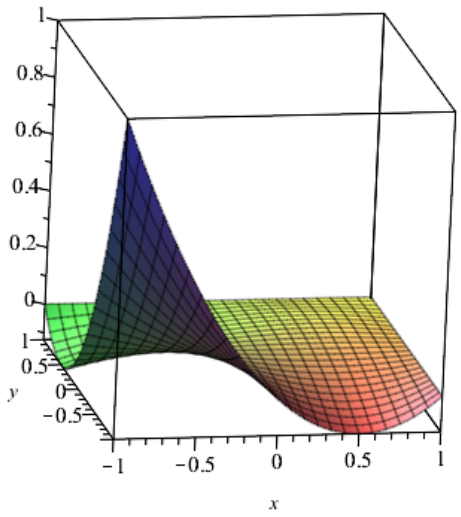
Table 4.3: x-lexicographic counting of the grid points of element Ω_k.

The element $\Omega_k = [x_{k-1/2}, x_{k+1/2}] \times [z_{k-1/2}, z_{k+1/2}]$ is mapped to a reference element $[-1,1] \times [-1,1]$ with a linear transformation:

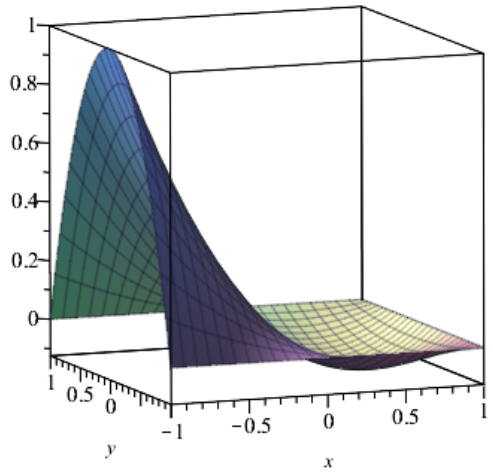
$$\boldsymbol{\xi}(\boldsymbol{x}) = (\xi_x, \xi_z)(\boldsymbol{x}) = \left(\frac{2(x - x_k)}{\Delta x_k}, \frac{2(z - z_k)}{\Delta z_k} \right). \tag{4.29}$$

For the change of variables, the corresponding Jacobian $|J|$ is needed. The Jacobian that belongs to this transformation is given by:

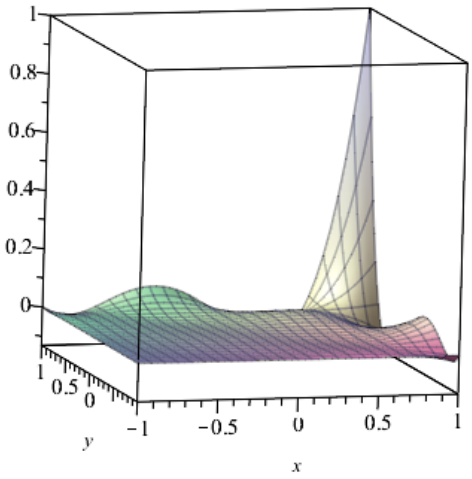
$$|J(\xi_x, \xi_z)| = \begin{vmatrix} \frac{\partial x}{\partial \xi_x} & \frac{\partial x}{\partial \xi_z} \\ \frac{\partial z}{\partial \xi_x} & \frac{\partial z}{\partial \xi_z} \end{vmatrix} = \begin{vmatrix} \frac{1}{2}\Delta x_k & 0 \\ 0 & \frac{1}{2}\Delta z_k \end{vmatrix} = \frac{1}{4} \Delta x_k \Delta z_k. \tag{4.30}$$



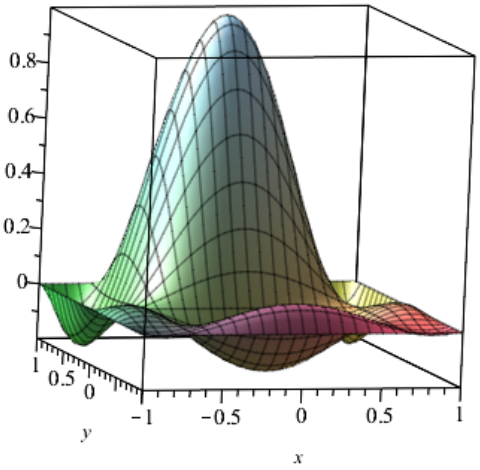
(a) Basis function $\phi_1(\xi_x, \xi_z)$ for $N = 2$.



(b) Basis function $\phi_4(\xi_x, \xi_z)$ for $N = 2$.



(c) Basis function $\phi_{25}(\xi_x, \xi_z)$ for $N = 4$.



(d) Basis function $\phi_{18}(\xi_x, \xi_z)$ for $N = 4$.

Figure 4.10: Some basis functions for different N.

After the basis functions are chosen, we can substitute

$$\forall \boldsymbol{x} \in \Omega_k : \begin{cases} \eta(\boldsymbol{\xi}(\boldsymbol{x})) = \phi_i(\boldsymbol{\xi}(\boldsymbol{x})), & i \in \{1, \dots, N_p\}, \quad (4.31a) \\ \varphi^k = \sum_{j=1}^{N_p} a_h^k(x_j^k, t)\phi_j(\boldsymbol{\xi}(\boldsymbol{x})), & \quad (4.31b) \end{cases}$$

into the weak formulation (4.26).

The first term of (4.26) can be rewritten to:

$$\int_{\Omega_k} \frac{\partial \varphi}{\partial t}\eta \, d\Omega = \int_{\Omega_k} \frac{\partial}{\partial t}\left[\sum_{j=1}^{N_p} a^k(\boldsymbol{x}_j^k, t)\phi_j\right]\phi_i \, d\Omega = \sum_{j=1}^{N_p} \frac{\partial}{\partial t}\left(a^k(\boldsymbol{x}_j^k, t)\right)\int_{\Omega_k} \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))\phi_j(\boldsymbol{\xi}(\boldsymbol{x})) \, d\Omega, \quad (4.32)$$

and the second term to:

$$\int_{\Omega_k} \boldsymbol{f}(\varphi) \cdot \nabla \frac{\eta}{\rho(z)} \, d\Omega = \int_{\Omega_k} \boldsymbol{f}(\varphi) \cdot \nabla \frac{\phi_i}{\rho(z)} \, d\Omega, \quad (4.33)$$

$$= \int_{\Omega_k} \rho(z) \begin{pmatrix} u(\boldsymbol{x}, t) \\ w(\boldsymbol{x}, t) \end{pmatrix} \varphi \cdot \begin{pmatrix} \frac{\partial}{\partial x}\frac{\phi_i(\boldsymbol{\xi}(x,z))}{\rho(z)} \\ \frac{\partial}{\partial z}\frac{\phi_i(\boldsymbol{\xi}(x,z))}{\rho(z)} \end{pmatrix} d\Omega, \quad (4.34)$$

$$= \int_{\Omega_k} \varphi\rho(z) \begin{pmatrix} u(\boldsymbol{x}, t) \\ w(\boldsymbol{x}, t) \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{\rho(z)}\frac{\partial \phi_i(\boldsymbol{\xi}(x,z))}{\partial x} \\ \frac{1}{(\rho(z))^2}\left(\rho(z)\frac{\partial \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))}{\partial \xi_z}\frac{\partial \xi_z}{\partial z} - \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))\frac{\partial \rho(z)}{\partial z}\right) \end{pmatrix} d\Omega, \quad (4.35)$$

$$= \sum_{j=0}^{N_p} a^k(\boldsymbol{x}_j^k, t)\int_{\Omega_k} \phi_j(\boldsymbol{\xi}(\boldsymbol{x}))\left(u(\boldsymbol{x}, t)\frac{\partial \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))}{\partial \xi_x}\frac{2}{\Delta x_k}\right.$$
$$\left. + \frac{w(\boldsymbol{x}, t)}{\rho(z)}\left(\rho(z)\frac{\partial \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))}{\partial \xi_z}\frac{2}{\Delta z_k} - \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))\frac{\partial \rho(z)}{\partial z}\right)\right) d\Omega. \quad (4.36)$$

This must hold for every test function ϕ_i with $i \in \{1, \dots, N_p\}$, thus, it can be cast into a matrix-vector form:

$$M^k \frac{\partial \boldsymbol{a}^k}{\partial t} = S^k \boldsymbol{a}^k - \boldsymbol{f}^k, \quad \text{with } \boldsymbol{a}^k = \begin{pmatrix} a^k(x_1^k, t) \\ a^k(x_2^k, t) \\ \vdots \\ a^k(x_{N_p}^k, t) \end{pmatrix}. \quad (4.37)$$

The third term of the weak formulation (4.26) results in the vector $\boldsymbol{f}^k$, which describes the fluxes on the boundary of the element. This vector is explained after the definition of the element mass matrix M^k and the element stiffness matrix S^k.

For the integration of the element matrices, one can choose to integrate inexact. In this case, the Lobatto-Gauss-Legendre quadrature rule is used for two-dimensional problems:

$$\int_{-1}^{1}\int_{-1}^{1} g(\xi_x, \xi_z) \, d\xi_x d\xi_z = \int_{-1}^{1} \sum_{i=0}^{N_x} \omega_{x,i} g(\xi_{x,i}, \xi_z) \, d\xi_z = \sum_{j=0}^{N_z}\sum_{i=0}^{N_x} \omega_{z,j}\omega_{x,i} g(\xi_{x,i}, \xi_{z,j}), \quad (4.38)$$

where the weights are given by

$$\omega_{x,i} = \frac{2}{N_x(N_x+1)(P_{N_x}(\xi_{x,i}))^2}, \qquad \omega_{z,j} = \frac{2}{N_z(N_z+1)(P_{N_z}(\xi_{z,j}))^2}.$$

The theorem of Cools (2002) says that this quadrature rule is exact for polynomials $g(\boldsymbol{x})$ with degree less than $\max(2N_x + 1, 2N_z + 1)$ [12]. More information on two-dimensional quadrature rules can be found in [8].

Element mass matrix
From Equations (4.32) and (4.37), a change of variables and the quadrature rule, the element mass matrix is written as:

$$M_{ij}^k = \int_{\Omega_k} \phi_j(\boldsymbol{\xi}(\boldsymbol{x}))\phi_i(\boldsymbol{\xi}(\boldsymbol{x}))\, d\Omega = \int_{-1}^{1}\int_{-1}^{1} \phi_j(\boldsymbol{\xi})\phi_i(\boldsymbol{\xi})\frac{1}{4}\Delta x_k \Delta z_k\, d\xi_x\, d\xi_z \tag{4.39}$$

$$\approx \sum_{p=0}^{N_x}\sum_{q=0}^{N_z} \frac{1}{4}\Delta x_k \Delta z_k \omega_{x,p}\omega_{z,q}\phi_j(\xi_{x,p},\xi_{z,q})\phi_i(\xi_{x,p},\xi_{z,q}) = \frac{1}{4}\Delta x_k \Delta z_k \omega_{x,i}\omega_{z,i}\delta_{ij}. \tag{4.40}$$

This means that the element mass matrix is a diagonal mass matrix when inexact integration is used. Consequently, the inverse mass matrix that is needed for the time integration, is easily computed. When exact integration would be used, the inverse mass matrix has to be found by solving linear equations. Moreover, when inexact integration is used, it is computational easy to change the polynomial degrees of N_x and N_z.

Element stiffness matrix
The element stiffness matrix is derived from (4.36) and (4.37), and is given by:

$$S_{ij}^k \approx \frac{1}{4}\Delta x_k \Delta z_k \omega_{x,j_x}\omega_{z,j_z}\Bigg(\frac{2}{\Delta x_k}u(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)\frac{\partial \ell_{x,i}(\xi_{j_x})}{\partial \xi_x}\delta_{i_z,j_z} \tag{4.41}$$

$$+ \frac{w(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)}{\rho(z(\xi_{z,j_z}))}\left[\frac{2}{\Delta z_k}\rho(z(\xi_{j_z}))\frac{\partial \ell_{z,i}(\xi_{j_z})}{\partial \xi_z}\delta_{xi,j_x} - \delta_{i,j}\frac{\partial \rho(z(\xi_{j_z}))}{\partial z}\right]\Bigg). \tag{4.42}$$

Note that the S_{ij}^k has only a contribution when the x-coordinates of gridpoints i and j, and/or the z-coordinates are the same. The derivation of the element stiffness matrix can be found in Appendix A.2.

Element fluxes
For $\boldsymbol{f}^k$, the third term of the weak formulation (4.26), some extra steps have to be taken. First of all, the integral over the element's boundary is split up into a sum of edges e:

$$\int_{\partial\Omega_k}\left[\boldsymbol{f}(\varphi_h)\frac{\eta}{\rho(z)}\right]\cdot\boldsymbol{n}\, d\Gamma = \sum_{e\in\Omega_k}\int_e\left[\boldsymbol{f}(\varphi_h)\frac{\eta}{\rho(z)}\right]\cdot\boldsymbol{n}_e\, d\Gamma = \sum_{e\in\Omega_k}\int_e \frac{\eta}{\rho(z)}h_{e,k}\, d\Gamma. \tag{4.43}$$

On the element boundaries e, the function φ is discontinuous, making $\boldsymbol{f}(\varphi)$ ambiguous. Therefore, $\boldsymbol{f}(\varphi)\cdot\boldsymbol{n}_e$ must be replaced by a numerical flux $h_{e,k}$ where $\boldsymbol{n}_e$ is the outward pointing vector normal vector of the element boundary e. The ambiguity that was solved by using φ^+ and φ^- in the one-dimension case, is in higher dimensions solved by naming them φ^{int} and φ^{ext} which are defined by the values in the interior and exterior of the element respectively [6].

As mentioned previously in Section 4.2, DG is not very sensitive to the choice of numerical fluxes for basis functions of polynomials of degrees $N \geq 3$ [22]. For this application the Lax-Friedrichs flux is chosen. The Lax-Friedrichs flux has more numerical diffusion than upwind; $\frac{1}{2}\left(\frac{\Delta x^2}{\Delta t}\right)$ [24] instead of $\left(\frac{\Delta x}{2} - \frac{\Delta t}{2}\right)$ [31]. We chose it with the idea that less dispersion will occur with Lax-Friedrichs.

In general, the local Lax-Friedrichs flux is defined by:

$$h_{e,k}(\varphi^{\text{int}},\varphi^{\text{ext}}) = \frac{1}{2}\left(\boldsymbol{f}(\varphi^{\text{int}})\cdot\boldsymbol{n}_{e,k} + \boldsymbol{f}(\varphi^{\text{ext}})\cdot\boldsymbol{n}_{e,k} - \alpha_{e,k}\left[\varphi^{\text{ext}} - \varphi^{\text{int}}\right]\right), \tag{4.44}$$

where $\alpha_{e,k}$ is an estimate of the biggest eigenvalue in absolute value of the Jacobian in the outward pointing normal direction $\frac{\partial}{\partial\varphi}\left(\boldsymbol{f}(\varphi(\boldsymbol{x},t))\cdot\boldsymbol{n}_{e,k}\right)$ for $\boldsymbol{x}$ on the elements boundary e at time t [6]. In general, the Lax-Friedrichs flux sets $\alpha_{e,k} = \frac{\Delta x}{\Delta t}$ while for the local Lax-Friedrichs flux the α is not a constant.

Even though, the value of $\alpha_{e,k}$ seems hard to calculate, in this case, it can be generalized. As we have taken a Cartesian grid, the normal vectors are the plus and minus standard unit vectors. Since $\frac{\partial}{\partial\varphi}\boldsymbol{f}(\varphi(\boldsymbol{x},t)) = \rho(z)\boldsymbol{u}(\boldsymbol{x},t)$, $\frac{\partial}{\partial\varphi}\left(\boldsymbol{f}(\varphi(\boldsymbol{x},t))\cdot\boldsymbol{n}_{e,k}\right)$ is not a matrix. For this reason, α is taken as the largest eigenvalue of $\frac{\partial}{\partial\varphi}\left(\boldsymbol{f}(\varphi(\boldsymbol{x},t))\otimes\boldsymbol{n}_{e,K}\right)$ [1] in absolute value:

$$\frac{\partial}{\partial\varphi}\left(\boldsymbol{f}(\varphi(\boldsymbol{x},t))\otimes\boldsymbol{n}_{e,K}\right) = \frac{\partial}{\partial\varphi}\left(\rho(z)\boldsymbol{u}(\boldsymbol{x},t)\varphi\otimes\boldsymbol{n}_{e,K}\right) = \begin{cases}\rho(z)\begin{pmatrix}\pm u(\boldsymbol{x},t) & 0\\ \pm w(\boldsymbol{x},t) & 0\end{pmatrix} & \text{eigenvalues: } \pm\rho(z)u(\boldsymbol{x},t), 0,\\ \rho(z)\begin{pmatrix}0 & \pm u(\boldsymbol{x},t)\\ 0 & \pm w(\boldsymbol{x},t)\end{pmatrix} & \text{eigenvalues: } \pm\rho(z)w(\boldsymbol{x},t), 0.\end{cases}$$

As a result, $\alpha_{e,k}$ is given by:

$$\alpha_{e,k} = \max_{(\boldsymbol{x},t)\in e}\left(\left|\rho(z)\boldsymbol{u}(\boldsymbol{x},\boldsymbol{t})\cdot\boldsymbol{n}_{e,k}\right|\right). \tag{4.45}$$

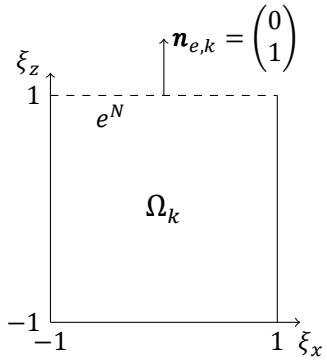


Figure 4.11: Element boundary e^N of the element Ω_k shown with its outward pointing normal vector $\boldsymbol{n}_e$ on its reference element.

Every element has 4 element boundaries. For example, the integral over the Northern boundary of the element (see Figure 4.11) on which the grid points $N_z(N_x+1)+1$,$N_z(N_x+1)+2$, ..., $(N_z+1)(N_x+1)$ lay, can be written as:

$$\int_{e^N}\frac{\eta}{\rho(z)}h_{e,k}\,d\Gamma = \int_{x_{k-1/2}}^{x_{k+1/2}}\frac{\phi_i}{\rho(z_{k+1/2})}h_{e,k}\,dx, \tag{4.46}$$

$$= \int_{-1}^{1}\frac{\phi_i(\xi_x,1)}{\rho(z_{k+1/2})}h_{e,k}(\varphi^{\text{int}},\varphi^{\text{ext}})\frac{\Delta x_k}{2}\,d\xi_x, \tag{4.47}$$

$$\approx \sum_{p=N_z(N_x+1)+1}^{N_p}\frac{\Delta x_k}{2}\omega_{x,p}\frac{\phi_i(\boldsymbol{\xi}_p)}{\rho(z_{k+1/2})}h_{e,k}(\varphi(\boldsymbol{\xi}_p^{\text{int}}),\varphi(\boldsymbol{\xi}_p^{\text{ext}})). \tag{4.48}$$

Since $\phi_i(\boldsymbol{\xi}_p) = \delta_{ip}$ is non-zero if $i = p$, and the normal vector corresponding to the Northern boundary is $\begin{pmatrix}0\\1\end{pmatrix}$, it can be rewritten as:

$$\int_{e^N}\frac{\phi_i}{\rho(z_{k+1/2})}h_{e,k}\,d\Gamma \approx \frac{\Delta x_k}{4}\frac{\omega_{x,i}}{\rho(z_{k+1/2})}\big(\rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{int}},t)\varphi(\boldsymbol{\xi}_i^{\text{int}}) + \rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{ext}},t)\varphi(\boldsymbol{\xi}_i^{\text{ext}}) - \alpha_{e,k}\left[\varphi(\boldsymbol{\xi}_i^{\text{ext}}) - \varphi(\boldsymbol{\xi}_i^{\text{int}})\right]\big), \tag{4.49}$$

[1] The definition of the outer product $\otimes$ is $\boldsymbol{u}\otimes\boldsymbol{v} = \boldsymbol{u}\boldsymbol{v}^T$.

the Northern boundary of Ω_k is shared with Ω_{k+K_x}

$$= \frac{\Delta x_k}{4}\frac{\omega_{x,i}}{\rho(z_{k+1/2})}\Big(\rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{int}},t)\sum_{j=0}^{N_p}a_j^k\phi_j(\boldsymbol{\xi}_i^{\text{int}}) + \rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{ext}},t)\sum_{j=0}^{N_p}a_j^{k+K_x}\phi_j(\boldsymbol{\xi}_i^{\text{ext}}) - \alpha_{e,k}\Big[\sum_{j=0}^{N_p}a_j^{k+K_x}\phi_j(\boldsymbol{\xi}_i^{\text{ext}}) - \sum_{j=0}^{N_p}a_j^k\phi_j(\boldsymbol{\xi}_i^{\text{int}})\Big]\Big), \tag{4.50}$$

grid point i of Ω_k is grid point $i - N_z(N_x+1)$ of Ω_{k+K_x}

$$= \frac{\Delta x_k}{4}\frac{\omega_{x,i}}{\rho(z_{k+1/2})}\Big(\rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{int}},t)a_i^k + \rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{ext}},t)a_{i-N_z(N_x+1)}^{k+K_x} - \alpha_{e,k}\Big[a_{i-N_z(N_x+1)}^{k+K_x} - a_i^k\Big]\Big), \tag{4.51}$$

$$= \frac{\Delta x_k\omega_{x,i}}{4\rho(z_{k+1/2})}\Big(\rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{int}},t) + \alpha_{e,k}\Big)a_i^k + \frac{\Delta x_k\omega_{x,i}}{4\rho(z(s_i))}\Big(\rho(z_{k+1/2})w(\boldsymbol{\xi}_i^{\text{ext}},t) - \alpha_{e,k}\Big)a_{i-N_z(N_x+1)}^{k+K_x}, \tag{4.52}$$

where $\alpha_{e,k} = \max_{(\boldsymbol{x},t)\in e^N}\left(\left|\rho(z)w(\boldsymbol{x},\boldsymbol{t})\right|\right)$. Here i was chosen arbitrarily, so it can be written in a matrix-vector form:

$$\int_{e^N}\frac{\phi_i}{\rho(z_{k+1/2})}h_{e,k}\,d\Gamma, \quad \forall i\in\{1,\dots,N_p\} \implies \hat{F}(k)\boldsymbol{a}^k + F^N(k)\boldsymbol{a}^{k+K_x}. \tag{4.53}$$

Similarly, the integrals of the other three element boundaries are calculated and casted in a matrix-vector form. Consequently, the flux vector $\boldsymbol{f}_k$ in (4.37) is defined as:

$$\boldsymbol{f}^k = F(k)\boldsymbol{a}^k + F^N(k)\boldsymbol{a}^{k+K_x} + F^E(k)\boldsymbol{a}^{k+1} + F^S(k)\boldsymbol{a}^{k-K_x} + F^W(k)\boldsymbol{a}^{k-1}. \tag{4.54}$$

4.3.3. Numerical results for the two-dimensional problem

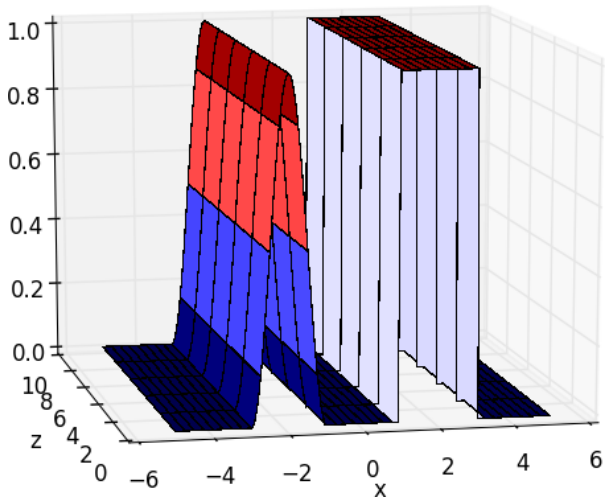
In order to check whether the element matrices are correctly defined, the two-dimensional problem is implemented and tested in Fortran, just as DALES is. Since the three-dimensional case is going to be implemented, only a few tests with the two-dimensional case are done to assure whether the element matrices are defined correctly.

The initial condition that is chosen to test the model with, is the same as in the one-dimensional case, given in Equation (4.12), extended to two dimensions:

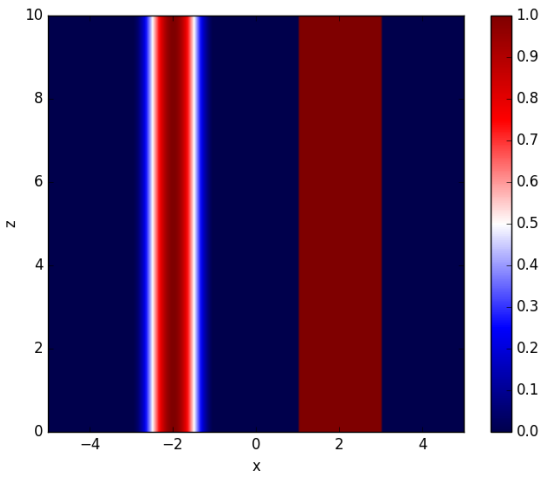
$$\varphi(x,z,0) = \varphi_0(x,z) = \begin{cases}\frac{1-\cos(\pi(x-1))}{2}, & x\in[-3,-1],\ z\in[0,10],\\ 1, & x\in[1,3],\ z\in[0,10],\\ 0, & x\notin[-3,-1]\cup[1,3],\ z\in[0,10].\end{cases} \tag{4.55}$$

In Figure 4.12 the initial condition is shown in a three-dimensional plot, a filled contour plot and a slice of the three-dimensional plot.

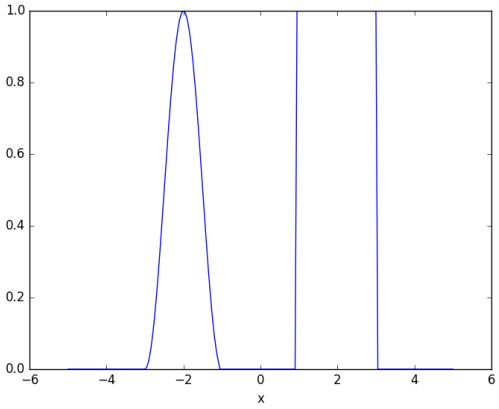
For the first test case, the velocities u in the x-direction and w in the z-directions are given by $u = 1$ and $w = 0$, which is actually the same test case that was used in the one-dimensional case. In Figure 4.13, the results are given after simulating till $t = 10$ s. The independence of z is clearly shown as was expected. In comparison with the one-dimensional test with upwind (see Figure 4.8), the results show more dispersion. This difference is not caused by choosing a different numerical flux, but because of



(a) Initial condition $\varphi(x, z, 0)$ in three-dimensions.



(b) Contourplot of initial condition $\varphi(x, z, 0)$.



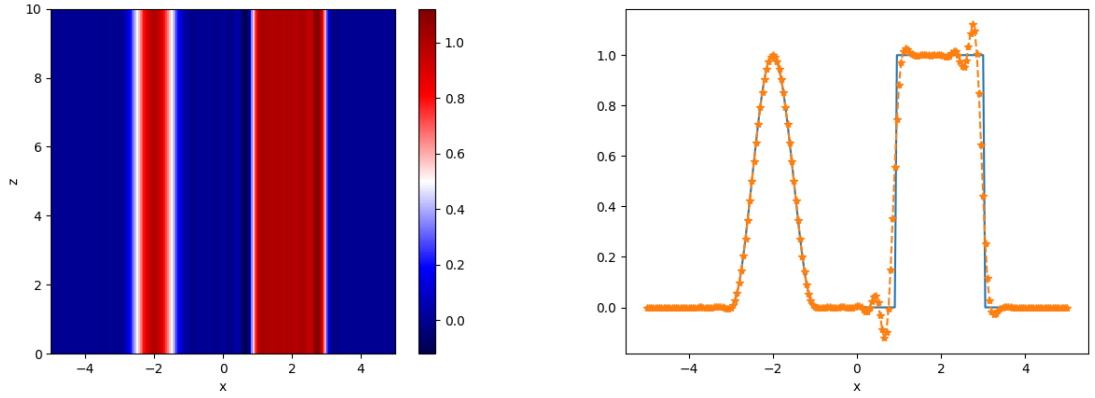
(c) Initial condition $\varphi(x, \tilde{z}, 0)$ in one-dimension for $\tilde{z} \in [0, 10]$.

Figure 4.12: The initial condition $\varphi(x, z, 0)$.

inexact integration that is chosen for DALES.

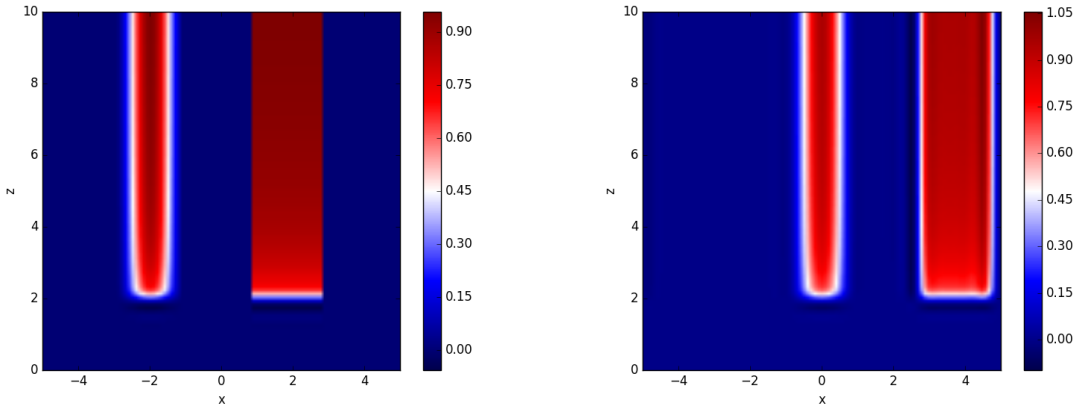
To be sure whether the model calculate the fluxes correctly, different velocities are used. Firstly, in Figure 4.14a, the result at t =2 s is given where $u = 0$ and $w = 1$. Secondly, the velocities $u = w = 1$ are tested. In both test cases the zero-flux boundary is shown perfectly. Clearly, the result is like one could foresee.

All in all, the results of the two-dimensional problem show indeed the one dimensional results extended in the z-direction. Moreover, the element matrices and fluxes seem to work correctly. This means that it can be assumed that the two-dimensional problem is correctly implemented and can be extended to three-dimensions.



(a) Contourplot. (b) $\varphi(x, z, 10)$ with $z = 0.15\,\mathrm{m}$.

Figure 4.13: $\varphi(x, z, 10)$, $K_x = K_z = 100$, $N_x = N_z = 2$ and $\Delta t = 2 \cdot 10^{-2}$.



(a) $u = 0, w = 1$. (b) $u = 1, w = 1$.

Figure 4.14: $\varphi(x, z, 2)$, $K_x = K_z = 100$, $N_x = N_z = 2$ and $\Delta t = 1 \cdot 10^{-2}$.

4.3.4. Three-dimensional problem

In this section the two-dimensional problem is extended to three dimensions. Moreover, the moment limiter is derived for three-dimensional problems.

The extension of the DG method for (4.24) from two dimensions to three dimensions is quite straightforward. For example, element k is given by:

$$\Omega_k = [x_{k-1/2}, x_{k+1/2}] \times [y_{k-1/2}, y_{k+1/2}] \times [z_{k-1/2}, z_{k+1/2}]. \tag{4.56}$$

As a result, there are more grid points and basis functions in every element. For this reason, the counting must be extended, we chose to use x-lexicographic counting, then y-lexicographic, and last but not least z-lexicographic counting.

Consequently, the quantity of interest, φ, is approximated by a larger weighted sum of basis functions:

$$\varphi^k(\boldsymbol{x}, t) = \sum_{j=1}^{N_p} a_h^k(\boldsymbol{x}_j^k, t)\phi_j(\boldsymbol{\xi}(\boldsymbol{x})). \tag{4.57}$$

where $N_p = (N_x + 1)(N_y + 1)(N_z + 1)$ and $\phi_j(x, y, z) = \ell_{x,j}(x)\ell_{y,j}(y)\ell_{z,j}(z)$, with $j \in \{1, \ldots, N_p\}$.

Moreover, the linear transformation has to be done for all three coordinates. Thereupon, the Jacobian $|J|$ becomes:

$$|J(\xi_x,\xi_y,\xi_z)| = \begin{vmatrix} \frac{\partial x}{\partial \xi_x} & \frac{\partial x}{\partial \xi_y} & \frac{\partial x}{\partial \xi_z} \\ \frac{\partial y}{\partial \xi_x} & \frac{\partial y}{\partial \xi_y} & \frac{\partial y}{\partial \xi_z} \\ \frac{\partial z}{\partial \xi_x} & \frac{\partial z}{\partial \xi_y} & \frac{\partial z}{\partial \xi_z} \end{vmatrix} = \frac{1}{8}\Delta x_k \Delta y_k \Delta z_k. \tag{4.58}$$

On top of that, an extra LGL weight is summed over in the quadrature rule:

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} g(\xi_x,\xi_y,\xi_z)\, d\xi_x d\xi_y d\xi_z = \int_{-1}^{1} \sum_{j=0}^{N_y}\sum_{i=0}^{N_x} \omega_{z,j}\omega_{x,i} g(\xi_{x,i},\xi_{y,j},\xi_z)\, d\xi_z, \tag{4.59}$$

$$= \sum_{m=0}^{N_z}\sum_{j=0}^{N_y}\sum_{i=0}^{N_x} \omega_{z,m}\omega_{y,j}\omega_{x,i} g(\xi_{x,i},\xi_{y,j},\xi_{z,m}), \tag{4.60}$$

where

$$\omega_{x,i} = \frac{2}{N_x(N_x+1)(P_{N_x}(\xi_{x,i}))^2}, \qquad \omega_{y,j} = \frac{2}{N_y(N_y+1)(P_{N_y}(\xi_{y,j}))^2}, \qquad \omega_{z,m} = \frac{2}{N_z(N_z+1)(P_{N_z}(\xi_{z,m}))^2}.$$

The LGL quadrature rule is still exact for functions of degrees $\max(2N_x+1, 2N_y+1, 2N_z+1)$ or less (Theorem Cools, 2002 [12]).

These minor changes influences the element matrices, such as an extra weight of the quadrature rule and/or an extra $\frac{1}{2}\Delta y_k$ of the Jacobian when variables are changed. In the next paragraphs the element matrices of (4.37) are shown.

Element mass matrix

The element matrix for the three-dimensional problem is given by:

$$M_{ij}^k = \int_{\Omega_k} \phi_j(\boldsymbol{\xi}(\boldsymbol{x}))\phi_i(\boldsymbol{\xi}(\boldsymbol{x}))\, d\Omega = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \phi_j(\boldsymbol{\xi})\phi_i(\boldsymbol{\xi})\frac{1}{8}\Delta x_k \Delta y_k \Delta z_k\, d\xi_x\, d\xi_y\, d\xi_z, \tag{4.61}$$

$$\approx \sum_{p=0}^{N_x}\sum_{q=0}^{N_y}\sum_{r=0}^{N_z} \frac{1}{8}\Delta x_k \Delta z_k \omega_{x,p}\omega_{y,q}\omega_{z,r}\phi_j(\xi_{x,p},\xi_{y,q},\xi_{z,r})\phi_i(\xi_{x,p},\xi_{y,q},\xi_{z,r}), \tag{4.62}$$

$$= \frac{1}{8}\Delta x_k \Delta y_k \Delta z_k \omega_{x,i}\omega_{y,i}\omega_{z,i}\delta_{ij}. \tag{4.63}$$

Hence, the element mass matrix is with inexact integration still a diagonal matrix.

Element stiffness matrix

The addition of the y-direction also gives an extra term in the element stiffness matrix, which is in the three-dimensional case defined as:

$$\begin{aligned} S_{ij}^k \approx \frac{1}{8}\Delta x_k \Delta y_k \Delta z_k \omega_{x,j_x}\omega_{y,j_y}\omega_{z,j_z}\Bigg(& u(\boldsymbol{x}(\xi_{x,j_x},\xi_{y,j_y},\xi_{z,j_z}),t)\frac{\partial \ell_{x,i}(\xi_{j_x})}{\partial \xi_x}\delta_{i_y,j_y}\delta_{i_z,j_z}\frac{2}{\Delta x_k} \\ & + v(\boldsymbol{x}(\xi_{x,j_x},\xi_{y,j_y},\xi_{z,j_z}),t)\frac{\partial \ell_{y,i}(\xi_{j_y})}{\partial \xi_y}\delta_{i_x,j_x}\delta_{i_z,j_z}\frac{2}{\Delta y_k} \\ & + \frac{w(\boldsymbol{x}(\xi_{x,j_x},\xi_{y,j_y},\xi_{z,j_z}),t)}{\rho(z(\xi_{z,j_z}))}\left[\rho(z(\xi_{j_z}))\frac{\partial \ell_{z,i}(\xi_{j_z})}{\partial \xi_z}\delta_{i_x,j_x}\delta_{i_y,j_y}\frac{2}{\Delta z_k} - \delta_{i,j}\frac{\partial \rho(z(\xi_{j_z}))}{\partial z}\right]\Bigg). \end{aligned} \tag{4.64}$$

Element fluxes
The boundary of element Ω_k consists of six faces, instead of four edges in the two dimensional case, which are defined as follows:

bottom $f_1^k = \{(x,y,z) \in \Omega_k : z = z_{k-1/2}\}$,
top $f_2^k = \{(x,y,z) \in \Omega_k : z = z_{k+1/2}\}$,
left $f_3^k = \{(x,y,z) \in \Omega_k : x = x_{k-1/2}\}$,
right $f_4^k = \{(x,y,z) \in \Omega_k : x = x_{k+1/2}\}$,
front $f_5^k = \{(x,y,z) \in \Omega_k : y = y_{k-1/2}\}$,
back $f_6^k = \{(x,y,z) \in \Omega_k : y = y_{k+1/2}\}$.

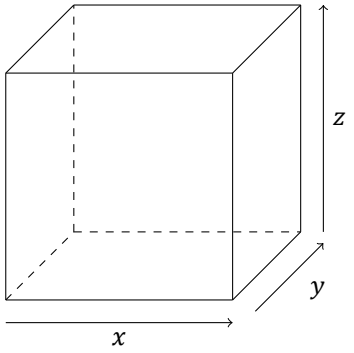


Figure 4.15: Element Ω_k with its 6 faces.

As a result, $\boldsymbol{f}^k$ of Equation (4.37) becomes:

$$\begin{aligned}\boldsymbol{f}^k = F(k)\boldsymbol{a}^k + F_1(k)\boldsymbol{a}^{k-K_xK_y} + F_2(k)\boldsymbol{a}^{k+K_xK_y} + F_3(k)\boldsymbol{a}^{k-1} \\ + F_4(k)\boldsymbol{a}^{k+1} + F_5(k)\boldsymbol{a}^{k-K_x} + F_6(k)\boldsymbol{a}^{k+K_x}.\end{aligned} \tag{4.65}$$

For example, the integral over f_2^k of element Ω_k with i such that $x_i \in f_2^k$ can be written as:

$$\begin{aligned}\int_{f_2^k} \frac{\phi_i}{\rho(z_{k+1/2})} h_{e,k}\, d\Gamma \approx \frac{\Delta x_k \Delta y_k}{8} \frac{\omega_{x,i}\omega_{y,i}}{\rho(z_{k+1/2})} \Big(\rho(z_{k+1/2}) w(\boldsymbol{\xi}_i^{\text{int}}, t) a_i^k + \rho(z_{k+1/2}) w(\boldsymbol{\xi}_i^{\text{ext}}, t) a_{i-N_z(N_y+1)(N_x+1)}^{k+K_xK_y} \\ - \alpha_{e,k}\Big[a_{i-N_z(N_y+1)(N_x+1)}^{k+K_xK_y} - a_i^k\Big]\Big),\end{aligned} \tag{4.66}$$

where $\alpha_{e,k} = \max_{(\boldsymbol{x},t)\in f_2^k} \left(\left|\rho(z_{k-1/2}) w(\boldsymbol{x},t)\right|\right)$. As one can see, there are a few minor changes compared to the two-dimensional case, namely the quadrature weight $\omega_{y,i}$ and the Jacobian change $\frac{1}{2}\Delta y_k$.

4.3.5. Numerical results for the three-dimensional problem

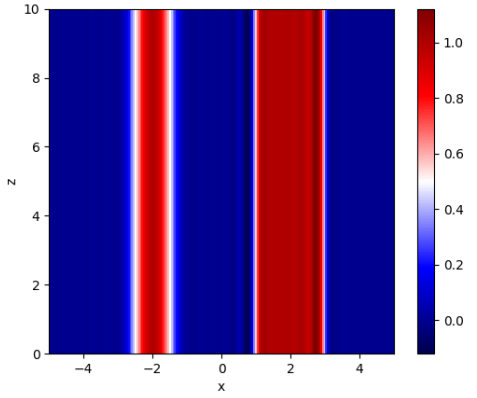
In this section, the numerical results of DG are shown. The test case that is used is the same test case as the one-dimensional test case and the two-dimensional test case extended to three dimensions. In all tests, $\Delta t = 0.99\text{CFL}_{L_2} \frac{u}{\Delta x}$ is chosen such that the time step is big enough to be computationally fast, but also small enough to avoid numerical instability.

Since the initial condition is independent of y and z, only the interesting (y,z)-contour plots are shown. The (x,y)- and (x,z)- contours are both shown to confirm the independence of y and z. Moreover, the plots only show one value of every node even though there are two values for the nodes on the element boundary. In the plots half cells are shown $[x_{k-1/2}, x_{k+1/2}) \times [y_{k-1/2}, y_{k+1/2}) \times [z_{k-1/2}, z_{k+1/2})$.

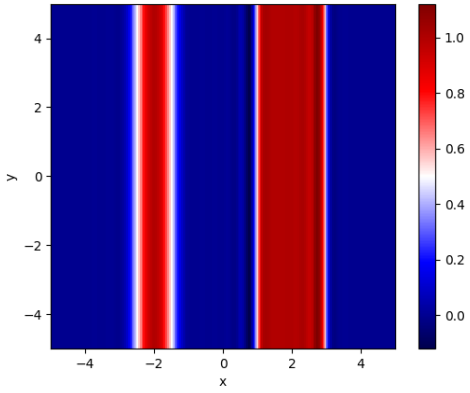
In Figure 4.16, the result after simulating 10 seconds with $u = 1$, $v = w = 0$ can be seen. Hence, the result is the same as in the two-dimensional case (Figure 4.13), only extended in the y-direction. Moreover, the numerical errors are calculated in the L_2- and ℓ_2-norm in the same way as described in Chapter 4.2.4. The numerical error in L_2-norm is 2.2096 and in ℓ_2-norm 5.3843.

When $\varphi(x,y,z,t)$ is approximated after 50 s, the dispersion becomes worse. However, the continuous part almost remained the same with exceptions of the small oscillations before and after the wave. These small oscillations are the result of the continuous part that is not smooth enough for DG to have no problems with it and when the dispersion is advected, they seem to get amplified. The numerical errors in L_2- and ℓ_2-norm are 2.6459 and 6.2813 respectively.

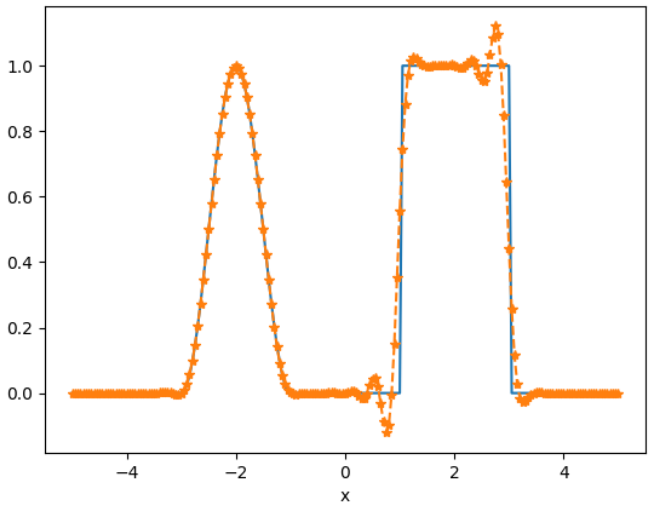
On top of that, the computational time per time step is calculated by averaging the computation time per time step using 2393 samples (which is a simulation till t =50 s). The average computational time



(a) $\varphi(x, -5, z, 10)$



(b) $\varphi(x, -5, z, 10)$

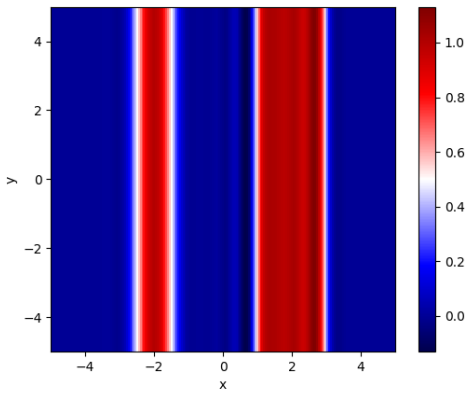


(c) $\varphi(x, -5, 0, 10)$

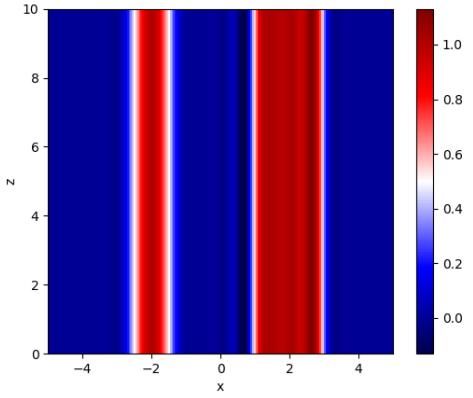
Figure 4.16: The approximation of $\varphi(x, y, z, 10)$ with DG using $K_x = 100$, $K_y = K_z = 2$, $N_x = N_y = N_z = 2$ and $\Delta t = 0.99\text{CFL}_2 \frac{u}{\Delta x}$.

for DG with $N = 2$ is 0.150 s per time step.

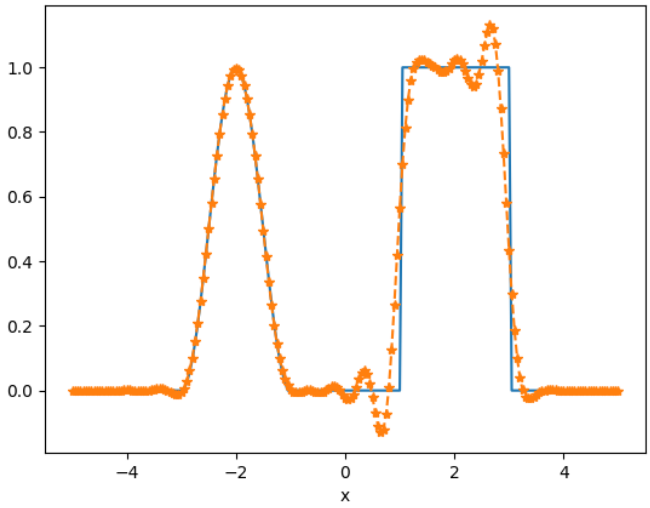
All in all, the same conclusion can be drawn as in the literature study prior to this thesis work [5] is drawn; DG works very well for smooth functions, but for non-smooth functions dispersions occur which worsen in time. Therefore, a limiter is needed to remove the dispersion.



(a) Filled contourplot of the approximation of $\varphi(x, y, 0, 50)$.



(b) Filled contourplot of the approximation of $\varphi(x, -5, z, 50)$.



(c) The approximation of $\varphi(x, -5, 0, 50)$.

Figure 4.17: The approximation of $\varphi(x, y, z, 50)$ with DG using $K_x = 100$, $K_y = K_z = 2$, $N_x = N_z = 2$ and $\Delta t = 0.99\text{CFL}_2 \frac{u}{\Delta x}$.

4.3.6. Moment limiter derivation and extension to three-dimensions

In Section 4.2.2, the moment limiter is introduced for the first time. For our three-dimensional problem, the moment limiter has to be extended to three dimensions. First, the derivation of the moment limiter is explained for one-dimensional problems. Thereafter, the extension to three dimensions and its corresponding limiting order are given.

Derivation of the one-dimensional moment limiter

Just like most limiters for DG, the limiter is based on the modal form of the solution. As noted before in Section 4.2, switching between nodal and modal form can easily be done with the VanderMonde matrix. For a one-dimensional problem, the modal form is written as:

$$\varphi = \sum_{i=0}^{p} \hat{a}_i^k P_i(\xi), \tag{4.67}$$

with $P_i(\xi)$ the ith Legendre polynomial and for example, the following linear transformation

$$\xi = \frac{2(x - x_k)}{\Delta x_k}, \tag{4.68}$$

is used such that the element $[x_k, x_{k+1}]$ is mapped to $[-1, 1]$.

In [18], Krivodonova explains that using a Taylor series shows that

$$\hat{a}_i^k \approx C\Delta x^i \frac{\partial^i \varphi(\xi)}{\partial x^i},\ i \in \{1, \dots, p\}, \tag{4.69}$$

when there are no discontinuities. This means that forward and backward differences using $\hat{a}_{i-1}^k$, $\hat{a}_{i-1}^{k+1}$ and $\hat{a}_{i-1}^{k-1}$ can be used to approximate the ith derivative $\hat{a}_i^k$. As a result, the derivative of φ is to be calculated to derive the moment limiter.

Using the fact that $\frac{\partial^2 \xi}{\partial x^2} = 0$, the first i derivatives of φ are given by:

$$\frac{\partial \varphi}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial \varphi}{\partial \xi}, \tag{4.70}$$

$$\frac{\partial^2 \varphi}{\partial x^2} = \left(\frac{\partial \xi}{\partial x}\right)^2 \frac{\partial^2 \varphi}{\partial \xi^2} + \frac{\partial^2 \xi}{\partial x^2}\frac{\partial \varphi}{\partial \xi} = \left(\frac{\partial \xi}{\partial x}\right)^2 \frac{\partial^2 \varphi}{\partial \xi^2}, \tag{4.71}$$

$$\frac{\partial^3 \varphi}{\partial x^3} = \frac{\partial}{\partial x}\left[\left(\frac{\partial \xi}{\partial x}\right)^2 \frac{\partial^2 \varphi}{\partial \xi^2}\right] = 2\frac{\partial \xi}{\partial x}\frac{\partial^2 \xi}{\partial x^2}\frac{\partial^2 \varphi}{\partial \xi^2} + \left(\frac{\partial \xi}{\partial x}\right)^3 \frac{\partial^3 \varphi}{\partial \xi^3} = \left(\frac{\partial \xi}{\partial x}\right)^3 \frac{\partial^3 \varphi}{\partial \xi^3}, \tag{4.72}$$

$$\vdots$$

$$\frac{\partial^i \varphi}{\partial x^i} = \left(\frac{\partial \xi}{\partial x}\right)^i \frac{\partial^i \varphi}{\partial \xi^i}. \tag{4.73}$$

For the derivative of $\frac{\partial \varphi}{\partial \xi}$, the derivative of the ith Legendre polynomial [39] is needed, which is defined as follows:

$$\frac{\partial}{\partial \xi} P_i(\xi) = (2i-1)P_{i-1}(\xi) + (2i-5)P_{i-3}(\xi) + (2i-7)P_{i-5}(\xi) + \dots, \tag{4.74}$$

and the second derivative is given by:

$$\frac{\partial^2}{\partial \xi^2} P_i(\xi) = \frac{\partial}{\partial \xi}\Big((2i-1)P_{i-1}(\xi) + (2i-5)P_{i-3}(\xi) + (2i-7)P_{i-5}(\xi) + \dots\Big), \tag{4.75}$$

$$= (2i-1)\Big((2i-3)P_{i-2}(\xi) + (2i-7)P_{i-4}(\xi) + \dots\Big) + \frac{\partial}{\partial \xi}\Big((2i-5)P_{i-3}(\xi) + (2i-7)P_{i-5}(\xi) + \dots\Big). \tag{4.76}$$

As one can see that the derivatives of the Legendre polynomials are, like the Legendre polynomials itself, recursive. The recursion can be written with the double factorial $(2i-1)!!$. The double factorial[35] is defined as:

$$a!! = \begin{cases} a(a-2)(a-4)\cdots 5\cdot 3\cdot 1 & a > 0 \text{ odd}, \quad (4.77a) \\ a(a-2)(a-4)\cdots 6\cdot 4\cdot 2 & a > 0 \text{ even}, \quad (4.77b) \\ 1 & a = -1, 0. \quad (4.77c) \end{cases}$$

Using this information, the $(i-1)$th derivative of φ can be written as follows:

$$\frac{\partial^{i-1}\varphi_k}{\partial x^{i-1}} = \left(\frac{\partial \xi}{\partial x}\right)^{i-1} \frac{\partial^{i-1}}{\partial \xi^{i-1}}\left(\sum_{i=0}^{p} \hat{a}_i^k P_i(\xi)\right), \tag{4.78}$$

$$= \left(\frac{\partial \xi}{\partial x}\right)^{i-1} \frac{\partial^{i-1}}{\partial \xi^{i-1}}\left(\sum_{0<m<i-1}\left[\hat{a}_m^k P_m(\xi)\right] + \hat{a}_{i-1}^k P_{i-1}(\xi) + \sum_{m>i-1}^{p} \hat{a}_m^k P_m(\xi)\right), \tag{4.79}$$

$$= \left(\frac{2}{\Delta x_k}\right)^{i-1}\left(0 + \hat{a}_{i-1}^k (2i-3)!! + \frac{\partial^{i-1}}{\partial \xi^{i-1}} \sum_{m>i-1}^{p} \hat{a}_m^k P_m(\xi)\right), \tag{4.80}$$

$$= \left(\frac{2}{\Delta x_k}\right)^{i-1}\left(\hat{a}_{i-1}^k (2i-3)!! + \frac{\partial^{i-1}}{\partial \xi^{i-1}} \sum_{m>i-1}^{p} \hat{a}_m^k P_m(\xi)\right), \tag{4.81}$$

and the ith derivative as:

$$\frac{\partial^{i}\varphi_k}{\partial x^{i}} = \left(\frac{2}{\Delta x_k}\right)^{i}\left(\hat{a}_i^k (2i-1)!! + \frac{\partial^{i}}{\partial \xi^{i}} \sum_{m>i}^{p} \hat{a}_m^k P_m(\xi)\right). \tag{4.82}$$

Using forward differencing, the ith derivative of φ can be approximated by:

$$\left(\frac{\partial^{i-1}\varphi_{k+1}}{\partial x^{i-1}} - \frac{\partial^{i-1}\varphi_k}{\partial x^{i-1}}\right)/\Delta x_k = \left(\frac{2}{\Delta x_k}\right)^{i}\left(\frac{1}{2}(2i-3)!!\,(\hat{a}_{i-1}^{k+1} - \hat{a}_{i-1}^k) + \frac{1}{2}\frac{\partial^{i-1}}{\partial \xi^{i-1}} \sum_{m>i-1}^{p} (\hat{a}_m^{k+1} - \hat{a}_m^k) P_m(\xi)\right). \tag{4.83}$$

Comparing the right-hand side of (4.83) with that of (4.82), we find that

$$\hat{a}_i^k = \frac{\hat{a}_{i-1}^{k+1} - \hat{a}_{i-1}^k}{2(2i-1)} + \mathcal{O}(\Delta x_k^{i+1}). \tag{4.84}$$

This means that the leading term of the ith derivative is allowed to be $2(2i-1)$ bigger than the forward and backward differences of the $(i-1)$th derivatives of the neighbouring cells. Therefore, the limited version of $\tilde{a}_i^k$ is chosen to be:

$$\tilde{a}_i^k = \text{minmod}(\hat{a}_i^k, \alpha_i(\hat{a}_{i-1}^{k+1} - \hat{a}_{i-1}^k), \alpha_i(\hat{a}_{i-1}^k - \hat{a}_{i-1}^{k-1})), \tag{4.85}$$

where

$$\frac{1}{2(2i-1)} \leq \alpha_i \leq 1. \tag{4.86}$$

When α_i is chosen outside the interval, it resulted in either loss of accuracy or numerical instability [18]. Using $\alpha_i = \frac{1}{2(2i-1)}$ more dissipation is added than when $\alpha_i = 1$ is used.

Three-dimensional moment limiter

The approximated function φ is defined as:

$$\varphi = \sum_{i=0}^{N_x}\sum_{j=0}^{N_y}\sum_{\ell=0}^{N_z} \hat{a}_{i,j,\ell}^{k_x,k_y,k_z} \psi_i(\xi_x)\psi_y(\xi_y)\psi_z(\xi_z), \tag{4.87}$$

where $\psi(\xi)$ is the basis function in one dimension. Likewise, the nodal DG is defined in (4.57), the three-dimensional basis functions are the tensor products of one-dimensional basis functions. In this case, the scaled Legendre polynomials are chosen, such that the function space of basis functions ψ_n is orthonormal:

$$\psi_n(\xi) = \sqrt{n+\frac{1}{2}} P_n(\xi), \quad n = \{i, j, \ell\} \tag{4.88}$$

When φ is differentiated $i > 0$ times in the x-direction, $j > 0$ times and $\ell > 0$ times in the y- and z-direction, the derivative is given by:

$$\frac{\partial^{i+j+\ell}\varphi}{\partial x^i \partial y^j \partial z^\ell} = \left(\frac{2}{\Delta x}\right)^i \left(\frac{2}{\Delta y}\right)^j \left(\frac{2}{\Delta z}\right)^\ell \left[\sqrt{i+\frac{1}{2}}\sqrt{j+\frac{1}{2}}\sqrt{\ell+\frac{1}{2}}(2i-1)!!\,(2j-1)!!\,(2\ell-1)!!\,\hat{a}_{i,j,\ell}^{k,m,p} + \frac{\partial^{i+j+\ell}}{\partial x^i \partial y^j \partial z^\ell}\sum_{u>i}^{N_x}\sum_{v>j}^{N_y}\sum_{w>\ell}^{N_z}\hat{a}_{u,v,w}^{k,m,p}\psi_i(\xi_x)\psi_y(\xi_y)\psi_z(\xi_z)\right]. \tag{4.89}$$

In contrast with the one-dimensional derivative, see Equation (4.82), the coefficient $\hat{a}_{i,j,\ell}^{k,m,p}$ is also multiplied with the scaling of the basis function and correspondingly, the influence of the y- and z-direction is present.

If φ is one time less differentiated, for example in the x-direction, it looks like this:

$$\frac{\partial^{i-1+j+\ell}\varphi}{\partial x^{i-1} \partial y^j \partial z^\ell} = \left(\frac{2}{\Delta x}\right)^{i-1} \left(\frac{2}{\Delta y}\right)^j \left(\frac{2}{\Delta z}\right)^\ell \left[\sqrt{i-\frac{1}{2}}\sqrt{j+\frac{1}{2}}\sqrt{\ell+\frac{1}{2}}(2i-3)!!\,(2j-1)!!\,(2\ell-1)!!\,\hat{a}_{i-1,j,\ell}^{k,m,p} + \frac{\partial^{i+j+\ell}}{\partial x^i \partial y^j \partial z^\ell}\sum_{u\geq i}^{N_x}\sum_{v>j}^{N_y}\sum_{w>\ell}^{N_z}\hat{a}_{u,v,w}^{k,m,p}\psi_i(\xi_x)\psi_y(\xi_y)\psi_z(\xi_z)\right]. \tag{4.90}$$

Comparing (4.89) with its approximation using forward differencing with (4.90), we find that:

$$\hat{a}_{i,j,\ell}^{k,m,p} = \frac{\hat{a}_{i-1,j,\ell}^{k+1,m,p} - \hat{a}_{i-1,j,\ell}^{k,m,p}}{2\sqrt{4i^2-1}} + \mathcal{O}(\Delta x^{i+1}\Delta y^{j+1}\Delta z^{\ell+1}). \tag{4.91}$$

Hence, the same can be done for the y- and z-direction, which all results to the same lower bound for α_n with $n = \{i, j, \ell\}$:

$$\frac{1}{2\sqrt{4n^2-1}} \leq \alpha_n \leq \frac{\sqrt{2n-1}}{\sqrt{2n+1}}, \quad n = \{i, j, \ell\}. \tag{4.92}$$

Again the lower bound is the most dissipative choice, while the upper bound is the least. Note that these bounds are the one-dimensional bounds (4.86) multiplied by $\frac{\sqrt{2n-1}}{\sqrt{2n+1}}$. This factor is a result of the scaling factors of the Legendre polynomials. This was also found by Krivodonova in [**?**] for two-dimensions.

Consequently, the limited version of $\tilde{a}_{i,j,\ell}^{k,m,p}$ is found by:

$$\tilde{a}_{i,j,\ell}^{k,m,p} = \text{minmod}(\hat{a}_{i,j,\ell}^{k,m,p}, \alpha_i(\hat{a}_{i-1,j,\ell}^{k+1,m,p} - \hat{a}_{i-1,j,\ell}^{k,m,p}), \alpha_i(\hat{a}_{i-1,j,\ell}^{k,m,p} - \hat{a}_{i-1,j,\ell}^{k-1,m,p}), \alpha_j(\hat{a}_{i,j-1,\ell}^{k,m+1,p} - \hat{a}_{i,j-1,\ell}^{k,m,p}), \alpha_j(\hat{a}_{i,j-1,\ell}^{k,m,p} - \hat{a}_{i,j-1,\ell}^{k,m-1,p}), \alpha_\ell(\hat{a}_{i,j,\ell-1}^{k,m,p+1} - \hat{a}_{i,j,\ell-1}^{k,m,p}), \alpha_\ell(\hat{a}_{i,j,\ell-1}^{k,m,p} - \hat{a}_{i,j,\ell-1}^{k,m,p-1})), \tag{4.93}$$

where α_n is in the interval (4.92).

In this thesis project, $\alpha_n = \frac{\sqrt{2n-1}}{\sqrt{2n+1}}$ is chosen, since the KNMI wants to have an advection scheme that is the least diffusive.

Limiting order for the three-dimensional moment limiter

The limiting order for three-dimensional problems is less straightforward than for one-dimensional problems, however, the idea stays the same: limit from the highest to the lowest order. This is more difficult, because different functions can have the same polynomial degree. For example, the functions x^2y, x^3, x^2z, and xyz are polynomials of degree three. Thus, the question is which coefficient is limited first.

In [18], Krivodonova shows the limiting order for two-dimensional problems when the polynomial degree of the basis functions is $N_x = N_y = N$. In short, the limiter stops when a coefficient $a_{i,i}$ does not need limiting or a symmetric pair $a_{i,j}$ and $a_{j,i}$ does not need limiting. The order in which is limited can be found in Table 4.4. First $a_{N,N}$ is limited, when no limiting is needed, the limiter is stopped. Otherwise, the coefficients $a_{N,N-1}$ and $a_{N-1,N}$ are limited. When both are limited, the limiter stops. If not, the next coefficients are checked.

$a_{N,N}$	
$a_{N,N-1}$	& $a_{N-1,N}$
$a_{N,N-2}$	& $a_{N-2,N}$
⋮	
$a_{N,0}$	& $a_{0,N}$
$a_{N-1,N-1}$	
$a_{N-1,N-2}$	& $a_{N-2,N-1}$
$a_{N-1,N-3}$	& $a_{N-3,N-1}$
⋮	
$a_{N-1,0}$	& $a_{0,N-1}$
$a_{1,1}$	
$a_{1,0}$	& $a_{0,1}$

Table 4.4: Limiting order for two-dimensional domains with polynomial orders $N_x = N_z = N$.

When the limiting order is extended for three-dimensional problems, there are no symmetrical coefficients anymore, because the coefficients depend on an uneven number of parameters. Instead of the symmetry of the coefficients, the combination of the three parameters i, j, ℓ is looked at. When multiple combinations of the three parameters i, j, ℓ exist, all the coefficients $a_{i,j,\ell}$, $a_{i,\ell,j}$, $a_{j,i,\ell}$, $a_{j,\ell,i}$, $a_{\ell,i,j}$ and $a_{\ell,j,i}$ must be limited. When all of them are not limited, the limiting is stopped.

In Table 4.5, the limiting order is given when the polynomial degree of the basis functions in the x-, y- and z- directions are equal. Each row shows the parameters i, j, ℓ and represents all the coefficients with all existing combinations of i, j, ℓ. The limiter stops when all the coefficients from the row do not need limiting. This limiting order is also described in the appendix of [13].

Bear in mind, when a parameter i, j or ℓ is 0, the backward and forward difference are undefined. Consequently, the terms with α_0 are left out in (4.93).

When N_x, N_y and N_z differ from each other, the limiting order changes. The ℓ-column (the fastest changing column) now stands for $\max(N_x, N_y, N_z)$ and the i-column for $\min(N_x, N_y, N_z)$. One should be careful with the combinations, several combinations may not exist.

Hence, the polynomial degree of the basis function is not taken into account. For example, the coefficients with $N, 0, 0$ are the corresponding weights of the basis functions with polynomial degree N. These coefficients has to be limited before the coefficient with $N-1, N-1, N-1$, which belongs to the basis function with polynomial degree $3N-3$. However, if one take the polynomial degree of the basis function into account, it would be the other way around. Nevertheless, the basis functions of the coefficients $N, 0, 0$ have a high order polynomial in one direction instead of lower order polynomials in all three dimensions. In Figure B.1, numerical results are shown when the incorrect limiting order is used; the basis function xyz is limited before the basis functions x^2, y^2 and z^2. This small difference in the limiting order lead to more peak clipping. In short, the limiting order is very important for the results of the moment limiter.

i	j	ℓ
N	N	N
N	N	$N-1$
$\vdots$	$\vdots$	$\vdots$
N	N	0
N	$N-1$	$N-1$
N	$N-1$	$N-2$
$\vdots$	$\vdots$	$\vdots$
N	$N-1$	0
N	$N-2$	$N-2$
N	$N-2$	$N-3$
$\vdots$	$\vdots$	$\vdots$
N	$N-2$	0
$\vdots$	$\vdots$	$\vdots$
N	0	0
$N-1$	$N-1$	$N-1$
$\vdots$	$\vdots$	$\vdots$
$N-1$	$N-1$	0
$\vdots$	$\vdots$	$\vdots$
1	0	0

Table 4.5: Limiting order for three dimensional domains using $N_x = N_y = N_z = N$. Each row shows the three parameters i, j, ℓ that represent all the coefficients with all existing combinations of i, j, ℓ.

4.3.7. Numerical results with the three-dimensional moment limiter

In this section, the numerical results are shown using DG with the moment limiter and basis functions of polynomial order $N_x = N_y = N_z = 2$ for a three-dimensional test case.

Limit after every time step

In Figure 4.18, the effect is shown when the moment limiter is used after every time step. Indeed the dispersion is removed, nevertheless, the peak of the continuous part is clipped. On top of that, there is some diffusion in the discontinuous part, which can also be seen in Figure 4.16 where there is no limiter used. Another interesting observation is the asymmetry at the discontinuity, which is the remainder of the bigger oscillations. The numerical error in L_2-norm is 2.1766 and in ℓ_2-norm 4.9466.

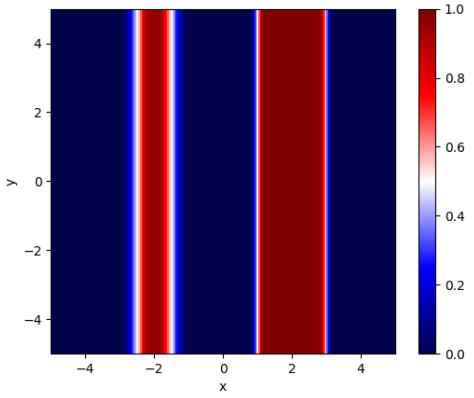
The approximation of φ after 50 s can be seen in Figure 4.19. In contrast with the simulation till 10 s, the peak is more clipped which means that more mass is lost. Recall that without limiter (see figure 4.17), no mass was lost, meaning that the limiter does not necessarily conserve mass. Moreover, more diffusion took place in the discontinuous part. The numerical errors of this simulation are 2.5570 and 5.7656 in the L_2- and ℓ_2-norm respectively. The average time step takes 0.152 s.

In comparison with the one-dimensional result of the moment limiter using exact integration (see Figure 4.8), more peak clipping takes place with inexact integration. Nevertheless, the discontinuous part looks better, because it has less diffusion. Even though, the use of exact integration looks better when φ is approximated after simulating 10 s, after 50 s the results with inexact integration show much less diffusion, making inexact integration a better option in this case. Like Krivodonova mentioned in [18], the moment limiter indicates to be especially beneficial for long-time simulations.

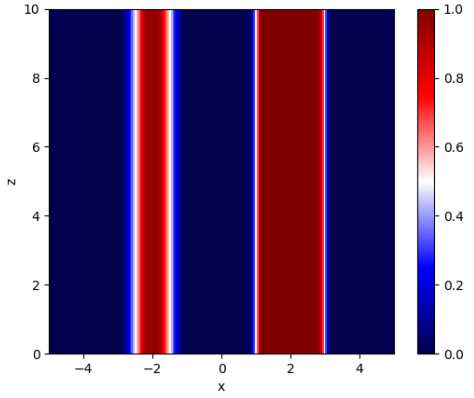
Limit after every RK3 step

Another option could be to limit after every RK3 step instead only once after a complete time step. The advantage of this is that no distinction has to be made between every RK3 step. Moreover, it will save some extra memory space in DALES. This is explained in Chapter 5.

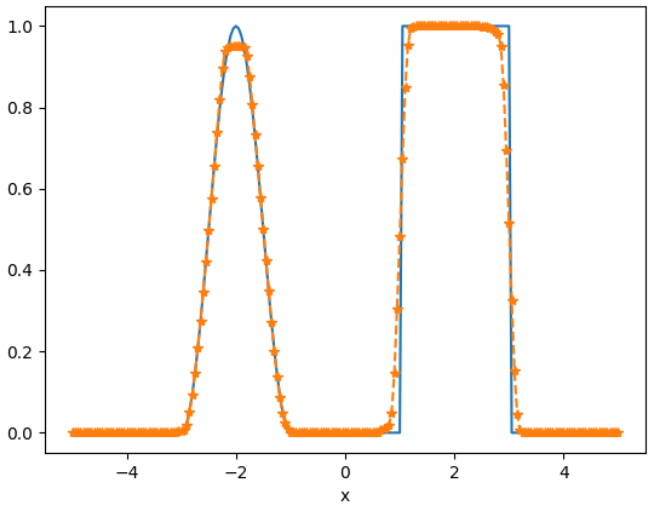
In Figures 4.20 and 4.21, the approximations of φ at $t = 10$ s and $t = 50$ s are given. With the naked eye one cannot see any difference between limiting every RK3 step instead of only once every time



(a) $\varphi(x, y, 0, 10)$



(b) $\varphi(x, -5, z, 10)$.



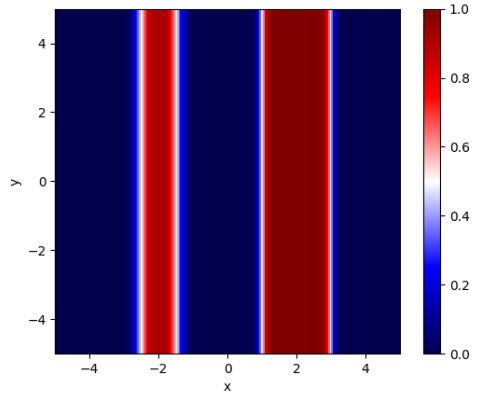
(c) $\varphi(x, -5, 0, 10)$

Figure 4.18: $\varphi(x, y, z, 10)$ with $K_x = 100$, $K_y = K_z = 2$ when limiter is used, $\Delta t = 2.089 \cdot 10^{-2}$.

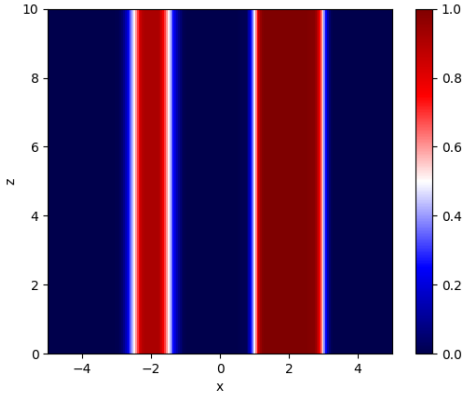
step. Though in numerical error there is a small difference. Limiting every RK3 step has a smaller numerical error in the order of 10^{-3}; The numerical error after simulating till t =10 s is 2.1759 and 4.9435, and till t =50 s 2.5551 and 5.7607 in L_2- and ℓ_2-norm respectively. The average time step takes longer than when only once every time step is limited, namely it takes 0.158 s.

Conclusion

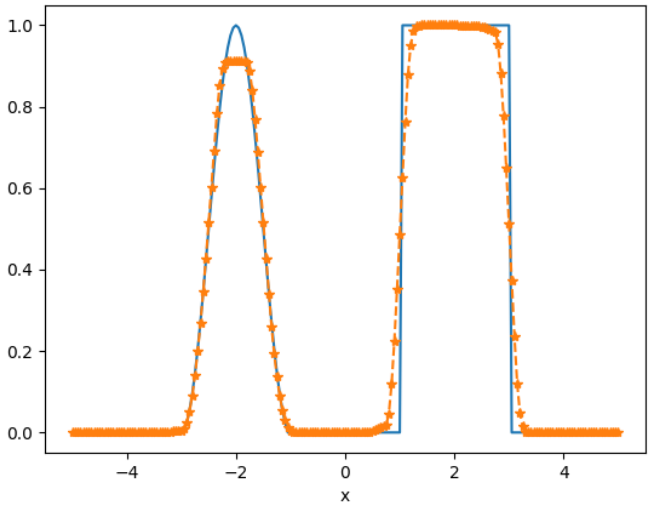
In short, using a limiter clearly improves the results by removing the oscillations. Two choices can made; limit every Rk3 step or only limit once after the last RK3 step. The numerical results show no significant difference, except that the accuracy is slightly better for limiting every RK3 step, but it also takes about 0.006 s longer to compute per time step.



(a) $\varphi(x, y, 0, 50)$.

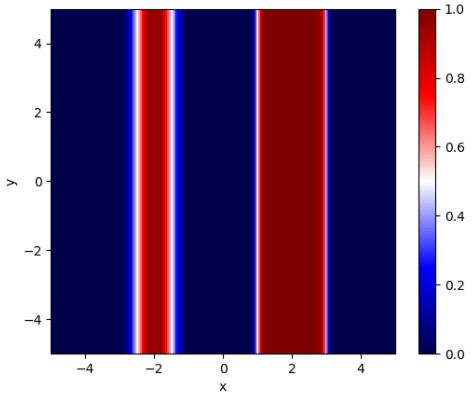


(b) $\varphi(x, -5, z, 50)$

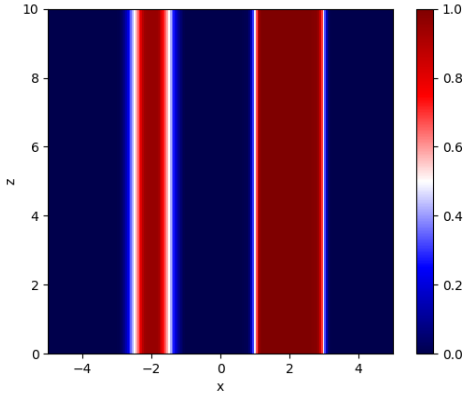


(c) $\varphi(x, -5, 0, 50)$, $\Delta t = 2.089 \cdot 10^{-2}$

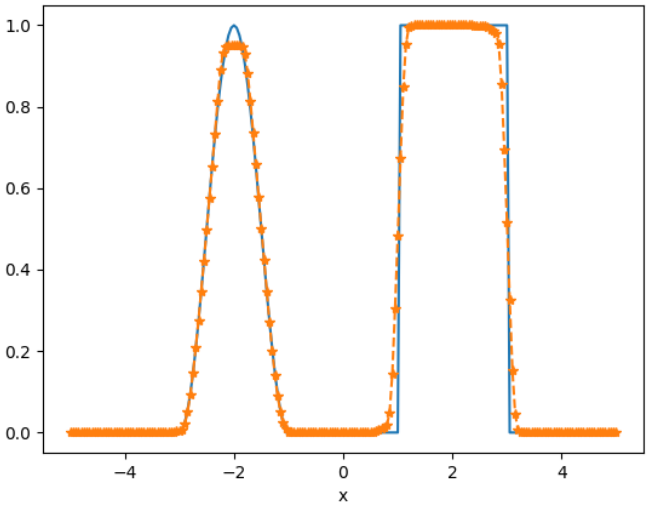
Figure 4.19: $\varphi(x, y, z, 50)$ with $K_x = 100$, $K_y = K_z = 2$ when limiter is used.



(a) $\varphi(x, y, 0, 10)$

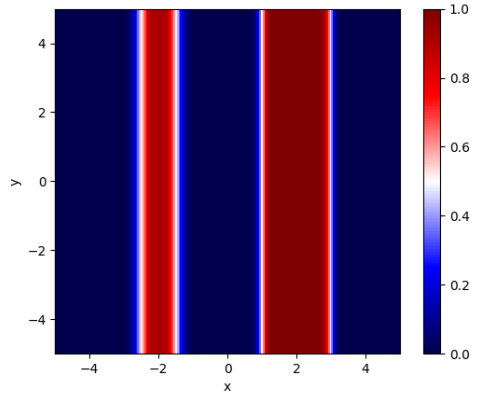


(b) $\varphi(x, -5, z, 10)$.

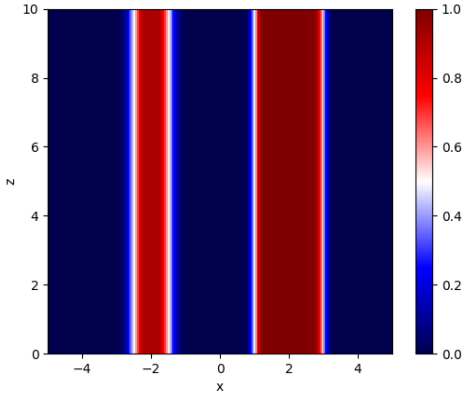


(c) $\varphi(x, -5, 0, 10)$

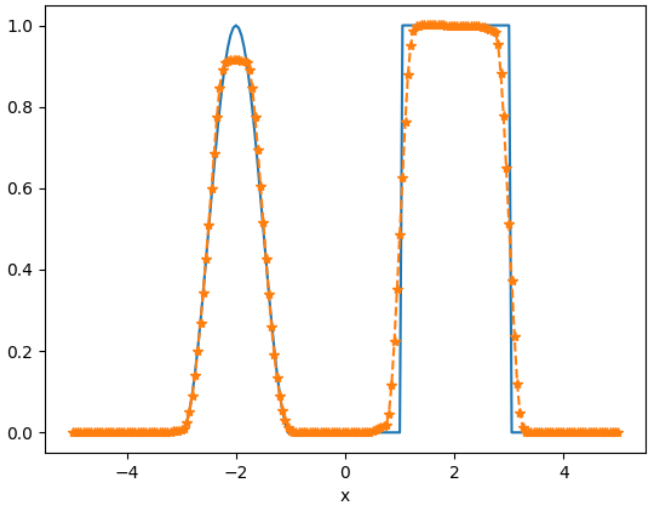
Figure 4.20: $\varphi(x, y, z, 10)$ with $K_x = 100$, $K_y = K_z = 2$ when limiter after every Rk3 step is used, $\Delta t = 2.089 \cdot 10^{-2}$.



(a) $\varphi(x, y, 0, 50)$.



(b) $\varphi(x, -5, z, 50)$



(c) $\varphi(x, -5, 0, 50)$, $\Delta t = 2.089 \cdot 10^{-2}$

Figure 4.21: $\varphi(x, y, z, 50)$ with $K_x = 100$, $K_y = K_z = 2$ when limiter after every RK3 step is used.

5

Comparison of WENO and DG in DALES

In this chapter, the discontinuous Galerkin method is compared with the WENO method in a DALES setup. First, the choices on how DG is implemented in DALES are explained. After that, the numerical results of both methods are compared.

5.1. Implementation choices

We have chosen to work first with a stripped version of DALES which we call the LES wrapper. In this stripped version, only the advection subroutine is used; all other subroutines are not included in the LES wrapper. This means that no subroutines can influence the numerical results except the advection scheme.

The LES wrapper did not take into account to simulate exactly the run time. As a result, when one wants to simulate $\varphi(x, y, z, 10)$, it is possible that the results show $\varphi(x, y, z, 10 + \beta\Delta t)$ with $0 \leq \beta < 1$. In our test case, $\Delta t = 2 \cdot 10^{-2}$ is chosen which satisfies the CFL condition for both WENO and DG for $u = 1$, $v = 0$ and $w = 0$.

Each DG cell exists of one DALES cell (see Figure 5.1). This is chosen such that the accuracy is as high as possible without increasing the computational cost by taking more DG cells in one DALES cell.

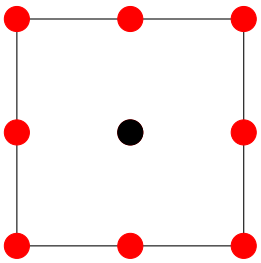


Figure 5.1: One DG cell as one DALES grid box in two-dimensions. The black node is the one node which is also the DALES variable φ_{FVM}, all other nodes are only DG variables.

The advection subroutines of DALES obtain the variable φ that must be advected as input and returns the tendency $\frac{\partial \varphi}{\partial t}$ as output. The input and output variable are one value for each grid cell, which is the cell average like with the finite volume method (FVM). For DG with $N_x = N_y = N_z = 2$, 27 nodal values are needed for each grid cell (see Figure 5.1). Therefore, a mapping is used to create the variables needed for DG, which will be referred to as mapping a (see Figure 5.3). One can test several mappings; we have chosen two mappings:

1. Cell average a: Give all 27 nodal values the cell average as value.
2. L_2-projection: Use the L_2-projection to the continuous finite element space.

For each method for mapping a, a name is given, Cell average a and L_2-projection.

Thereafter, DG can be applied on φ to find the tendency $\frac{\partial \varphi}{\partial t}$ of every 27 nodal points. Again a mapping, refered to as mapping b, is needed to return the needed output of DALES. In this thesis, two mappings are explained and tested:

1. Cell average of tendency: Take the cell average of the tendencies of the DG values.
2. Cell average b: Calculate the advected $\varphi_{\text{DG}}(\boldsymbol{x}, t + \beta\Delta t)$. Thereafter, the cell average of φ_{DG} is determined to find $\varphi_{\text{FVM}}(\boldsymbol{x}, t + \beta\Delta t)$ with which the tendency of φ_{FVM} is calculated.

When a limiter is needed, extra steps have to be taken. The limiter is used on φ itself and not the tendency, so the tendency needs to be integrated in time before it can be limited. With the limited $\tilde{\varphi}$, the tendency can be recalculated which can be again mapped to the tendency that is needed for DALES. In Figure 5.2, a summary of all steps are given in a flow chart.

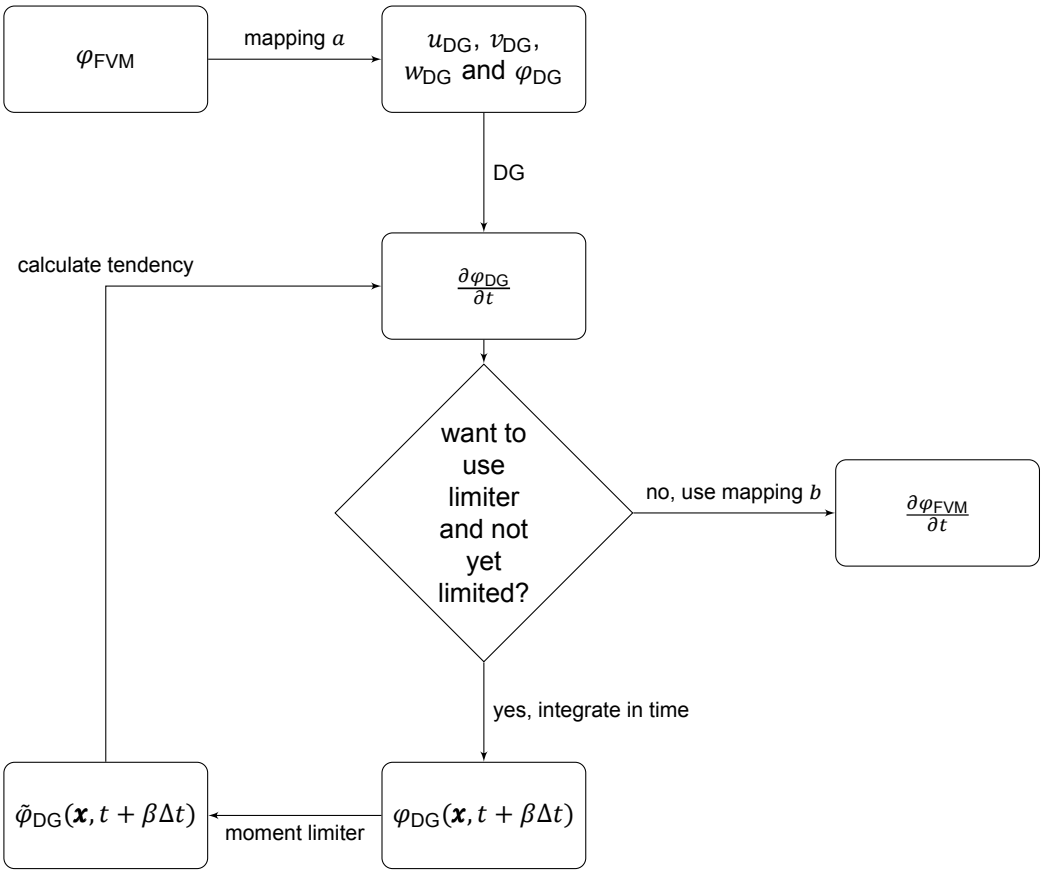


Figure 5.2: Flow chart of the DG advection scheme of DALES in general. β denotes the RK3 coefficient.

One can ask why the DG variable is not saved every time step instead of using mappings; this would probably result in the same numerical results as in Chapter 4. However, this would bring several complications with it. First of all, the time integration is done outside the advection scheme meaning that it would be done twice. Second of all, DALES uses operational splitting which means that the variable is also influenced outside the advection scheme. Consequently, extra mappings for all other tendencies of the variable are needed from their FVM variable to the DG variable, which are not straightforward. Third of all, the advection scheme can be used for multiple variables, which would also acquire more memory space. This means that for each variable a corresponding DG variable must be saved. On the

one hand, this could be solved by using dynamic variable names, nevertheless, Fortran does not support this. On the other hand, an allocatable multiple dimensional vector could be used which requires some creativity in implementation, like the number of variables that are going to be advected with DG, must be known before it can be allocated. In short, saving the DG variable did not seem to be a good option.

In Figure 5.2, it is shown that there is no distinction made of every RK3 step in our advection scheme. In other words, the limiter is used every RK3 step instead only once every time step. There are two reasons for this choice. Firstly, the other option would acquire to save all DG variables of all variables that use our advection scheme, which is not easy as discussed above. Secondly, the numerical accuracy is slightly better for more or less the same computational time.

5.1.1. Mappings

In this section the ideas of the mappings a and b are explained and how they work.

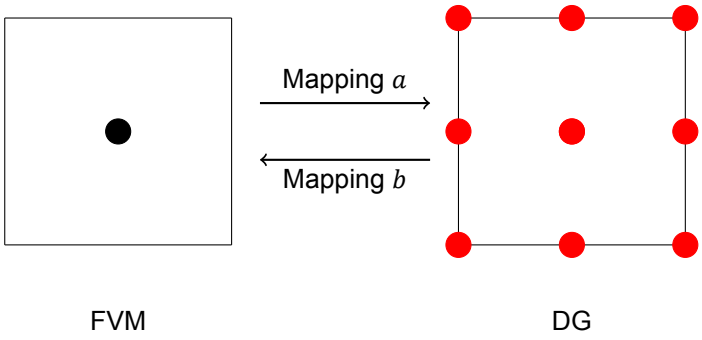


Figure 5.3: A two-dimensional illustration for mappings a and b.

Mappings a

Cell average a

Cell average a is the most simple method that can be used as mapping a. Since the LES filtered variable is the cell average of the grid box, all values of the element are more or less the average. Henceforth, all nodal values of the grid cell are chosen to be the cell average of the grid cell. This can mean that at the element boundaries the function can be discontinuous, which is one of the advantages of DG, as allowing discontinuities gives an extra degree of freedom to find the approximated function.

L_2-projection

Instead of using the cell average for all the nodal values of the element, the nodal values can be calculated by using its neighbours. The interpolation method has to satisfy certain conditions; it has to conserve mass and the nodal values should lay near the cell averages. Therefore, the L_2-projection to the continuous space is chosen.

The L_2- projection φ_{CG} is the optimal solution of:

$$\min \|\varphi_{\text{FVM}} - \varphi_{\text{CG}}\|_{L_2(\Omega)}. \tag{5.1}$$

This can be rewritten to:

$$\int_\Omega \varphi_{\text{CG}}\eta \, d\Omega = \int_\Omega \varphi_{\text{FVM}}\eta \, d\Omega, \ \forall \eta, \tag{5.2}$$

where the test function η is an arbitrary piecewise continuous function.

Recall that in Chapter 4.2.1, the initial condition of DG is similarly calculated. The difference between them is the L_2-projection is done to the continuous space instead of the discontinuous space. When

the FVM variable is projected to the continuous space, there are less degrees of freedom than when it is projected to the discontinuous space; if the FVM variable is L_2-projected to the discontinuous space, it results in the previous mapping, cell average a.

Since the computational domain is partitioned into non-overlapping elements, the integrals over the domain of (5.2) can be written as:

$$\sum_{k:\Omega_k \subset \Omega} \int_{\Omega_k} \varphi_{\text{CG}} \eta \, d\Omega = \sum_{k:\Omega_k \subset \Omega} \int_{\Omega_k} \varphi_{\text{FVM}} \eta \, d\Omega. \tag{5.3}$$

As told above, (5.3) is almost the same as Equation (4.2b), the only difference is that it is solved globally instead of locally per element. Rather than allowing multiple values on the boundaries, φ^{int} and φ^{ext}, there is only one value per node. Thereupon, a global matrix M is assembled using the element matrices M^k.

The first integral of (5.3) can be written as:

$$\sum_{k:\Omega_k \subset \Omega} \int_{\Omega_k} \varphi_{\text{CG}} \phi_i \, d\Omega \approx \sum_{k:\Omega_k \subset \Omega} \sum_{j=1}^{N_p} a(\boldsymbol{x}_j, t) M_{ij}^k, \tag{5.4}$$

and the integral over one element for the cell average as follows:

$$\int_{\Omega_k} \varphi_{\text{FVM}} \phi_i \, d\Omega \approx \frac{1}{8} \Delta x_k \Delta y_k \Delta z_k \omega_{x,i} \omega_{y,i} \omega_{z,i} \varphi_{\text{FVM}}(\Omega_k) = M_{ii}^k \varphi_{\text{FVM}}(\Omega_k). \tag{5.5}$$

All in all, a matrix-vector equation is to be solved:

$$M\boldsymbol{a} = \boldsymbol{\varphi}_{\text{FVM}}, \tag{5.6}$$

where $\boldsymbol{a}$ is the vector of the continuous Galerkin weights and $\boldsymbol{\varphi}_{\text{FVM}}$ represents $\int_\Omega \varphi_{\text{FVM}} \phi_i \, d\Omega$.

For the assembly of the matrix-vector equation, a mapping is needed from our nodal value m of element (i, j, k) to the nodal value in the physical space $h(i, j, k, m)$. Hence, a nodal value (i, j, k, m) does not necessarily have a unique number $h(i, j, k, m)$ (see Figure 5.4). In other words, the mapping is a many-to-one function. With this mapping, every element mass matrix can be assembled in the global mass matrix and $\boldsymbol{\varphi}_{\text{FVM}}$ can be calculated. This goes similarly to the finite element method, see Section 6.2.6 of [31] for more information.

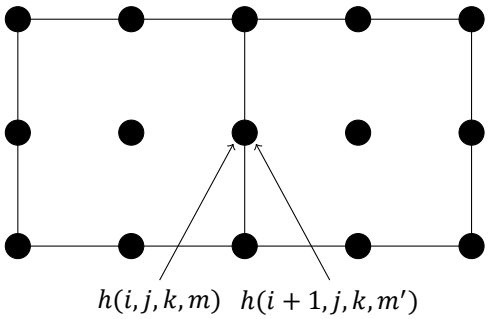


Figure 5.4: Many-to-one mapping from node m of element (i, j, k) and node m' of element $(i + 1, j, k)$ to node $h(i, j, k, m)$ of the domain.

Mappings b
In general, the cell average of element Ω_k of a three-dimensional function $g(\boldsymbol{x})$ is calculated as follows:

$$g_{\Omega_k,\text{FVM}} = \frac{1}{\Delta x \Delta y \Delta z} \int_{\Omega_k} g(\boldsymbol{x})\, d\Omega_k, \tag{5.7}$$

$$= \frac{1}{\Delta x \Delta y \Delta z} \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} g(\boldsymbol{x}(\boldsymbol{\xi}))\frac{1}{8}\Delta x \Delta y \Delta z\, d\boldsymbol{\xi}, \tag{5.8}$$

$$\approx \frac{1}{8}\sum_{m=0}^{N_z}\sum_{j=0}^{N_y}\sum_{i=0}^{N_x} \omega_{z,m}\omega_{y,j}\omega_{x,i} g(\xi_{x,i}, \xi_{y,j}, \xi_{z,m}). \tag{5.9}$$

Here the function g can be φ_{DG} or its tendency. This is how the mapping cell average of tendency works.

For cell average b, extra steps are taken which are shown in Figure 5.5. The time integration step is done by doing one RK3 step and the cell average of φ_{DG} is calculated by (5.9). For the tendency of φ_{FVM}, the difference is taken between the input variable $\varphi_{\text{FVM}}(t)$ and $\varphi_{\text{FVM}}(t + \beta\Delta t)$ with β the RK3 coefficient.

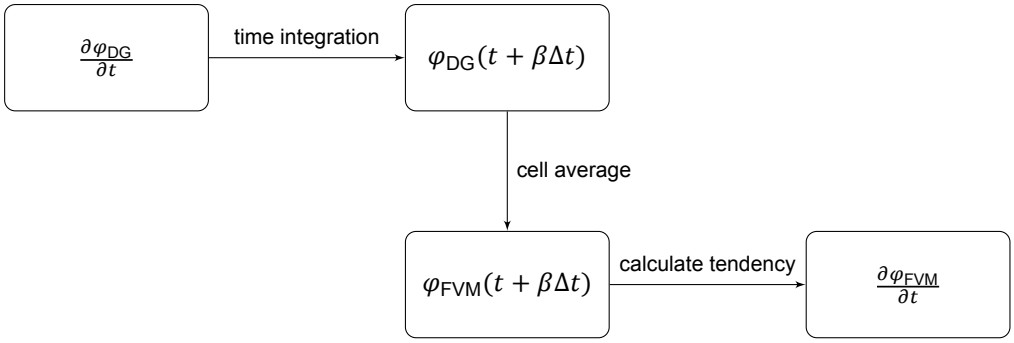


Figure 5.5: Flow chart of method cell average b. β denotes the RK3 coefficient.

Invertible mappings
When mappings are used, the question arises whether the actions are invertible. For mappings a and b, this property is not immediately clear. In this paragraph, it is looked at whether the mappings are invertible.

A mapping is invertible if the function is a bijection, for which the following two conditions must hold:

1. The function is injective.
2. The function is surjective.

In Figure 5.6, the two properties are illustrated.

The mappings a and b are needed to map from a FVM variable to a DG variable and back. A mapping from FVM to DG is surely surjective which means that every FVM variable has a DG variable and the other way around. However, the mapping is not injective; multiple DG variables can be mapped to the same FVM variable since its a cell average of multiple DG values. This means that there exists no invertible mapping from FVM variable to a DG variable. Similarly, there exists no invertible mappings from FVM to CG and from CG to DG.

To sum up, there exist no mappings a and b which are invertible.

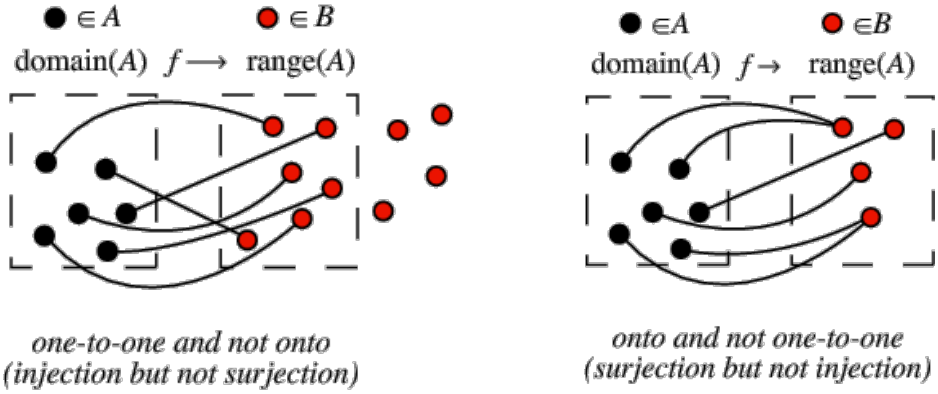


Figure 5.6: Injection and surjection. Images taken from [36] and [37].

5.2. Numerical results of DG

In this section, the numerical results of DG are shown with different combinations of mappings. First of all, it should be mentioned that smaller domains are tested, because there was not enough memory to compute bigger domains after the L_2-projection was implemented.

5.2.1. Cell average a and cell average of tendency

In Figure 5.7 and Figure 5.8, the approximations of φ after 5 s and 25 s are shown. The (x, z)-slices are not shown, since these are same as the (x, y)-slices. The computation took about 2.482×10^{-3} s per time step. One can see that it looks as diffusive as using upwind without DG which is also the case (See Figure 3.3 in [5]). The numerical error in ℓ_2-norm for continuous φ is 2.7909 and 4.3186 for $t = 5$ s and $t = 25$ s respectively. For the discontinuous φ the numerical errors are 3.73844 and 6.1688 respectively. The diffusion could be caused by the mapping from the DALES variable to the DG variable.

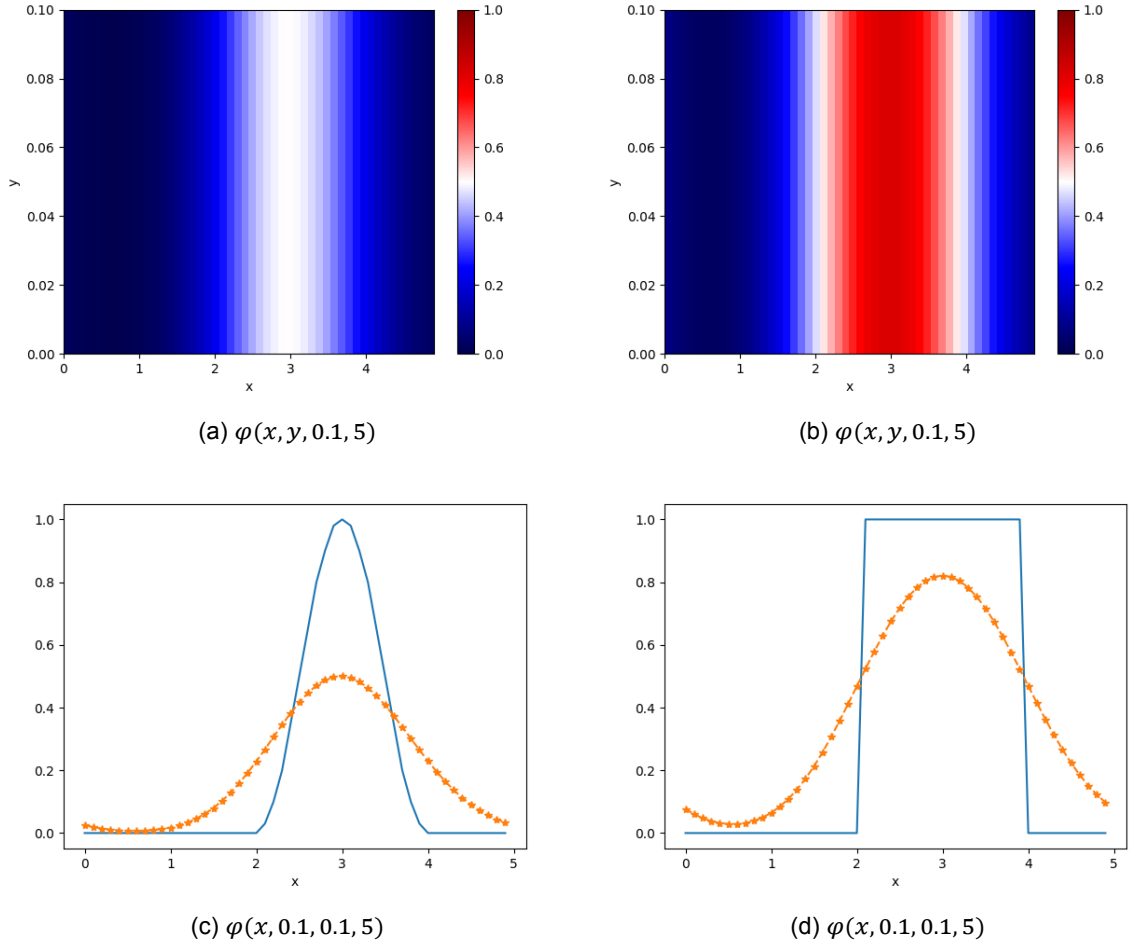
Whereas one could expect, the results are smooth to a degree that there is no effect after applying a limiter. Indeed, it had no effect and the numerical results are exactly the same with identical numerical errors. Except the average computational time of a time step is longer, 3.590×10^{-3} s. The figures can be found in Appendix B.2 as Figures B.2 and B.3.

5.2.2. L_2-projection and cell average of tendency

When the L_2-projection is used instead of cell average a, the diffusion becomes significantly less, see Figure 5.9 and Figure 5.10. To be exact, the method is underdiffusive which means that a lot of oscillations are created. An other striking result is the discontinuity in the z-direction that arose after simulating. Further testing excluded the incorrect implementation of the L_2-projection as the cause of the discontinuity. Most likely, the discontinuity is caused by the boundary condition at the bottom and/or the periodic boundary conditions which use the ghost cells.

Just as with the cell average a method, the limiter has no influence. In both cases of with and without limiter, the numerical errors in ℓ_2-norm for the continuous part and discontinuous of φ are 1.8889 and 3.90622, and 3.2956 and 9.5786 for both $t = 5$ s and $t = 5$ s. However, the computational time with limiter is 3.900×10^{-3} s while it was 2.755×10^{-3} s without limiter.

Another test has been done by adding extra diffusion by using the moment limiter with an other parameter α. The most diffusive $\alpha_n = \frac{1}{2\sqrt{4n^2-1}}$, which is still stable, is chosen. However, this did not have any impact; still the same results are obtained.



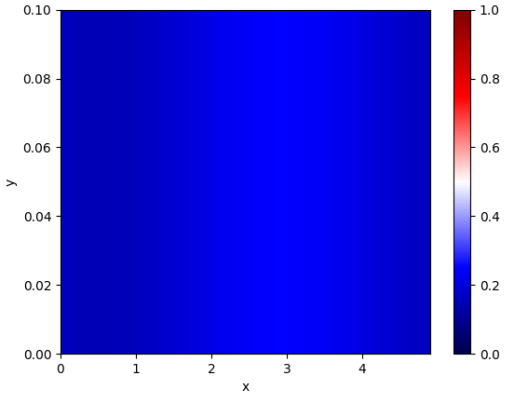
(a) $\varphi(x, y, 0.1, 5)$

(b) $\varphi(x, y, 0.1, 5)$

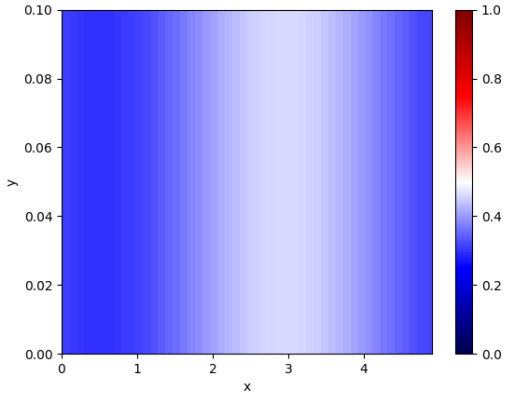
(c) $\varphi(x, 0.1, 0.1, 5)$

(d) $\varphi(x, 0.1, 0.1, 5)$

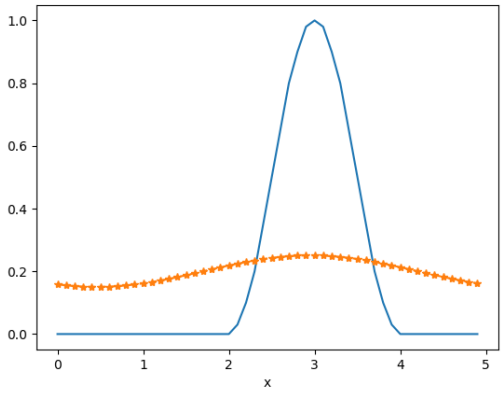
Figure 5.7: Continuous and discontinuous $\varphi(x, y, z, 5)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$, using cell average a and cell average of tendency.



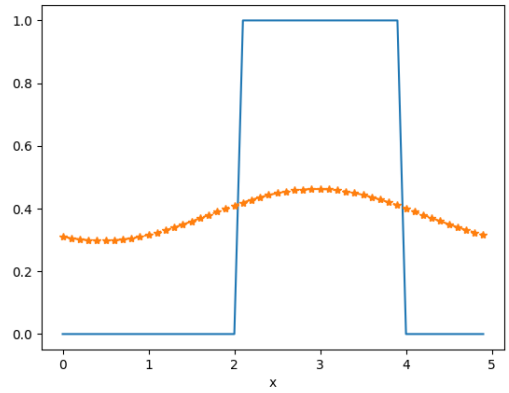
(a) $\varphi(x, y, 0.1, 25)$



(b) $\varphi(x, y, 0.1, 25)$

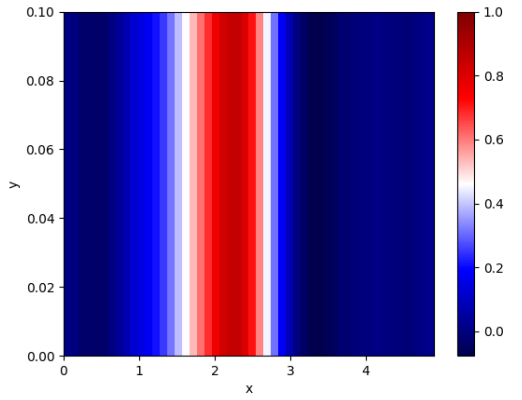


(c) $\varphi(x, 0.1, 0.1, 25)$

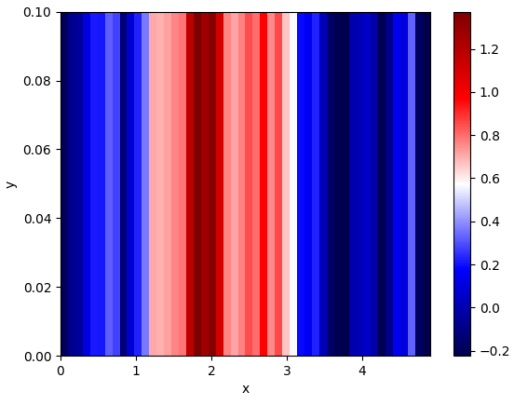


(d) $\varphi(x, 0.1, 0.1, 25)$

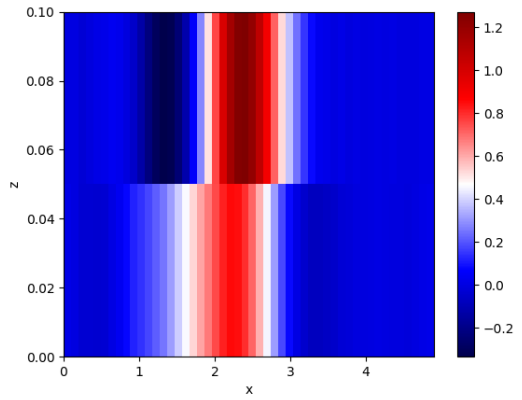
Figure 5.8: Continuous and discontinuous $\varphi(x, y, z, 25)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$, using cell average a and cell average of tendency.



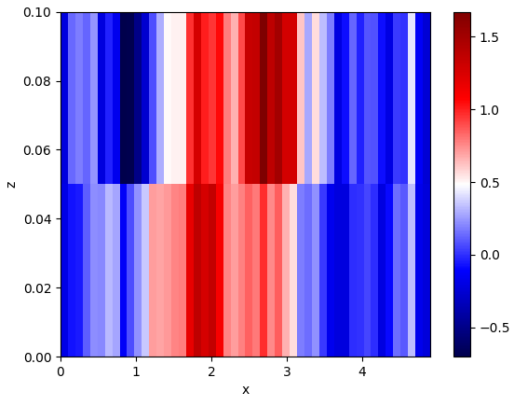
(a) $\varphi(x, y, 0.1, 5)$



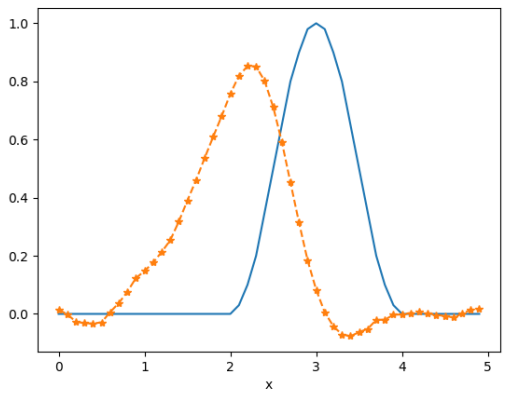
(b) $\varphi(x, y, 0.1, 5)$



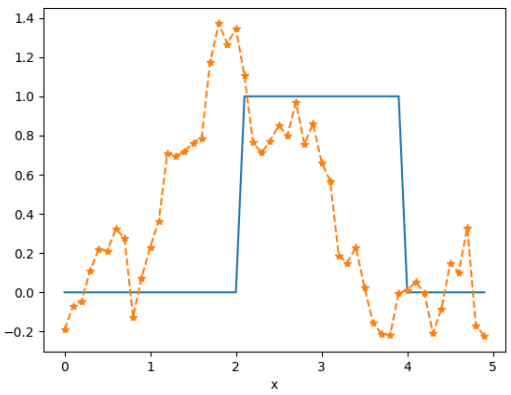
(c) $\varphi(x, 0.1, z, 5)$



(d) $\varphi(x, 0.1, z, 5)$

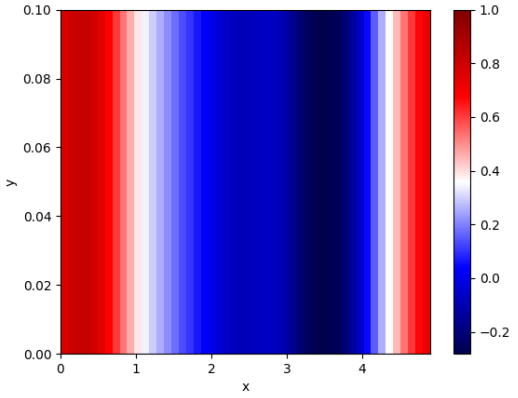


(e) $\varphi(x, 0.1, 0.1, 5)$

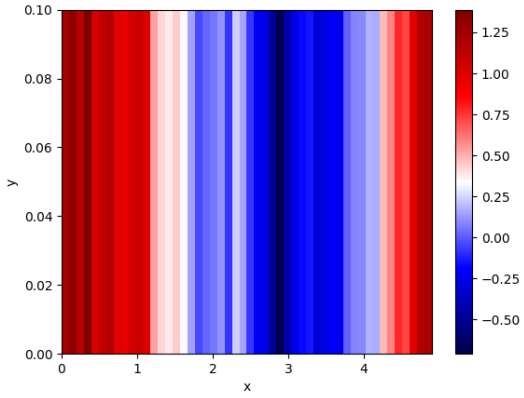


(f) $\varphi(x, 0.1, 0.1, 5)$

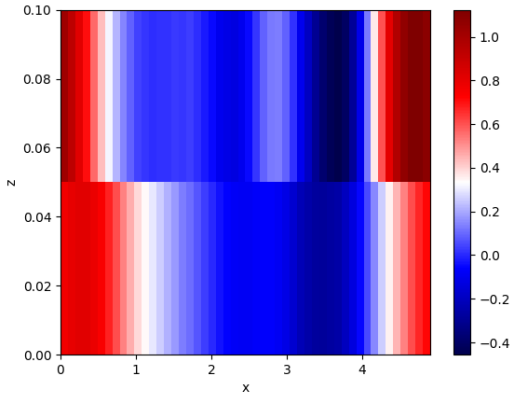
Figure 5.9: Continuous and discontinuous $\varphi(x, y, z, 5)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$, using L_2-projection and cell average of tendency.



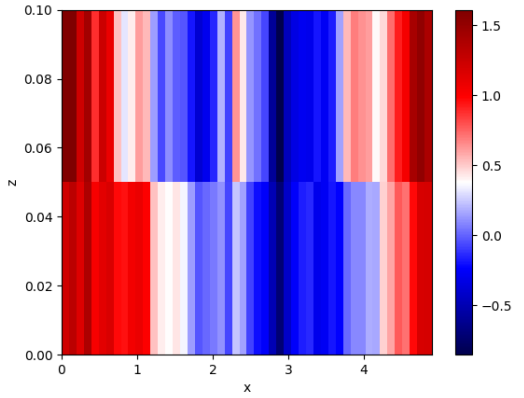
(a) $\varphi(x, y, 0.1, 25)$



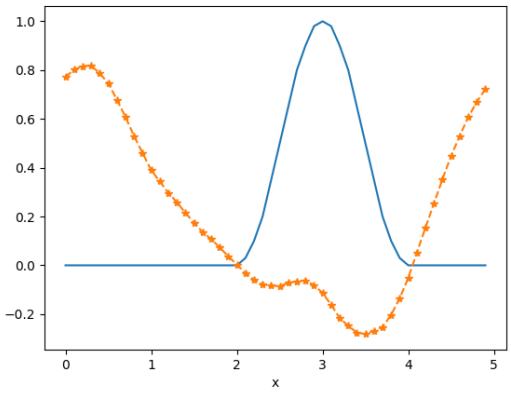
(b) $\varphi(x, y, 0.1, 25)$



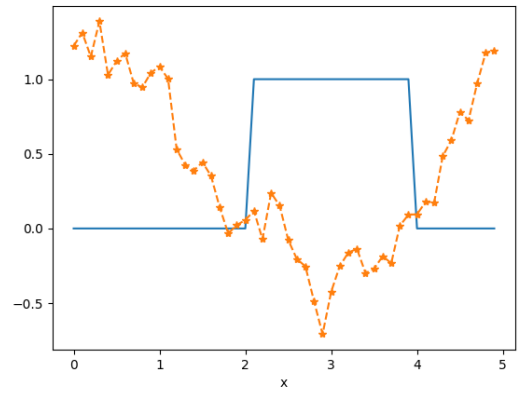
(c) $\varphi(x, 0.1, z, 25)$



(d) $\varphi(x, 0.1, z, 25)$



(e) $\varphi(x, 0.1, 0.1, 25)$



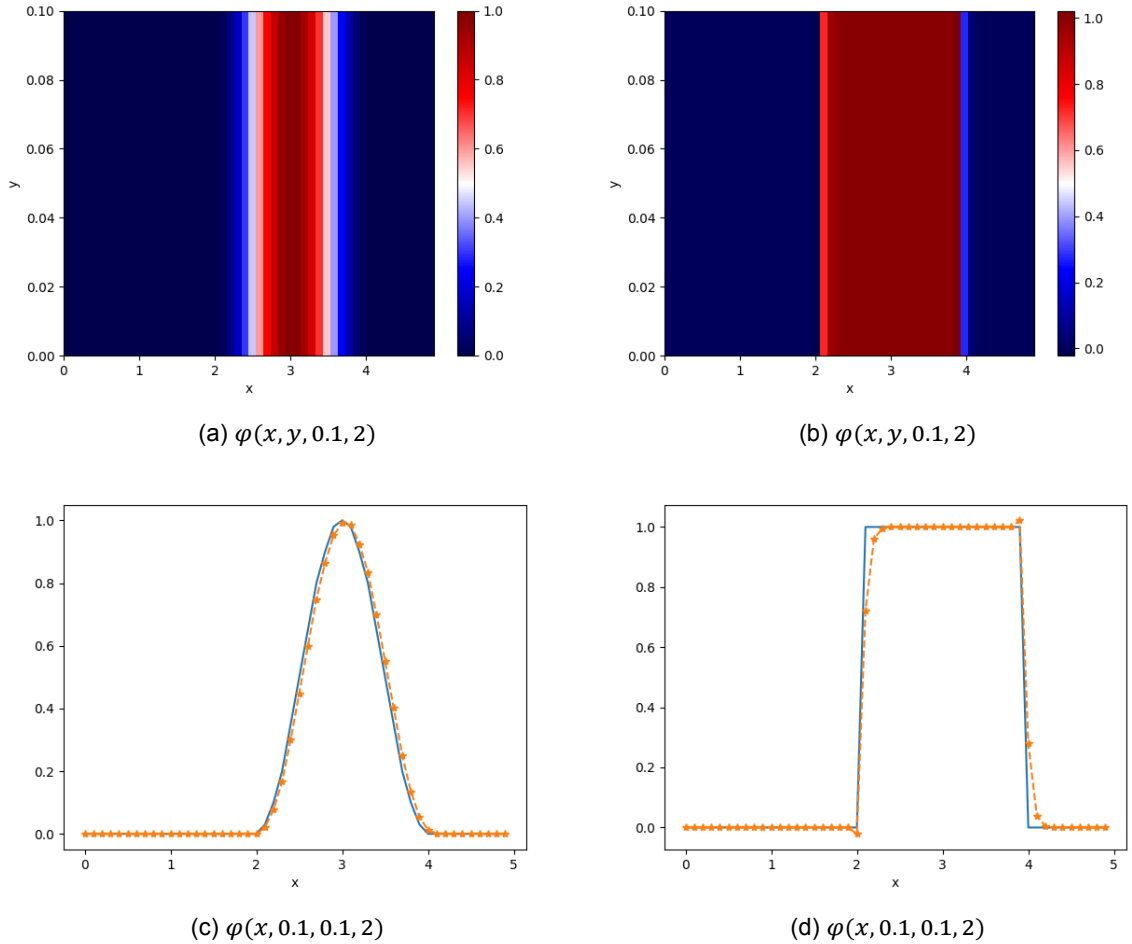
(f) $\varphi(x, 0.1, 0.1, 25)$

Figure 5.10: Continuous and discontinuous $\varphi(x, y, z, 25)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$, using L_2-projection and cell average of tendency.

5.2.3. L_2-projection and cell average b

As KNMI wants an advection scheme with the least diffusion as possible, the L_2-projection is used as mapping a combined with the cell average b to test whether the oscillations are removed.

In Figure 5.11, the results of this method is shown. At first glance, the results are highly improved, however, the results are given for t =2 s and not for t =5 s. The simulation till t =2 s shows that the 'numerical' velocity of this method is much lower than the actual velocity which would not have been clear from the simulations till t =5 s. All in all, this method cannot be used.



(a) $\varphi(x, y, 0.1, 2)$ (b) $\varphi(x, y, 0.1, 2)$

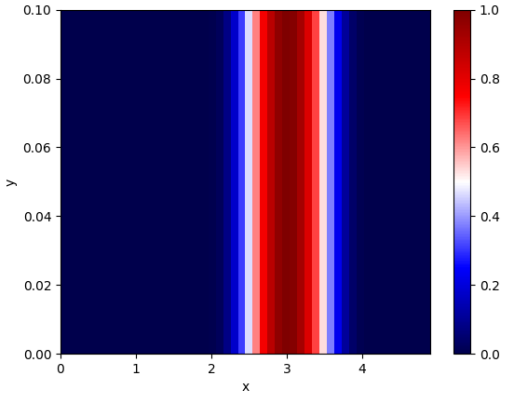
(c) $\varphi(x, 0.1, 0.1, 2)$ (d) $\varphi(x, 0.1, 0.1, 2)$

Figure 5.11: Continuous and discontinuous $\varphi(x, y, z, 2)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$, using L_2-projection and cell average b.

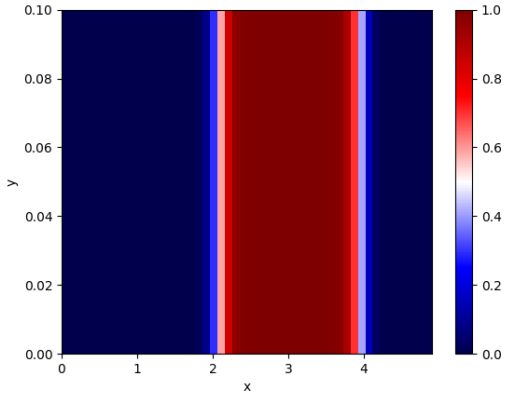
5.3. Numerical results of WENO

The WENO method is the most accurate advection scheme implemented in DALES. Therefore, the DG method will be compared to the fifth order WENO. In the literature study prior to this work [5], more information on WENO can be found, including numerical results for a one-dimensional test case.

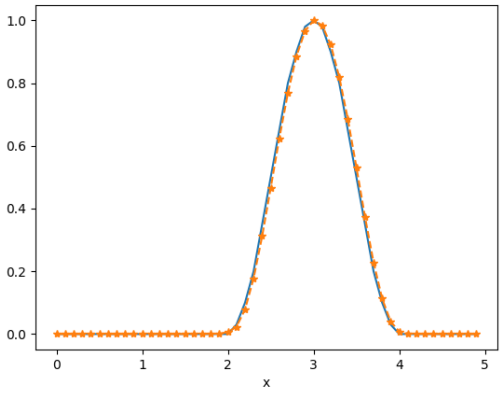
In Figure 5.12, the approximation of $\varphi(\boldsymbol{x}, 5\,\mathrm{s})$ is given. The continuous part is perfectly kept, but some diffusion has taken place around the discontinuous part. If we compare this to DG with limiter outside the LES wrapper (see Figure 4.18), the continuous part is more accurately simulated with WENO while the discontinuous part is less accurate with WENO. In addition, when the limiter is used every RK3 step (see Figure 4.20), there is less diffusion with DG. The numerical errors in ℓ_2-norm are 0.0021 and 0.1167 for the continuous and discontinuous initial condition respectively.



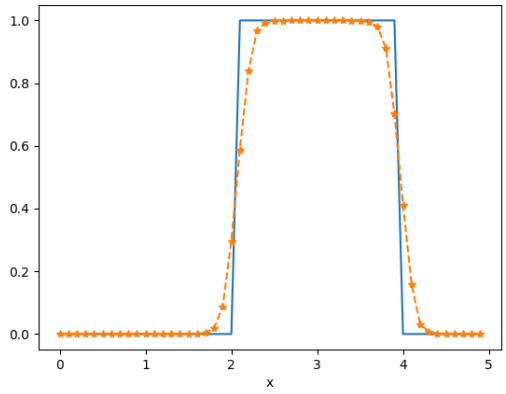
(a) $\varphi(x, y, 0.1, 5)$



(b) $\varphi(x, y, 0.1, 5)$



(c) $\varphi(x, 0.1, 0.1, 5)$



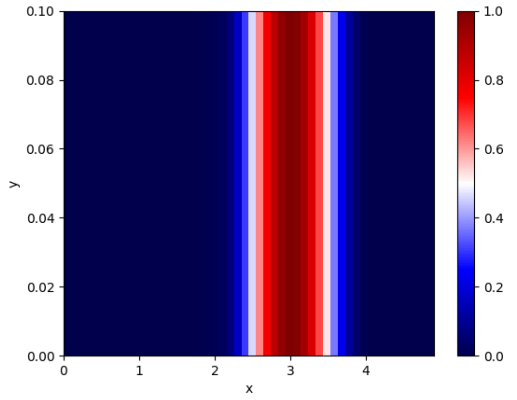
(d) $\varphi(x, 0.1, 0.1, 5)$

Figure 5.12: Continuous and discontinuous $\varphi(x, y, z, 5)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$ using WENO.

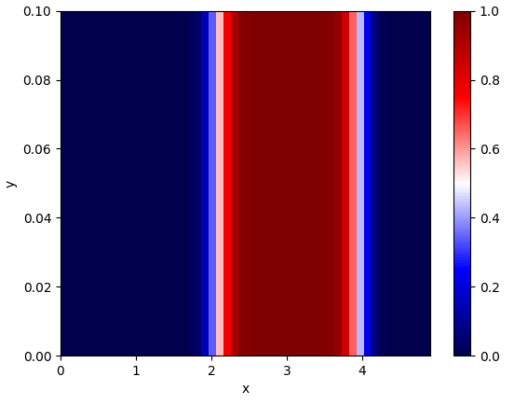
The results of simulating till $t = 25$ s with WENO can be seen in Figure 5.13. Simulating four more periods did not have a lot of effect, like the numerical errors are 0.0024 and 0.1565 for continuous and discontinuous φ. The average computational time step takes 2.110×10^{-4} s. Even though, the numerical results of WENO seems quite accurate. For long term time simulations, the advection scheme has more diffusion than wanted, which can be seen in Figure B.4. In addition, the comparison of WENO and DG method outside the LES wrapper can be found in Appendix B.3.

5.4. Conclusion

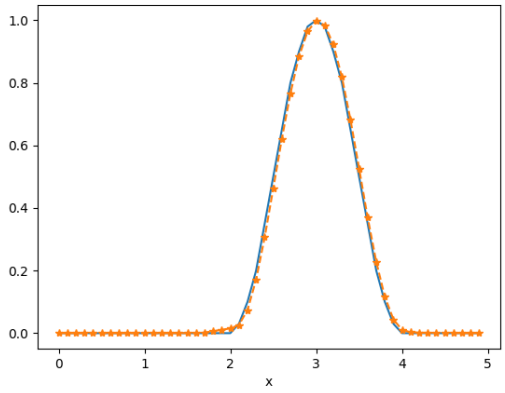
All in all, in the LES wrapper the WENO method gives much better numerical results than the DG method. On top of that, the computational time of the WENO method, without parallelization, is 10 times less (see Table 5.1). This was to be expected, since DG solved 27 nodal values per element instead of one value per element. The tested mappings that are needed for the implementation in DALES, do not give accurate results. While the combination cell average a and cell average of the tendency has too much diffusion, the combination, L_2-projection and cell average of tendency, is underdiffusive. The combination L_2-projection and cell average b has a very long delay in time, therefore, it is a fruitless method. Since the methods are less accurate than WENO, we did not test them in DALES. However, when the DG method does not need any mapping from FVM variable to DG variable and back, the DG method is a very good opponent of the WENO method with good perspectives for the future.



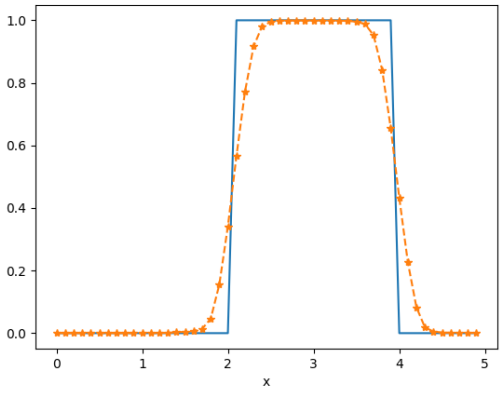
(a) $\varphi(x, y, 0.1, 25)$



(b) $\varphi(x, y, 0.1, 25)$



(c) $\varphi(x, 0.1, 0.1, 25)$



(d) $\varphi(x, 0.1, 0.1, 25)$

Figure 5.13: Continuous and discontinuous $\varphi(x, y, z, 25)$ with $\Delta x = \Delta y = \Delta z = 0.1$, $\Delta t = 2 \cdot 10^{-2}$ using WENO.

Method	Comp. time	Run time	Continuous	Discontinuous
Cell average a &	$2.482 \cdot 10^{-3}$s	5 s	2.7909	3.7384
Cell average of tendency		25 s	4.3186	6.1688
L_2-projection &	$2.755 \cdot 10^{-3}$s	5 s	1.8889	3.2956
Cell average of tendency		25 s	3.9062	9.5786
WENO	$2.114 \cdot 10^{-4}$s	5 s	0.0021	0.1167
		25 s	0.0024	0.1565

Table 5.1: Each method with its average computational time per time step and numerical errors in ℓ_2-norm for continuous and discontinuous initial condition. Note that the WENO method is 10 times faster than the other two methods.

6

Conclusion and further recommendations

This chapter concludes this thesis. First, each subject of the thesis is summarized and concluded. Besides, some implemenation problems are addressed. Thereafter, the results of the thesis are discussed and suggestions are made for further research.

6.1. Conclusion

The aim of this thesis was the development of a fast and accurate advection scheme for the large-eddy simulation, DALES. Therefore, the discontinuous Galerkin (DG) method is proposed and the corresponding research question of the thesis was:

> Can DG be used as an advection solver in DALES such that:
>
> - the computational time is less than the WENO method, and/or
> - the numerical accuracy is better than the WENO method, while the computational time is not doubled?

This section summarizes the results and concludes each subject this thesis encountered. Last but not least, the research question is answered.

The problems of numerical weather prediction and climate models are mathematically modelling atmospheric processes and to evaluate the models as accurate and efficient as possible. One of these atmospheric processes that could be solved more accurately and efficiently is the advection. Advection is one of the most important processes that takes place in the atmosphere. The Dutch Atmospheric Large-Eddy Simulation (DALES) that is used, among other things, for parametrization development, still has problems with its advection schemes. Therefore, the DG method is suggested which is known for its high scalability, geometric flexibility and allowance of discontinuous approximations.

In Chapter 3, the important numerical methods for the advection equation of DALES are discussed. The grid spacing and time integration method with its corresponding time step constraints are mentioned. Thereafter, the shortcomings of the implemented advection schemes of DALES are named. The most accurate advection scheme is the WENO method, which is still too diffusive when steep gradients are present. Moreover, it has the longest computation time.

The DG method is a combination of the finite element method and the finite volume method. This method solves the weak form of the differential equations instead of solving the differential equations like the finite difference method does. The idea of DG works well for smooth applications, DG shows no problems like time lags or diffusion. However, at discontinuities there are non-physical oscillations in the approximations. This holds for one-, two- and three-dimensional DG methods. The dispersion can be resolved by using a limiter, for example the moment limiter. In this thesis, the moment limiter

is extended to three-dimensional problems on tensor-product meshes. Like most limiters, the tested moment limiter removes the dispersion errors, but adds too much artificial diffusion which reduces the order of the method. Limiting is normally done after each time step, but can also be done multiple times in every time step. In Chapter 4 tests are done in which the moment limiter is used after every RK3 step. The effect of limiting more often is very minimal. The numerical accuracy is slightly better, but the computational time is also slightly longer.

In Chapter 5, the DG method is tested in a stripped version of DALES, the LES wrapper, such that only the advection scheme is tested. When the DG method can be used in DALES, mappings are needed from the cell average, like the finite volume (FVM) variable, to the DG variable and vice versa. Several mappings are tested, however, none of the methods obtained accurate results like the numerical results that were assessed outside the LES wrapper. Therefore, the DG advection scheme has not been tested within DALES. Moreover, the DG method was, as expected, ten times more computational expensive than the WENO method in a non-parallel environment.

All in all, from Chapter 4 it can be concluded that DG with moment limiter is a quite accurate numerical method to solve the advection equation. Thus, DG can be used as an advection solver in DALES, however, when mappings are needed from a FVM variable to a DG variable and vice versa, DG does not have many more benefits than WENO has. To answer the research question, DG can be used as an advection solver in DALES. However, the numerical accuracy is not improved when these mappings are used. Moreover, the computational time did not decrease compared to WENO in a non-parallel environment, which we do expect in a parallel environment since DG method is known for its high scalability. Nevertheless, we suspect that DG is the future for the numerical weather prediction models. Nowadays we see more computers with a many-core architecture and shared memory which is perfect for DG with its high scalability due to a compact stencil. Moreover, there are a lot of options, which are not investigated in this thesis, to tune DG. In Section 6.3, several recommendations are given and one in particular that works around the problem with the mappings from FVM to DG, which also reduces the computational and memory cost.

6.2. Implementation problems

During this project, complications arose during the implementation of DG in the LES wrapper.

First of all, we had chosen to not assemble a global matrix, but to solve a linear system of equations per element. The global matrices are more sparse than the element matrices. However, we chose not to use global matrices, because we thought that parallelization can be done without extra changes.

As the boundary conditions are solved outside the advection scheme by using ghost cells, all element matrices are stored by every x_i-, y_j- and z_k-coordinate. The program worked, however, it was extremely slow, more than 100 times slower than the DG method outside the LES wrapper. With the use of profiling, we found out that more time was spent in the sparse library than outside the LES wrapper. Since the DG method outside the LES wrapper was implemented and stored for an element number instead needing the three numbers i, j, k, we adjusted the DG method in the LES wrapper such that the element number was used. This decreased the computational time by 10%, nonetheless, outside the LES wrapper, the method was much faster. With the advice of Fredrik Jansson, we have chosen to drop the sparse matrices and implemented the element stiffness matrices with full matrices. As a result, the method was much faster and even faster than the DG method outside the LES wrapper which still used the sparse matrix library of Fortran.

Secondly, when we checked whether the advection scheme was compliant with DALES, we remembered that for the boundary conditions i, j, k are needed instead of the element number. Consequently, the method had again to be adjusted to take care of this.

6.3. Further remarks and recommendations

In this section, some obtained results are criticized and suggestions for further research are given.

6.3.1. Remarks

First of all, the obtained time shifts mentioned in Section 3.3, are questionable since the time shift did not occur in DALES, see Chapter 5. One would think that in the one-dimensional test case the time simulation was not stopped at the exact end time. However, this has been checked multiple times and the time integration method was exactly the same as DG had been tested with, which did not have any time shift. In other words, the time integration method is not the cause of the time shifts. Further research is needed to establish the source of the time lags.

In Chapter 4.3, we have chosen to use inexact integration for two reasons. First of all, one can easily change the polynomial degrees of the basis functions. Second of all, the element matrices can easily be derived and the element mass matrix becomes a diagonal matrix. The numerical results of DG with moment limiter show better results for inexact integration after a longer time simulation, but without moment limiter, exact integration gives less dispersion than when inexact integration is used. Nevertheless, there has not been done further research on the effects of exact and inexact integration.

In addition, the test cases of this thesis were done with constant velocity. In this case, there are no differences between the upwind flux and Lax-Friedrichs flux, nonetheless, this can be the case when a space-dependent velocity $\boldsymbol{u}(\boldsymbol{x})$ is used. Therefore, more research should be done on this topic.

Moreover, by accident some results are made with a wrongly implemented limiting order of the three-dimensional moment limiter (see Appendix B.2). These results showed that the limiting order is very important. It would be interesting to see whether the numerical results can be improved by adjusting the limiting order.

On top of that, the moment limiter does not make the DG method flawless. Even in smooth regions, peak clipping takes place, since the moment limiter limits while it is not needed. This can be solved by using a shock detection method. For example, by using the troubled-cell indicator of Vuik [34]. The method can detect discontinuous regions where limiting is needed, which reduces the computational cost. However, for three-dimensional non-uniform tensor meshes, which can be used within DALES, more research is needed on this topic before the indicator can be used in DALES.

The computation time is another important property for the advection scheme of DALES. The DG method in the LES wrapper was ten times slower than the WENO method, however, the DG method could be faster when a global sparse matrix is assembled instead of solving the equations per element. Recall that this had not been chosen to allow parallelization without changing the code. Moreover, tests should be done with parallelization before it can be concluded that DG is slower than WENO.

6.3.2. Recommendations for future work

As one can notice, when DG is implemented in DALES, many extra information is created and is later on reduced to one value (See Figure 5.3). Consequently, the computational cost becomes higher without the advantage of being able to use the extra nodal values in other subroutines which then need to be interpolated. An other drawback is the extra time integration step that has to be made before the limiter can be applied, while the time integration is done later on in the program. Consequently, it can be looked at to limit the tendency instead φ itself.

Another option is to recreate the DALES model with the DG method as a basis. As a result, no unnecessary extra information is created. Obviously, this would cost a lot of time to do since a lot of different processes are added on top of the four conservation laws discussed in Chapter 2.

Nevertheless, Chapter 5 describes that a DALES cell is chosen as one DG cell. However, multiple DALES grid boxes can be used as one DG cell as was discussed in Chapter 5 of the literature study prior to this work [5]. Consequently, the computational and memory cost can be reduced. An example of a DG cell in two-dimensions can be seen in Figure 6.1. This subject will be picked up in the near future.

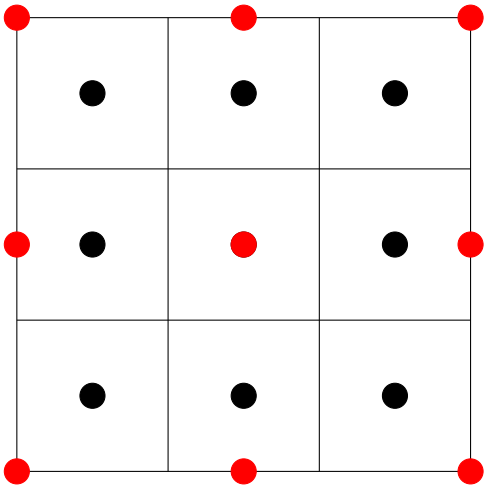


Figure 6.1: Nine DALES cells as one DG cell in two-dimensions. The red points are the DG nodal points and the black points are the centres of the DALES cells. Note that the centre point is both a DG and DALES nodal point.

A

Element matrices

In this appendix, some more information is given on the derivation of element matrices. First, the derivation of the fluxes in one-dimension is shown when the Lax-Friedrichs flux is used instead of the upwind flux. Second, the complete derivation of the element stiffness matrix for DALES in two-dimensions is given.

A.1. Lax-Friedrich's flux for one-dimensional advection equation

The element fluxes are derived from the following:

$$[f(\varphi)\eta]_{x_{k-1/2}}^{x_{k+1/2}} = \hat{F}_{k+1/2}\ell_i(x_{k+1/2}^-) - \hat{F}_{k-1/2}\ell_i(x_{k-1/2}^+), \forall i \in \{0, \dots, N\}, \tag{A.1}$$

where the numerical flux $\hat{F}_{k\pm1/2}$ is given by the local Lax-Friedrich's flux.

The local Lax-Friedrich's flux is calculated by:

$$\hat{F}_{k-1/2} = \frac{1}{2}\left(f(\varphi_{h,k-1/2}^-) + f(\varphi_{h,k-1/2}^+) - \alpha\left[\varphi_{h,j-1/2}^+ - \varphi_{h,j-1/2}^-\right]\right), \tag{A.2}$$

where

$$\alpha_{k-1/2} = \max(|f'(\varphi_{k-1/2}^-)|, |f'(\varphi_{k-1/2}^+)|) = \max(|u_{k-1/2}|, |u_{k-1/2}|) = |u_{k-1/2}|, \tag{A.3}$$

because the definition of our flux function is $f(\varphi) = u\varphi$.

The numerical flux contributes only at the element boundaries, in other words, for $i = 0$ and $i = N$. For the left boundary, $x = x_{k-1/2}$, the corresponding numerical flux is:

$$\hat{F}_{k-1/2} = \frac{1}{2}\left(u_{k-1/2}a_N^{k-1} + u_{k-1/2}a_0^k - |u_{k-1/2}|\left[a_0^k - a_N^{k-1}\right]\right), \tag{A.4}$$

and for the right boundary, $x = x_{k+1/2}$:

$$\hat{F}_{k+1/2} = \frac{1}{2}\left(u_{k+1/2}a_N^k + u_{k+1/2}a_0^{k+1} - |u_{k+1/2}|\left[a_0^{k+1} - a_N^k\right]\right). \tag{A.5}$$

Thus, also (A.1) is only non-zero when $i = 0$ and $i = N$:

$$0 - \hat{F}_{k-1/2}\cdot 1 = -\frac{1}{2}u_{k-1/2}a_N^{k-1} - \frac{1}{2}u_{k-1/2}a_0^k + \frac{1}{2}|u_{k-1/2}|a_0^k - \frac{1}{2}|u_{k-1/2}|a_N^{k-1}, \tag{A.6}$$

$$\hat{F}_{k+1/2}\cdot 1 - 0 = \frac{1}{2}u_{k+1/2}a_N^k + \frac{1}{2}u_{k+1/2}a_0^{k+1} - \frac{1}{2}|u_{k+1/2}|a_0^{k+1} + \frac{1}{2}|u_{k+1/2}|a_N^k. \tag{A.7}$$

All in all, the matrix-vector equation of Equation (A.1) is:

$$\begin{pmatrix} 0 & \dots & 0 & F_{km,0N} \\ \vdots & \ddots & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & 0 \end{pmatrix}\begin{pmatrix} a_0^{k-1} \\ a_1^{k-1} \\ \vdots \\ a_N^{k-1} \end{pmatrix} + \begin{pmatrix} F_{k,00} & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & F_{k,NN} \end{pmatrix}\begin{pmatrix} a_0^k \\ a_1^k \\ \vdots \\ a_N^k \end{pmatrix} + \begin{pmatrix} 0 & \dots & \dots & 0 \\ \vdots & \ddots & & \vdots \\ 0 & & \ddots & \vdots \\ F_{kp,N0} & \dots & \dots & 0 \end{pmatrix}\begin{pmatrix} a_0^{k+1} \\ a_1^{k+1} \\ \vdots \\ a_N^{k+1} \end{pmatrix}, \tag{A.8}$$

where the elements are given by:

$$F_{km,0N} = -\frac{1}{2}u_{k-1/2} - \frac{1}{2}|u_{k-1/2}|, \tag{A.9}$$

$$F_{k,00} = -\frac{1}{2}u_{k-1/2} + \frac{1}{2}|u_{k-1/2}|, \tag{A.10}$$

$$F_{k,NN} = \frac{1}{2}u_{k+1/2} + \frac{1}{2}|u_{k+1/2}|, \tag{A.11}$$

$$F_{kp,N0} = \frac{1}{2}u_{k+1/2} - \frac{1}{2}|u_{k+1/2}|. \tag{A.12}$$

A.2. Derivation of the element stiffness matrix for DALES in two-dimensions

The element stiffness matrix of Equation (4.26) is defined by:

$$S_{ij}^k = \int_{\Omega_k} \phi_j(\boldsymbol{\xi}(\boldsymbol{x}))\Bigg(u(\boldsymbol{x},t)\frac{\partial\phi_i(\boldsymbol{\xi}(\boldsymbol{x}))}{\partial\xi_x}\frac{2}{\Delta x_k} + \frac{w(\boldsymbol{x},t)}{\rho(z)}\Bigg(\rho(z)\frac{\partial\phi_i(\boldsymbol{\xi}(\boldsymbol{x}))}{\partial\xi_z}\frac{2}{\Delta z_k} - \phi_i(\boldsymbol{\xi}(\boldsymbol{x}))\frac{\partial\rho(z)}{\partial z}\Bigg)\Bigg)\,d\Omega, \tag{A.13}$$

$$= \int_{-1}^{1}\int_{-1}^{1} \phi_j(\boldsymbol{\xi})\Bigg(u(\boldsymbol{x}(\boldsymbol{\xi}),t)\frac{\partial\phi_i(\boldsymbol{\xi})}{\partial\xi_x}\frac{2}{\Delta x_k} + \frac{w(\boldsymbol{x}(\boldsymbol{\xi}),t)}{\rho(z(\xi_z))}\Bigg(\rho(z(\xi_z))\frac{\partial\phi_i(\boldsymbol{\xi})}{\partial\xi_z}\frac{2}{\Delta z_k} - \phi_i(\boldsymbol{\xi})\frac{\partial\rho(z(\xi_z))}{\partial\xi_z}\Bigg)\Bigg)\frac{1}{4}\Delta x_k\Delta z_k\,d\xi_x\,d\xi_z, \tag{A.14}$$

$$\approx \sum_{p=0}^{N_x}\sum_{q=0}^{N_z}\frac{1}{4}\Delta x_k\Delta z_k\omega_{x,p}\omega_{z,q}\phi_j(\xi_{x,p},\xi_{z,q})\Bigg[u(\boldsymbol{x}(\xi_{x,p},\xi_{z,q}),t)\frac{\partial\phi_i(\xi_{x,p},\xi_{z,q})}{\partial\xi_x}\frac{2}{\Delta x_k} + \frac{w(\boldsymbol{x}(\xi_{x,p},\xi_{z,q}),t)}{\rho(z(\xi_{z,q}))}\Bigg(\rho(z(\xi_{z,q}))\frac{\partial\phi_i(\xi_{x,p},\xi_{z,q})}{\partial\xi_z}\frac{2}{\Delta z_k} - \phi_i(\xi_{x,p},\xi_{z,q})\frac{\partial\rho(z(\xi_{z,q}))}{\partial\xi_z}\Bigg)\Bigg]. \tag{A.15}$$

Since for the basis functions holds $\phi_j(\xi_{x,p},\xi_{z,q}) = 1$ if $p = j_x$ and $q = j_z$, the element matrix can be written as:

$$S_{ij}^k \approx \frac{1}{4}\Delta x_k\Delta z_k\omega_{x,j_x}\omega_{z,j_z}\Bigg(u(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)\frac{\partial\phi_i(\xi_{j_x},\xi_{j_z})}{\partial\xi_x}\frac{2}{\Delta x_k} + \frac{w(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)}{\rho(z(\xi_{z,j_z}))}\Bigg[\rho(z(\xi_{j_z}))\frac{\partial\phi_i(\xi_{x,j_x},\xi_{z,j_z})}{\partial\xi_z}\frac{2}{\Delta z_k} - \phi_i(\xi_{x,j_x},\xi_{z,j_z})\frac{\partial\rho(z(\xi_{z,j_z}))}{\partial z}\Bigg]\Bigg). \tag{A.16}$$

Recall that we have defined the basis function ϕ_j by the tensor-product of the one-dimensional Lagrange polynomials:

$$\phi_j(x,z,t) = \ell_{x,j}(x)\ell_{z,j}(z). \tag{A.17}$$

As a result, Equation (A.16) can be reduced to:

$$S_{ij}^k \approx \frac{1}{4}\Delta x_k\Delta z_k\omega_{x,j_x}\omega_{z,j_z}\Bigg(u(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)\frac{\partial\ell_{x,i}(\xi_{j_x})}{\partial\xi_x}\ell_{z,i}(\xi_{j_z})\frac{2}{\Delta x_k} + \frac{w(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)}{\rho(z(\xi_{z,j_z}))}\Bigg[\rho(z(\xi_{j_z}))\frac{\partial\ell_{z,i}(\xi_{j_z})}{\partial\xi_z}\ell_{x,i}(\xi_{j_x})\frac{2}{\Delta z_k} - \phi_i(\xi_{j_x},\xi_{j_z})\frac{\partial\rho(z(\xi_{j_z}))}{\partial z}\Bigg]\Bigg), \tag{A.18}$$

$$= \frac{1}{4}\Delta x_k\Delta z_k\omega_{x,j_x}\omega_{z,j_z}\Bigg(\frac{2}{\Delta x_k}u(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)\frac{\partial\ell_{x,i}(\xi_{j_x})}{\partial\xi_x}\delta_{i_z,j_z} + \frac{w(\boldsymbol{x}(\xi_{x,j_x},\xi_{z,j_z}),t)}{\rho(z(\xi_{z,j_z}))}\Bigg[\frac{2}{\Delta z_k}\rho(z(\xi_{j_z}))\frac{\partial\ell_{z,i}(\xi_{j_z})}{\partial\xi_z}\delta_{xi,j_x} - \delta_{i,j}\frac{\partial\rho(z(\xi_{j_z}))}{\partial z}\Bigg]\Bigg). \tag{A.19}$$

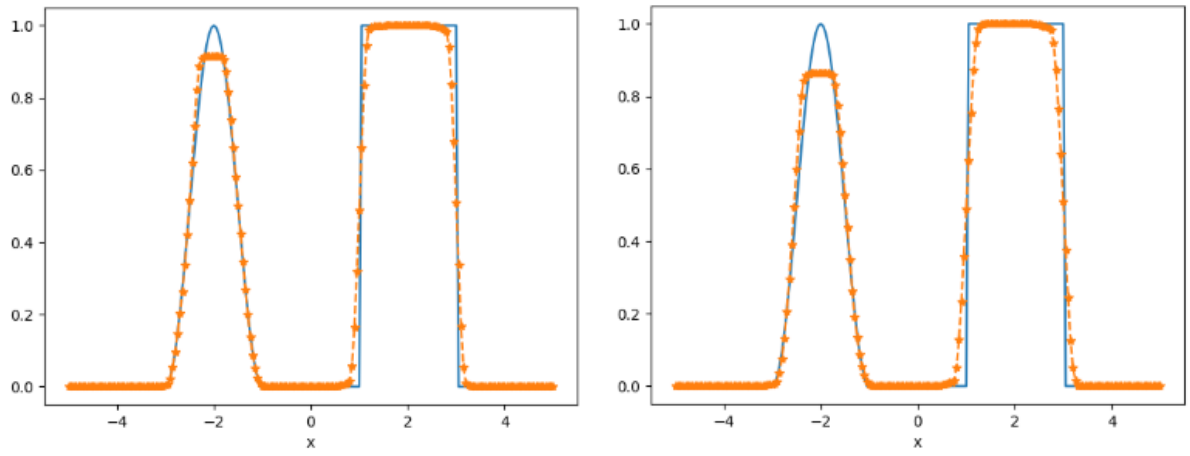
B

Extra figures

In this appendix, extra figures are shown which could be interesting for the reader.

B.1. Wrong limiting order

In this section, the limiting order was not correctly implemented. Here coefficient $\hat{a}^k_{1,1,1}$ was limited before the coefficients $\hat{a}^k_{2,0,0}$, $\hat{a}^k_{0,2,0}$ and $\hat{a}^k_{0,0,2}$ instead the other way around, which was the only difference with the correct limiting order. We suspect that the limiter has been stopped after limiting $\hat{a}^k_{1,1,1}$ meaning that the coefficients $\hat{a}^k_{2,0,0}$ were not limited, leading to more peak clipping than with the correct limiting order (see Figures B.1).



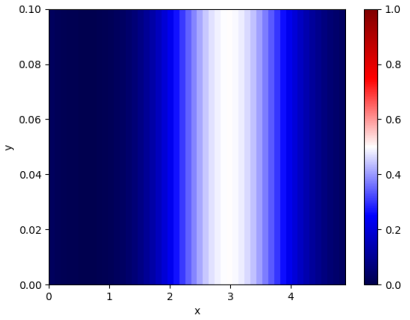
(a) t =10 s.

(b) t =50 s.

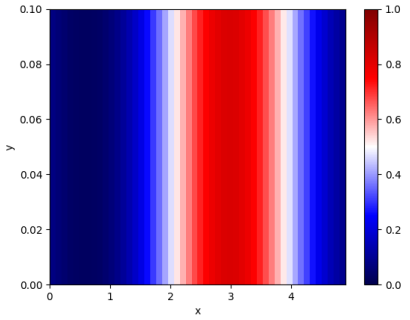
Figure B.1: $\varphi(x,-5,0,t)$ with $K_x = 100$, $K_y = K_z = 2$ when limiter with the wrong limiting order is used, $\Delta t = 2.089 \cdot 10^{-2}$.

B.2. DG in DALES

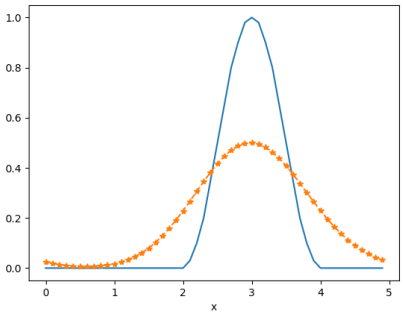
In Figures B.2 and B.3, the numerical results are shown when the DG method with limiter is used with the combination cell average a and cell average of tendency as mappings a and b. These figures are shown to confirm that limiting has indeed no effect.



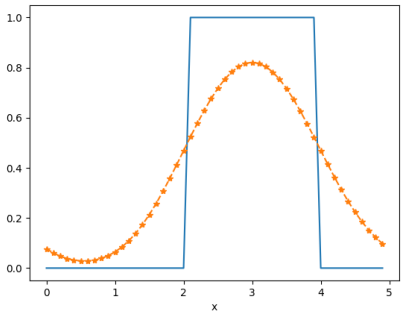
(a) $\varphi(x, y, 0.1, 5)$



(b) $\varphi(x, y, 0.1, 5)$

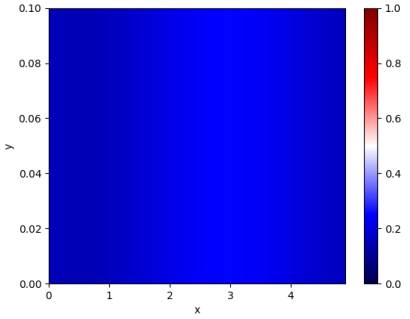


(c) $\varphi(x, 0.1, 0.1, 5)$

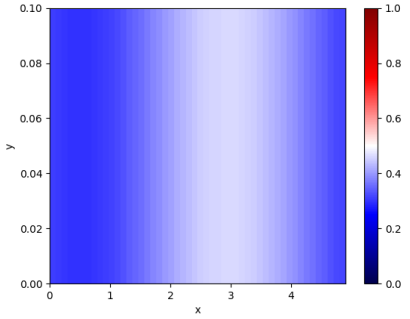


(d) $\varphi(x, 0.1, 0.1, 5)$

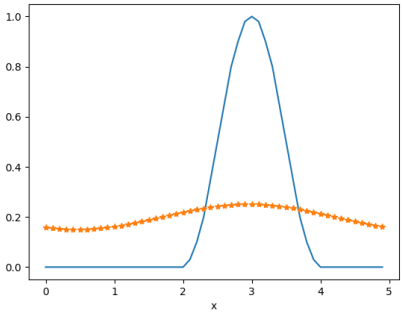
Figure B.2: Continuous and discontinuous $\varphi(x, y, z, 5)$ with $K_x = 100$, $K_y = K_z = 2$, $\Delta t = 2 \cdot 10^{-2}$, using cell average a, cell average of tendency and the moment limiter.



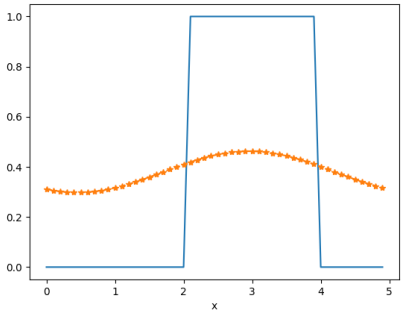
(a) $\varphi(x, y, 0.1, 25)$



(b) $\varphi(x, y, 0.1, 25)$



(c) $\varphi(x, 0.1, 0.1, 25)$

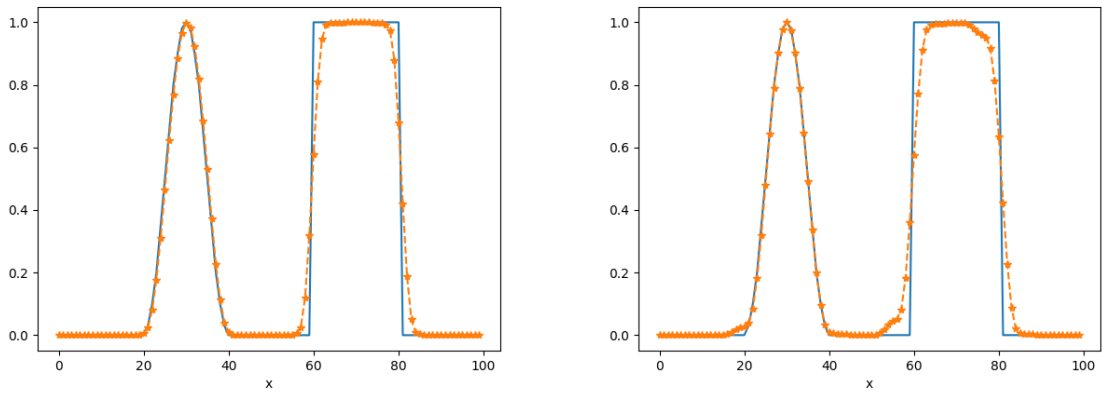


(d) $\varphi(x, 0.1, 0.1, 25)$

Figure B.3: Continuous and discontinuous $\varphi(x, y, z, 25)$ with $K_x = 10$, $K_y = K_z = 2$, $\Delta t = 2 \cdot 10^{-2}$, using cell average a, cell average of tendency and the moment limiter.

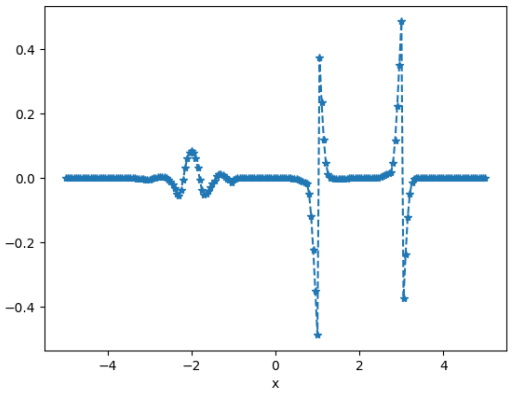
B.3. DG versus WENO

For the comparison of DG and WENO, we wanted to compare the numerical results of DG, see Figures 4.20 and 4.21, with WENO. In Figure B.4, these results of WENO are shown. Figure B.5 shows the difference $\varphi_0 - \varphi(t)$ where $\varphi(t)$ is approximated by the two methods. From these figures, we can clearly see that the WENO method can approximate continuous functions better than the DG method. Nevertheless, for discontinuous functions the DG method is slightly better especially for longer time-simulations. Note that at the discontinuous part, the error of DG does not seem to to evolve in time. In other words, for longer time periods discontinuous functions are approximated better by the DG method.

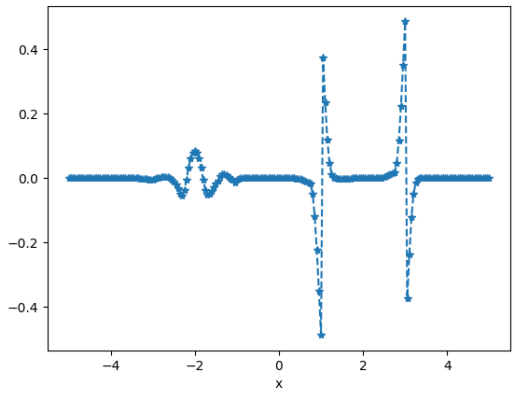


(a) t =25 s (b) t =50 s

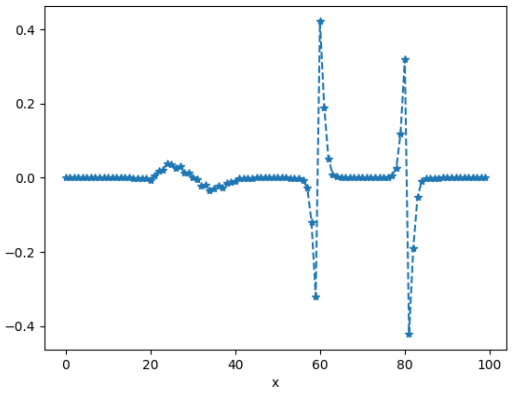
Figure B.4: $\varphi(x, 0, 0, t)$ with $K_x = 100$, $K_y = K_z = 2$, $\Delta t = 2 \cdot 10^{-2}$ using WENO.



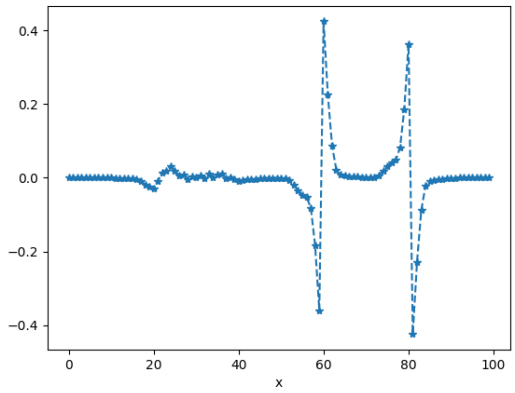
(a) DG at t =25 s, using the moment limiter every RK3 step.



(b) DG at t =50 s, using the moment limiter every RK3 step.



(c) WENO at t =25 s.



(d) WENO at t =50 s.

Figure B.5: The differences in initial φ_0 with the approximated φ at time t for DG and WENO.

Bibliography

[1] P. Bechtold. Atmospheric thermodynamics, May 2009. URL https://www.ecmwf.int/sites/default/files/elibrary/2015/16954-atmospheric-thermodynamics.pdf. Retrieved on 2017-05-18.

[2] L.C. Berselli, L.C. Iliescu, and W.J. Layton. *Mathematics of Large Eddy Simulation of Turbulent Flows*. Springer, Berlin, Heidelberg, 2006.

[3] R. Biswas, K. D. Devine, and J. E. Flaherty. Parallel, adaptive finite element methods for conservation laws. *Applied Numerical Mathematics*, 14(1-3):255–283, apr 1994. doi: 10.1016/0168-9274(94)90029-9.

[4] S.J. Böing. *The Interaction Between Deep Convective Clouds and Their Environment*. PhD thesis, TU Delft, 2014.

[5] C. Caljouw. Discontinuous galerkin methods for numerical weather prediction, June 2017. URL http://ta.twi.tudelft.nl/nw/users/vuik/numanal/caljouw_scriptie.pdf. Literature Study.

[6] B. Cockburn and C.-W. Shu. The runge–kutta discontinuous galerkin method for conservation laws v. *Journal of Computational Physics*, 141(2):199 – 224, 1998. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1998.5892. URL http://www.sciencedirect.com/science/article/pii/S0021999198958922.

[7] B. Cockburn and C.-W. Shu. Runge-kutta discontinuous galerkin methods for convection-dominated problems. *Journal of Computing*, 16(3):173–261, 2001. doi: 10.1023/a:1012873910884.

[8] S. Deng. Quadrature formulas in two dimensions, 2010. URL http://math2.uncc.edu/~shaodeng/TEACHING/math5172/Lectures/Lect_15.PDF.

[9] R. Desai. The cyclone [image]. Educational Blog, May 2011. URL http://drrajivdesaimd.com/2011/05/14/the-cyclone/comment-page-1/. Retrieved on 2017-04-13.

[10] A. Dosio. *Turbulent Dispersion in the Atmospheric Convective Boundary Layer*. PhD thesis, Wageningen Universiteit, May 2005.

[11] Federal Aviation Administration. *Aviation Weather [Image]*. Aviation Supplies & Academics, Inc., 1975. URL https://www.aviationweather.ws/046_Wind_Shear.php.

[12] A. Geletu. Orthogonal polynomials, quadratures & sparse-grid methods for probability integrals, 2010. URL https://www.tu-ilmenau.de/fileadmin/media/simulation/Lehre/Vorlesungsskripte/Lecture_materials_Abebe/OrthoPolys_and_SparseGrids2.pdf.

[13] L. Gryngarten, A. Smith, and S. Menon. Discontinuous galerkin method with the spectral deferred correction time-integration scheme and a modified moment limiter for adaptive grids. *Communications in Applied Mathematics and Computational Science*, 7(2):133–174, 2012.

[14] A. Harten, B. Engquist, S. Osher, and S.R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. *Journal of Computational Physics*, 71(2):231–303, aug 1987. doi: 10.1016/0021-9991(87)90031-3.

[15] T. Heus, C. C. van Heerwaarden, H. J. J. Jonker, A. Pier Siebesma, S. Axelsen, K. van den Dries, O. Geoffroy, A. F. Moene, D. Pino, S. R. de Roode, and J. Vilà-Guerau de Arellano. Formulation of the dutch atmospheric large-eddy simulation (DALES) and overview of its applications. *Geoscientific Model Development*, 3(2):415–444, sep 2010. doi: 10.5194/gmd-3-415-2010.

[16] J. R. Holton and G. J. Hakim. *An Introduction to Dynamic Meteorology*. Academic Press. Elsevier, 2013.

[17] G.T. Huntington, D.A. Benson, and A.V. Rao. A comparison of accuracy and computational efficiency of three pseudospectral methods. *AIAA Paper 2007-6405*, 2007.

[18] L. Krivodonova. Limiters for high-order discontinuous galerkin methods. *Journal of Computational Physics*, 226(1):879–896, sep 2007.

[19] P. Lesaint and P.A. Raviart. On a finite element method for solving the neutron transport equation. *Publications mathématiques et informatique de Rennes*, (S4):1–40, 1974. URL http://eudml.org/doc/273730.

[20] R.J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[21] X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200–212, nov 1994. doi: 10.1006/jcph.1994.1187.

[22] S. Marras, J. F. Kelly, M. Moragues, A. Müller, M. A. Kopera, M. Vázquez, F. X. Giraldo, G. Houzeaux, and O. Jorba. A review of element-based galerkin methods for numerical weather prediction: Finite elements, spectral elements, and discontinuous galerkin. *Archives of Computational Methods in Engineering*, 23(4):673–722, 2016. ISSN 1886-1784.

[23] Met Office. Clouds, 2012. URL http://www.metoffice.gov.uk/binaries/content/assets/mohippo/pdf/library/factsheets/12_0674-factsheet-1_clouds.pdf.

[24] L. Rezzolla. Numerical methods for the solution of partial differential equations, 2011. URL http://www.aei.mpg.de/~rezzolla/lnotes/Evolution_Pdes/evolution_pdes_lnotes.pdf. Retrieved on 2017-09-22.

[25] K. Saniee. A simple expression for multivariate lagrange interpolation. *SIAM Undergraduate Research Online*, 1, january 2008.

[26] C.-W. Shu. Discontinuous galerkin methods: General approach and stability, n.d. URL https://www3.nd.edu/~zxu2/acms60790S15/DG-general-approach.pdf. Retrieved on 2017-10-06.

[27] C.-W. Shu and Osher S. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, aug 1988. doi: 10.1016/0021-9991(88)90177-5.

[28] C.-W. Shu and Osher S. Efficient implementation of essentially non-oscillatory shock-capturing schemes, II. *Journal of Computational Physics*, 83(1):32–78, jul 1989. doi: 10.1016/0021-9991(89)90222-2.

[29] Thomson Higher Education [Image]. Temperature changes of adiabetic compression and expansion., Retrieved on 2017-10-15. URL https://atmos.washington.edu/~hakim/101/adiabatic/.

[30] J. van der Dussen. *Stratocumulus Transitions in Present-day and Future Climate*. mathesis, TU Delft, June 2015.

[31] J. van Kan, A. Segal, and F. Vermolen. *Numerical Methods in Scientific Computing*. Delft Academic Press, 2014.

[32] C. Vuik and D.J.P. Lahaye. Scientific computing (wi201), 2015.

[33] C. Vuik, P. van Beek, F. Vermolen, and J. van Kan. *Numerieke Methoden voor Differentiaalvergelijkingen*. VSSD, 2006.

[34] M.J. Vuik. *Multiwavelets and Outlier Detection for Troubled-cell indiation in Discontinuous Galerkin Methods*. PhD thesis, TU Delft, 2017.

[35] E. W. Weisstein. Double Factorial, n.d.. URL http://mathworld.wolfram.com/DoubleFactorial.html. Retrieved on 2017-09-04.

[36] E. W. Weisstein. Injection, n.d.. URL http://mathworld.wolfram.com/Injection.html. Retrieved on 2017-10-13.

[37] E. W. Weisstein. Surjection, n.d.. URL http://mathworld.wolfram.com/Surjection.html. Retrieved on 2017-10-13.

[38] P. Wesseling. Von Neumann stability conditions for the convection-diffusion equation. *IMA journal of Numerical Analysis*, 16(4):583–598, 1996.

[39] Wikipedia. Legendre polynomials, 2017. URL https://en.wikipedia.org/wiki/Legendre_polynomials. Retrieved 2017-09-04.

[40] J. Wille. Discontinuous Galerkin applied to a generic two-phase flow model in a porous medium, July 2009. URL http://ta.twi.tudelft.nl/nw/users/vuik/numanal/wille_eng.html.