The WENO Method for Non-Equidistant Meshes

Philip Rupp

September 11, 2012, Berlin

Contents

1 Introduction

The WENO method in general is an approximation principle for functions based on isolated and discrete points in an closed interval. Due to [3], the first WENO Scheme was introduced by X.-D. Liu, S. Osher and T. Chan 1994 in the Journal of Computational Physics vol. 115.
The acronym stands for **W**eighted **E**ssentially **N**on-**O**scillatory and is an improvement of the non-weighted ENO procedure. The aim of both, the ENO and WENO method, is to get a preferable smooth interpolation e.g. to counteract over- and undershoots near discontinuities. The WENO method uses a linear combination of possible interpolation functions weighted on their individual smoothness.

The common application area of the WENO method is to solve partial differential equations (PDE), especially physical, hyperbolic conversation laws, in a numerical way. Therefore it has to be combined with e.g. an explicit Euler-procedure or a Runge-Kutta-Method as temporal discretization.

As it is known by basic interpolation theory, the order of accuracy depends on the number of used nodes. We will remain as general as possible while presenting the necessary equations, but the explicit forms, examples and final results will be presented for a 5th order WENO method. However the used pattern can be generalized easily.

At last we demonstrate and implementation of the WENO method in a short c-program and discuss the results in Section 4. Therefore, we programed the later explained WENO polynomial reconstruction method for given cell averages. As example, we use a step function taken from [1] and discussed in Section 2.2. The example of a solved differential equation will be given by the simple linear transport equation. The whole source code can be reviewed within the appendix.

1.1 Settings and Conditions

In the course of this work we will derive the necessary formulas concerning the WENO method. Here we will not be limited to equidistant meshes. Rather we will consider completely general grids. Though all used meshes shall be constant in time and each space dimensions. According to those characteristics we define a one dimensional mesh through its grid values in a relative way:

$$x_i = x_0 + \sum_{k=0}^{i} g_k \Delta x. \tag{1}$$

So the difference of any two grid points with $i \geq j$ yields to:

$$x_i - x_j = \sum_{k=j+1}^{i} g_k \Delta x = d_{i,j}. \tag{2}$$

As shown, the used numeration of the grid values is ascending with x_0 as left hand starting point. The $g_k > 0$ are the direct distances of the points x_{i-1} and x_i normed by a parameter Δx. That parameter Δx makes the mesh independent of the exact scaling and can be chosen freely as a constant value; e.g. $\Delta x = 1$, as we will do in most of the further calculations.
This construction has several possible advantages:

- Since the parameter Δx is user defined, the g_k can always be varied while their ratios $\frac{g_a}{g_b}$ for every $a, b \in \mathbb{N}$ do not change. So all meshes, with equal ratios, lead to e.g. the same coefficients of the later introduced smoothness indicators.

- One could define $g_k = 1$ for the largest cell range. So all other g_k's, and even the $d_{i,j}$'s, indicate the relative rate concerning the largest cell range, given by Δx.

- It is possible to set $g_k = g(k)$ for any chosen function. This leads to very easily constructed regular grids.

Within this report all marginal problems will be avoided. We assume that all required points and values outside the relevant interval are known. Also extreme solutions and special exceptional cases will not be discussed.
Later-on calculations with the WENO method for more than one dimension are based on a simple 'dimension by dimension fashion', as so called in [3]. So it is absolutely sufficient to concentrate on the one dimensional case, like we will do.

As interpolating functions we will refer to the mostly used Lagrange polynomials. Such a polynomial based on the $n + 1$ given points $(x_i; y_i)$ with $i \in [m; m + n + 1] \subset \mathbb{N}$ is defined by:

$$P(x) = \sum_{i=m}^{m+n} y_i L_i^m(x) = \sum_{i=m}^{m+n} \left(y_i \prod_{i \neq j=m}^{m+n} \frac{x - x_j}{x_i - x_j} \right) \tag{3}$$

Please note, that concerning algebraic functions we will use e.g. $L^k(x)$ as index and $(L(x))^k$ as power. For the rest of this paper, n will be the order of accuracy concerning a WENO interpolation and reconstruction based on $n + 1$ points and n cells, respectively the later-on large stencil. Since those two methods are closely related we will also try to use consistent expressions for repetitive objects. But always keep in mind that some objects, e.g. the interpolation/reconstruction polynomials, base on different concepts and got different explicit forms.

2 The WENO Schemes

2.1 The Interpolation Problem

Like in [3], we will discuss two separated, but related, problems. They are called 'interpolation' and 'reconstruction' and we will start with the first one. The interpolation is a Finite Differences (FD) method using the values y_i at the nodes x_i of the grid. The final task is to compute an additional value between two of those nodes. To stay general we chose an interval around the node x_j with a range of s grid steps to the left and $s+1$ to the right. The searched value may belong to a point between x_j and x_{j+1}, e.g. $x_{j+\frac{1}{2}}$. So the considered stencil contains the points $\{x_{j-s}, x_{j-s+1}, ...x_j, x_{j+\frac{1}{2}}, x_{j+1}, ...x_{j+s+1}\}$.

The total number of used points is $n+1 = s+s+2$ and the degree of an interpolation polynomial based on those points would be $n = 2s+1$, which obviously just can take uneven numbers. This large interval will be called S and forms our 'large stencil'. As next step we divide this Stencil in a maximal number of 'small stencils' having a constant order, formed out of the grid nodes and containing $x_{j+\frac{1}{2}}$. That would be exactly $\frac{n+1}{2} = s+1$ intervals as can be seen in Figure 1, using the example of fifth order. The small stencils have a length of $s+2$ grid points, so their polynomials got a degree of $s+1 = \frac{n+1}{2}$. To name the interpolation polynomials we chose $Q(x)$ as interpolation based on the large stencil S and $P_k(x)$ for the small stencils S_k with the node x_{j-s+k} as most left starting point.

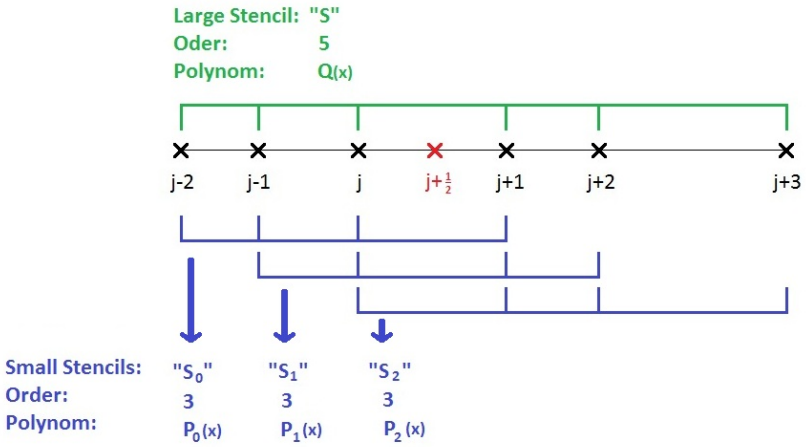

Figure 1: Setting of the Stencils for a 5th order interpolation of $x_{j+\frac{1}{2}}$

Following Equation (3) the semi-explicit form of the interpolation polynomials for a 6-point large stencil are shown below. The K_i^j are analog to the L_i^j, but possess degree 5 and not 3.

$$\begin{aligned} Q(x) &= \sum_{i=j-2}^{j+3} y_i K_i^{j-2} \qquad (4) \\ Q(x) &= y_{j-2}K_{j-2}^{j-2} + y_{j-1}K_{j-1}^{j-2} + y_j K_j^{j-2} + y_{j+1}K_{j+1}^{j-2} + y_{j+2}K_{j+2}^{j-2} + y_{j+3}K_{j+3}^{j-2}. \end{aligned}$$

$$\begin{aligned} P_k(x) &= \sum_{i=j-2+k}^{j+1+k} y_i L_i^k \qquad (5) \\ P_0(x) &= y_{j-2}L_{j-2}^{j-2} + y_{j-1}L_{j-1}^{j-2} + y_j L_j^{j-2} + y_{j+1}L_{j+1}^{j-2}, \\ P_1(x) &= y_{j-1}L_{j-1}^{j-1} + y_j L_j^{j-1} + y_{j+1}L_{j+1}^{j-1} + y_{j+2}L_{j+2}^{j-1}, \\ P_2(x) &= y_j L_j^j + y_{j+1}L_{j+1}^j + y_{j+2}L_{j+2}^j + y_{j+3}L_{j+3}^j. \end{aligned}$$

Since the combination of the small stencils forms the large stencil, a linear weighted combination of the interpolation polynomials $P_k(x)$ also forms the polynomial $Q(x)$. Therefore we chose special, so called linear weights, $C_k(x)$ to satisfy the following condition:

$$Q(x) = \sum_{k=0}^{\frac{n-1}{2}} C_k(x) P_k(x). \qquad (6)$$

Remember, that $Q(x)$ is of degree n and the P_k are of degree $\frac{n+1}{2}$. So those linear weights have to represent polynomials of degree $\frac{n-1}{2}$. The next step is to combine the Equations (4), (5) and (6) including just known functions and the searched linear weights. Equating the coefficients of the y_i lets us form a linear system of equation to calculate the $C_k(x)$ as shown in the example for 5th/3rd order:

$$\begin{aligned} K_{j-2}^j &= C_0 L_{j-2}^{j-2}, \qquad (7) \\ K_{j-1}^j &= C_0 L_{j-1}^{j-2} + C_1 L_{j-1}^{j-1}, \\ K_j^j &= C_0 L_j^{j-2} + C_1 L_j^{j-1} + C_2 L_j^j, \\ K_{j+1}^j &= C_0 L_{j+1}^{j-2} + C_1 L_{j+1}^{j-1} + C_2 L_{j+1}^j, \\ K_{j+2}^j &= C_1 L_{j+2}^{j-1} + C_2 L_{j+2}^j, \\ K_{j+3}^j &= C_2 L_{j+3}^j. \end{aligned}$$

These $n+1$ equations for $\frac{n+1}{2}$ unknown polynomials can be solved easily and elegant, using the Definition (2):

$$\begin{aligned} C_0 &= \frac{(x-x_{j+2})(x-x_{j+3})}{d_{j+2,j-2}d_{j+3,j-2}}, \\ C_1 &= (x-x_{j-2})(x-x_{j+3})\left(\frac{1}{d_{j-2,j-1}d_{j+3,j-1}} - \frac{1}{d_{j+2,j-2}d_{j+3,j-2}}\right), \\ C_2 &= \frac{(x-x_{j-1})(x-x_{j-2})}{d_{j+3,j-2}d_{j+3,j-1}}. \end{aligned} \tag{8}$$

Our final aim is to get a high order, smooth function interpolating our given points. The linear weights help us just to get the high order polynomial $Q(x)$, witch we could also calculate directly. For now we keep them at the back of our minds and define some kind of 'smoothness indicators':

$$\beta_k = \sum_{i=1}^{\frac{n+1}{2}} \int_{x_j}^{x_{j+1}} (\Delta x)^{2i+1} \left(\frac{d^i}{dx^i} P_k\right)^2 \mathrm{d}x. \tag{9}$$

Roughly speaking they sum up the different derivatives of the corresponding polynomial $P_k(x)$. So they do some kind of oscillation measuring for the interpolated functions within the interval $[x_i; x_{i+1}]$. The higher this smoothness indicator is, the less smooth is the function inside the cell with the interpolating point. The constant factor $(\Delta x)^{2i+1}$ is the same as in Definition (1) and makes the smoothness indicator independent of the exact grid scaling. All meshes with the same structure and relative grid-step-ratio will lead to the same β-coefficients.

Actually the explicit choice of this smoothness indicator is up to the user. Other forms and formulas would work too, but within the literature this version is established. Unfortunately most of the current publications use just equidistant WENO concepts. For this reason the introduced smoothness indicator is actually designed for equidistant grids. There might be better smoothness indicators, especially for the case of non-equidistant meshes. However, as you can see, the β_k's are computed directly from the interpolation polynomials, so we can give them an explicit, analytical form.

Specifying the short term $\dot{f}(x) = \frac{d}{dx}f(x)$ and using 5th order and Definition (3) and (2) the derivatives of each L_j^m are in general form:

$$\begin{aligned} a_i^m &= \prod_{\substack{j=j-2+m \\ j\neq i}}^{j+1+m} \frac{1}{d_{i,j}}, \qquad (10) \\ \dot{L}_i^m &= a_i^m \sum_{\substack{k=m \\ k\neq i}}^{3+m} \prod_{\substack{j=m \\ j\neq i; j\neq k}}^{3+m} (x - x_j), \\ \ddot{L}_i^m &= 2a_i^m \sum_{\substack{k=m \\ k\neq i}}^{3+m} (x - x_k), \\ \dddot{L}_i^m &= 6a_i^m. \end{aligned}$$

Analogously, we also define for the later use:

$$a_i = \prod_{\substack{j=j-2 \\ j\neq i}}^{j+3} \frac{1}{d_{i,j}}. \qquad (11)$$

So for a 5th order interpolation the first smoothness indicator, belonging to the most left stencil, is:

$$
\begin{aligned}
\beta_0 &= y_{j-2}^2 \int_{x_j}^{x_{j+1}} \left(\Delta x \left(\dot{L}_{j-2}^{j-2} \right)^2 + (\Delta x)^3 \left(\ddot{L}_{j-2}^{j-2} \right)^2 + (\Delta x)^5 \left(\dddot{L}_{j-2}^{j-2} \right)^2 \right) \mathrm{d}x, \quad (12) \\
&+ y_{j-1}^2 \int_{x_j}^{x_{j+1}} \left(\Delta x \left(\dot{L}_{j-1}^{j-2} \right)^2 + (\Delta x)^3 \left(\ddot{L}_{j-1}^{j-2} \right)^2 + (\Delta x)^5 \left(\dddot{L}_{j-1}^{j-1} \right)^2 \right) \mathrm{d}x, \\
&+ y_{j}^2 \int_{x_j}^{x_{j+1}} \left(\Delta x \left(\dot{L}_{j}^{j-2} \right)^2 + (\Delta x)^3 \left(\ddot{L}_{j}^{j-2} \right)^2 + (\Delta x)^5 \left(\dddot{L}_{j}^{j-1} \right)^2 \right) \mathrm{d}x, \\
&+ y_{j+1}^2 \int_{x_j}^{x_{j+1}} \left(\Delta x \left(\dot{L}_{j+1}^{j-2} \right)^2 + (\Delta x)^3 \left(\ddot{L}_{j+1}^{j-2} \right)^2 + (\Delta x)^5 \left(\dddot{L}_{j+1}^{j-1} \right)^2 \right) \mathrm{d}x, \\
&+ y_{j-2} y_{j-1} 2 \int_{x_j}^{x_{j+1}} \left(\Delta x \dot{L}_{j-2}^{j-2} \dot{L}_{j-1}^{j-2} + (\Delta x)^3 \ddot{L}_{j-2}^{j-2} \ddot{L}_{j-1}^{j-2} + (\Delta x)^5 \dddot{L}_{j-2}^{j-2} \dddot{L}_{j-1}^{j-2} \right) \mathrm{d}x, \\
&+ y_{j-2} y_{j} \quad 2 \int_{x_j}^{x_{j+1}} \left(\Delta x \dot{L}_{j-2}^{j-2} \dot{L}_{j}^{j-2} + (\Delta x)^3 \ddot{L}_{j-2}^{j-2} \ddot{L}_{j}^{j-2} + (\Delta x)^5 \dddot{L}_{j-2}^{j-2} \dddot{L}_{j}^{j-2} \right) \mathrm{d}x, \\
&+ y_{j-2} y_{j+1} 2 \int_{x_j}^{x_{j+1}} \left(\Delta x \dot{L}_{j-2}^{j-2} \dot{L}_{j+1}^{j-2} + (\Delta x)^3 \ddot{L}_{j-2}^{j-2} \ddot{L}_{j+1}^{j-2} + (\Delta x)^5 \dddot{L}_{j-2}^{j-2} \dddot{L}_{j+1}^{j-2} \right) \mathrm{d}x, \\
&+ y_{j-1} y_{j} \quad 2 \int_{x_j}^{x_{j+1}} \left(\Delta x \dot{L}_{j-1}^{j-2} \dot{L}_{j}^{j-2} + (\Delta x)^3 \ddot{L}_{j-1}^{j-2} \ddot{L}_{j}^{j-2} + (\Delta x)^5 \dddot{L}_{j-1}^{j-2} \dddot{L}_{j}^{j-2} \right) \mathrm{d}x, \\
&+ y_{j-1} y_{j+1} 2 \int_{x_j}^{x_{j+1}} \left(\Delta x \dot{L}_{j-1}^{j-2} \dot{L}_{j+1}^{j-2} + (\Delta x)^3 \ddot{L}_{j-1}^{j-2} \ddot{L}_{j+1}^{j-2} + (\Delta x)^5 \dddot{L}_{j-1}^{j-2} \dddot{L}_{j+1}^{j-2} \right) \mathrm{d}x, \\
&+ y_{j} y_{j+1} \quad 2 \int_{x_j}^{x_{j+1}} \left(\Delta x \dot{L}_{j}^{j-2} \dot{L}_{j+1}^{j-2} + (\Delta x)^3 \ddot{L}_{j}^{j-2} \ddot{L}_{j+1}^{j-2} + (\Delta x)^5 \dddot{L}_{j}^{j-2} \dddot{L}_{j+1}^{j-2} \right) \mathrm{d}x.
\end{aligned}
$$

This pattern is the same for the other two indicators, just changing the corresponding points and functions. The different colors represent the terms belonging to the different derivatives. As this formula shows, each β_k has 10 coefficients just depending on the grid and not on the specific y_i-values, making 30 factors all in all. Those coefficients, which we will name $g_{i,j}^k$ ='coefficient of $(y_i y_j)$ respective β_k', can be computed and stored at the beginning of the interpolation. We will not specify the whole calculation, because it is simple and long algebra, but show the final results within the appendix under Equation (55). For all equidistant meshes these coefficients reduce to the same set of real numbers. This makes the use of those grid much easier and the corresponding formulas much shorter.

We can now combine the smoothness indicators with the linear weights to form our nonlinear weights. Because it shall be something like a convex combination at the end, it is helpful to define an auxiliary number:

$$\alpha_k = \frac{C_k}{(\epsilon + \beta_k)^2}. \quad (13)$$

As you see the α's contain another value, named ϵ. This quantity prevents the denominator from becoming zero, e.g. for constant functions. It's a previously chosen number, mostly 10^{-6} and has no actual use for the interpolation. Notice further, that these α_k's consist of the C_k's, weighted by the smoothness factors β_k. The more smooth a function is within the relevant interval, the lower is the smoothness factor and the higher becomes the ratio of the corresponding polynomial.

These α-values are again weighted by their combined sum to get a 'kind of convex combination' to form the 'nonlinear weights' ω_k:

$$\begin{aligned} \omega_k &= \frac{\alpha_k}{\sum_{i=0}^{\frac{n-1}{2}} \alpha_i}, \qquad (14) \\ \sum_{k=0}^{\frac{n-1}{2}} \omega_k &= 1. \end{aligned}$$

Note that they get their sign from the C_k's and $\omega_k \geq 0$ and $C_k \geq 0$ is just given in the space of the interval $[x_i; x_{i+1}]$. The proof is shown in [2]. This fact just counts for the interpolation part and within the following reconstruction the C_k can get negative even more easily. But for the used order of 5 and the explicit computed points during this report, they will always be positive. But in general, those sets don't really form convex combinations. They have always a sum of one, but are not necessary positive.

At the end of this derivation we can construct the interpolation polynomial $I(x)$, interpolating our given node points between x_j and x_{j+1} using the predefined, low order polynomial P_k:

$$I(x) = \sum_{i=0}^{\frac{n-1}{2}} \omega_i P_i. \qquad (15)$$

As you see the order of accuracy of the interpolation polynomial $I(x)$ is n, consisting of the $\frac{n+1}{2}$-order interpolation polynomials $P_k(x)$ and the $\frac{n-1}{2}$-order linear weights. So roughly speaking this originated function is of higher order than the P_k's, but mostly is not that strongly oscillating as the calculated $Q(x)$. It combines advantages of both and makes the WENO procedure perfect for interpolating piecewise smooth functions with discontinuities in between.

2.1.1 Why those unique C_k's can always be found

Following ideas from [2] it is surely correct to say $Q(x_i) = y_i$ within the large stencil S, due to the definition of the interpolation polynomial. Within each small stencil S_k an analog equation can be formed: $P_k(x_i) = y_i$. Outside each small stencil, within the large one, the P_k take unknown values, not necessary zero in the nodes. So we should claim:

$$C_k(x_i) = 0 \text{ for each } x_i \in S \backslash S_k. \tag{16}$$

The $\frac{n+1}{2}$ C_k's are polynomials of degree $\frac{n-1}{2}$. They got now a sum of $\frac{n^2+2n+1}{4}$ parameters. We already can form $\frac{n^2-1}{4}$ independent equations out of (16), due to the fundamental theorem of algebra. Combining them with Equation (6), the linear weights have to form again a 'kind of convex set', **not** satisfying $C_k > 0$, but the main requirement

$$\sum_{k=0}^{\frac{n-1}{2}} C_k(x_i) = 1 \text{ for each } x_i \in S. \tag{17}$$

To keep track of the dependency of these equations, we define a general form for the linear weights:

$$C_k(x) = \sum_{i=0}^{\frac{n-1}{2}} r_k^i (x)^i. \tag{18}$$

We get a simplified form of Equation (17), that clearly shows its $\frac{n-1}{2}$ degree polynomial character:

$$\sum_{i=0}^{\frac{n-1}{2}} \left((\sum_{j=0}^{\frac{n-1}{2}} r_j^i) \right) (x_z)^i = 1 \text{ for each } x_z \in S. \tag{19}$$

The polynomial on the left side has to be identical to the constant function $r(x) \equiv 1$. This leads to exactly $\frac{n+1}{2}$ simple, independent equations:

$$\begin{aligned} \sum_{j=0}^{\frac{n-1}{2}} r_j^i &= 0 \text{ for each } i \in \left[1, \frac{n-1}{2}\right], \\ \sum_{j=0}^{\frac{n-1}{2}} r_j^0 &= 1. \end{aligned} \tag{20}$$

Together with Equation (16) we get $\frac{n^2+2n+1}{4}$ independent equations for the same number of parameters. The system of $C_k(x)$ got a unique solution. We will now discuss a short example of a 5th order interpolation. At first we define the general form for our 3 second degree polynomials $C_k(x)$:

$$C_k(x) = a_k(x)^2 + b_k x + c_k. \tag{21}$$

So we got all in all 9 parameters to determine. 6 independent fixing equations are formed out of (16):

$$\begin{aligned} a_0(x_i)^2 + b_0 x_i + c_0 &= 0 \text{ for } i \in \{j+2, j+3\}, \\ a_1(x_i)^2 + b_1 x_i + c_1 &= 0 \text{ for } i \in \{j-2, j+3\}, \\ a_2(x_i)^2 + b_2 x_i + c_2 &= 0 \text{ for } i \in \{j-2, j-1\}. \end{aligned} \tag{22}$$

Furthermore Equation (17) turns into:

$$(a_0 + a_1 + a_2)(x_i)^2 + (b_0 + b_1 + b_2)x_i + (c_0 + c_1 + c_2) = 1 \text{ for each } x_i \in S. \tag{23}$$

This is a simple polynomial of degree two, coinciding on 6 points with the constant polynomial $v(x) \equiv 1$. Both sides have to be identical and a comparison of coefficients gives us the remaining 3 equations, to compute the unique solution:

$$\begin{aligned} (a_0 + a_1 + a_2) &= 0, \\ (b_0 + b_1 + b_2) &= 0, \\ (c_0 + c_1 + c_2) &= 1. \end{aligned} \tag{24}$$

At least we want to mention, that the set of C_k even forms a basis of $\mathbb{P}_{\frac{n-1}{2}}$. This easily can be seen as following definition, based on Equation (16) and the fundamental theorem of algebra, forms a maximal, independent set:

$$C_k(x) = \gamma_k \prod_{x_i \in S \setminus S_k} (x - x_i) \text{ with } \gamma_k \in \mathbb{R}. \tag{25}$$

With this definition and Equation (17) we also got an additional way to calculate the linear weights:

$$\sum_{i=1}^{k} \gamma_i \prod_{x_m \in S \setminus S_i} \left(x_{j-\frac{n-1}{2}+k} - x_m \right) = 1 \text{ for all } k \in \left[1; \frac{n-1}{2} \right]. \tag{26}$$

The so defined system only includes the minimal, independent set of equations and clearly shows again the unique solvability. As you will simply check, all statements are equal and compatible with the explicit 5th-order form given in (8).

2.2 The Reconstruction Problem

In addition to the interpolation task, the WENO method also can be used for reconstructing functions. The aim of such a reconstruction is also a smooth estimation of a certain value between two known grid points. The difference lays in the starting conditions. Within a reconstruction not the values at the nodes are given, but the cell average of the function. Those types of problems are usually called Finite Volume (FV) methods. The general setting still is the same as during the interpolation part. Within this report the cell average of the function $u(x)$ within the interval $[x_i, x_{i+1}]$ is defined as:

$$\overline{y}_i = \frac{1}{d_{i+1,i}} \int_{x_i}^{x_{i+1}} u(x)\,\mathrm{d}x. \tag{27}$$

Since we don't want to develop the whole WENO construction from scratch, it should be helpful to use our knowledge from the WENO interpolation. If we can calculate all grid values out of the given cell averages we could completely use the formulas given in Section 2.1. To reach this, another auxiliary function is introduced, like it is done in [3]. This function is generated out of the function $f(x)$, which we want to reconstruct:

$$U_k(x) = \int_{x_{j-2+k}}^{x} u(z)\,\mathrm{d}z. \tag{28}$$

This is an integral function with its variable in the upper limit. The lower limit is not important and can be chosen as any optional fixed point. We will always choose x_k as most left node point of the used stencil. While later-on $U_k(x)$ shall be based on the small stencil S_k, we will set $U(x)$ to be the analog function based on the large stencil S with $U(x)$ as $x_k = \inf S$. The outcome is a relation between the given cell averages y_j and the grid values of our previously defined function $U_k(x)$ in all relevant points x_i within the used stencil:

$$U_k(x_i) = \int_{x_{j-2+k}}^{x_i} u(z)\,\mathrm{d}z = \sum_{j=j-2+k}^{i-1} \int_{x_j}^{x_{j+1}} u(z)\,\mathrm{d}z = \sum_{j=j-2+k}^{i-1} d_{j+1,j}\overline{y}_j. \tag{29}$$

Furthermore the corresponding derivative interpolates the not known function $u(x)$, with $\tilde{u}(x)$ being a primitive of it:

$$\frac{d}{dx}U_k(x) = \frac{d}{dx}\left(\int_{x_{j-2+k}}^{x} u(z)\,\mathrm{d}z\right) = \frac{d}{dx}\left(\tilde{u}(x) - \tilde{u}(x_{j-2+k})\right) = u(x). \tag{30}$$

Especially this holds for the node points. Surely this derivative also satisfies the reconstruction of the cell averages defined in (27).

$$\frac{1}{d_{i+1,i}} \int_{x_j}^{x_{j+1}} \frac{d}{dx} U_k(x)\,\mathrm{d}x = \frac{1}{d_{i+1,i}} \int_{x_j}^{x_{j+1}} u(x)\,\mathrm{d}x = \overline{y}_i. \tag{31}$$

Now we take again the stencils and interpolating polynomials used before and shown in Figure 1. But this time the functions $P_k(x)$ and $Q(x)$ shall reconstruct the 'real' function $u(x)$ out of Equation (27) within the corresponding stencil. This means precisely that both got the same average values concerning the grid cells. We have just shown, that the derivative $\frac{d}{dx}U_k(x)$ fulfills this requirement. All we have to do is to construct an interpolation polynomial $\tilde{P}_k(x)$ for the node points of $U_k(x)$ within the stencil S_k, so that $\tilde{P}_k(x_i) = U_k(x_i)$ for $x_i \in S_k$. The derivative $\frac{d}{dx}\tilde{P}_k(x) = P_k(x)$ of this function can be used as reconstruction function for the values of $u(x)$. This works analogously for the reconstructing polynomial $Q(x) = \frac{d}{dx}\tilde{Q}(x)$ within the large stencil, with $\tilde{Q}(x_i) = U(x_i)$ for $x_i \in S$. This leads to:

$$\begin{aligned} \int_{x_j}^{x_{j+1}} P_k(z)\,\mathrm{d}z &= \int_{x_j}^{x_{j+1}} \frac{d}{dx} U_k(x)\,\mathrm{d}x = d_{j+1,j}\overline{y_i}, \\ \int_{x_j}^{x_{j+1}} Q(z)\,\mathrm{d}z &= \int_{x_j}^{x_{j+1}} \frac{d}{dx} U(x)\,\mathrm{d}x = d_{j+1,j}\overline{y_i}. \end{aligned} \tag{32}$$

To summarize, we just have to calculate the node points of the function $U_k(x)$ (respectively $U(x)$) with Equation (29). Then we interpolate them in terms of the Lagrange polynomial in (3). At last we determine the derivative of that interpolation polynomial and use it as reconstruction (in terms of same cell averages) for the unknown function $u(x)$. The general equation for the interpolation of the primitive auxiliary function $U_k(x)$, based on the stencil S_k and a order 5 reconstruction is:

$$\begin{aligned} U_k(x) &= \sum_{i=j-2+k}^{k+3} \left(\sum_{h=j-2+k}^{i-1} d_{h+1,h}\overline{y}_h \right) \prod_{\substack{s=j-2+k \\ s\neq i}}^{j+1+k} \frac{x - x_s}{x_i - x_s}, \\ U(x) &= \sum_{i=j-2}^{j+3} \left(\sum_{h=j-2}^{i-1} d_{h+1,h}\overline{y}_h \right) \prod_{\substack{s=j-2 \\ s\neq i}}^{j+3} \frac{x - x_s}{x_i - x_s}. \end{aligned} \tag{33}$$

This leads to the general 3rd and 5th order reconstruction polynomial as its derivative for the same stencil and point combination:

$$P_k(x) = \sum_{i=j-2+k}^{j+1+k} \left(a_i^k \left(\sum_{h=j-2+k}^{i-1} d_{h+1,h}\overline{y}_h \right) \sum_{t=j-2+k}^{j+1+k} \prod_{\substack{s=j-2+k \\ s\neq i; s\neq t}}^{j+1+k} (x - x_s) \right), \quad (34)$$

$$Q(x) = \sum_{i=j-2}^{j+3} \left(a_i^k \left(\sum_{h=j-2}^{i-1} d_{h+1,h}\overline{y}_h \right) \sum_{t=j-2}^{j+3} \prod_{\substack{s=j-2 \\ s\neq i; s\neq t}}^{j+3} (x - x_s) \right).$$

With a_i^k and a_i from Equation (10) and (11). These complicated looking functions are just the combination of the Equations (3),(10), (28), (29), (30), and (31). They reconstruct the polynomial of degree three or five with given cell averages. Please note, that e.g. the polynomial Q_k has an order of accuracy of n, if $n+1$ points and n cells are used, though it is just a polynomial of degree $n-1$. This is, because it is a reconstruction and not an interpolation. To satisfy the main requirement (32) it has to be integrated and gets a degree of n, counting as interpolating function for the given cell averages.

the next step is to find the explicit forms of the linear weights $C_k(x)$. Again, they shall be the linking part between the reconstruction functions based on the small stencils S_k, with $P_k(x)$, and the high degree polynomial $Q(x)$, based on the large stencil S. So the general setting is equal to the interpolation part. Since the reconstruction polynomials all are based on the same reconstruction function $u(x)$, they got the same grid values according to Equation (30). So they also count as order $n-1$ and $\frac{n-1}{2}$ interpolation polynomials for those grid points based on n and $\frac{n+1}{2}$ points. The general linear weights have to be polynomials of degree $\frac{n-1}{2}$ again. According to that, the arguments for their existence and uniqueness are the analog ones as before. The only difference is the much more complicated form they got within the reconstruction problem. So due to simplicity we just calculate them at one special point. In Equation (57) of the appendix you can find the explicit linear weights, computed at the node point x_{j+1}.

Actually this was the main point concerning the reconstruction problem. All work left is the calculation of the smoothness indicators. But as you easily can see in Equation (12) of Section 2.1, their coefficients are not depending of the actual given data, but just on the mesh. Also emember, that our reconstruction polynomials are already the first derivatives of the interpolating auxiliary function. They are based on $\dot{L}_i^m$ in a fifth order reconstruction. Their derivatives, used to compute the smoothness indicators, are combinations of the given $\ddot{L}_i^m$ and $\dddot{L}_i^m$, that are also used within the interpolation part to get the coefficients g_i^j. So all we have to do now is to copy the coefficients of the smoothness indicators from Equation (55), but skip the first summand, which

is colored red, because it belongs to the not used first derivative. As node points y_i we take the values of the corresponding auxiliary function $U_k(x_i)$. This might not be the most efficient way, but within this report it works good enough to use. The concrete reconstruction values are computed with a formula similar to Equation (15), using again Definition (13) and (14).

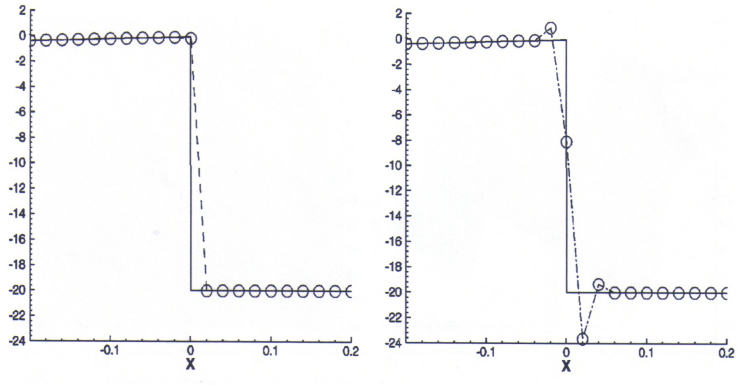

Figure 2: Reconstruction of cell boundaries. Solid lines: exact function; symbols: aproximations. Left: 5th order WENO; Right: 5th order Traditional[1]

As said before, the probably greatest advantage of the WENO schemes over other interpolation and reconstruction methods is in the case of non-smooth functions. In Figure 2, taken from [1], the difference between a traditional fifth order reconstruction with Equation (34) and the WENO reconstruction is presented. Like it is characteristic for most reconstruction concepts, around the discontinuity appear quite strong oscillations, while the WENO reconstructed function looks smooth, fits the exact solution and avoids over- and undershoots. In here the reconstructed function was $u(x) = 2x$ for $x \leq 0$ and $u(x) = -20$ for $x > 0$. The used uniform mesh was constructed by $x_i = 0.02(i - 0.4965)$.

3 Solving Differential Equations

The main usage of the WENO schemes lays in the field of applied numerical science, especially to solve hyperbolic conservation laws. Those are mostly homogenous partial differential equations. In this report, we will discuss the example of an one-dimensional case with the general form:

$$\partial_t u(x,t) + \partial_x f(u(x,t)) = 0, \tag{35}$$

where $u(x,t)$ is the searched function with time coordinate t and a single space dimension x, $\partial_k = \frac{\partial}{\partial_k}$ shall be the short term of the partial derivative with respect to k and $f(z)$ represents the concrete conservation function. We will discuss this problem again for different boundary conditions. As before we call the procedure an FV method for given cell averages, defined in (27). For given node values it represents a FD scheme and will be described afterwards.

3.1 The Finite Volume Scheme

If we got given cell averages $\overline{y}_i$, in terms of Equation (27) out of Section 2.2, we need a FV method again. Therefore we have to transform the partial differential equation into its integral form, taking '$\frac{1}{d_{i+1,i}} \int_{x_i}^{x_{i+1}} \mathrm{d}x$' on every term:

$$\partial_t \, \overline{y}_i(t) + \frac{1}{d_{i+1,i}} \left(f[y_{i+1}(t)] - f[y_i(t)] \right) = 0. \tag{36}$$

Since we can compute the node values from the given cell averages, we can reduce the equation by its space dimension. It actually results in an ordinary differential equation with respect to the time. This type of equation can be solved with other concepts, e.g. the explicit Euler method. This is a low order method and is just use for the simplicity of presentation. In practice one would rather use a 3rd order Runge-Kutta-Method.

The basic idea of the explicit Euler method is simple. We suppose a linear change of each cell averages within a nonuniform time-grid with $t_{j+1} - t_j = \Delta t_j$. So we can assume:

$$\overline{y}_i(t_{j+1}) = \overline{y}_i(t_j) + \Delta t_j \partial_t \, \overline{y}_i(t)|_{t=t_j}. \tag{37}$$

Using (36), this leads to:

$$\overline{y}_i(t_{j+1}) = \overline{y}_i(t_j) + \frac{\Delta t_j}{d_{i+1,i}} \left(f[y_i(t_j)] - f[y_{i+1}(t_j)] \right). \tag{38}$$

This equation is an explicit formula to approximate all desired cell averages at any chosen time coordinate. Of course this method of time discretization is very simple and not used often, but it shows the possibility of using the WENO method to solve such differential equations.

3.2 The Finite Difference Scheme

The FD method is based again on the node values, while cell averages are not given. One simple idea is to use the FV scheme of Section 3.1 and calculate the needed cell averages via numerical quadrature like e.g. Newton's integration. But because of the multiple transformations of the given data the accuracy is suffering.

So it seems best to use just the grid values and take the partial differential equation in its differential form like it is in (35). To solve it, we need to compute the derivative of the given, concrete function $f(u)$. We will see that the WENO interpolation scheme is a good choice to handle this problem. Just some little adjustments have to be done. First of all the linear weights C_k change since our Definition (6) changes into a differential analogon, again using $\dot{f}(x) = \frac{d}{dx}f(x)$. To show the analogy we also choose related symbols:

$$\dot{Q}(x) = \sum_{k=0}^{\frac{n-1}{2}} \tilde{C}_k(x)\dot{P}_k(x). \tag{39}$$

n is still the order of accuracy concerning the used WENO method, just as of the polynomial $Q(x)$ based on the large stencil S. As one can see, the general setting stays again exactly the same as in Section 2. Actually this Equation (39) has the same form as the definition of the linear weights within the reconstruction problem in Section 2.2. There, the reconstruction polynomials $Q(x)$ and $P_k(x)$ were derivatives of the auxiliary functions $U(x)$ and $U_k(x)$, which have been interpolated based on the previously computed grid values. And since we just need the derivative values at the grid points, e.g. x_{i+1}, we can copy the linear weights of Equation (57) for a fifth order interpolation. We even can nearly copy the polynomials of Equation (34) of Section 2.2 and take them as interpolation polynomials for the searched derivatives $\dot{P}_k(x)$ and $\dot{Q}(x)$. The summations of the cell averages have to be changed into the now given node values. The concrete Forms are presented in Equation (40). Of course we also just need their values at the node point x_{i+1}. As you can see, the reconstruction and interpolation problems are really interlaced.

The explicit forms of the 5th order polynomials are:

$$\begin{aligned}\dot{P}_k(x) &= \sum_{i=j-2+k}^{j+1+k} \left(a_i^k y_i \sum_{t=j-2+k}^{j+1+k} \prod_{\substack{s=j-2+k \\ s\neq i; s\neq t}}^{j+1+k} (x - x_s) \right), \qquad (40)\\ \dot{Q}(x) &= \sum_{i=j-2}^{j+3} \left(a_i^k y_i \sum_{t=j-2}^{j+3} \prod_{\substack{s=j-2 \\ s\neq i; s\neq t}}^{j+3} (x - x_s) \right).\end{aligned}$$

The next change concerns the smoothness indicators, which are now called $\tilde{\beta}_k$. Unlike Definition (9), the sum of integrals starts with the second derivative of the corresponding P_k.

$$\tilde{\beta}_k = \sum_{i=2}^{\frac{n+1}{2}} \int_{x_j}^{x_{j+1}} (\Delta x)^{2i+1} \left(\frac{d^i}{dx^i} P_k \right)^2 \mathrm{d}x. \qquad (41)$$

This arises from the fact, that we actually want to interpolate the first derivative and its smoothness is measured with higher derivatives. Furthermore the polynomial P_k got an order of $\frac{n+1}{2}$ and at most that much derivatives, not using its own, concrete function values. So that has to be the end of the summation. Fortunately we don't need to compute the explicit fifth order equations for the smoothness indicators again, but roughly can take the ones from the WENO interpolation part in Section 2.1. We just need to delete the first of the three summands, which belongs to the first derivative and is colored red within this paper. The concrete calculation of the derivative values is done with a similar formula as before, using the analogon to Definition (13) and (14):

$$\partial_x u(x)|_{x=x_i} = \dot{I}(x_i) = \sum_{k=0}^{\frac{n-1}{2}} \tilde{\omega}_k(x_i) \dot{P}_k(x_i). \qquad (42)$$

With this approximation we are able to solve our differential equation. The approximation for the derivative of the differential function $f(u)$ computed at x_i is given by:

$$\partial_x f[u(x,t_j)]|_{x=x_i} = \left(\partial_u f(u)|_{u=u(x_i,t_j)} \right) \left(\sum_{k=0}^{\frac{n-1}{2}} \tilde{\omega}_k(x_i) \dot{P}_k(x_i) \right). \qquad (43)$$

Note that on the right side the addiction to t is given by the fact, that all used node values belong to the special t_j-coordinate from the left side of the

equation. So also the used polynomials and weights actually depend on t. Now we choose again an explicit Euler method for the part of time discretization, as it is discussed in section 3.1. But this time we assume:

$$y_i(t_{j+1}) = y_i(t_j) + \Delta t_j \partial_t y_i(t)|_{t=t_j}. \tag{44}$$

Combining this with Equation (35) and (42) gives us the final formula:

$$y_i(t_{j+1}) = y_i(t_j) - \Delta t_j \left(\partial_u f(u)|_{u=u(x_i,t_j)}\right) \sum_{k=0}^{\frac{n-1}{2}} \tilde{\omega}_k(x_i)\dot{P}_k(x_i). \tag{45}$$

4 The Implementation

Now we want to present the practical use of the WENO method. Therefore we implemented a c-program for reconstruction, in terms of Section 2.2, and solving partial differential equations (PDE), in terms of Section 3.1. The whole program can be seen in the appendix. The implementation just consists of formulas and functions described in this report. The results will be shown in a qualitative way, quantitative error values will not be computed.

Note that this is just a prototype program. Its only use is to show that the presented concepts do not just work theoretically, but also in practice. So it is not optimized and got a lot of opportunities for improvement.

4.1 WENO Reconstruction of a Step-Function

As example for a reconstructed function we take the piecewise defined polynomial presented in Section 2.2:

$$u(x) = \begin{cases} 2x & , x \leq 0 \\ -20 & , else \end{cases}. \tag{46}$$

The used mesh includes the same amount of points within the analysed range of $[-0.2; 0.2]$, as one in Section 2.2 did. But we use a non equidistant mesh, manually adjusted to the discontinuity. So the highest point density is set around the $x = 0$ and the grid can be referred to as adapted. Using the general form of Definition (1) the grid is given by the recursive formula for $i > 0$:

$$x_i = x_{i-1} + 0.06(1 - 0.93e^{-130x_{i-1}^2}) \ , \ x_0 = -0.19993. \tag{47}$$

Due to generalisation and simplicity, the cell averages, used as start values in the implementation, were computed with a 6th order Newton's integration. The analytical formula for calculating the cell averages of function (46) within the cell $[x_i; x_{i+1}]$ would be:

$$\overline{y}_i = \begin{cases} x_{i+1} + x_i & , x_{i+1} \leq 0 \\ -20 & , x_i \geq 0 \\ \frac{-(x_i)^2 - 20x_{i+1}}{d_{i+1,i}} & , else \end{cases} . \tag{48}$$

The results of the reconstruction process can be seen in Figure 3. The equidistant reconstruction result uses the same grid as it is used in [1] and Section 2.2, but still is computed with our general, non-equidistant version of the WENO program. As said before, both reconstructions are based on the same number of node points. Though, the non equidistant mesh, which is adapted to the discontinuity, produce much better results.

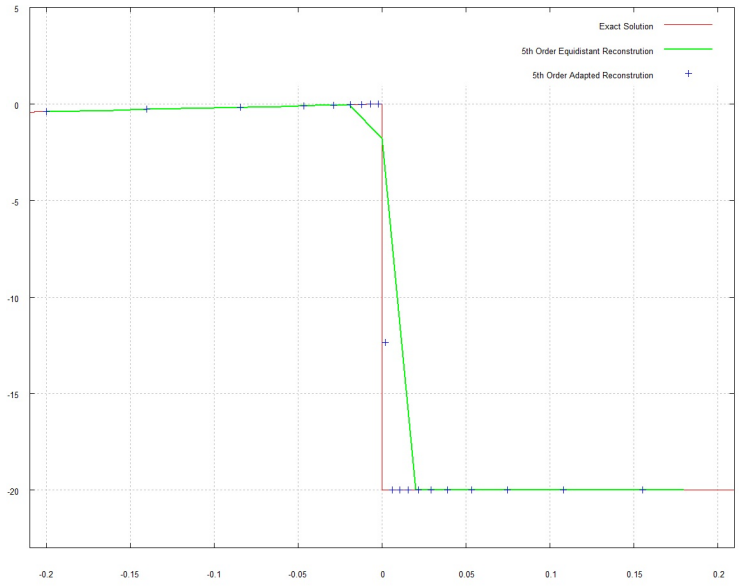

Figure 3: Reconstruction of Function (46) with our implementation of the WENO method compared to the equidistant case

Please note, that we actually just implemented a program to solve PDE's. The reconstruction can be achieved by manually setting the range of the time approximation with '$endt = 0$' to zero. With this modification, just one simple reconstruction is performed, and the actual PDE-solving-part of the implementation is skipped.

4.2 Solution of the Linear Transport Equation

As the main usage of the WENO method is to solve partial differential equations, we also want to present a simple implementation and their results. We have chosen a quite simple physical conservation law called 'linear transport equation'. The complete initial boundary value problem is given by its general form as:

$$\begin{aligned} \partial_t u(x,t) + \partial_x u(x,t) &= 0, \\ u(x,0) &= g(x). \end{aligned} \tag{49}$$

The analytical and exact solution in general is given as:

$$u(x,t) = g(x-t). \tag{50}$$

As one can see, the initial function $g(x)$ is simply shifted to the right, as the time increases. This fact explains the name of the used PDE and gives us a comparable function for the solutions of our WENO implementations. The physical interpretation of the linear transport equation could be a salt concentration in a smooth river, that is transported by its flow, or dust, carried by a steady wind. Of course within this simple model all kinds of diffusion or turbulences are neglected.

We want to present the example of the following initial function for $t_0 = 0$:

$$g(x) = \begin{cases} \frac{x}{2} & , |x| \leq 2 \\ 0 & , else \end{cases}. \tag{51}$$

Obviously $g(x)$ got 2 discontinuities at $x = \pm 2$, where the function makes a relatively huge step from a linear function, to the constant polynomial of zero. The analytical formula for calculating the cell averages of the interval $[x_i; x_{i+1}]$ is:

$$\overline{y}_i = \begin{cases} 0 & , x_{i+1} < -2 \\ 0 & , x_i > 2 \\ \frac{x_i + x_{i+1}}{4 d_{i+1,i}} & , x_i \geq -2 \ \& \ x_{i+1} \leq +2 \\ \frac{1}{d_{i+1,i}} (\frac{(x_{i+1})^2}{4} - 1) & , x_i < -2 \ \& \ x_{i+1} \geq -2 \\ \frac{1}{d_{i+1,i}} (1 - \frac{(x_i)^2}{4}) & , x_i \leq +2 \ \& \ x_{i+1} > +2 \end{cases}. \tag{52}$$

As Equation (50) shows, the exact solution for the initial value problem is a shifted version of the start function:

$$u(x,t) = \begin{cases} \frac{x-t}{2} & , |x-t| \leq 2 \\ 0 & , else \end{cases} . \tag{53}$$

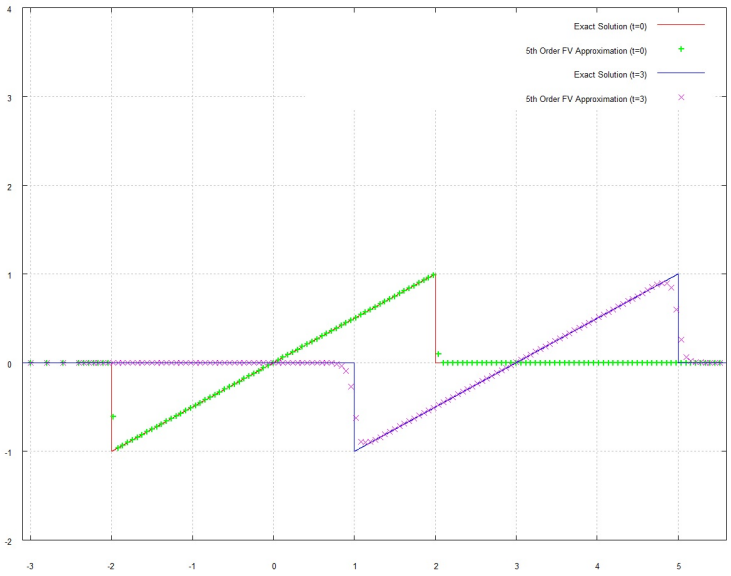

Figure 4: Solution of the PDE (49) with the initial function (51) for $t = 0;3$ using an adapted and non-equidistant grid

This quite extreme example is shown in Figure 4., with the reconstructed initial condition at $t = 0$ and the computed solution for $t = 3$. Here we have chosen another type of equidistant grid, recursive defined for $i > 0$ as:

$$\begin{aligned} x_i &= x_{i-1} + 0.2k \ , \ x_0 = -3, \\ k &= \begin{cases} 0.3 & , -2.5 < x < 5.5 \\ 1 & , else \end{cases} . \end{aligned} \tag{54}$$

This mesh got a small cell range within the movement area of the discontinuities for $t \in [0;3]$ and a bigger cell range outside. The time discretisation was done with an equidistant grid, using the parameter $\Delta t = 0.005$. As obvious, the reconstruction at $t = 0$ fits much better at the exact solution than the approximation for $t = 3$. Especially the sharp edges of the discontinuity became round and smooth. That phenomenon is characteristic for the multiple polynomial reconstructions during the approximation procedure. However, neither the initial function, nor the end approximation shows visible overshoots.

5 Conclusion and Future Prospects

To summarise the results of this report, the WENO method is a good choice to interpolate or reconstruct non smooth functions. The convex combination including linear and non-linear weights reduces overshoots and oscillations near discontinuities to a minimum. The general, non-equidistant equations are mostly long and complicated functions, but contain just simple algebra. Also the different concepts of e.g. interpolation and reconstruction, with different boundary values, have a lot in common and are closely related to each other. Combined with time discretisation priciples, like the Runge-Kutta method or the used explicit Euler method, the WENO concept is also well-suited to solve PDE's.

further the simple, but significant, examples showed the practical use of the introduced WENO method. It was proven, that a implementation of the presented procedures is able to reconstruct functions out of given cell averages and can solve partial differential equations. Even if we did not explicitly compare the WENO method with traditional polynomial reconstruction concepts, the results are free of overshoots and do not show any kind of oscillation around their discontinuities. This makes the WENO method a strong tool in the area of numerical calculation, which should be worked on to improve it even more.

As last point we want to give some proposals for potential future researches on the topic of non-equidistant WENO methods. They mainly include improvements of the concepts presented in this report and our corresponding implementation.

- The whole implementation is just a rough prototype program. Its only use was to show the possibility of a working WENO method. So there is much space for correction and perfection. E.g. the parameter, which do not depend on the concrete values, but on the grid, could be computed within a separated, outsourced function. The equations for all quantities might also be reworked to increase their numerical accuracy.

- We presented the ordinary formulas for the smoothness indicators in Equation (9). But actually this special form was designed and tested for equidistant meshes. There are several ideas to improve the accuracy or computation time of the WENO concept, by using different smoothness indicators. N. *Črnjarić − Žic*, S. *Maćešić* and Bojan *Crnković* e.g.described a different method for the smoothness indication based on non-uniform meshes.[1]

[1]Efficient implementation of WENO schemes to nonuniform meshes - Università degli Studi di Ferrara - Oct. 2007

- The principle of time discretisation can surely be optimised. As mentioned before, the explicit Euler method is actually not used. An implemented Runge-Kutta-Method would extremely raise the accuracy of the PDE solutions. Also an effective implementation of temporal adaptive grids could increase the whole efficiency. The focus of high point density could be shifted to the area of highest interest during the computation. Therefore a lot of optimisation is needed. But the main problem might be to implement a concept to adapt the given cell averages to the newly generated mesh cells.

- We just introduced the WENO method for the starting conditions of given node values and cell averages. Surely, the WENO concept can be specialized also to different kinds of given quantities. Also different types of interpolation and reconstruction polynomials could be used. One could analise, weather it would make sense to change several parts of the discussed WENO method to improve e.g. the accuracy for special problems.

6 Appendix

The coefficients of the smoothness indicators out of the interpolation problem of Section 2.1. They also can be modified and used for the reconstruction problem in Section 2.2 and are closely related to the smoothness indicators for the first derivative interpolation of Section 3.2:

$$
\begin{aligned}
g^0_{j-2,j-2} = \quad & (a^0_{j-2})^2\big[\frac{(d_{j+1,j})^3\Delta x}{15}(5d_{j,j-1}d_{j+1,j-1}+2(d_{j+1,j})^2) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j}-d_{j,j-1})^3+(d_{j+1,j-1}+d_{j+1,j})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^0_{j-1,j-1} = \quad & (a^0_{j-1})^2\big[\frac{(d_{j+1,j})^3\Delta x}{15}(5d_{j,j-2}d_{j+1,j-2}+2(d_{j+1,j})^2) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j}-d_{j,j-2})^3+(d_{j+1,j-2}+d_{j+1,j})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^0_{j,j} = \quad & (a^0_j)^2\big[\Delta x\big(\frac{9((x_{j+1})^5-(x_j)^5)}{5} \\
& +3((x_j)^4-(x_{j+1})^4)(x_{j-2}+x_{j-1}+x_{j+1}) \\
& +\frac{2((x_{j+1})^3-(x_j)^3)}{3}(2(x_{j-2})^2+2(x_{j-1})^2+2(x_{j+1})^2 \\
& \quad +7x_{j-2}(x_{j-1}+x_{j+1})+7x_{j-1}x_{j+1}) \\
& +2((x_j)^2-(x_{j+1})^2)(x_{j-2}+x_{j-1}+x_{j+1})(x_{j-2}(x_{j-1}+x_{j+1}) \\
& \quad +x_{j-1}x_{j+1}) \\
& +(x_{j+1}-x_j)(x_{j-2}(x_{j-1}+x_{j+1})+x_{j-1}x_{j+1})^2\big) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j}-d_{j,j-2}-d_{j,j-1})^3+(d_{j+1,j-2}+d_{j+1,j-1})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^0_{j+1,j+1} = \quad & (a^0_{j+1})^2\big[\Delta x\big(\frac{9(x_{j+1})^5}{5}-3(x_{j+1})^4(x_{j-2}+x_{j-1}+x_j) \\
& +\frac{2(x_{j+1})^3}{3}(2(x_{j-2})^2+2(x_{j-1})^2+2(x_j)^2 \\
& +7x_{j-2}(x_{j-1}+x_j)+7x_{j-1}x_j) \\
& -2(x_{j+1})^2(x_{j-2}+x_{j-1}+x_j)(x_{j-2}(x_{j-1}+x_j)+x_{j-1}x_j) \\
& +x_{j+1}(x_{j-2}(x_{j-1}+x_j)+x_{j-1}x_j)^2 \\
& +\frac{x_j}{15}(5(x_j)^3(x_{j-2}+x_{j-1})-15(x_{j-2})^2(x_{j-1})^2 \\
& \quad -5(x_j)^2(x_{j-2}+x_{j-1})^2-2(x_j)^4)\big) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j-2}+d_{j+1,j-1}+d_{j+1,j})^3-(d_{j,j-2}+d_{j,j-1})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big],
\end{aligned}
\tag{55}
$$

$$
\begin{aligned}
g^0_{j-2,j-1} = \quad & 2a^0_{j-2}a^0_{j-1}\Big[\frac{(d_{j+1,j})^3\Delta x}{30}(5d_{j+1,j-2}d_{j,j-1}+5d_{j,j-2}d_{j+1,j-1}+4(d_{j+1,j})^2)\\
+ \quad & 2d_{j+1,j}(\Delta x)^3(d_{j,j-1}d_{j+1,j-2}+d_{j,j-2}d_{j+1,j-1}+2(d_{j+1,j})^2)\\
+ \quad & 36d_{j+1,j}(\Delta x)^5\Big],\\
g^0_{j-2,j} = \quad & 2a^0_{j-2}a^0_{j}\Big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+1,j-1})^2+7(d_{j,j-1})^2\\
& +5d_{j,j-2}d_{j+1,j-1}+5d_{j+1,j-2}d_{j,j-1}+d_{j,j-1}d_{j+1,j-1})\\
+ \quad & 2d_{j+1,j}(\Delta x)^3(d_{j,j-2}(d_{j,j-1}+d_{j+1,j-1})\\
& +(d_{j+1,j-1})^2+(d_{j+1,j})^2+(d_{j,j-1})^2)\\
+ \quad & 36d_{j+1,j}(\Delta x)^5\Big],\\
g^0_{j-2,j+1} = \quad & 2a^0_{j-2}a^0_{j+1}\Big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+7(d_{j+1,j-1})^2+2(d_{j,j-1})^2\\
& +5d_{j,j-2}d_{j+1,j-1}+5d_{j+1,j-2}d_{j,j-1}+d_{j,j-1}d_{j+1,j-1})\\
+ \quad & 2d_{j+1,j}(\Delta x)^3(d_{j,j-1}d_{j+1,j-2}+d_{j+1,j-1}d_{j+1,j-2}\\
& +(d_{j+1,j})^2+(d_{j,j-1})^2+(d_{j+1,j-1})^2)\\
+ \quad & 36d_{j+1,j}(\Delta x)^5\Big],\\
g^0_{j-1,j} = \quad & 2a^0_{j-1}a^0_{j}\Big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+1,j-2})^2+7(d_{j,j-2})^2\\
& +5d_{j,j-1}d_{j+1,j-2}+5d_{j+1,j-1}d_{j,j-2}+d_{j,j-2}d_{j+1,j-2})\\
+ \quad & 2d_{j+1,j}(\Delta x)^3(d_{j,j-1}(d_{j,j-2}+d_{j+1,j-2})\\
& +(d_{j+1,j-2})^2+(d_{j+1,j})^2+(d_{j,j-2})^2)\\
+ \quad & 36d_{j+1,j}(\Delta x)^5\Big],\\
g^0_{j-1,j+1} = \quad & 2a^0_{j-1}a^0_{j+1}\Big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j,j-2})^2+7(d_{j+1,j-2})^2\\
& +5d_{j+1,j-1}d_{j,j-2}+5d_{j,j-1}d_{j+1,j-2}+d_{j+1,j-2}d_{j,j-2})\\
+ \quad & 2d_{j+1,j}(\Delta x)^3(d_{j+1,j-1}(d_{j+1,j-2}+d_{j,j-2})\\
& +(d_{j,j-2})^2+(d_{j+1,j})^2+(d_{j+1,j-2})^2)\\
+ \quad & 36d_{j+1,j}(\Delta x)^5\Big],\\
g^1_{j-1,j-1} = \quad & (a^1_{j-1})^2\Big[\frac{(d_{j+1,j})^3\Delta x}{15}(2(d_{j+2,j})^2+2(d_{j+2,j+1})^2+d_{j+2,j+1}d_{j+2,j})\\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j}+d_{j+2,j})^3+(d_{j+1,j}-d_{j+2,j+1})^3)\\
+ \quad & 36d_{j+1,j}(\Delta x)^5\Big],
\end{aligned}
$$

$$
\begin{aligned}
g^0_{j,j+1} \;=\;& 2a^0_j a^0_{j+1}\big[\Delta x\big(\frac{3((x_{j+1})^5-(x_j)^5)}{10}\\
&-\frac{2((x_{j+1})^4-(x_j)^4)}{3}(x_{j-2}+x_{j-1})\\
&+\frac{((x_{j+1})^3-(x_j)^3)}{3}((x_{j-2})^2+5x_{j-2}x_{j-1}+(x_{j-1})^2-\frac{x_jx_{j+1}}{2})\\
&-((x_{j+1})^2-(x_j)^2)(x_{j-2}(x_{j-1})^2+x_{j-1}(x_{j-2})^2\\
&\quad-\frac{x_jx_{j+1}}{3}(x_{j-2}+x_{j-1}))\\
&+(x_{j+1}-x_j)((x_{j-2})^2(x_{j-1})^2-x_{j-2}x_{j-1}x_jx_{j+1}))\\
+\;& 2d_{j+1,j}(\Delta x)^3(3d_{j,j-2}d_{j,j-1}+2(d_{j+1,j-1})^2+(d_{j+1,j-2})^2\\
&+d_{j,j-2}d_{j+1,j-2}+d_{j+1,j-1}d_{j,j-2})\\
+\;& 36d_{j+1,j}(\Delta x)^5\big],\\
g^1_{j,j} \;=\;& (a^1_{j+1})^2\big[\Delta x\big(\frac{9((x_{j+1})^5-(x_j)^5)}{5}\\
&-3((x_{j+1})^4-(x_j)^4)(x_{j-1}+x_{j+1}+x_{j+2})\\
&+\frac{2((x_{j+1})^3-(x_j)^3)}{3}(2(x_{j-1})^2+2(x_{j+1})^2+2(x_{j+2})^2\\
&\quad+7x_{j-1}(x_{j+1}+x_{j+2})+7x_{j+1}x_{j+2})\\
&-2((x_{j+1})^2-(x_j)^2)(x_{j-1}+x_{j+1}+x_{j+2})(x_{j-1}(x_{j+1}+x_{j+2})\\
&\quad+x_{j+2})+x_{j+1}x_{j+2})\\
&+(x_{j+1}-x_j)(x_{j-1}(x_{j+1}+x_{j+2})+x_{j+1}x_{j+2})^2)\\
+\;& \frac{4(\Delta x)^3}{9}((d_{j+1,j-1}-d_{j+2,j+1})^3+(d_{j+2,j}+d_{j+1,j}-d_{j,j-1})^3)\\
+\;& 36d_{j+1,j}(\Delta x)^5\big],\\
g^1_{j+1,j+1} \;=\;& (a^1_{j+1})^2\big[\Delta x\big(\frac{9((x_{j+1})^5-(x_j)^5)}{5}\\
&-3((x_{j+1})^4-(x_j)^4)(x_{j-1}+x_j+x_{j+2})\\
&+\frac{2((x_{j+1})^3-(x_j)^3)}{3}(2(x_{j-1})^2+2(x_j)^2+2(x_{j+2})^2\\
&\quad+7x_{j-1}(x_j+x_{j+2})+7x_jx_{j+2})\\
&-2((x_{j+1})^2-(x_j)^2)(x_{j-1}+x_j+x_{j+2})(x_{j-1}(x_j+x_{j+2})+x_jx_{j+2})\\
&+(x_{j+1}-x_j)(x_{j-1}(x_j+x_{j+2})+x_jx_{j+2})^2)\\
+\;& \frac{4(\Delta x)^3}{9}((d_{j+1,j-1}-d_{j+2,j+1})^3+(d_{j+2,j}+d_{j+1,j}-d_{j,j-1})^3)\\
+\;& 36d_{j+1,j}(\Delta x)^5\big],
\end{aligned}
$$

$$
\begin{aligned}
g^1_{j+2,j+2} = \quad & (a^1_{j+2})^2\big[\frac{(d_{j+1,j})^3\Delta x}{15}(2(d_{j,j-1})^2+2(d_{j+1,j-1})^2+d_{j+1,j-1}d_{j,j-1}) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j}-d_{j,j-1})^3+(d_{j+1,j}+d_{j+1,j-1})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^1_{j-1,j} = \quad & 2a^1_{j-1}a^1_j\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+2,j+1})^2+7(d_{j+2,j})^2 \\
& -5d_{j,j-1}d_{j+2,j+1}-5d_{j+1,j-1}d_{j+2,j}+d_{j+2,j}d_{j+2,j+1}) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3((d_{j+2,j+1})^2 \\
& +(d_{j+1,j})^2+(d_{j+2,j})^2-d_{j,j-1}(d_{j+2,j}+d_{j+2,j+1})) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^1_{j-1,j+1} = \quad & 2a^1_{j-1}a^1_{j+1}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+2,j})^2+7(d_{j+2,j+1})^2 \\
& -5d_{j+1,j-1}d_{j+2,j}-5d_{j,j-1}d_{j+2,j+1}+d_{j+2,j+1}d_{j+2,j}) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3((d_{j+2,j})^2+(d_{j+1,j})^2+(d_{j+2,j+1})^2 \\
& -d_{j+1,j-1}(d_{j+2,j+1}+d_{j+2,j})) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^1_{j-1,j+2} = \quad & 2a^1_{j+2}a^1_{j-1}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(-5d_{j+2,j+1}d_{j,j-1}-5d_{j+2,j}d_{j+1,j-1}+4(d_{j+1,j})^2) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3(2(d_{j+1,j})^2-d_{j,j-1}d_{j+2,j+1}-d_{j+2,j}d_{j+1,j-1}) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^1_{j,j+1} = \quad & 2a^0_ja^0_{j+1}\big[\Delta x\big(\frac{3((x_{j+1})^5-(x_j)^5)}{10} \\
& -\frac{2((x_{j+1})^4-(x_j)^4)}{3}(x_{j+2}+x_{j-1}) \\
& +\frac{((x_{j+1})^3-(x_j)^3)}{3}((x_{j+2})^2+5x_{j+2}x_{j-1}+(x_{j-1})^2-\frac{x_jx_{j+1}}{2}) \\
& -((x_{j+1})^2-(x_j)^2)(x_{j+2}(x_{j-1})^2+x_{j-1}(x_{j+2})^2 \\
& \quad -\frac{x_jx_{j+1}}{3}(x_{j+2}+x_{j-1})) \\
& +(x_{j+1}-x_j)((x_{j+2})^2(x_{j-1})^2-x_{j+2}x_{j-1}x_jx_{j+1})\big) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3(2(d_{j+1,j-1})^2+(d_{j+2,j+1})^2-3d_{j+2,j}d_{j,j-1} \\
& +d_{j+2,j}d_{j+2,j+1}-d_{j+1,j-1}d_{j+2,j}) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^1_{j,j+2} = \quad & 2a^1_{j+2}a^1_j\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+1,j-1})^2+7(d_{j,j-1})^2 \\
& -5d_{j+2,j}d_{j+1,j-1}-5d_{j+2,j+1}d_{j,j-1}+d_{j,j-1}d_{j+1,j-1}) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3((d_{j+1,j-1})^2+(d_{j+1,j})^2+(d_{j,j-1})^2 \\
& -d_{j+2,j}(d_{j,j-1}+d_{j+1,j-1})) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big],
\end{aligned}
$$

$$
\begin{aligned}
g^1_{j+1,j+2} = \quad & 2a^1_{j+2}a^1_{j+1}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+7(d_{j+1,j-1})^2+2(d_{j,j-1})^2 \\
& -5d_{j+2,j}d_{j+1,j-1}-5d_{j+2,j+1}d_{j,j-1}+d_{j,j-1}d_{j+1,j-1}) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3(-d_{j,j-1}d_{j+2,j+1}-d_{j+1,j-1}d_{j+2,j+1} \\
& +(d_{j+1,j})^2+(d_{j,j-1})^2+(d_{j+1,j-1})^2) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+3,j+3} = \quad & (a^2_{j+3})^2\big[\frac{(d_{j+1,j})^3\Delta x}{15}(2(d_{j+1,j})^2+5d_{j+2,j+1}d_{j+2,j}) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+2,j+1}-d_{j+1,j})^3+(d_{j+2,j}+d_{j+1,j})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+2,j+2} = \quad & (a^2_{j+2})^2\big[\frac{(d_{j+1,j})^3\Delta x}{15}(5d_{j+3,j+1}d_{j+3,j}+2(d_{j+1,j})^2) \\
+ \quad & \frac{4(\Delta x)^3}{9}((d_{j+1,j}+d_{j+3,j})^3+(d_{j+1,j}-d_{j+3,j+1})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+1,j+1} = \quad & (a^0_{j+1})^2\big[\Delta x\big(\frac{9(x_{j+1})^5}{5}-3(x_{j+1})^4(x_{j+3}+x_{j+2}+x_j) \\
& +\frac{2(x_{j+1})^3}{3}(2(x_{j+3})^2+2(x_{j+2})^2+2(x_j)^2 \\
& \quad +7x_{j+3}(x_{j+2}+x_j)+7x_{j+2}x_j) \\
& -2(x_{j+1})^2(x_{j+3}+x_{j+2}+x_j)(x_{j+3}(x_{j+2}+x_j)+x_{j+2}x_j) \\
& +x_{j+1}(x_{j+3}(x_{j+2}+x_j)+x_{j+2}x_j)^2 \\
& +\frac{x_j}{15}(5(x_j)^3(x_{j+3}+x_{j+2})-15(x_{j+3})^2(x_{j+2})^2 \\
& \quad -5(x_j)^2(x_{j+3}+x_{j+2})^2-2(x_j)^4)\big) \\
+ \quad & \frac{4(\Delta x)^3}{9}((+d_{j+1,j}-d_{j+3,j+1}-d_{j+2,j+1})^3-(d_{j+3,j}+d_{j+2,j})^3) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+3,j} = \quad & 2a^2_{j+3}a^2_j\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+7(d_{j+2,j})^2+2(d_{j+2,j+1})^2 \\
& +5d_{j+3,j+1}d_{j+2,j}+5d_{j+3,j}d_{j+2,j+1}+d_{j+2,j+1}d_{j+2,j}) \\
+ \quad & 2d_{j+1,j}(\Delta x)^3(d_{j+2,j+1}d_{j+3,j}+d_{j+2,j}d_{j+3,j} \\
& +(d_{j+1,j})^2+(d_{j+2,j+1})^2+(d_{j+2,j})^2) \\
+ \quad & 36d_{j+1,j}(\Delta x)^5\big],
\end{aligned}
$$

$$
\begin{aligned}
g^2_{j,j} = \quad & (a^2_j)^2\big[\Delta x\big(\frac{9((x_{j+1})^5-(x_j)^5)}{5} \\
& +3((x_j)^4-(x_{j+1})^4)(x_{j+3}+x_{j+2}+x_{j+1}) \\
& +\frac{2((x_{j+1})^3-(x_j)^3)}{3}(2(x_{j+3})^2+2(x_{j+2})^2+7x_{j+3}(x_{j+2}+x_{j+1}) \\
& \quad +7x_{j+2}x_{j+1}) \\
& +2((x_j)^2-(x_{j+1})^2)(x_{j+3}+x_{j+2}+x_{j+1})(x_{j+3}(x_{j+2}+x_{j+1}) \\
& \quad +x_{j+2}x_{j+1}) \\
& +(x_{j+1}-x_j)(x_{j+3}(x_{j+2}+x_{j+1})+x_{j+2}x_{j+1})^2\big) \\
& + \frac{4(\Delta x)^3}{9}((d_{j+1,j}+d_{j+3,j}+d_{j+2,j})^3-(d_{j+3,j+1}+d_{j+2,j+1})^3) \\
& + 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+3,j+2} = \quad & 2a^2_{j+3}a^2_{j+2}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(5d_{j+3,j}d_{j+2,j+1}+5d_{j+3,j+1}d_{j+2,j}+4(d_{j+1,j})^2) \\
& + 2d_{j+1,j}(\Delta x)^3(d_{j+2,j+1}d_{j+3,j}+d_{j+3,j+1}d_{j+2,j}+2(d_{j+1,j})^2) \\
& + 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+3,j+1} = \quad & 2a^2_{j+3}a^2_{j+1}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+2,j})^2+7(d_{j+2,j+1})^2 \\
& +5d_{j+3,j+1}d_{j+2,j}+5d_{j+3,j}d_{j+2,j+1}+d_{j+2,j+1}d_{j+2,j}) \\
& + 2d_{j+1,j}(\Delta x)^3(d_{j+3,j+1}(d_{j+2,j+1}+d_{j+2,j}) \\
& +(d_{j+2,j})^2+(d_{j+1,j})^2+(d_{j+2,j+1})^2) \\
& + 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+2,j+1} = \quad & 2a^2_{j+2}a^2_{j+1}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+3,j})^2+7(d_{j+3,j+1})^2 \\
& +5d_{j+2,j+1}d_{j+3,j}+5d_{j+2,j}d_{j+3,j+1}+d_{j+3,j+1}d_{j+3,j}) \\
& + 2d_{j+1,j}(\Delta x)^3(d_{j+2,j+1}(d_{j+3,j+1}+d_{j+3,j}) \\
& +(d_{j+3,j})^2+(d_{j+1,j})^2+(d_{j+3,j+1})^2) \\
& + 36d_{j+1,j}(\Delta x)^5\big], \\
g^2_{j+2,j} = \quad & 2a^2_{j+2}a^2_{j}\big[\frac{(d_{j+1,j})^3\Delta x}{30}(2(d_{j+1,j})^2+2(d_{j+3,j+1})^2+7(d_{j+3,j})^2 \\
& +5d_{j+2,j}d_{j+3,j+1}+5d_{j+2,j+1}d_{j+3,j}+d_{j+3,j}d_{j+3,j+1}) \\
& + 2d_{j+1,j}(\Delta x)^3(d_{j+2,j}(d_{j+3,j}+d_{j+3,j+1}) \\
& +(d_{j+3,j+1})^2+(d_{j+1,j})^2+(d_{j+3,j})^2) \\
& + 36d_{j+1,j}(\Delta x)^5\big],
\end{aligned}
$$

$$
\begin{aligned}
g^2_{j,j+1} \;=\;& 2a_j^2a_{j+1}^2\big[\Delta x\big(\frac{3((x_{j+1})^5-(x_j)^5)}{10}\\
&-\frac{2((x_{j+1})^4-(x_j)^4)}{3}(x_{j+3}+x_{j+2})\\
&+\frac{((x_{j+1})^3-(x_j)^3)}{3}((x_{j+3})^2+5x_{j+3}x_{j+2}+(x_{j+2})^2-\frac{x_jx_{j+1}}{2})\\
&-((x_{j+1})^2-(x_j)^2)(x_{j+3}(x_{j+2})^2+x_{j+2}(x_{j+3})^2\\
&\quad-\frac{x_jx_{j+1}}{3}(x_{j+3}+x_{j+2}))\\
&+(x_{j+1}-x_j)((x_{j+3})^2(x_{j+2})^2-x_{j+3}x_{j+2}x_jx_{j+1}))\\
+\;& 2d_{j+1,j}(\Delta x)^3(3d_{j+3,j}d_{j+2,j}+2(d_{j+2,j+1})^2+(d_{j+3,j+1})^2\\
&+d_{j+3,j}d_{j+3,j+1}+d_{j+2,j+1}d_{j+3,j})\\
+\;& 36d_{j+1,j}(\Delta x)^5\big].
\end{aligned}
$$

The reconstruction polynomials of Section 2.2, computed at the node x_{j+1}:

$$
\begin{aligned}
P_0(x_{j+1}) \;=\;& d_{j-1,j-2}\big[a^0_{j-1}d_{j+1,j-2}d_{j+1,j}+a^0_jd_{j+1,j-2}d_{j+1,j-1} \qquad (56)\\
&+a^0_{j+1}(d_{j+1,j-2}d_{j+1,j-1}+d_{j+1,j-2}d_{j+1,j}+d_{j+1,j-1}d_{j+1,j})\big]\overline{y}_{j-2}\\
+\;& d_{j,j-1}\big[a^0_jd_{j+1,j-2}d_{j+1,j-1}\\
&+a^0_{j+1}(d_{j+1,j-2}d_{j+1,j-1}+d_{j+1,j-2}d_{j+1,j}+d_{j+1,j-1}d_{j+1,j})\big]\overline{y}_{j-1}\\
+\;& d_{j+1,j}\big[a^0_{j+1}(d_{j+1,j-2}d_{j+1,j-1}+d_{j+1,j-2}d_{j+1,j}+d_{j+1,j-1}d_{j+1,j})\big]\overline{y}_j,
\end{aligned}
$$

$$
\begin{aligned}
P_1(x_{j+1}) \;=\;& d_{j,j-1}\big[-a^1_jd_{j+2,j+1}d_{j+1,j-1}\\
&+a^1_{j+1}(d_{j+1,j-1}d_{j+1,j}-d_{j+2,j+1}d_{j+1,j}-d_{j+2,j+1}d_{j+1,j-1})\\
&+a^1_{j+2}d_{j+1,j}d_{j+1,j-1}\big]\overline{y}_{j-1}\\
+\;& d_{j+1,j}\big[+a^1_{j+1}(d_{j+1,j-1}d_{j+1,j}-d_{j+2,j+1}d_{j+1,j}-d_{j+2,j+1}d_{j+1,j-1})\\
&+a^1_{j+2}d_{j+1,j}d_{j+1,j-1}\big]\overline{y}_j\\
+\;& d_{j+2,j+1}\big[+a^1_{j+2}d_{j+1,j}d_{j+1,j-1}\big]\overline{y}_{j+1},
\end{aligned}
$$

$$
\begin{aligned}
P_2(x_{j+1}) \;=\;& d_{j+1,j}\big[a^2_{j+1}(d_{j+2,j+1}d_{j+3,j+1}-d_{j+1,j}d_{j+3,j+1}-d_{j+1,j}d_{j+2,j+1})\\
&-a^2_{j+2}d_{j+3,j+1}d_{j+1,j}-a^2_{j+3}d_{j+2,j+1}d_{j+1,j}\big]\overline{y}_j\\
-\;& d_{j+2,j+1}\big[a^2_{j+2}d_{j+3,j+1}d_{j+1,j}\\
&+a^2_{j+3}d_{j+2,j+1}d_{j+1,j}\big]\overline{y}_{j+1}\\
-\;& d_{j+3,j+2}\big[a^2_{j+3}d_{j+2,j+1}d_{j+1,j}\big]\overline{y}_{j+2},
\end{aligned}
$$

$$
\begin{aligned}
Q(x_{j+1}) \;=\;& d_{j-1,j-2}\big[a_{j-1}d_{j+1,j-2}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}\\
&+a_j d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1}\\
&+a_{j+1}(d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1}+d_{j+1,j-2}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}\\
&+d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}-d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\\
&-d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1})\\
&-a_{j+2}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1}\\
&-a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\big]\overline{y}_{j-2}\\
+\;& d_{j,j-1}\big[a_j d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1}\\
&+a_{j+1}(d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1}+d_{j+1,j-2}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}\\
&+d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}-d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\\
&-d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1})\\
&-a_{j+2}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1}\\
&-a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\big]\overline{y}_{j-1}\\
+\;& d_{j+1,j}\big[a_{j+1}(d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1}+d_{j+1,j-2}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}\\
&+d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1}-d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\\
&-d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1})\\
&-a_{j+2}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1}\\
&-a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\big]\overline{y}_{j}\\
-\;& d_{j+2,j+1}\big[a_{j+2}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1}\\
&+a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\big]\overline{y}_{j+1}\\
-\;& d_{j+3,j+2}\big[a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}\big]\overline{y}_{j+2}.
\end{aligned}
$$

The linear weights for the reconstruction problem of Section 2.2 and the FD PDE solving method of 3.2, computed at the node x_{j+1}:

$$
\begin{aligned}
C_0(x_{j+1}) \;=\; & [a_{j-1}d_{j+1,j-2}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1} \\
& +a_j d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1} \\
& +a_{j+1}(d_{j+1,j-2}d_{j+1,j-1}d_{j+2,j+1}d_{j+3,j+1} + d_{j+1,j-2}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1} \\
& +d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}d_{j+3,j+1} - d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1} \\
& -d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1}) \\
& -a_{j+2}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1} \\
& -a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}] \\
& \cdot[a^0_{j-1}d_{j+1,j-2}d_{j+1,j} + a^0_j d_{j+1,j-2}d_{j+1,j-1} \\
& +a^0_{j+1}(d_{j+1,j-2}d_{j+1,j-1} + d_{j+1,j-2}d_{j+1,j} + d_{j+1,j-1}d_{j+1,j})]^{-1},
\end{aligned}
\tag{57}
$$

$$
\begin{aligned}
C_1(x_{j+1}) \;=\; & \big([a^2_{j+2}d_{j+3,j+1}d_{j+1,j} + a^2_{j+3}d_{j+2,j+1}d_{j+1,j}]C_2 \\
& +[a_{j+2}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+3,j+1} \\
& +a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}]\big) \\
& \cdot[a^1_{j+2}d_{j+1,j}d_{j+1,j-1}]^{-1},
\end{aligned}
$$

$$
\begin{aligned}
C_2(x_{j+1}) \;=\; & [a_{j+3}d_{j+1,j-2}d_{j+1,j-1}d_{j+1,j}d_{j+2,j+1}] \\
& \cdot[a^2_{j+3}d_{j+2,j+1}d_{j+1,j}]^{-1}.
\end{aligned}
$$

'solving pde.c'

```
/* PDE solving program - Philip Rupp - 9\11\12 */

#include <stdio.h>
#include <math.h>
#include "reconstruction.c"


/* This is a prototype program to solve partial differential
    equations. Therefore a combination of the
   WENO method and the explicite Euler method was implemented. All
        used functions are outsourced into a file
    called 'reconstruction.c'. */




/* Used Parameters:

    startv:                         Start value of the grid
    endv:                           End value of the grid
    endt:                           End time of the PDE computation
    timer:                          Timestep of the PDE computation
    x:                              Auxiliary quantity
    calcv:                          Function values for determining of
        the cell averages via Newton integration
    i,j:                            Indices for while-routines
    n:                              Number of used grid points
    nt:                             Number of computed timesteps
    grid:                           x-values of the grid-steps
    aval:                           Average values of the used cells
    reulst:                         Endvalues of the computation
    cellr:                          Parameter for the grid scaling

 */


int main (void) {


long i,j,n, nt;
double x, cellr=0.2, startv=-3, endv=7, endt=3, timer=0.005;

    /* Determining of the necessary node points and time steps */

i=0;
x=startv;
while (x < endv){
    x=x+cellr*moduf(x);
 i++;
}
n=i;

nt=endt/timer+1;
```

```
52
55      /* Constructing of the used grid and the interpolated function
            values */

double grid[n], aval[n], result[n], calcv[7*n+7];
58

i=1;
61 grid[0]=startv;
while (i < n){
      grid[i]=grid[i-1]+cellr*moduf(grid[i-1]);
64
      i++;
}
67
i=0;
while (i < n){
70
      calcv[7*i]=val(grid[i]);
      calcv[7*i+1]=val(grid[i]+(grid[i+1]-grid[i])/6.);
73    calcv[7*i+2]=val(grid[i]+2*(grid[i+1]-grid[i])/6.);
      calcv[7*i+3]=val(grid[i]+3*(grid[i+1]-grid[i])/6.);
      calcv[7*i+4]=val(grid[i]+4*(grid[i+1]-grid[i])/6.);
76    calcv[7*i+5]=val(grid[i]+5*(grid[i+1]-grid[i])/6.);
      calcv[7*i+6]=val(grid[i+1]);

79    i++;
}

82 /* Determining of the average values */

average(calcv,n,aval);
85

/* Start of the PDE solving algorithsm */
88

i=0;
while (i < nt){
91
      /* Reconstructing of the given cell averages */

94    reconstruct (n,grid,aval,result,cellr);


97    j=0;
      while (j < n){

100         /* The fundamental PDE solving equation */

            aval[j]=aval[j]-timer*(1./(grid[j+1]-grid[j]))*(pde(result
                [j+1])-pde(result[j]));
103


            j++;
```

```
106         }


109         i++;
    }

112


115         /* Output of the computed results */


118 output(n, grid, result);


121

    printf("\n\n");
    printf("The approximation was successfully computed!\n");
124 printf("All results are stored in a file called 'output.txt'\n");
    printf("The used grid had %d nodes, while the PDE computation used
        %d timeteps.\n", n, nt);
    printf("\n\n");
127


130         return 0;
    }
```

'reconstruction.c'

```
/* PDE solving program - Philip Rupp - 9\11\12 */



/* This is an include-file for the main implementation 'solving
   pde.c'. */


/* The explicit partial dirrerential equation function */

double pde(double x){

    double y;

    y=x;

    return y;
}



/* Data Output in txt-file */

void output(long range, double *grid, double *value){
                long i=0;
                FILE *datei = fopen("output.txt", "w");

                while (i < range){
                      fprintf(datei, "%g\t%g\n",grid[i], value[i])
                          ;
                      i++;
                }
}


/* Non-equidistant grid modulation function */

double moduf(double x){

        double y;

        y=(1-0.93*(exp(-130*pow(0+x,2))));

        if(x < -2.5 || x > 5.5){

         y=1;
        }
        else{

         y=0.3;
        }



```

```
        return y;
}

/* Initial condition for the reconstructed function */

double val(double x){
        double y;

        if (fabs(x) <= 2){
        y=x/2.;
    }
    else {
        y=0;
    }


        return y;
}



/* Cell average determining via Newton Integration */

void average(double *calcval, long n, double *aval){

    long i=0;
    double x,c[7]={41./840,216./840,27./840,272./840,
        27./840,216./840,41./840};

    while (i < n){
        aval[i]=(c[0]*calcval[7*i]+c[1]*calcval[7*i+1]+c[2]*
            calcval[7*i+2]+c[3]*calcval[7*i+3]+c[4]*calcval[7*i+4]+
            c[5]*calcval[7*i+5]+c[6]*calcval[7*i+6]);

        i++;
    }

}



/* Reconstruction of the node points out of given cell averages -
   Boundries are neglected */

/* Used Parameters:

    value0, value1, value2:     Auxiliary quantity
    a,a2:                       Coefficients of the interpolation
        polynomials (a2 for the large stencil)
    alpha:                      Auxiliary quantity for the end
        computation of the convex combination
    x, b:                       Auxiliary quantity
    beta:                       Coefficients of the smoothness
        indicators
    c:                          Linear weights
```

```
    i,k,j:                          Indices for while-routines
    pcoeff:                         Coefficients of the calculated
        polynomials

    Transfered Parameters:

    n:                              Number of used grid points
    grid:                           x-values of the grid-steps
    aval:                           Average values of the used cells
    reulst:                         Endvalues of the computation
    cellr:                          Parameter for the grid scaling

 */


void reconstruct (long n,double *grid, double *aval, double *
    result, double cellr) {

long i=0,k=0,j=0;
double value0[4][n], value1[4][n], value2[4][n], a[4][n-3], a2[6][
    n-5], w[3], alpha[4], pcoeff[9][n-2], x[3], b[3], c[3][n], beta
    [30][n];


/* Computing of the grid-depending parameters a_k and the
    coefficients of beta and P_k */

/* Determining of a_k - a2 belongs to the large stencil*/
i=0;
k=0;
j=0;
while (k < n-3){

    while (i < 4){
        a[i][k]=1;
        j=0;
        while (j<4){
        if (j != i){
        a[i][k]=a[i][k]*1./(grid[k+i]-grid[k+j]);
        }
        j++;
        }
        i++;
    }
    i=0;
    k++;
}


i=0;
k=0;
j=0;
while (k < n-5){

    while (i < 6){
        a2[i][k]=1;
```

```
            j=0;
            while (j<6){
            if (j != i){
            a2[i][k]=a2[i][k]*1./(grid[k+i]-grid[k+j]);
            }
            j++;
            }
            i++;
        }
        i=0;
        k++;
}

/* Determining of the coefficients of P_k - Boundries are
    neglected */

j=2;
while (j < n-2){

b[0]=a[1][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-grid[j]);
b[1]=a[2][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-grid[j-1]);
b[2]=a[3][j-2]*((grid[j+1]-grid[j-2])*(grid[j+1]-grid[j-1])+(grid[
    j+1]-grid[j-2])*(grid[j+1]-grid[j])+(grid[j+1]-grid[j-1])*(grid
    [j+1]-grid[j]));

pcoeff[0][j+1]=(grid[j-1]-grid[j-2])*(b[0]+b[1]+b[2]);
pcoeff[1][j+1]=(grid[j]-grid[j-1])*(b[1]+b[2]);
pcoeff[2][j+1]=(grid[j+1]-grid[j])*b[2];

b[0]=-a[1][j-1]*(grid[j+2]-grid[j+1])*(grid[j+1]-grid[j-1]);
b[1]=a[2][j-1]*((grid[j+1]-grid[j-1])*(grid[j+1]-grid[j])-(grid[j
    +2]-grid[j+1])*(grid[j+1]-grid[j])-(grid[j+2]-grid[j+1])*(grid[
    j+1]-grid[j-1]));
b[2]=a[3][j-1]*(grid[j+1]-grid[j])*(grid[j+1]-grid[j-1]);


pcoeff[3][j+1]=(grid[j]-grid[j-1])*(b[0]+b[1]+b[2]);
pcoeff[4][j+1]=(grid[j+1]-grid[j])*(b[1]+b[2]);
pcoeff[5][j+1]=(grid[j+2]-grid[j+1])*b[2];

b[0]=a[1][j]*((grid[j+2]-grid[j+1])*(grid[j+3]-grid[j+1])-(grid[j
    +1]-grid[j])*(grid[j+3]-grid[j+1])-(grid[j+1]-grid[j])*(grid[j
    +2]-grid[j+1]));
b[1]=-a[2][j]*(grid[j+3]-grid[j+1])*(grid[j+1]-grid[j]);
b[2]=-a[3][j]*(grid[j+2]-grid[j+1])*(grid[j+1]-grid[j]);


pcoeff[6][j+1]=(grid[j+1]-grid[j])*(b[0]+b[1]+b[2]);
pcoeff[7][j+1]=(grid[j+2]-grid[j+1])*(b[1]+b[2]);
pcoeff[8][j+1]=(grid[j+3]-grid[j+2])*b[2];

    j++;
}


/* Determining of the concrete values of the C_k at the node -
```

```
    Boundries are neglected */

j=2;
while (j < n-2){

c[2][j+1]=(a2[5][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-grid[j-1])
    *(grid[j+1]-grid[j])*(grid[j+2]-grid[j+1]))*1./(a[3][j]*(grid[j
    +2]-grid[j+1])*(grid[j+1]-grid[j]));

c[1][j+1]=((a[2][j]*(grid[j+3]-grid[j+1])*(grid[j+1]-grid[j])+a
    [3][j]*(grid[j+2]-grid[j+1])*(grid[j+1]-grid[j]))*c[2][j+1]-a2
    [4][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-grid[j-1])*(grid[j
    +1]-grid[j])*(grid[j+3]-grid[j+1])-a2[5][j-2]*(grid[j+1]-grid[j
    -2])*(grid[j+1]-grid[j-1])*(grid[j+1]-grid[j])*(grid[j+2]-grid[
    j+1]));
c[1][j+1]=c[1][j+1]/(a[3][j-1]*(grid[j+1]-grid[j])*(grid[j+1]-grid
    [j-1]));

c[0][j+1]=a2[1][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-grid[j])*(
    grid[j+2]-grid[j+1])*(grid[j+3]-grid[j+1])+a2[2][j-2]*(grid[j
    +1]-grid[j-2])*(grid[j+1]-grid[j-1])*(grid[j+2]-grid[j+1])*(
    grid[j+3]-grid[j+1]);
c[0][j+1]=c[0][j+1]+a2[3][j-2]*((grid[j+1]-grid[j-2])*(grid[j+1]-
    grid[j-1])*(grid[j+2]-grid[j+1])*(grid[j+3]-grid[j+1])+(grid[j
    +1]-grid[j-2])*(grid[j+1]-grid[j])*(grid[j+2]-grid[j+1])*(grid[
    j+3]-grid[j+1])+(grid[j+1]-grid[j-1])*(grid[j+1]-grid[j])*(grid
    [j+2]-grid[j+1])*(grid[j+3]-grid[j+1]));
c[0][j+1]=c[0][j+1]-a2[3][j-2]*((grid[j+1]-grid[j-2])*(grid[j+1]-
    grid[j-1])*(grid[j+1]-grid[j])*(grid[j+2]-grid[j+1])+(grid[j
    +1]-grid[j-2])*(grid[j+1]-grid[j-1])*(grid[j+1]-grid[j])*(grid[
    j+3]-grid[j+1]));
c[0][j+1]=c[0][j+1]-a2[4][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-
    grid[j-1])*(grid[j+1]-grid[j])*(grid[j+3]-grid[j+1]);
c[0][j+1]=c[0][j+1]-a2[5][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-
    grid[j-1])*(grid[j+1]-grid[j])*(grid[j+2]-grid[j+1]);
c[0][j+1]=c[0][j+1]*1./(a[1][j-2]*(grid[j+1]-grid[j-2])*(grid[j
    +1]-grid[j])+a[2][j-2]*(grid[j+1]-grid[j-2])*(grid[j+1]-grid[j
    -1])+a[3][j-2]*((grid[j+1]-grid[j-2])*(grid[j+1]-grid[j-1])+(
    grid[j+1]-grid[j-2])*(grid[j+1]-grid[j])+(grid[j+1]-grid[j-1])
    *(grid[j+1]-grid[j])));

j++;
}


/* Determining of the coefficients beta_k within [j,j+1] -
    Boundries are neglected - order can be identified by the used a
    -values */

j=2;
while (j < n-2){
b[0]=36*(grid[j+1]-grid[j])*pow(cellr,5);

beta[1][j+1]=4*(pow(cellr,3)/9.)*(pow(grid[j+1]-2*grid[j]+grid[j
    -2],3)+pow(2*grid[j+1]-grid[j-2]-grid[j],3));
beta[1][j+1]=(beta[1][j+1]+b[0])*pow(a[1][j-2],2);
```

```
beta[2][j+1]=(4*pow(cellr,3)/9.)*(pow(grid[j+1]+grid[j-2]-3*grid[j]+grid[j-1],3)+pow(2*grid[j+1]-grid[j-1]-grid[j-2],3));
beta[2][j+1]=(beta[2][j+1]+b[0])*pow(a[2][j-2],2);

beta[3][j+1]=(4*pow(cellr,3)/9.)*(pow(3*grid[j+1]-grid[j]-grid[j-2]-grid[j-1],3)-pow(2*grid[j]-grid[j-1]-grid[j-2],3));
beta[3][j+1]=(beta[3][j+1]+b[0])*pow(a[3][j-2],2);

beta[7][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j]-grid[j-1])*((grid[j]-grid[j-2])+(grid[j+1]-grid[j-2]))+pow(grid[j+1]-grid[j-2],2)+pow(grid[j+1]-grid[j],2)+pow(grid[j]-grid[j-2],2));
beta[7][j+1]=2*(beta[7][j+1]+b[0])*a[1][j-2]*a[2][j-2];

beta[8][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j+1]-grid[j-1])*((grid[j+1]-grid[j-2])+(grid[j]-grid[j-2]))+pow(grid[j]-grid[j-2],2)+pow(grid[j+1]-grid[j],2)+pow(grid[j+1]-grid[j-2],2));
beta[8][j+1]=2*(beta[8][j+1]+b[0])*a[1][j-2]*a[3][j-2];

beta[9][j+1]=4*((grid[j+1]-grid[j])*pow(cellr,3)/2.)*(3*(grid[j]-grid[j-2])*(grid[j]-grid[j-1])+2*pow(grid[j+1]-grid[j-1],2)+pow(grid[j+1]-grid[j-2],2));
beta[9][j+1]=beta[9][j+1]+4*((grid[j+1]-grid[j])*pow(cellr,3)/2.)*((grid[j]-grid[j-2])*(grid[j+1]-grid[j-2])+(grid[j+1]-grid[j-1])*(grid[j]-grid[j-2]));
beta[9][j+1]=2*(beta[9][j+1]+b[0])*a[2][j-2]*a[3][j-2];

beta[11][j+1]=(4*pow(cellr,3)/9.)*(pow(2*grid[j+1]-grid[j-1]-grid[j+2],3)+pow(grid[j+2]+grid[j+1]+grid[j-1]-3*grid[j],3));
beta[11][j+1]=(beta[11][j+1]+b[0])*pow(a[1][j-1],2);

beta[12][j+1]=(4*pow(cellr,3)/9.)*(pow(2*grid[j+1]-grid[j-1]-grid[j+2],3)+pow(grid[j+2]+grid[j+1]+grid[j-1]-3*grid[j],3));
beta[12][j+1]=(beta[12][j+1]+b[0])*pow(a[2][j-1],2);

beta[13][j+1]=4*(pow(cellr,3)/9.)*(pow(grid[j+1]-2*grid[j]+grid[j-1],3)+pow(2*grid[j+1]-grid[j-1]-grid[j],3));
beta[13][j+1]=(beta[13][j+1]+b[0])*pow(a[3][j-1],2);

beta[17][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*(3*(grid[j]-grid[j+2])*(grid[j]-grid[j-1])+2*pow(grid[j+1]-grid[j-1],2)+pow(grid[j+1]-grid[j+2],2));
beta[17][j+1]=beta[17][j+1]+2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j]-grid[j+2])*(grid[j+1]-grid[j+2])+(grid[j+1]-grid[j-1])*(grid[j]-grid[j+2]));
beta[17][j+1]=2*(beta[17][j+1]+b[0])*a[1][j-1]*a[2][j-1];

beta[18][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j]-grid[j+2])*((grid[j]-grid[j-1])+(grid[j+1]-grid[j-1]))+pow(grid[j+1]-grid[j],2)+pow(grid[j]-grid[j-1],2)+pow(grid[j+1]-grid[j-1],2));
beta[18][j+1]=2*(beta[18][j+1]+b[0])*a[1][j-1]*a[3][j-1];

beta[19][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j+1]-grid[
```

```
    j+2])*((grid[j]-grid[j-1])+(grid[j+1]-grid[j-1]))+pow(grid[j
    +1]-grid[j],2)+pow(grid[j]-grid[j-1],2)+pow(grid[j+1]-grid[j
    -1],2));
beta[19][j+1]=2*(beta[19][j+1]+b[0])*a[2][j-1]*a[3][j-1];

beta[20][j+1]=4*(pow(cellr,3)/9.)*(pow(grid[j]-2*grid[j+1]+grid[j
    +2],3)-pow(2*grid[j]-grid[j+1]-grid[j+2],3));
beta[20][j+1]=(beta[20][j+1]+b[0])*pow(a[3][j],2);

beta[21][j+1]=4*(pow(cellr,3)/9.)*(pow(grid[j+1]-2*grid[j]+grid[j
    +3],3)+pow(2*grid[j+1]-grid[j+3]-grid[j],3));
beta[21][j+1]=(beta[21][j+1]+b[0])*pow(a[2][j],2);

beta[22][j+1]=4*(pow(cellr,3)/9.)*(pow(3*grid[j+1]-grid[j]-grid[j
    +3]-grid[j+2],3)-pow(2*grid[j]-grid[j+2]-grid[j+3],3));
beta[22][j+1]=(beta[22][j+1]+b[0])*pow(a[1][j],2);

beta[25][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j+1]-grid[
    j+2])*(grid[j]-grid[j+3])+(grid[j]-grid[j+2])*(grid[j+1]-grid[j
    +3])+2*pow(grid[j+1]-grid[j],2));
beta[25][j+1]=2*(beta[25][j+1]+b[0])*a[3][j]*a[2][j];

beta[26][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j+1]-grid[
    j+3])*((grid[j]-grid[j+2])+(grid[j+1]-grid[j+2]))+pow(grid[j
    +1]-grid[j],2)+pow(grid[j]-grid[j+2],2)+pow(grid[j+1]-grid[j
    +2],2));
beta[26][j+1]=2*(beta[26][j+1]+b[0])*a[1][j]*a[3][j];

beta[27][j+1]=2*(grid[j+1]-grid[j])*pow(cellr,3)*((grid[j+1]-grid[
    j+2])*((grid[j+1]-grid[j+3])+(grid[j]-grid[j+3]))+pow(grid[j]-
    grid[j+3],2)+pow(grid[j+1]-grid[j],2)+pow(grid[j+1]-grid[j
    +3],2));
beta[27][j+1]=2*(beta[27][j+1]+b[0])*a[1][j]*a[2][j];


j++;
}


/* Calculation of the node values of the used auxiliary-integral
    function U */

j=2;
while (j < n-2){


     value0[0][j-2]=0;
     value0[1][j-2]=(grid[j-1]-grid[j-2])*aval[j-2];
     value0[2][j-2]=value0[1][j-2]+(grid[j]-grid[j-1])*aval[j-1];
     value0[3][j-2]=value0[2][j-2]+(grid[j+1]-grid[j])*aval[j];

     value1[0][j-1]=0;
     value1[1][j-1]=(grid[j]-grid[j-1])*aval[j-1];
     value1[2][j-1]=value1[1][j-1]+(grid[j+1]-grid[j])*aval[j];
     value1[3][j-1]=value1[2][j-1]+(grid[j+2]-grid[j+1])*aval[j+1];

```

```
    value2[0][j]=0;
    value2[1][j]=(grid[j+1]-grid[j])*aval[j];
    value2[2][j]=value2[1][j]+(grid[j+2]-grid[j+1])*aval[j+1];
    value2[3][j]=value2[2][j]+(grid[j+3]-grid[j+2])*aval[j+2];


    j++;
    }

/* Computing of the values with the WENO method - Boundries are
   neglected */

j=2;
while (j < n-3){


    b[0]=value0[1][j-2]*value0[1][j-2]*beta[1][j+1]+value0[2][j
        -2]*value0[2][j-2]*beta[2][j+1]+value0[3][j-2]*value0[3][j
        -2]*beta[3][j+1];
    b[0]=b[0]+value0[1][j-2]*value0[2][j-2]*beta[7][j+1]+value0
        [1][j-2]*value0[3][j-2]*beta[8][j+1]+value0[2][j-2]*value0
        [3][j-2]*beta[9][j+1];

    b[1]=value1[1][j-1]*value1[1][j-1]*beta[11][j+1]+value1[2][j
        -1]*value1[2][j-1]*beta[12][j+1]+value1[3][j-1]*value1[3][j
        -1]*beta[13][j+1];
    b[1]=b[1]+value1[1][j-1]*value1[2][j-1]*beta[17][j+1]+value1
        [1][j-1]*value1[3][j-1]*beta[18][j+1]+value1[2][j-1]*value1
        [3][j-1]*beta[19][j+1];

    b[2]=value2[3][j]*value2[3][j]*beta[20][j+1]+value2[2][j]*
        value2[2][j]*beta[21][j+1]+value2[1][j]*value2[1][j]*beta
        [22][j+1];
    b[2]=b[2]+value2[3][j]*value2[2][j]*beta[25][j+1]+value2[3][j
        ]*value2[1][j]*beta[26][j+1]+value2[2][j]*value2[1][j]*beta
        [27][j+1];

    x[0]=pcoeff[0][j+1]*aval[j-2]+pcoeff[1][j+1]*aval[j-1]+pcoeff
        [2][j+1]*aval[j];
    x[1]=pcoeff[3][j+1]*aval[j-1]+pcoeff[4][j+1]*aval[j]+pcoeff
        [5][j+1]*aval[j+1];
    x[2]=pcoeff[6][j+1]*aval[j]+pcoeff[7][j+1]*aval[j+1]+pcoeff
        [8][j+1]*aval[j+2];


    alpha[0]=c[0][j+1]/pow(pow(10,-6)+b[0],2);
    alpha[1]=c[1][j+1]/pow(pow(10,-6)+b[1],2);
    alpha[2]=c[2][j+1]/pow(pow(10,-6)+b[2],2);
    alpha[3]=alpha[0]+alpha[1]+alpha[2];


    result[j+1]=(alpha[0]/alpha[3])*x[0]+(alpha[1]/alpha[3])*x
        [1]+(alpha[2]/alpha[3])*x[2];
```

```
346         j++;
    }

349 /* Boundary values are continued as linear function */


352         x[0]=(result[4]-result[3])/fabs(grid[4]-grid[3]);
            x[1]=(result[n-3]-result[n-4])/fabs(grid[n-4]-grid[n-3]);

355         result[2]=result[3]-fabs((grid[3]-grid[2]))*x[0];
            result[1]=result[2]-fabs((grid[2]-grid[1]))*x[0];
            result[0]=result[1]-fabs((grid[0]-grid[1]))*x[0];
358         result[n-2]=result[n-3]+fabs((grid[n-3]-grid[n-2]))*x[1];
            result[n-1]=result[n-2]+fabs((grid[n-1]-grid[n-2]))*x[1];


361 }
```

References

[1] J. Shi, C. Hu, C.-W. Shu; A technique of treating negative weights in WENO schemes; Journal of Computational Physics 175 (2002) pp. 108-116

[2] E. Carlini, R. Ferretti, G. Russo; A weighted essentially nonoscillatory, large time-step scheme for Hamilton-Jacobi equations; SIAM J. Sci. Comput. 27 (2005) pp. 1071-1082

[3] C.-W. Shu; High order weighted essentially nonoscillatory schemes for convection dominated problems; SIAM Review 51 pp. 82-100

List of Figures