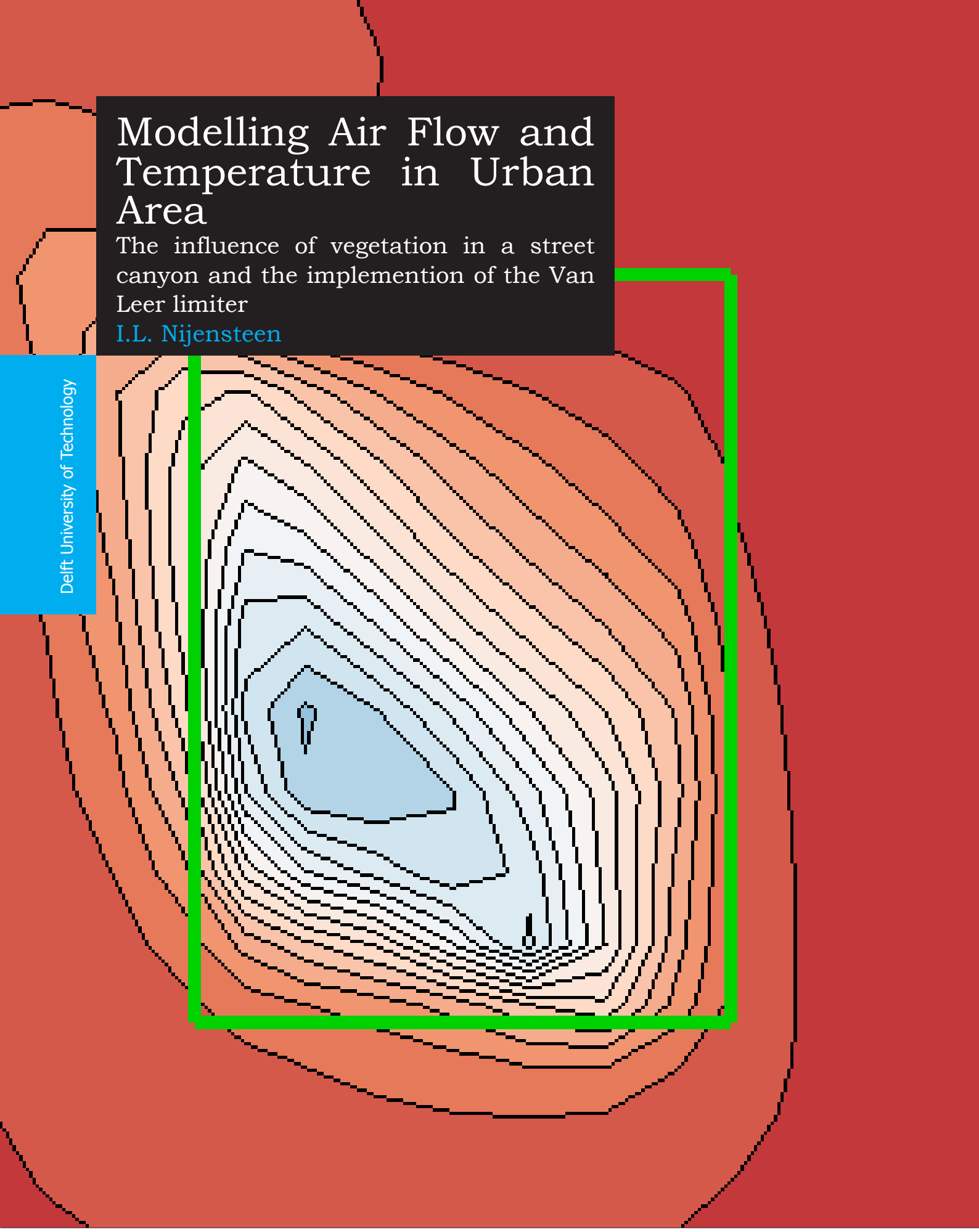
Modelling Air Flow and Temperature in Urban Area

The influence of vegetation in a street canyon and the implemention of the Van Leer limiter

I.L. Nijensteen

Delft University of Technology



TU Delft Delft University of Technology

Challenge the future

Modelling Air Flow and Temperature in Urban Area

The influence of vegetation in a street canyon and the implemention of the Van Leer limiter

by

I.L. Nijensteen (4536940)

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Applied Mathematics and Applied Physics

at the Delft University of Technology,
to be defended publicly on Friday August 21, 2020 at 10:00 AM.

Supervisors:	Prof. dr. ir. C. Vuik,	TU Delft
	Prof. dr. ir. S. Kenjereš,	TU Delft
Thesis committee:	Prof. dr. ir. J. M. Thijssen,	TU Delft
	Assoc. Prof. dr. ir. J. W. van der Woude,	TU Delft

TU Delft Delft University of Technology

Abstract

In this thesis, a model for air flow and temperature in urban areas is studied. In particular, the influence of vegetation in a street canyon is investigated. This includes the effect of vegetation on air flow and the cooling property of vegetation to lower temperature in street canyons.
Computational Fluid Dynamics simulations with vegetation in a street canyon (green facades and trees) are performed. To model this, the Reynolds Averaged Navier Stokes equations in 3D are closed using the k-ε turbulence model with source and sink terms to account for the effects of vegetation on air flow. In the thermal equation, the Simple Gradient Diffusion Hypothesis is used to close the equations and a cooling power term is introduced to account for the transpirational cooling property of vegetation.
The results regarding the vertical velocities in the street canyon are compared with earlier simulations by Gromke et al.

The simulations involving temperature showed significant effects of the vegetation in the street canyon. At street level, the green facades yielded a stronger cooling effect than trees. However, the cooling effect of trees is stronger halfway the canyon and just above the canyon in comparison with green facades. An important note is that the temperature drop inside the tree canopy strongly suggests that the cooling power in the simulation is modelled stronger than one would expect in reality.

Another part of this research is the implementation of a Higher Order Scheme. This is done with the Van Leer limiter. The simulations with this new numerical method yielded very similar results compared with the UDS simulations. One outlier is noticed in the velocity in the y-direction for the empty street canyon. A possible explanation could be the sensitivity of the method near the ground.

The study of the effect of vegetation in a street canyon could be extended in the future by comparing combinations of vegetation and adjusting the parameters of the green facades. Furthermore, the Van Leer limiter could be extended to the temperature part of the simulations in the future, since sharper gradients are expected here, so the improvement with respect to UDS and QUDS could be more significant. Finally, also other flux limiters, such as the Koren scheme, could be tested.

Contents

List of Symbols

Acronyms

CDS	Central Difference Scheme
CFD	Computational Fluid Dynamics
FVM	Finite Volume Method
LAD	Leaf Area Density
LUDS	Linear Upwind Difference Scheme
MUSCL	Monotonic Upstream-centered Scheme for Conservation Laws
QUDS	Quadratic Upwind Difference Scheme
QUICK	Quadratic Upstream Interpolation for Convective Kinematics
RANS	Reynolds Averaging Navier Stokes
TKE	Turbulent Kinetic Energy
TVD	Total Variation Diminishing
UDS	Upwind Difference Scheme
UHI	Urban Heat Island

Physics Constants

κ	Von Kármán constant	0.435
Pr	Prandtl number	0.71
Pr	Turbulent Prandtl number	0.9
σ_k	Model constant for the k-equation	1.0
σ_ϵ	Model constant for the ϵ-equation	1.3
C_μ	Model constant of the eddy viscosity	0.09
C_D	Drag coefficient	0.3
$C_{1\epsilon}$	Model constant for the ϵ-equation	1.44
$C_{2\epsilon}$	Model constant for the ϵ-equation	1.92

Parameters

δ_{ij}	Kronecker delta	$-$
ϵ	Dissipation of turbulent kinetic energy	$\mathrm{m}^2/\mathrm{s}^3$
$\hat{\tau}$	Shear stress	$\mathrm{kg}/(\mathrm{m}\cdot\mathrm{s}^2)$
$\hat{p}$	Instantaneous pressure	$\mathrm{kg}/(\mathrm{m}\cdot\mathrm{s}^2)$
$\hat{u}$	Instantaneous velocity	m/s

ν	Kinematic viscosity	m^2/s
ν_t	Turbulent viscosity	m^2/s
ω	Under-relaxation parameter	$-$
ρ	Density of air	kg/m^3
a	Leaf area density	$1/\mathrm{m}$
k	Turbulent kinetic energy	$\mathrm{m}^2/\mathrm{s}^2$
P_0	Transpirational energy losses of vegetation	W/m^2
P_c	Volumetric cooling power	W/m^3
P_k	Production of turbulent kinetic energy	$\mathrm{m}^2/\mathrm{s}^3$
S_ϵ	Production of turbulent dissipation related to effects of vegetation	$\mathrm{m}^2/\mathrm{s}^3$
S_k	Production of turbulent kinetic energy related to effects of vegetation	$\mathrm{m}^2/\mathrm{s}^3$
$S_{m,i}$	Sink/source term	$\mathrm{m}^2/\mathrm{s}^3$

1

Introduction

In the summer of 2019 various heat records were broken in Europe. Although a part of humanity enjoys these hot summers, it is confirmed by various studies how human-induced climate change has altered the likelihood and intensity of heat waves during summer.[1] In Figure 1.1, the intensity of high temperatures in 2019 is visualised. Extremely high temperatures have many negative effects on the quality of human life. Especially in urban area, these high temperatures during summer urge for changes in the blue print of urban environment to ensure quality of life.
The study of temperature distribution in the urban environment is often characterised using the concept of Urban Heat Islands. An Urban Heat Island (UHI) is an urban area that is significantly warmer than its surrounding rural area due to human activities. The UHI is defined as the difference in air temperature between the urban canyon and the rural environment. [2] The UHI generally depends on urban characteristics, such as street geometry and urban vegetation fraction.
Furthermore, the Urban Heat Island decreases air quality with the increasing concentration of pollutants such as ozone, and decreases water quality.

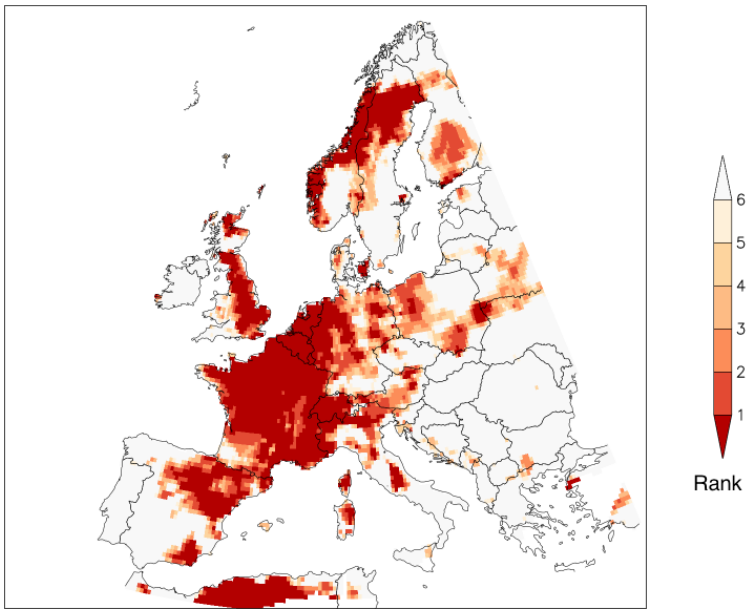


Figure 1.1: Rank of annual maximum temperatures observed in Europe in 2019 compared to 1950 –2018. Source: [1]

One of the key suggestions for solving the problem of Urban Heat Islands is to integrate green zones in the city. Vegetation decreases local temperature by shading and evapotranspiration, which results in relieving the urban heat island effect. [3] Besides the cooling property of vegetation, green zones in the city have many other benefits. Implementing green zones will improve the air quality, due to the filtering capacity of the vegetation, which lowers the concentration of pollutants. Furthermore, carbon

dioxide is absorbed by the process of photosynthesis. Green zones decrease the chance of inundation, reduces noise pollution and contributes to social cohesion. [4]

However, the problem of UHI is not simply solved by adding vegetation. The study of Gromke and Blocken [5] showed that in a simple street canyon with buildings on both sides the concentration of pollutants can increase due to the presence and specific configuration of vegetation. Such effects can occur by the influence of the flow field on the dispersion of the heat and pollutant concentration. Thus, vegetation can have a significant effect on the flow field. [6] So, when searching for a solution to the UHI problem, specific configurations of the vegetation in the street canyon have to be carefully tested to determine their effectiveness.

Figure 1.2: A park with green zones in the city is often a good place to cool down during heat summer days. Source: Wageningen University.

In this thesis, a model for the analysis of air flow and temperature in urban area will be studied. To arrive at this point, a turbulence model will be described. The obtained equations from this model cannot be solved analytically. Using several numerical methods, a model is built which can compute numerical solutions.

The goal of this study is to gain insight in the model of the air flow and temperature in street canyons. This research aims to implement total variation diminishing in the numerical solving methods to the already existing model to improve the accuracy of the solutions.

In order to reach this goal, several research questions are specified:

1. How to derive a model to solve the air flow in urban area?
2. How to derive a model to solve the temperature in urban area?
3. How can we use numerical methods to solve the differential equations from the derived model for air flow and temperature?
4. How can we improve the existing RANS Solver with the implementation of a Higher Order Scheme (Van Leer Limiter)?

2

Air Flow in Urban Area

To model air flow in urban area the behaviour of this physical phenomenon is studied. It is known that to describe the fluid dynamics of flows the Navier Stokes equations need to be solved. These transport equations are described in Section 2.1. These equations cannot be solved analytically. However, it is possible to solve these equations numerically using Computational Fluid Dynamics (CFD). For the purpose of this research, the Reynolds averaged Navier-Stokes (RANS) approach is used, which is treated in Section 2.2. All turbulence scales are modelled and therefore the computational costs are reduced enormously. A disadvantage of the approach is that it lacks the influence of smaller scale motions. This chapter explains the transport equations and the RANS approach, and discusses a way to implement sink/source terms in the equations due to vegetation.

2.1. Transport equations

A good starting point for this derivation is the Navier-Stokes equations [7] for the instantaneous velocity, $\hat{u}_i$:

$$\underbrace{\frac{\partial \hat{u}_i}{\partial t}}_{\text{acceleration}} + \underbrace{\hat{u}_j \frac{\partial \hat{u}_i}{\partial x_j}}_{\text{convection}} = - \underbrace{\frac{1}{\rho}\frac{\partial \hat{p}}{\partial x_i}}_{\text{pressure gradient}} + \underbrace{\frac{1}{\rho}\frac{\partial \hat{\tau_{ij}}}{\partial x_j} + \hat{S}_{m,i}}_{\text{diffusion}} \tag{2.1}$$

The subscripts i and j denote the three dimensions, since modelling flow through urban area is three dimensional problem. For the purpose of this research, air is assumed to be incompressible. This means that the density ρ is constant. $\hat{p}$ is the instantaneous pressure, $\hat{\tau}_{ij}$ the shear stress and $S_{m,i}$ is the sink/source term. Finally, shear stress for stationary and incompressible fluids can be defined as follows:

$$\hat{\tau}_{ij} = \rho\nu\left(\frac{\partial \hat{u}_i}{\partial x_j} + \frac{\partial \hat{u}_j}{\partial x_i}\right) \tag{2.2}$$

In this equation, ν is the kinematic viscosity. [8]

2.2. Reynolds Averaged Navier-Stokes approach

Equation 2.1 cannot be solved analytically generally spoken, thus numerical simulations are used. The first possibility is to use Direct Numerical Simulations (DNS). With this method, the velocity is directly calculated from the Navier-Stokes equations. However, this method can only be useful, when the values of the Reynolds number are smaller than 10^4. This depends among other things on the characteristics of turbulent flow. Turbulent flow, especially in the atmospheric boundary layer, is heavily influenced by coherent flow structures such as eddies. [9] An eddy is the swirling of a fluid and the reverse current created of a fluid in a turbulent regime. To acquire reliable results the grid parameters Δx, Δy and Δz must be very small. These constraints result in time consuming calculations, which explains why this method is only used for values of the Reynolds number smaller than 10^4. In an average urban area,

the Reynolds number easily exceeds this value, so a different approach is needed to do computations.

Accordingly, a second possibility, the *Reynolds averaged Navier-Stokes equations* (RANS), is used to calculate the velocity. These equations are often used in environmental engineering. To obtain these equations, the property of turbulent flow structures is used in such a way that all variables, such as velocity, pressure and temperature, continuously fluctuate in a stochastic way. This means that exact values cannot be predicted, which also explains why turbulent flow is so difficult to model. By using the environmental engineering approach, it is assumed that these small fluctuations are not needed. Only the general behaviour of the flow structure is needed for the model. This general behaviour can be calculated using the Reynolds averaged Navier-Stokes equations. [8]

In the RANS-model, an arbitrary instantaneous property $\hat{x}$ (e.g. velocity or pressure) is written as $\hat{x} = \bar{x} + x'$, where $\bar{x}$ is the time averaged part and x' the fluctuating part. This is visualised in Figure 2.1.

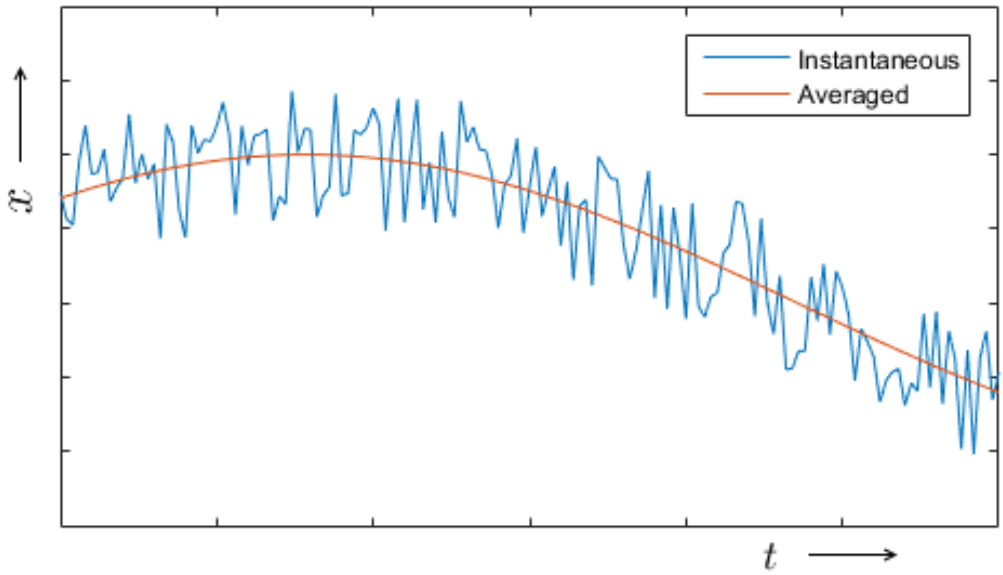


Figure 2.1: An arbitrary instantaneous property with the averaged part in blue. Source: Filipovic [8].

Every instantaneous property can be split in these two parts, which yields the following equation from the original equation 2.1:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}[\nu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) - \overline{u_i' u_j'}] + S_{m,i} \tag{2.3}$$

where ν is the viscosity and $\overline{u_i' u_j'}$ are the Reynolds stresses.

A complete derivation of this equation can be found in literature, such as [8]. With this set of equations, $\overline{u_i}$ and $\overline{u_j}$ for $i, j = 1, 2, 3$ can be computed. Under the assumption of constant density, the continuity equation gives the following:

$$\nabla \cdot \overline{\mathrm{u}} = 0 \tag{2.4}$$

The term $-\overline{u_i' u_j'}$ is added in equation 2.3, compared to equation 2.1. This term is often called the Reynolds stress and without this term, the solutions would always result in laminar flow. This nonlinear term requires additional modelling to close the RANS equation in order to have a unique solution. However, this is still less computationally expensive than solving the Navier-Stokes equations directly. The Reynolds stress gives rise to a new problem, because the equations of 2.3 are not closed. Such a problem is called a turbulence closure problem. [6]

2.3. The k-ε turbulence model

Turbulence closure models are used to find a solution to the Navier-Stokes equations in the airflow model. There are many turbulence closure models, but in this research the k-ε turbulence model of Launder and Sharma is used, since in this field of study it is the mostly used model. Two new differential equations are introduced in order to close the Reynolds averaged Navier-Stokes equations. The first differential equation is for the turbulent kinetic energy (TKE), k. This is a measure of the turbulence of the flow. The second differential equation is for the turbulent dissipation, ϵ, which is a measure of amount of turbulence that is dissipated due to viscous effects. [8] The Reynolds stresses, $-\overline{u_i' u_j'}$, can now be defined in terms of these two variables. This approach of the k-ε turbulence model is commonly used and in particular accurate for incompressible constant-density flows. Since air flow through urban area also has these properties, the k-ε model is suitable for this research.

The k-ε model makes the following assumption:

$$\overline{u_i' u_j'} = \frac{2}{3} k \delta_{ij} + \nu_t \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{2.5}$$

In this equation, δ_{ij} is the Kronecker delta, k the turbulent kinetic energy (often referred to as TKE) and ν_t the turbulent viscosity. With this rewritten problem, a closed set of equations is obtained if k and ν_t are known. For the eddy viscosity the following approximation is used:

$$\nu_t = C_\mu \frac{k^2}{\epsilon} \tag{2.6}$$

This expression follows from dimensional analysis. Quantity k^2/ϵ is a unit of time and can be seen as a time scale of the turbulence. C_μ is a model constant.

k-equation

This model assumes that turbulence can be expressed by the turbulent kinetic energy k. This energy is dependent on $\overline{u_i' u_j'}$ for $i, j = 1, 2, 3$, but it can be simplified using the following approximation (with $i = 1, 2, 3$):

$$k = \frac{1}{2} \overline{u_i^2} \tag{2.7}$$

However, k is still unknown. By deriving a partial differential equation for k, k can be solved. This can be done by substituting $x' = \bar{x} + x'$ for all instantaneous properties into equation 2.1 and then subtracting equation 2.3 from this.

After some other operations, which are beyond the scope of this project, the following partial differential equation is found for k:

$$\frac{\partial k}{\partial t} + u_j \frac{\partial k}{\partial x_j} = \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] + P_k - \epsilon + S_k \tag{2.8}$$

Often this equation is referred to as the k-equation. Some new variables and constants are introduced in this equation. P_k is the production of turbulent kinetic energy and σ_k another model constant for the k-equation. S_k is the production term of turbulent kinetic energy related to the effects of vegetation and ϵ is the dissipation of turbulent kinetic energy into heat due to viscosity. The production term, P_k, is defined as following:

$$P_k = -\overline{u_i' u_j'} \frac{\partial u_i}{\partial x_j} \tag{2.9}$$

For now, the turbulent dissipation ϵ, $S_{m,i}$ and the production of turbulent kinetic energy resultant from vegetation, P_k^v, are unknown terms.

ε-equation

Finally, an expression for the turbulent diffusion ϵ is derived. This is done by deriving a partial differential equation for ϵ. In the previous subsection 2.3 the turbulent diffusion is defined as follows:

$$\epsilon = \nu(\frac{\partial \overline{u_i'}}{\partial x_j})^2 \tag{2.10}$$

To obtain a partial differential equation for turbulent dissipation, the equation has to be differentiated with respect to x_j, then multiplied by $\frac{\partial u_i'}{\partial x_j}$ and lastly averaged. [8] Since this whole derivation is rather complex, it is left out here. The final result is the following:

$$\frac{\partial \epsilon}{\partial t} + u_j \frac{\partial \epsilon}{\partial x_j} = \frac{\partial}{\partial x_j}[(\nu + \frac{\nu_t}{\sigma_\epsilon})\frac{\partial \epsilon}{\partial x_j}] + C_{1\epsilon} P_k \frac{\epsilon}{k} - C_{2\epsilon}\frac{\epsilon^2}{k} + S_\epsilon \tag{2.11}$$

In this equation, $C_{1\epsilon}$, $C_{2\epsilon}$ and σ_ϵ are model constants. S_ϵ is the production of turbulent dissipation related to vegetation effects. With this last equation, a closed model has been formulated, if there is no vegetation. Some model constants have standard values in the $k-\epsilon$ model, which are given in the Table 2.1.

The effect of vegetation on the flow field is included by using the sink/source terms in the transport equations ($S_{m,i}$, S_k and S_ϵ). The vegetation acts on the flow through various interactions. These interactions are discussed in more detail in the next section.

2.4. Vegetation effects

Until now, the approach in modelling air flow has been generally applicable. However, in this research the influence of vegetation on air flow is studied in particular. In order to do so, a proper way to model vegetation is examined. First, some properties of green zones are discussed.

Vegetation differs from most other solid objects, when looking at the interaction with air flow. Before air flow arrives at an object (e.g. a tree or building), it has a mean kinetic energy. Typically, at such an obstruction the air encounters a lot of resistance, so a great part of the air is slowed down, which results in shear stresses around the object. This leads to eddies with the same scale as the object dimensions and is called *shear*. [10] If this happens, energy is converted in both situations (vegetation or solid objects). The energy contained by this shear, is often referred to as shear kinetic energy.

However, vegetation interacts in a very different way with the air, since the resistance of individual leaves cannot be modelled as a solid object. The air is certainly slowed down, but the eddies are much smaller, since the leaves have much smaller dimensions than the totality of the vegetation. This results in many smaller eddies inside and behind the vegetation. These eddies as a whole are called the *wake*. This leads to much smaller turbulence cascade, which results in reaching the dissipative state faster. This process is unique to porous obstacles. [10]

The sink/source term in the moment equation is the form drag force. This force occurs when a leaf is positioned perpendicular to the air flow. Because the pressure upwards is larger than the pressure downwards on the leaf, a force is exerted on the leaf in the direction of the flow. Furthermore, there is a viscous drag force present. This force occurs when a leaf is parallel positioned to the air flow. Due to the no slip boundary, the air flow must have no velocity. This results in a shear force dependent on the fluid viscosity. This shear force has a significant contribution for low Reynolds numbers, so it can be neglected for now. This results in the following equation for the body forces F_i:

$$F_i = \frac{1}{2} C_D a |u| u_i \tag{2.12}$$

The sink/source terms of k and ϵ depend on the shortcut of the cascade process and the generation of large eddies due to the form drag. The eddies can be seen as the 'swirl' in the turbulent flow behind the object. The energy hold by this wake, is called the wake kinetic energy. So, mean flow motion is converted into wake turbulence.

The vegetation object itself generates eddies of the size of the obstacle itself due to shear. The eddies generated by the leaves are dissipated more rapidly due to the shortcut of the energy cascade. This shortcut of the cascade process scales with the product of the velocity and the turbulent kinetic energy. The formula for turbulent kinetic energy source term is:

$$S_k = \frac{1}{2} C_D a(\beta_p |u^3| - \beta_d |u| k) \tag{2.13}$$

where β_p and β_d are model constants that express the significance of the processes. β_p is a production constant and β_d is a destruction constant.[11] From this equation, the sink/source term of ϵ can be derived:

$$S_\epsilon = \frac{1}{2} C_D a(C_{4\epsilon} \beta_p |u^3| \frac{\epsilon}{k} - C_{5\epsilon} \beta_d |u| \epsilon) \tag{2.14}$$

where $\frac{k}{\epsilon}$ is the characteristic time and $C_{4\epsilon}$ and $C_{5\epsilon}$ are model constants.

The modelling of the velocity field depends strongly on the values of the introduced constants. The model constants proposed by Katul are proved to have a good agreement with measured data, according to earlier research by Kenjeres and Ter Kuile. [6]

Table 2.1: Model constants as proposed by Kenjeres and Ter Kuile [2013]

C_D	β_p	β_d	$C_{4\epsilon}$	$C_{5\epsilon}$
0.3	1.0	5.1	0.9	0.9

3

Temperature distribution in Urban Areas

The temperature in an environment is influenced by the presence of vegetation. Plants can cool the environment by a process known as transpiration. When the atmosphere heats up, plants will release excess water from their leaves into the air.

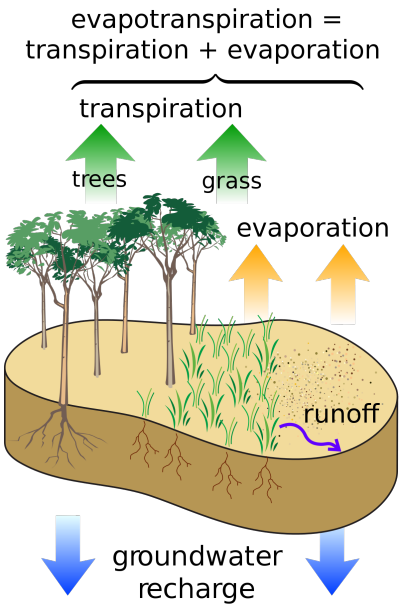


Figure 3.1: Water cycle of the Earth's surface is showed, with the individual components of transpiration and evaporation that make up evapotranspiration. Source: Wikipedia

By releasing this evaporated water, plants cool themselves and the surrounding environment. This process is visualised in Figure 3.1. It is similar to sweating for animals: by sweating they cool their skin. This cooling process can be seen as a temperature sink. In this Chapter, the equation of thermal energy is discussed and is closed using the Simple Gradient Diffusion Hypothesis. Finally, the influence of vegetation on the temperature distribution is described.

3.1. Equations for Thermal Energy

The equation for thermal energy is used to find the temperature distribution:

$$\frac{\partial \hat{T}}{\partial t} + \hat{u_i}\frac{\partial \hat{T}}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\frac{\nu}{\mathrm{Pr}}\frac{\partial \hat{T}}{\partial x_i}\right) \tag{3.1}$$

where $\hat{T}$ is the instantaneous temperature, ν the kinematic viscosity and Pr the dimensionless Prandtl number. The Prandtl number is defined as the ratio of momentum diffusivity to thermal diffusivity:

$$\mathrm{Pr} = \frac{\nu}{a} \tag{3.2}$$

where a is the thermal diffusivity ($a = \frac{k}{\rho c_p}$). In Equation 3.1, it is assumed that ρ is constant and furthermore, thermal buoyancy is neglected. As a result of this assumption, the temperature distribution depends on the velocity field, but vice versa the velocity field does not depend on temperature.

For the Reynolds averaged temperature, the following equation is obtained:[8]

$$\frac{\partial \overline{T}}{\partial t} + \overline{u_i}\frac{\partial \overline{T}}{\partial x_i} = \frac{\partial}{\partial x_i}[\frac{\nu}{\mathrm{Pr}}\frac{\partial \overline{T}}{\partial x_i} - \overline{T'u_i'}] + S_T \tag{3.3}$$

In this equation, $\overline{T'u_j'}$ are the thermal stresses and S_T is the production term that accounts for the cooling effects of the transpiration from vegetation. The thermal stresses are unknown, so as with the $k-\epsilon$-model, a model is needed to calculate these terms.

3.2. Closing the Reynolds averaged thermal energy equation

To close the Reynolds averaged thermal equation (3.3), one can choose between two models: the generalised gradient diffusion hypothesis and the Simple Gradient Diffusion Hypothesis. In this thesis, the latter is used. Both hypotheses are relatively easy to implement, since they do not introduce additional transport equations. The Simple Gradient Diffusion Hypothesis states the following for the thermal stresses:

$$\overline{T'u_j'} = -\frac{\nu_t}{\mathrm{Pr}_t}\frac{\partial T}{\partial x_j} \tag{3.4}$$

where Pr_t is the turbulent Prandtl number, defined as the ratio between the momentum eddy diffusivity and the heat transfer eddy diffusivity. In general, a turbulent Prandtl number between 0.7 and 0.9 is used. For this research, $\mathrm{Pr}_t = 0.71$ is used, in accordance with a paper of Manickathan et al. [12].

A disadvantage of the Simple Gradient Diffusion Hypothesis is that it is less effective in simulations with strong buoyancy effects. However, buoyancy is neglected in this thesis, so this will not effect the performance of the simulations in this thesis.

3.3. Vegetation: sink and source terms

The temperature model is closed, if the sink and source terms from vegetation effects are ignored. In the momentum equations, vegetation has effects through production terms for turbulent kinetic energy and turbulent dissipation. Vegetation also effects the thermal equations. In general, the presence of vegetation can decrease temperature. This effect can be obtained by providing shade and transpiration, as discussed earlier in this Chapter. Due to the vegetation, heat is absorbed to evaporate moisture that is transpired through the leaves of the vegetation. Heat is extracted from the air and the vegetation itself. Furthermore, heat is exchanged between the vegetation and the air.

From the argumentation above, it can be concluded that the thermal sink term S_T depends on the leaf area density a. For a low value of the leaf area density, there is limited surface, where transpiration can occur, so the cooling effect is less than for a larger leaf area density. The volumetric cooling power, P_c, is used to account for the leaf area density. P_c represents the ability of vegetation to cool the surrounding air per cubic meter. With the assumption that the volumetric cooling power P_c is linearly depended on the leaf area density a, the following relation is obtained:

$$P_c = P_0 \cdot a \tag{3.5}$$

where P_0 is a measure of transpirational energy losses of the vegetation per unit leaf area density. From the research of Filipovic [8], the value of $P_0 = 250 W/m^2$ is used.

The thermal sink term also depends on the temperature difference between the leaves and the air, the heat capacity of the air c_p and the velocity of the air flowing along the leaves.

Using the Buckingham Pi Theorem, dimensional analysis gives the following dependencies:

$$S_T = K_P \frac{P_c}{c_p} \left(\frac{c_p(T - T_{leaf})}{|\overline{U}|^2}\right)^{\alpha} \tag{3.6}$$

The scaling factor α is often assumed to be 0, which means that the thermal heat sink of the vegetation is simply an empirical constant. [10] K_p is a model constant that will set to -1, since this is a sink term. Using $\alpha = 0$ yields the following expression for the thermal sink term:

$$S_T = -\frac{P_c}{c_p} \tag{3.7}$$

In Table 3.1 the values for the model constants can be found.

Table 3.1: Model constants for the thermal energy equations

Pr	Pr_t	K_p	α	c_p	P_0
-	-	-	-	m^2/Ks^2	W/m^2
0.71	0.9	-1	0	1005	250

This concludes the model of the temperature distribution. With the use of Reynolds averaging, the simple gradient diffusion hypothesis and a description of the sink term caused by vegetation, a closed model to describe temperature is derived.

4

Numerical Methods

To solve the transport and heat equations, numerical methods are used. In this research, the geometries and flows are too complex to solve the derived equations analytically. Therefore, numerical methods are needed to solve the partial differential equations. A numerical solution is evaluated by both the consistency and stability. First of all, the Finite Volume Method (FVM) is the most fundamental technique that is pointed out and discussed in Section 4.1. When using the finite volume method, a differencing scheme should be specified. The choice of a difference scheme can be delicate and in Section 4.2 the most common schemes are treated.

4.1. Finite Volume Method

The transport equations are solved numerically by using the Finite Volume Method (FVM). This means that the calculation of the full domain is divided into a finite number of so called control volumes or grid cells. For each of these cells, discretised transport equations are integrated in time and space and then solved.

4.1.1. Discretisation

First of all, the differential form of the transport equations is expressed as:

$$\underbrace{\frac{\partial}{\partial t}(\rho\phi)}_{\text{unsteady}} + \underbrace{\frac{\partial}{\partial x_j}(\rho u_j \phi)}_{\text{convective}} = \underbrace{\frac{\partial}{\partial x_j}(\Gamma_\phi \frac{\partial \phi}{\partial x_j})}_{\text{diffusive}} + \underbrace{S_\phi}_{\text{source}} \tag{4.1}$$

where ϕ is the dependent variable and the meaning of the different terms is indicated by the brackets. This equation needs to be integrated in space over the control volume, which gives a new equation:

$$\int_V \frac{\partial}{\partial t}(\rho\phi)dV + \int_V \frac{\partial}{\partial x_j}(\rho u_j \phi)dV = \int_V \frac{\partial}{\partial x_j}(\Gamma_\phi \frac{\partial \phi}{\partial x_j})dV + \int_V S_\phi dV \tag{4.2}$$

If Gauss' theorem [1] is applied, the convective and diffusive terms are transformed from a volume to a surface integral, resulting in:

$$\int_V \frac{\partial}{\partial t}(\rho\phi)dV + \int_S \rho\phi \mathrm{u} \cdot \mathrm{n}dS = \int_S \Gamma_\phi \nabla\phi \cdot \mathrm{n}dS + \int_V S_\phi dV \tag{4.3}$$

where S is the surface surrounding control volume V and n is the normal unit vector to this surface S. The unsteady and source terms are still volume integrals and can be approximated by replacing the integral by the mean value of the variable multiplied with the volume of the control volume. The mean value is estimated to be equal to the value in the centre of the control volume. This results in the following approximation: [13]

[1] Gauss' theorem states the following: $\int_S \mathrm{A} \cdot \mathrm{n}dS = \int_V (\nabla \cdot \mathrm{A})dV$

$$\int_V q dV = \bar{q}V \approx q_P V \tag{4.4}$$

where subscript P denotes that the value is evaluated in the centre of the control volume.

4.1.2. Control Volume

In general, the cell control volume can have any shape and number of faces, but the most convenient is the geometry with the hexahedral cells with six faces. Because the problems in this research are three dimensional, each subdomain (a synonym for the control volume) has 6 direct neighbours, which are distinguished using the following names: W (West), E (East), N (North), S (South), T (Top) and B (Bottom). Capital letters denote the centre of a grid cell, while small letters denote the boundary surface of the grid cell with the control volume P. A schematic overview if this situation, is shown in figure 4.1. [14]

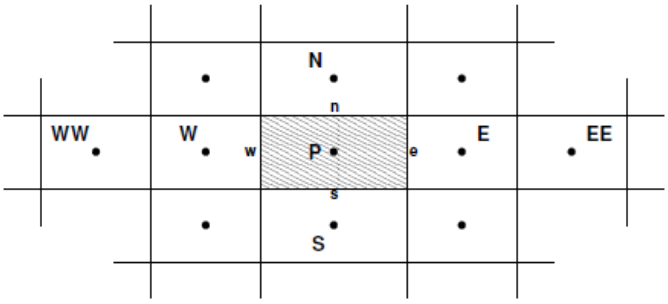


Figure 4.1: Two dimensional overview of a control volume P with its nearest neighbours.

4.2. Differencing Schemes

The next step in the process of numerical computation is the choice of a differencing scheme. This is a delicate step, since one of the most important contributions to the total error in computational fluid dynamics is the discretisation error. Reducing this error is done by applying the appropriate differencing scheme and a sufficiently fine resolution. [15]

4.2.1. Central Difference Scheme

Using the central difference method (CDS) is the most straightforward, but this can be inconvenient in situations with convection as dominating factor in the heat transfer, especially when the applied grids are not fine enough. Symmetrical interpolation and relatively easy implementation are benefits of CDS. The CDS can be formulated with the following equation:

$$\phi_e = \phi_E f_{1P} + \phi_P (1 - f_{1P}) \tag{4.5}$$

where the f represents the interpolation factor for a particular node: [15]

$$f_{1P} = \frac{\overline{Pe}}{\overline{Pe} + \overline{eE}} \tag{4.6}$$

In figure 4.2 a graphic representation of this differencing scheme is given.

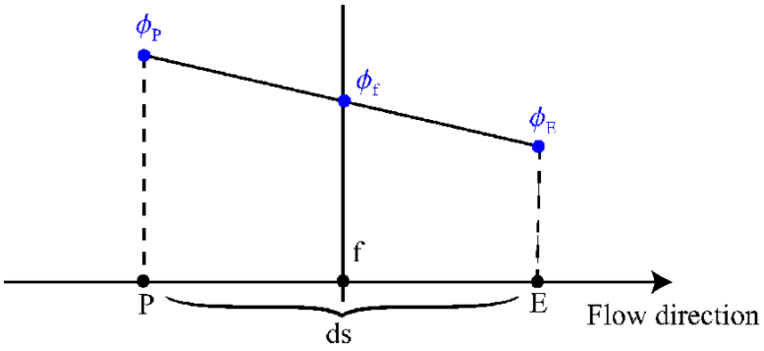


Figure 4.2: A graphic representation of the central differencing scheme.

4.2.2. Upwind Difference Scheme

The upwind difference scheme (UDS) is another possible method. The big advantage of UDS is that it is unconditionally stable, i.e. it is guaranteed that the solutions are bounded. Unfortunately, the UDS is only accurate up to first order and suffers from a phenomenon called "false diffusion". Since it is only accurate up to first order, a very fine grid is needed. This upwind difference scheme can be written as following:

$$\phi_e = \begin{cases} \phi_P & \text{if} (\mathrm{v} \cdot \mathrm{n})_e \geq 0 \\ \phi_E & \text{if} \, (\mathrm{v} \cdot \mathrm{n})_e < 0 \end{cases} \tag{4.7}$$

4.2.3. Linear Upwind Difference Scheme

Linear upwind difference scheme (LUDS) uses a linear extrapolation from the two closest upstream neighbours, in stead of just one, which is the case for UDS. [15] This looks like:

$$\phi_e = \begin{cases} \phi_P + (\phi_P - \phi_W)(1 - \lambda_w) & \text{if} (\mathrm{v} \cdot \mathrm{n})_e \geq 0 \\ \phi_E + (\phi_E - \phi_E E)\lambda_e & \text{if} (\mathrm{v} \cdot \mathrm{n})_e < 0 \end{cases} \tag{4.8}$$

where the λ's are interpolation factors for one particular note, with index and direction. For instance:

$$\lambda_P = \frac{x_e - x_p}{(x_e - x_P) + (x_E - x_e)} \tag{4.9}$$

LUDS is second order accurate, thus it gives better results than UDS. However, one should notice that LUDS is not unconditionally bounded and could give oscillatory and nonphysical solutions.

4.2.4. Quadratic Upwind Difference Scheme

The last possibility is the quadratic upwind difference scheme (QUDS or QUICK), which logically uses a quadratic extrapolation. This differencing scheme determines the value of a cell face with quadratic interpolation using values of three cells. [10] One value downwind is used, the other two values are from the other direction. For a flow direction from west to east for example this scheme evaluates the value at the west cell face as:

$$\phi_w = \frac{6}{8}\phi_W + \frac{3}{8}\phi_P - \frac{1}{8}\phi_{WW} \tag{4.10}$$

$$\phi_e = \frac{6}{8}\phi_P + \frac{3}{8}\phi_E - \frac{1}{8}\phi_W \tag{4.11}$$

In figure 4.3 a schematic overview of QUICK is given.
The QUICK scheme is still unbounded and has stability problems when taking time steps.

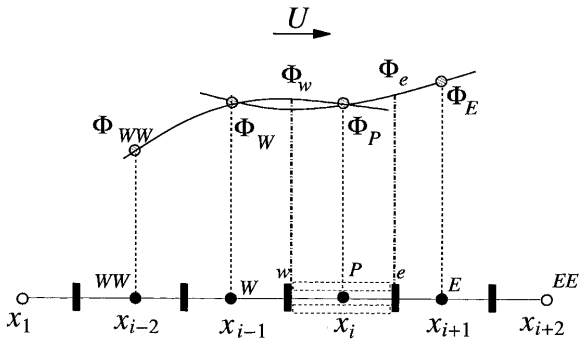


Figure 4.3: Schematic overview of the QUICK differencing scheme. Source: Zondag [16]

4.2.5. Time integration

For the unsteady term in the transport equations the backward Euler method is used: [6]

$$\int_{t_n}^{t_{n+1}} \frac{d\phi(t)}{dt} = \phi^{n+1} - \phi^n = \int_{t_n}^{t_{n+1}} f(t, \phi(t))dt \tag{4.12}$$

4.2.6. Conclusions on the discussed Differencing Schemes

To wrap up the previous sections, some findings can be concluded. First of all, high order schemes oscillate on coarse grids but converge faster to an accurate solution than low order scheme as the grid is refined. Secondly, first-order UDS is inaccurate and should not be used in most cases. When using this method, high accuracy cannot be obtained on affordable grids, since it gives a large diffusive error. Lastly, CDS is the simplest scheme of second-order accuracy and offers a fairly good concession with accuracy, simplicity and efficiency. [17]

4.2.7. Non-linear Schemes

Linear schemes are always vulnerable to nonphysical oscillations (so called wiggles) under some circumstances. This limitation is predicted in Godunov's theorem which states that there no linear convection scheme with greater than first-order truncation error can be monotonic.
A method is called *monotonicity-preserving* if [18]

$$\phi_i^n \geq \phi_{i+1}^n \text{for all } i$$

implies that

$$\phi_i^{n+1} \geq \phi_{i+1}^{n+1} \text{for all } i$$

To solve to this problem, the use of non-linear discretisations is introduced. These schemes adjust themselves in correspondence with the local solution, to maintain bounded behaviour. One of the most effective approaches to construct a non-linear scheme is proved to be the use of so called *flux limiters*. Flux limiters are simple functions which define the convection scheme based on a ratio of local gradients in the solution field. [19]

4.3. Total Variation Diminishing

By introducing correction terms, the method can be improved enormously. The idea with high-resolution methods is to combine the best features of both methods. This means that second-order accuracy is obtained where possible, but is not insisted in regions where the solution is not behaving smoothly.

Second-order accurate methods such as the Lax-Wendroff or Beam-Warming give much better accuracy on smooth solutions than the upwind method, but they both fail near discontinuities and often oscillations appear. [18] Upwind methods have the advantage of keeping a solution monotonically varying in regions where the solution should be monotone, even though the accuracy is not very high. The term $\phi_i^n - \phi_{i-1}^n$ can be modified in the relevant flux equations by applying a limiter that changes

the magnitude of the actually used correction, depending on the behaviour of the solution. However, this limiting process is complicated, since the solution to a hyperbolic problem typically consists of a superposition of waves in different families. At a given point and time, some waves may be smooth while others are not. [18] Using this fact, flux-limiter and slope-limiter methods can be obtained. In this research, the scope is limited by slope-limiter methods. The question arises on how much the slope should be limited. This can be fixed with a mathematical prescription. Before this can be done, a measure for oscillations is needed. For a grid function ϕ the total variation is defined:

$$TV(\phi) = \sum_{i=-\infty}^{\infty} |\phi_i - \phi_{i-1}| \tag{4.13}$$

When the method gives oscillations, the total variation of ϕ^n is expected to increase with time. The goal is to avoid oscillations by requiring that the method does not increase the total variation. A two-level method is called *total variation diminishing* (TVD) if, for any set of data ϕ^n, the values ϕ^{n+1} computed by the method satisfy $TV(\phi^{n+1}) \leq TV(\phi^n)$. The name is a bit confusing, since the total variation does not actually need to diminish; it may remain constant in time. If a method is TVD, then in particular data that are initially monotone $\phi_i^n \geq \phi_{i+1}^n$ (for all i) remains monotone in all next time steps. Any TVD method preserves the monotonicity. [18]

The general expression for higher order differencing scheme can be rewritten (with the addition of a limiter $\psi(r)$ as

$$\Phi = \Phi = \frac{1}{4}[(1+\beta)\psi(r)(\Phi_D - \Phi_C) + (1-\beta)\psi(\frac{1}{r})(\Phi_C - \Phi_U)] \tag{4.14}$$

In order to provide the conditions for the total variation diminishing (TVD), the limiter should be inside the specific area. By introducing nondimensional variables $\widetilde{\Phi}$ and r defined by

$$\widetilde{\Phi} = \frac{\Phi - \Phi_U}{\Phi_D - \Phi_U} \tag{4.15}$$

and the gradient ratio r is defined as:

$$r = (\frac{\partial \phi}{\partial x})_u / (\frac{\partial \phi}{\partial x})_f \tag{4.16}$$

which reduces in the case of a uniform mesh to:

$$r = \frac{\Phi_C - \Phi_U}{\Phi_D - \Phi_C} \tag{4.17}$$

the value at the cell-face can be calculated as

$$\widetilde{\Phi}_f = \widetilde{\Phi}_C + \frac{1}{4}\widetilde{\Phi}_C[(1+\beta)\psi(r)\frac{1}{r} + (1-\beta)\psi(\frac{1}{r})] \tag{4.18}$$

This expression can be further simplified using symmetry properties of the limiter functions. The final form is

$$\psi(r) = r \cdot \psi\left(\frac{1}{r}\right) \tag{4.19}$$

$$\widetilde{\Phi}_f = \widetilde{\Phi}_C + \frac{\psi(r)}{2}(1 - \widetilde{\Phi}_C) \tag{4.20}$$

This form can be seen as a general formulation of the TVD scheme from which a number of specific schemes known in literature can be extracted depending on the formulation of the limiter. There are many different classes of limiters that ensure that the solution remains inside the stability region.

4.3.1. Van Leer Limiter

In this research, the Van Leer limiter (also referred to as MUSCL limiter) is implemented in the CFD code. This limiter function is defined as follows: [19]

$$\psi(r) = \max[0, \min(2r, \frac{r+1}{2}, 2] \tag{4.21}$$

Van Leer proposed this scheme as a slope-limited form of another scheme (Fromm's scheme). This approach offers a combination of accuracy in smooth flow, good resolution of sharp gradients and reasonable convergence characteristics. [19] Furthermore, this limiter is the only classical limiter which does not introduce a local first-order term for smooth solutions. A plot with the Van Leer limiter function overlaid on the TVD region is shown in Figure 4.4.

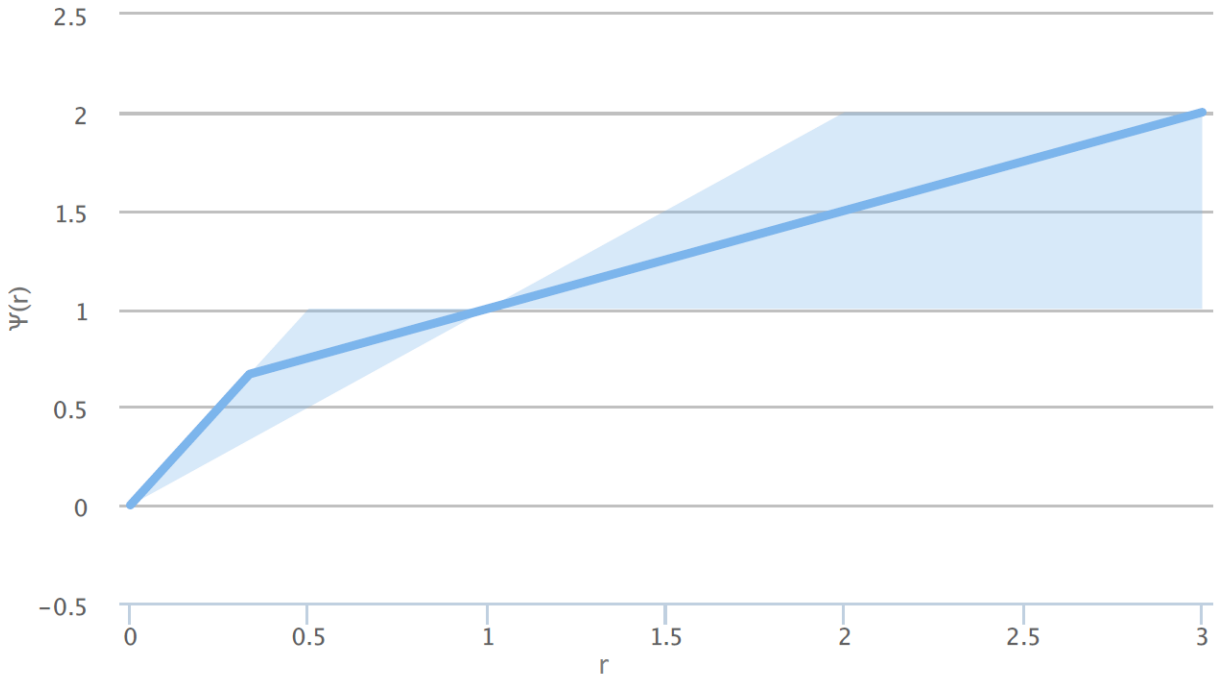


Figure 4.4: Van Leer limiter function overlaid on the second-order TVD region. Source: Nikola Mirkov.

4.4. Deferred Correction

As discussed, the convection diffusion equations from previous sections can be discretised using a finite volume method. This discretisation process results in a system of algebraic equations. These equations can be linear or non-linear, depending on the nature of the partial differential equations. There are several methods for solving these equations, which can be divided in for example direct and iterative methods. One specific approach, namely deferred correction, is discussed in this section.

If all terms with nodal values of the unknown variable are kept on the left hand side of the obtained equations after the discretisation, the computational molecule can become very large. The size of the computational molecule influences the storage capacity and the effort, which are needed to solve the linear equation system. So it is preferred to keep this as small as possible. In order to do so only the nearest neighbours of node P are kept on the left hand sides of the equations. Nonetheless, approximations which produce such simple computational molecules are often not accurate enough. This means it is needed to use approximations that refer to more nodes than just the nearest neighbours. A solution to this problem is to leave only the terms containing the nearest neighbours on the left-hand side of the discretised equations and bring all other terms to the right-hand side. This requires that these terms are evaluated using values from the previous iteration. However, this can lead to the divergence of iterations because the explicitly treated terms can be considerably large. [17]
A better method is to compute the terms that are approximated with a higher order approximation

explicitly and put them on the right-hand side of the equation. Then a simpler approximation to these terms (one that gives a small computational molecule) can be made and these values can be put on the left-hand side of the equation (with unknown variable values) and on the right-hand side (computing it with explicitly using existing values). The right-hand side is the difference between two approximations of the same term, and is very likely to be small. Hence, it should cause no problems in the iterative solution procedure. If the iterations converge, the low order approximation terms vanish and the obtained solution corresponds to the higher-order approximation. Since iterative methods are usually necessary due to the non-linearity of the equations to be solved, adding a small term to the part treated explicitly increases the computing effort by only a small bit. However, the memory and computing time required are very much reduced when the size of the computational molecule in the part of the equation treated implicitly is small.
This technique is used in the implementation of the Van Leer limiter in the FORTRAN code. In general, it is used when treating higher-order approximations, grid non-orthogonality and corrections needed to avoid undesired effects like oscillations in the solution. Because the right-hand side of the equation can be regarded as a ”correction” this method is called deferred correction. [17] In short, deferred correction is applied because it strengthens the diagonal dominance of a system matrix. This translates to a faster and more stable convergence of the iterative solution process.

4.5. Under relaxation

A convergence criterion is introduced to determine whether the numerical solution is sufficient. For each time step the numerical solution is compared to the numerical solution of the previous time step. Since this is a very computationally expensive task to do for every grid point, a trick is used. The comparison is done by using a large scale criterion and choosing specific control volumes to test. The large scale criterion collects the information from all control volumes. For each variable, this so called residual contains information on the convergence of the variable. If all residuals of the variables are less than the criterion, then the convergence criterion is met. If one of the variables grows unlimited, the simulation is said to be unstable. [8]

The stability of a simulation is determined by, amongst other things, the grid definition, the choice of differencing scheme and the size of the domain. To increase the stability of the simulation, the process of under relaxation parameters can be used. This process uses the following technique: the value of a variable ϕ_{n+1} of a certain time step depends on both the previous value and the predicted value of this time step. If ϕ_n is the value of the nth time step and $\widetilde{\phi}_{n+1}$ the value of the next step, then the under relaxation process defines the value for the next time step, ϕ_{n+1} as follows:

$$\phi_{n+1} = \omega\widetilde{\phi}_{n+1} + (1-\omega)\phi_n \tag{4.22}$$

where the under relaxation parameter $0 < \omega \leq 1$ determines how much the next iteration depends on the previous iteration. The process of under relaxation decreases the chances of divergence, but the disadvantage is that it slows down the simulation.

5

Street Canyon Simulations

The model that is introduced in Chapters 2 and 3 about modelling airflow and temperature in urban areas, can be solved numerically using a Computational Fluid Dynamics (CFD) code written and tested by professor S. Kenjereš. In the last 10 years many improvements and additions have been made in this code by bachelor and master students of the TU Delft. In this Chapter the most important sections of this code is explained. Furthermore, the set up of the street canyon simulations is defined.

5.1. Boundary Conditions

The code in FORTRAN is flexible in the dimension sizes, but in this research all domains are box shaped and formulated in the Cartesian coordinate system. In Figure 5.1 an example of a domain is shown. A box shaped domain is bounded by six faces and boundary conditions are defined for each face.

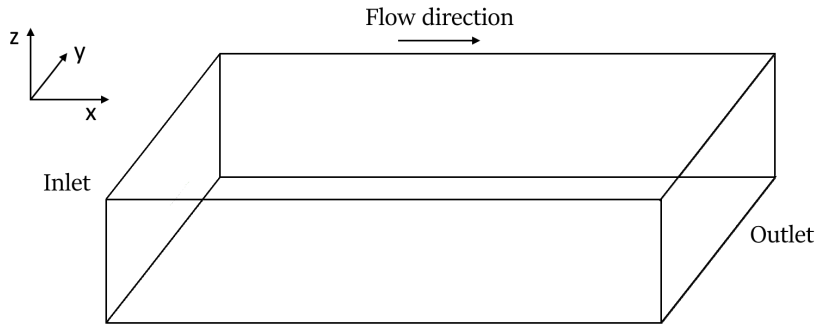


Figure 5.1: Example of a domain. The inlet is by default the left face and the outlet is the right face of the domain.

Most boundary conditions are not complicated to define using some straightforward assumptions. The bottom of the domain is the earth surface, so this is modelled as a wall. From the no slip condition can be deduced that the velocity must be zero at the bottom. If buildings are placed inside the domain, these is also modelled as walls. These walls can be modelled using standard wall functions. In the next subsection these wall functions are discussed in more detail. At the left face of the box the air enters the domain (inlet). For the velocity, turbulent kinetic energy (TKE) and the dissipation inlet profiles are specified. This means that for the entire left boundary of the domain the values of these variables are known explicitly. In the same way, at the right face of the box the air exits the domain; this is called an outlet. The other 3 faces are neither walls, inlets or outlets and are modelled as symmetry faces. Explicitly, this means that velocity must be parallel to these boundary faces. As a result of this fact, the domain sizes must be chosen large enough; if one of the variables is still fluctuating at a symmetry boundary its component perpendicular to the plane is set to zero, resulting in a loss of information. [8]

5.2. Wall Functions

Wall functions are often used to describe a turbulent flow near a wall. To obtain a formula for these walls, three different regions are considered; the viscous sub-layer, the logarithmic layer and the overlap region. The viscous sublayer is close to the wall, where turbulent effects are no longer dominant. As a result, the velocity in this region depends on viscosity. The logarithmic layer is assumed to be fully turbulent, without viscous effects. The transition from viscous to a turbulent regime is described by the overlap region. [10]
The gradient in the viscous sublayer is extremely large. To model this layer accurately, a fine grid is needed. Using dimensional analysis, the generalised logarithmic wall function can be obtained:

$$U^+ = \frac{1}{\kappa} \ln E \frac{u_* y}{\nu} \tag{5.1}$$

where $\kappa = 0.435$ is the Von Kármán constant, $E = 8.432$, y is just the vertical coordinate and ν the kinematic viscosity. The following equation is obtained for the wall shear stress: [10]

$$\tau_t = \tau_w = \rho C_\mu^{1/4} k_p^{1/2} \frac{U}{\ln y^+ / \kappa} \tag{5.2}$$

where the index P indicates that the values are evaluated at the first cell next to the wall. With this equation the velocity in the grid point closest to the wall can be calculated. An important remark is that equations 5.1 and 5.2 are only valid for smooth walls. When modelling air flow around vegetation, wall roughness will probably play a role. [6]

5.3. Initial Conditions

In general, a partial differential equation cannot be solved without initial conditions. However, if the numerical scheme yields stable calculations, the solution will approach an equilibrium solution regardless of the initial conditions. This is not true in general, but for our purpose this can be assumed. Still, if difficult initial conditions are chosen, the numerical solution will approach the equilibrium very slowly. Thus, the initial conditions are chosen constant throughout the domain and within the range of the inlet profiles.

5.4. Grid Definition

As discussed in Chapter 4, the domain must be divided into a finite number of control volumes. The discretised domain is called a grid or a mesh. With the addition of buildings and vegetation (in general called obstacles) in the domain, the division of the domain in control volumes is a delicate process. A requirement of these obstacles is that the buildings and vegetation must be box shaped to limit the difficulty. The distribution of these control volumes must satisfy certain resolution criteria. This means that the grid should sufficiently resolve significant gradients of all variables. [14] The values of the variables will vary to a greater extent around obstacles than farther away from these obstacles. Due to this fact, the grid is much finer at the edge of these obstacles than at other places in the domain. In this way, the fluctuations can be calculated more accurately. As a result, the control volumes are non uniform. However, in this paper all control volumes are rectangular shape. Argued in the same way, the grid becomes coarser when moving away from an obstacle. For this reason a *cell expansion factor* is defined; this is the ratio of adjacent grid cells. If the size of the cells at the obstacles is known, then the entire grid can be calculated. Doing this saves time and costs less storage memory. Furthermore, a maximum cell size is defined. An example of a grid is given in Figure 5.2.

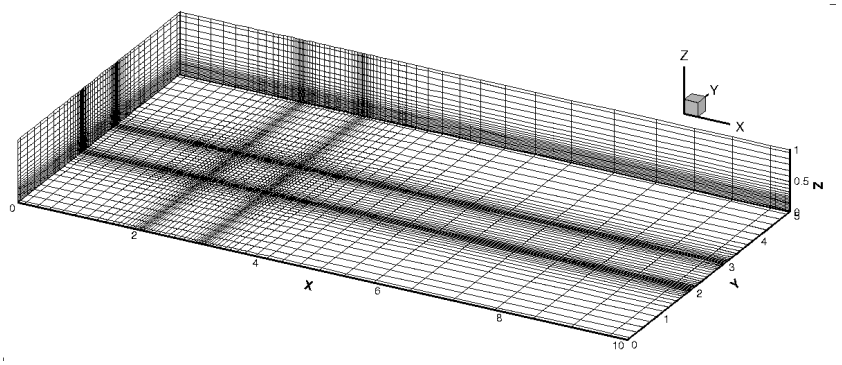


Figure 5.2: Example of a grid. Source: Filipovic [8]

5.5. Vegetation in Street Canyon

To study the effect of vegetation in a street canyon on air flow, some wind tunnel experiments from a study by Gromke [20] are simulated using Fortran code of professor Kenjeres. In Gromke's simulation, a scaled street canyon (1:150) with and without a porous body representing vegetation was used to perform wind tunnel experiments. This set-up is shown in Figure 5.4

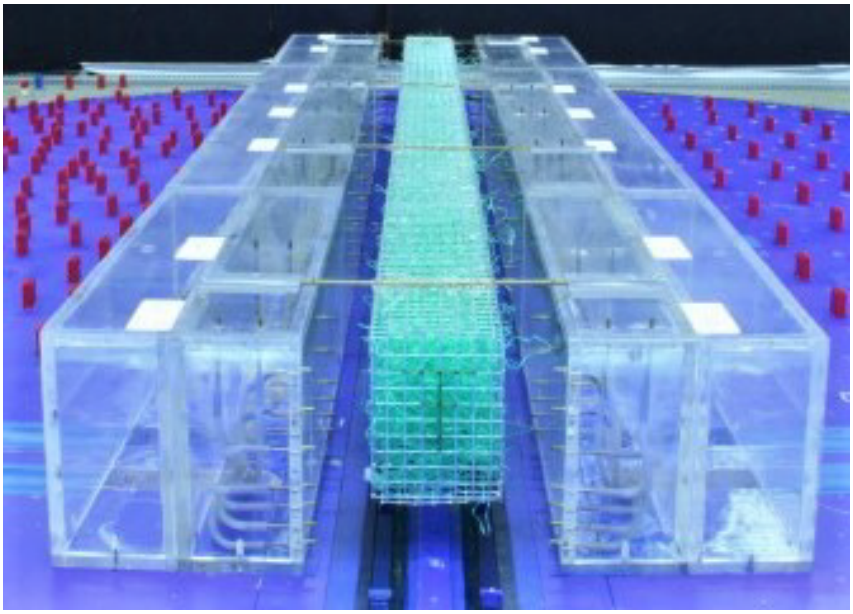


Figure 5.3: Set up for the wind tunnel experiments by Gromke [20]. Source: Gromke [5]

For the set up of the simulation, the domain sizes are the same as in Gromke's experiments and the height of the buildings is $H = 18m$. In Figure 5.4, the domain with 2

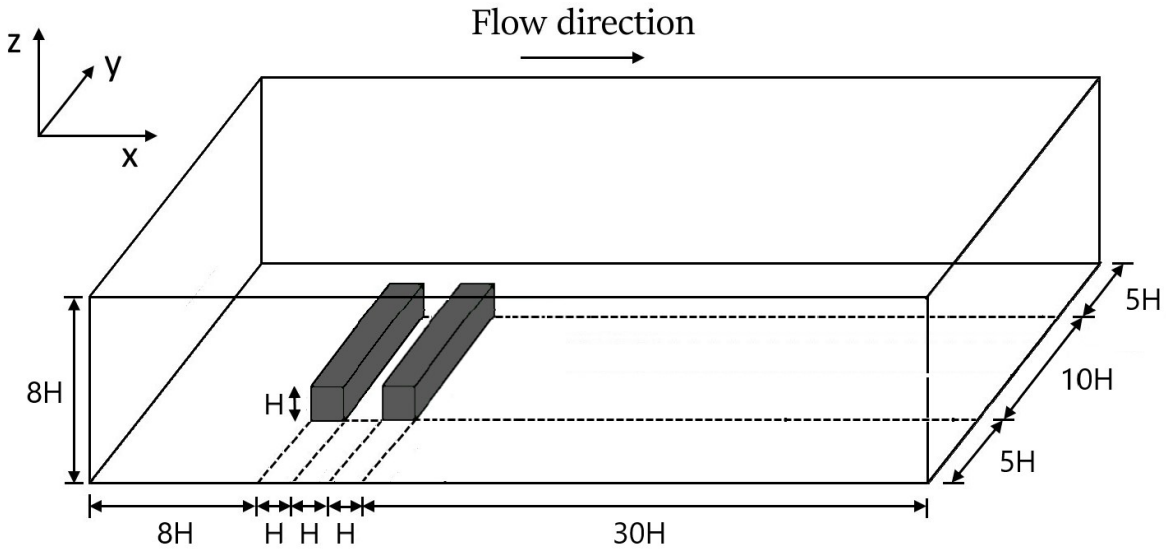


Figure 5.4: Set up of the empty street canyon for the CFD simulations. Source: Marleen van Soest [11]

The vegetation is tested at 2 different locations: as trees in the middle of the two buildings and as facade greening at the sides facing the street canyon. A third possibility is greening at the roof of the buildings. However, this option is left out of this research. Of course, a street canyon without vegetation is also part of the simulations as a status quo. The set up of these different vegetation options is displayed in Figure 5.5

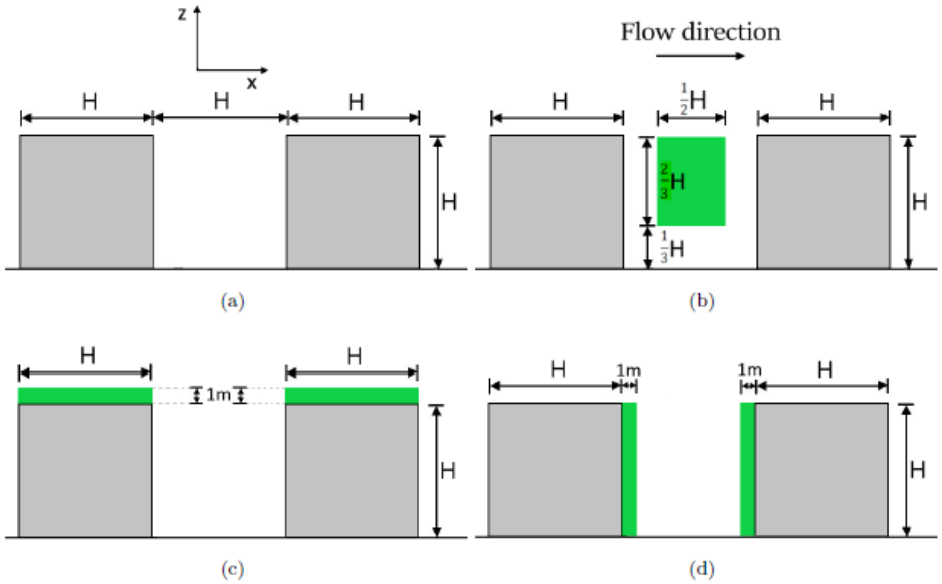


Figure 5.5: Set up of street canyon for different vegetation options: (a) a street canyon without vegetation, (b) trees in the middle of the street canyon, (c) roof greening and (d) facade greening. The building height is $H = 18m$. This image comes from the thesis of Marleen van Soest. [11]

The leaf area density (LAD) of the vegetation is derived from the density of the porous material used in Gromke's experiment and the thickness of the leaves. From Gromke's paper, it is known that the material had a material volume fraction of 3.0% and the thickness of the leaves is estimated at $5mm$. This results in a LAD of $60m^{-1}$.

5.5.1. Boundary Conditions

For the inlet profile, the following profiles are used to match the wind tunnel experiments. The velocity profile is described by:

$$u(z) = u_{\text{ref}}(\frac{z}{H})^{0.3} \tag{5.3}$$

where $u_{\text{ref}} = 4.7m/s$ and $H = 18m$. u is the velocity in the x-direction, v is the velocity in the y-direction and w is the velocity in the z-direction.

The turbulent energy profile is given by:

$$k(z) = \frac{u_*^2}{C_\mu}(1 - \frac{z}{\delta}) \tag{5.4}$$

where $u_* = 0.52m/s$, $C_\mu = 0.09$ and $\delta = 8H$.

Finally, the dissipation profile is:

$$\epsilon(z) = u_f^3 \frac{(1 - z/\delta)^2}{\kappa(z + z_0)} \frac{1 + 5.75z}{z_0} \tag{5.5}$$

where $u_f = 0.15m/s$ is the friction velocity, $\delta = 5m$, $z_0 = 0.033m$ and $\kappa = 0.435$ is the Von Kármán constant.

The boundary roughness height at the ground is set at $z_0 = 0.033m$, which gives a roughness parameter $e = 0.099m$

5.5.2. Computation

The grid settings are more or less copied from Gromke's experiments. The smallest grid dimensions are set to $0.9m \times 0.9m \times 0.9m$ and the largest grid dimensions to $3.0m \times 3.0m \times 3.0m$. This results in a grid of about 2.3 million cells. All under-relaxation parameters are set to 0.1. In Table 2.1 the used tree model coefficients are presented as proposed by earlier research by Katul and later verified by research of Ter Kuile and Kenjeres. For the velocity simulations, the drag coefficient C_D is linearly increased from 0.0 to 0.30 with steps of 0.03 to ensure convergent results per 300 iterations. At $C_D = 0.30$, 15000 iterations are done to ensure that the convergence criterion is met.

6

Results

In this Chapter the results of the simulations are showed. The obtained results are compared with the results of 2 papers of Gromke [20] [5]. For each vegetation option, the normalised vertical velocity $w* = W/u_{ref}$, the magnitude of the total velocity $|U| = \sqrt{u^2 + v^2 + w^2}$, the turbulent kinetic energy and temperature are displayed in contour plots. The contour plots show a cross section of the street canyon exactly in the middle of the street canyon. The two white blocks, in for example Figure 6.1, represent the two buildings. The flow direction is from left to right, as visualised in Figure 5.4. Furthermore, the different vegetation options are compared to each other in profile plots of the magnitude of the total velocity, the turbulent kinetic energy and the temperature. These profile plots are made for three different heights:

1. Street level: $z/H = 0.1$
2. Halfway in the canyon: $z/H = 0.5$
3. Just above the buildings: $z/H = 1.1$

Thus these profile plots have a fixed y- and z-coordinate.

6.1. Vertical velocity

In Figures 6.1, 6.2 and 6.3 the contour plots of the normalised velocity in the z-direction in street canyon are showed at $y/H = 10.5$. This vertical velocity is particular interesting to analyse, since it contains information about the decrease (or increase) of the concentration of pollutants in the street canyon.

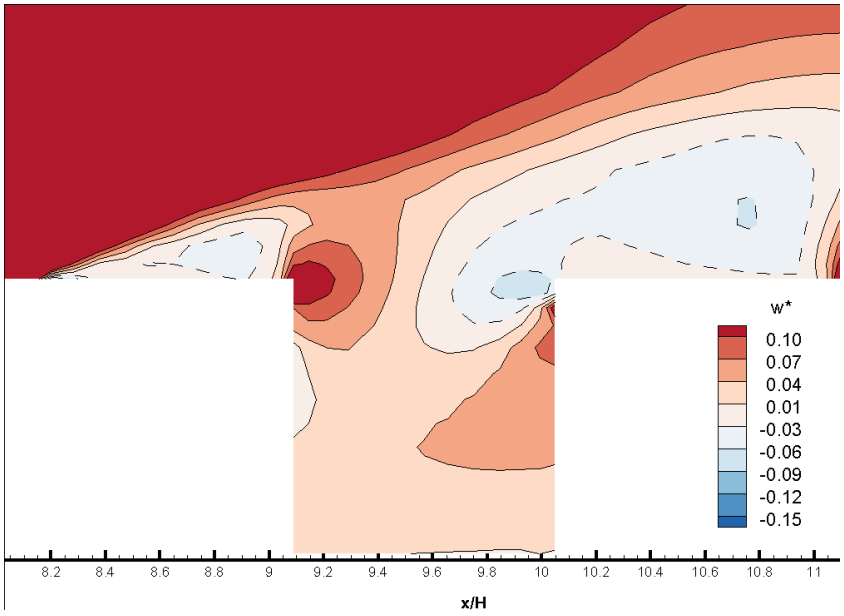


Figure 6.1: Contour plot of the velocity in the z-direction normalised by the reference velocity $w*$ at $y/H = 10.5$ from the street canyon without vegetation.

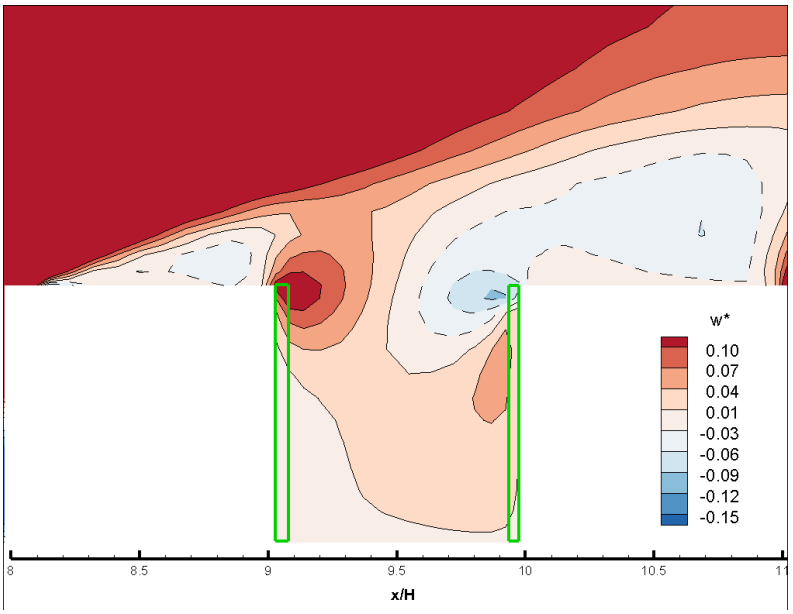


Figure 6.2: Contour plot of the velocity in the z-direction normalised by the reference velocity $w*$ at $y/H = 10.5$ from the street canyon with the green facades.

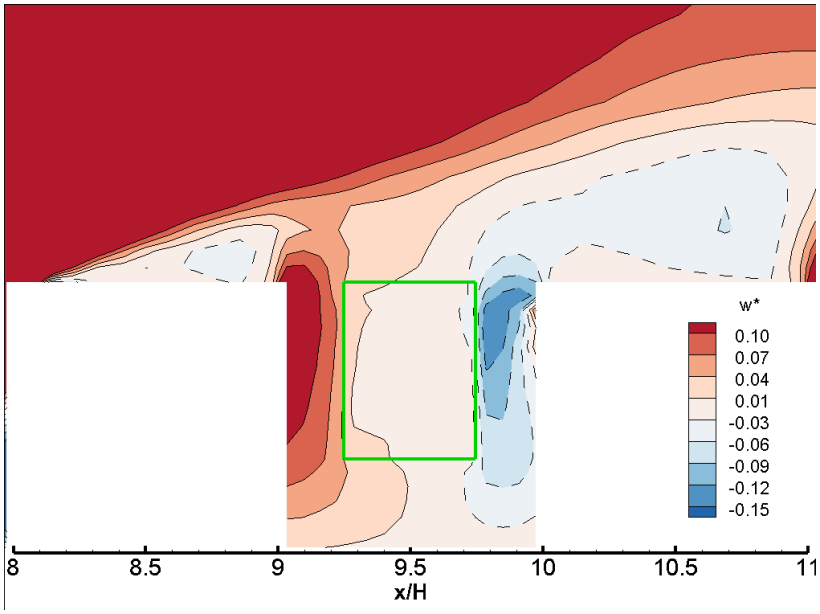


Figure 6.3: Contour plot of the velocity in the z-direction normalised by the reference velocity $w*$ at $y/H = 10.5$ from the street canyon with trees in the middle.

The impact of the different locations of vegetation on air flow can be assessed by comparing Figures 6.1, 6.2 and 6.3. It can be seen that the tree option has bigger influence on the vertical velocity than the green facade. The tree option causes higher values of the vertical velocity close to the wall.

The behaviour of the velocity in the vertical direction in the street canyon without vegetation and the street canyon with green facades is not as one would predict on beforehand. It is expected that there are both positive and negative values, which results in circulation. However, in Figures 6.1 and 6.2 there are only positive values visible. This phenomenon is most likely caused by the fact that in the chosen set up, only 1 street canyon is present. To obtain more realistic results, 10 sequential buildings can be created in the set up. By extracting results from the last street canyon, the results should probably behave more naturally.

The results plotted in the contour plots in Figures 6.1, 6.2 and 6.3 can be compared with contour plots from earlier research by Gromke [20]. Gromke performed both wind tunnel experiments and simulations using the k-ε model for an empty street canyon and a canyon with trees in the middle. The contour plots of the experiments and simulations are displayed in Figure 6.4 and 6.5

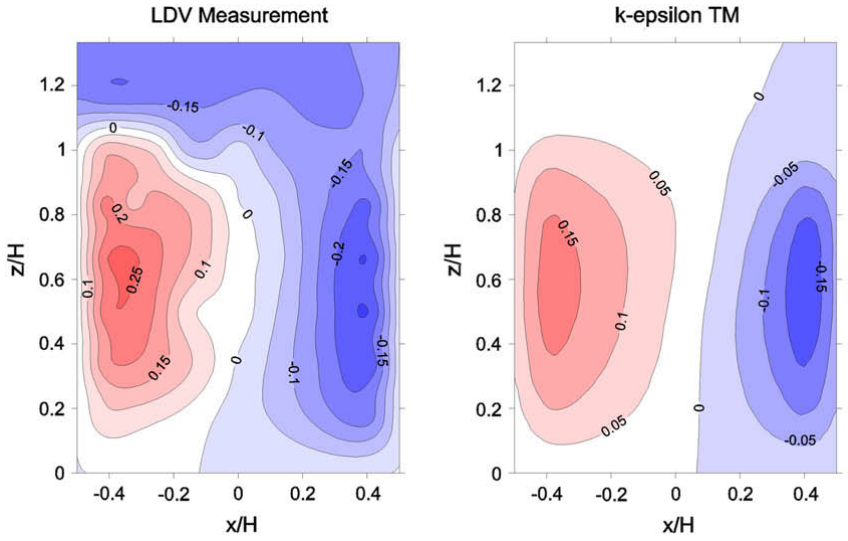


Figure 6.4: Contour plots from the measurement (left) and simulation (right) of Gromke [20]: normalised vertical velocities $w*$ at $y/H = 10.5$ in an empty street canyon.

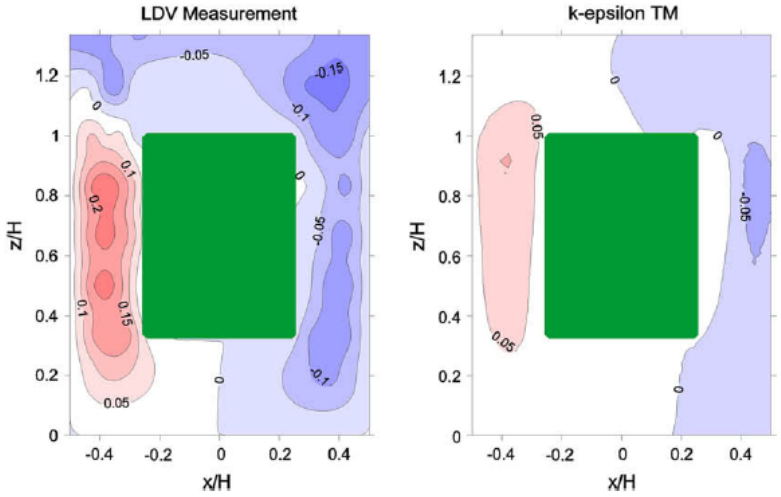


Figure 6.5: Contour plots from the measurements (left) and simulations (right) of Gromke [20]: normalised vertical velocities $w*$ at $y/H = 10.5$ in a street canyon with trees.

First, the results of the empty street canyon can be compared. In Figure 6.4, it can be seen that the measurement and simulation results of Gromke are qualitatively seen more or less the same. However, the agreement with the contour plot in Figure 6.1 is poor. Most likely, one of the simulation settings of this research is not in line with those of Gromke's simulations. This simulation has been done earlier by Marleen van Soest [11] and this yielded more similar results.
Secondly, the street canyon with trees in the middle is compared to the measurement and simulation by Gromke et al. [20]. The contour plots of this research in Figure 6.3 and the contour plots of Gromke in Figure 6.5 show good agreement. A difference is that the highest positive (on the left of the street canyon) and negative values (on the right) are a bit higher situated in the street canyon. This difference could be caused by the fact that the obstacle roughness was not specified in Gromke's paper [20]. In the simulation of this research the obstacle roughness was estimated to be $1.0 \cdot 10^{-6} m$.

6.2. Magnitude total velocity

Not only the vertical velocity component is analysed, also the magnitude of the total velocity is contained in the analysis. In Figures 6.6, 6.7 and 6.8 the contour plots of the magnitude of the total velocity in the street canyon are showed at $y/H = 10.5$ for the empty street canyon, the street canyon

with green facades and the street canyon with trees.

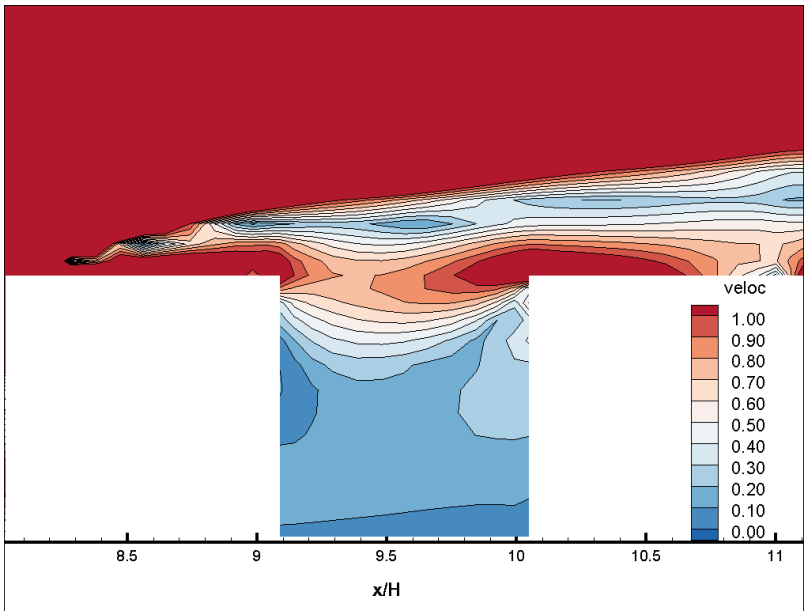


Figure 6.6: Contour plot of the magnitude of the total velocity at $y/H = 10.5$ from the empty street canyon.

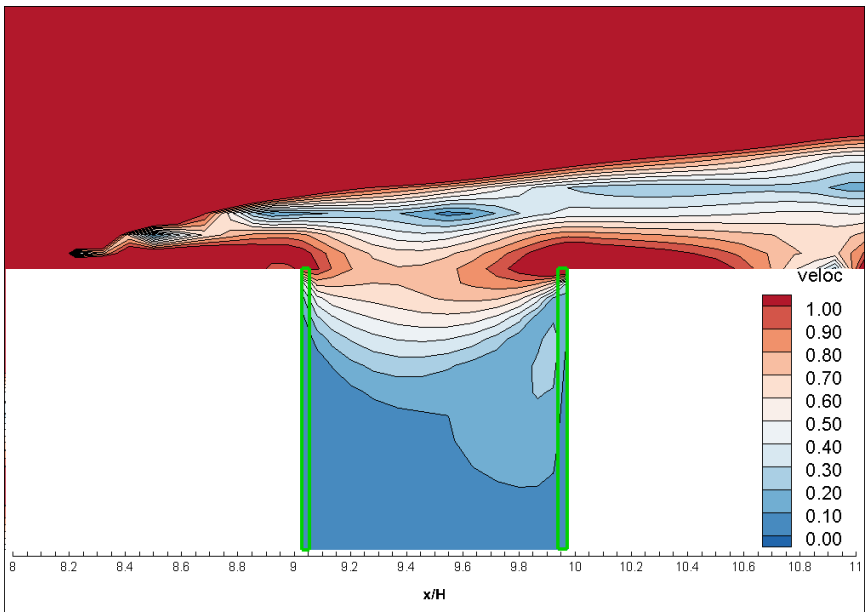


Figure 6.7: Contour plot of the magnitude of the total velocity at $y/H = 10.5$ from the street canyon with green facades

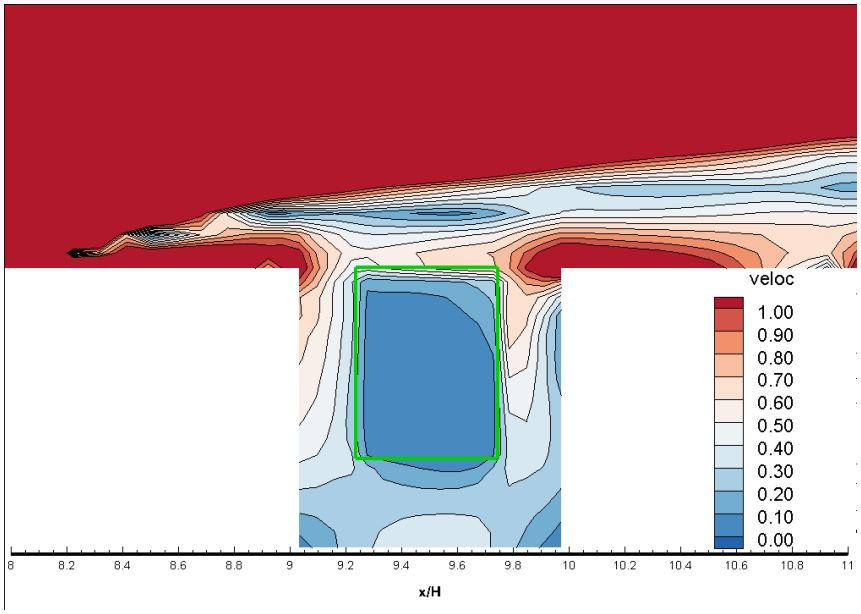


Figure 6.8: Contour plot of the magnitude of the total velocity at $y/H = 10.5$ from the street canyon with trees in the middle.

From the contour plots in Figures 6.6, 6.7 and 6.8 it can be seen that both green facades and trees influence the magnitude of the total velocity. It seems that the green facades slows down the air flow, whereas the air flow around the trees is much higher.

Furthermore, profile plots are made at street level ($z/H = 0.1$), halfway in the canyon ($z/H = 0.5$) and just above the buildings ($z/H = 1.1$). These plots only display the values inside the street canyon ($9 < x/H < 10$).

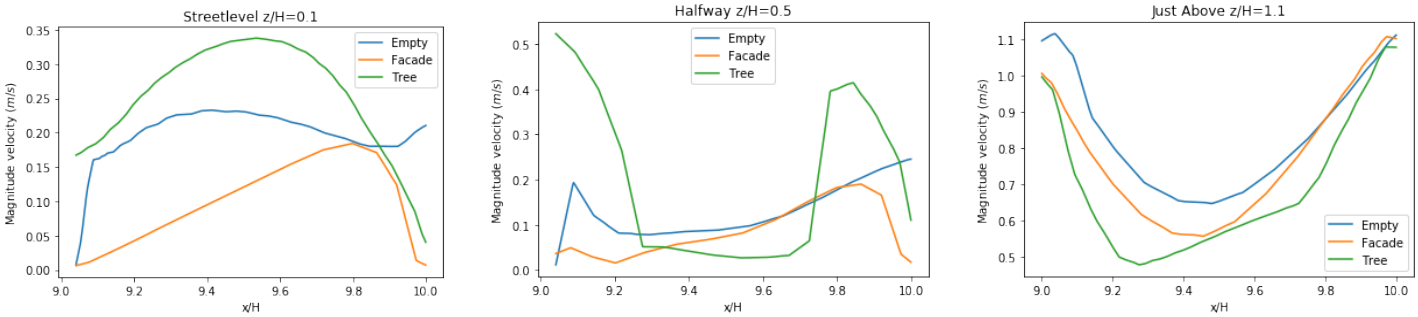


Figure 6.9: Profile plots of the magnitude of the total velocity at street level, halfway the canyon and just above the canyon

From the profile plots in Figure 6.9 can be seen that the green facade and the trees influence the magnitude of the total velocity on all three heights.

At street level the street canyon with trees has the highest velocity. This is probably due to the circulation of the air under the canopy.
Halfway in the canyon $z/H = 0.5$, the green facade has little impact in terms of velocity $|u|$. The green facade causes a smaller velocity near the walls. For the trees, this intersection goes through the trees, so logically the velocity inside the canopy is small with high peaks near the wall. These high peaks indicate a circulation around the canopy.
Just above the street canyon $z/H = 1.1$, trees and green facades also influences the velocity. Both vegetation options cause smaller values of the velocity.

6.3. Turbulent Kinetic Energy

In Figures 6.10, 6.11 and 6.12 the contour plots of the turbulent kinetic energy in the street canyon are showed at $y/H = 10.5$ for the empty street canyon, the street canyon with green facades and the street canyon with trees. The turbulent kinetic energy is a measure of turbulence of the air flow.

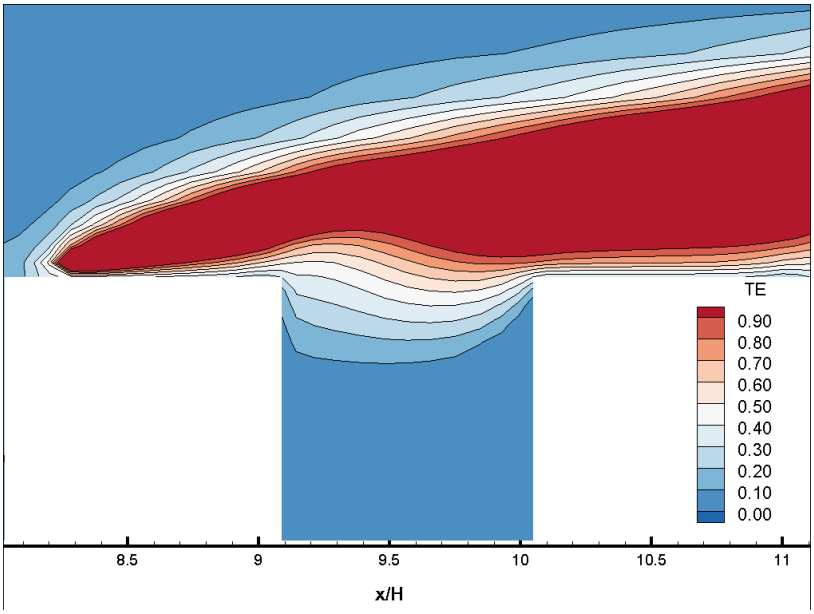


Figure 6.10: Contour plot of the turbulent kinetic energy at $y/H = 10.5$ from the street canyon without vegetation.

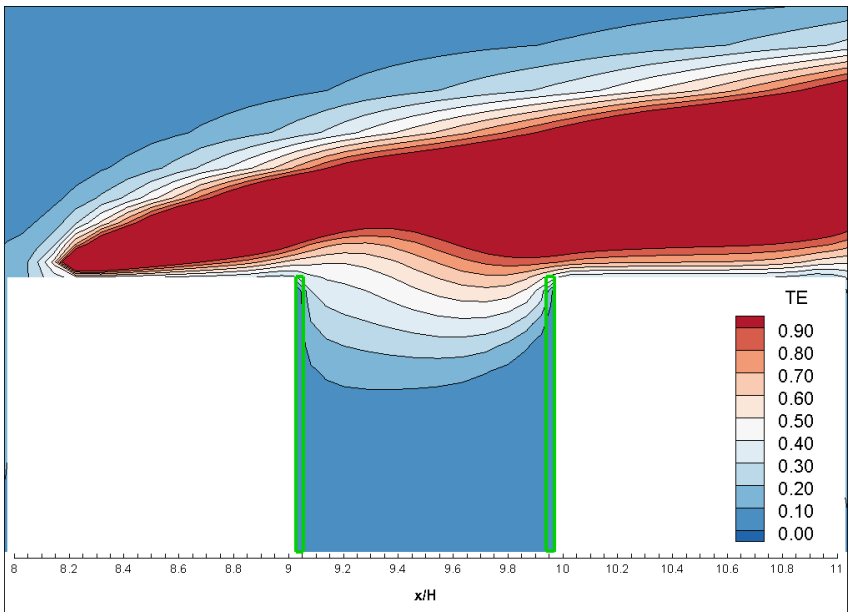


Figure 6.11: Contour plot of the turbulent kinetic energy at $y/H = 10.5$ from the street canyon with the green facades.

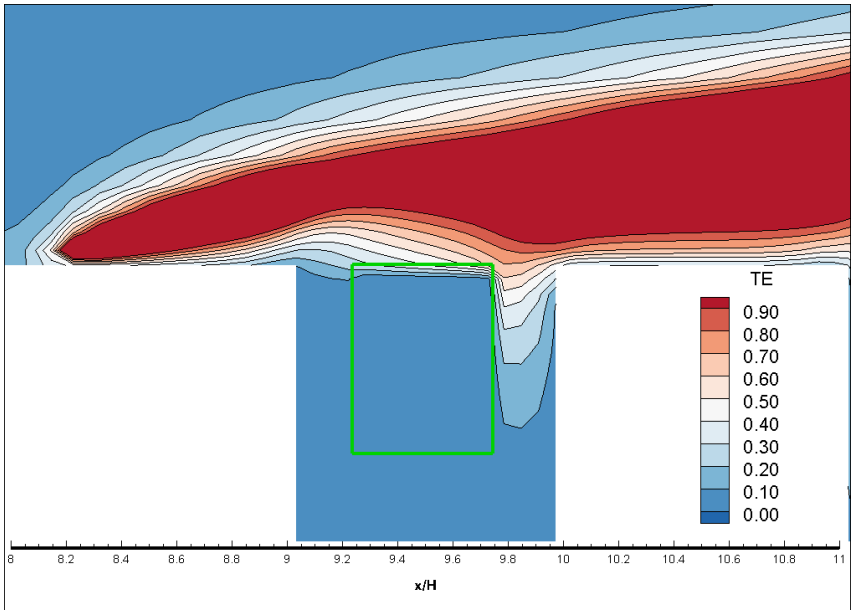
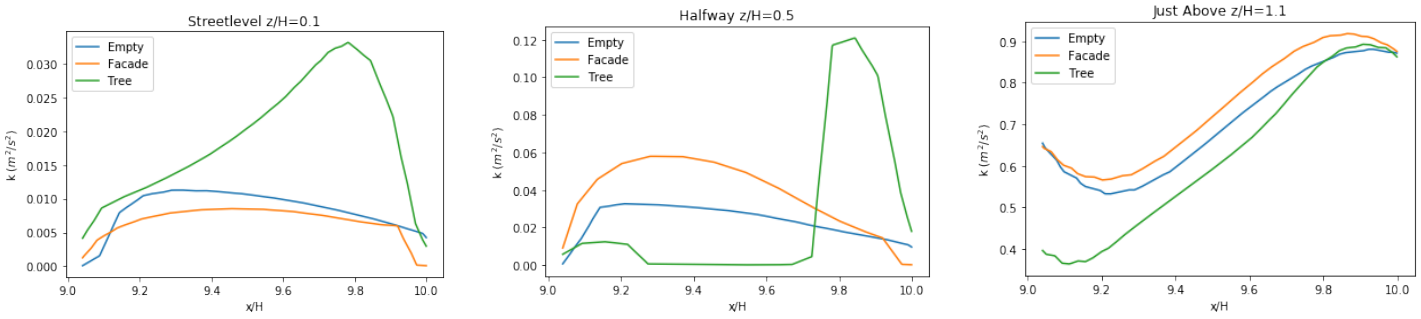


Figure 6.12: Contour plot of the turbulent kinetic energy at $y/H = 10.5$ from the street canyon with trees in the middle.

From the contour plots of the TKE in Figure 6.10, 6.11 and 6.12 that the trees in the street canyon influence the TKE much more than the green facades do. Especially, at the right of the street canyon, the trees cause higher values of the TKE.

Profile plots are made at street level ($z/H = 0.1$), halfway in the canyon ($z/H = 0.5$) and just above the buildings ($z/H = 1.1$). These plots only display the values inside the street canyon ($9 < x/H < 10$).



(a) (b) (c)

Figure 6.13: Profile plots of the turbulent kinetic energy at street level, halfway the canyon and just above the canyon

From the profile plots in Figure 6.13 can be seen that the green facade and the trees influence the turbulent kinetic energy on all three heights. At street level and halfway the canyon, the TKE of the street canyon with trees is much higher than for the empty street canyon and the street canyon with green facades. On the contrary, the TKE of the street canyon with trees is lower than the TKE of the other two situations just above the street canyon.

6.4. Temperature

In Figures 6.14, 6.15 and 6.16 the contour plots of the temperature in the street canyon are showed at $y/H = 10.5$ for the empty street canyon, the street canyon with green facades and the street canyon with trees.

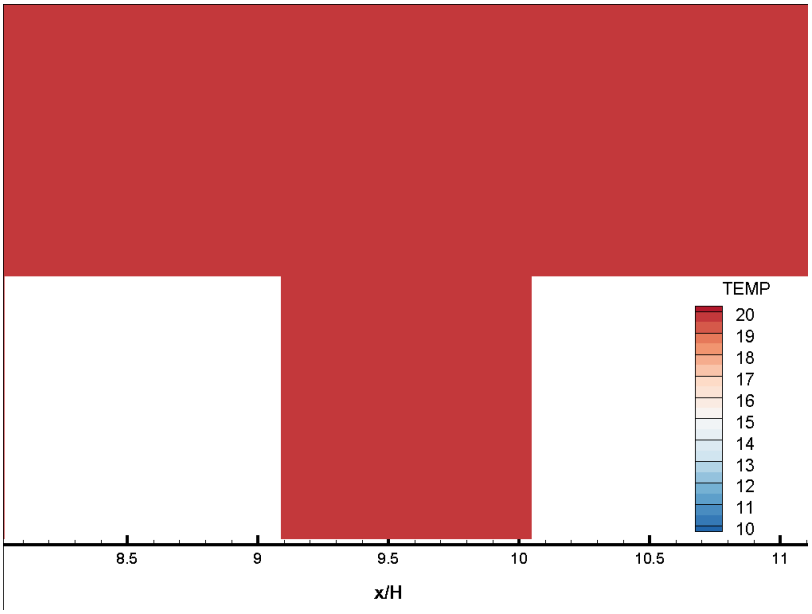


Figure 6.14: Contour plot of the temperature at $y/H = 10.5$ from the street canyon without vegetation.

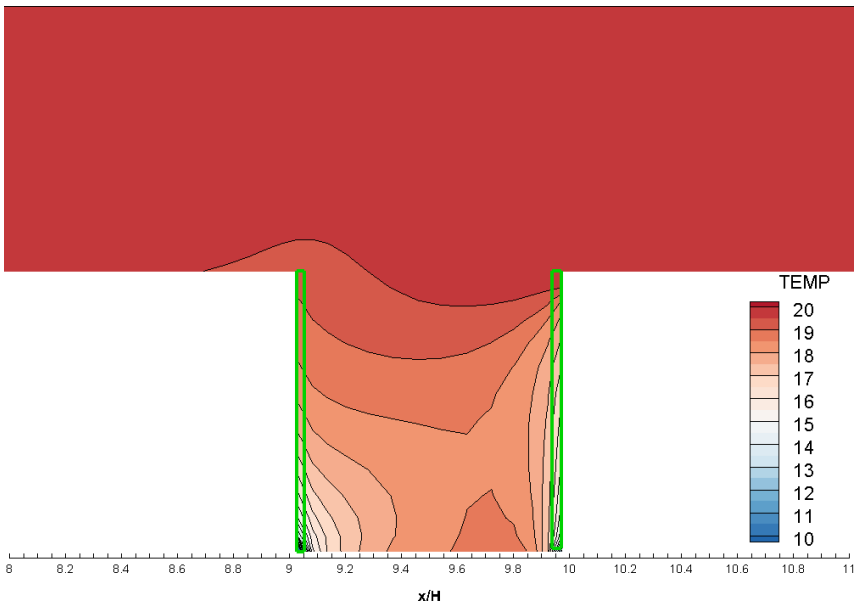


Figure 6.15: Contour plot of the temperature at $y/H = 10.5$ from the street canyon with the green facades.

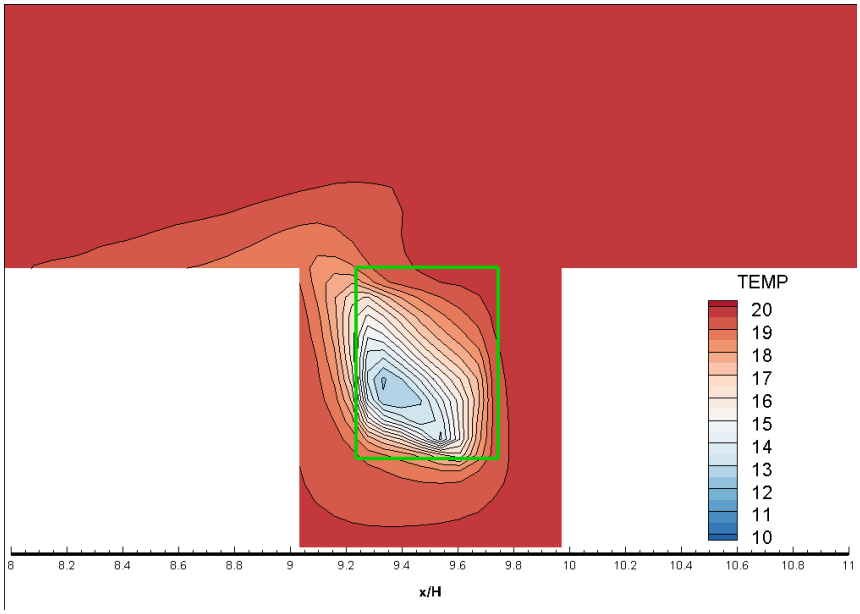
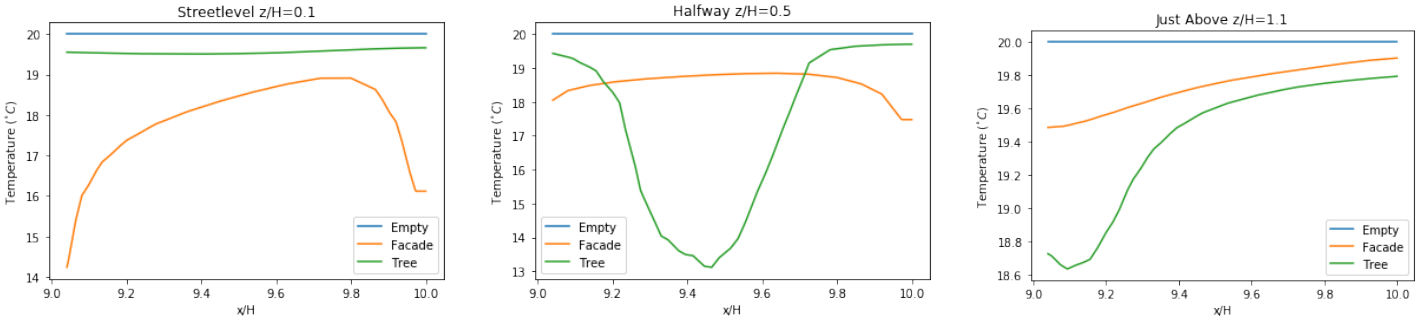


Figure 6.16: Contour plot of the temperature at $y/H = 10.5$ from the street canyon with trees in the middle.

The impact of the different locations of vegetation on air flow can be assessed by comparing Figures 6.14, 6.15 and 6.16. As expected, both the green facades and the trees display a difference in the temperature distribution compared to the empty street canyon. In the empty street canyon, there is a uniform distribution of $20°C$ everywhere. The green facade obviously cools the volume near the facade down the most. However, the temperature of the whole street canyon is lowered by the green facades. The trees also cool down the street canyon. Inside the tree canopy the temperature is lowered enormously. However, on the ground and on the right side of the street canyon the temperature drop is very small.

To take a closer look at the differences between the effect of the green facades and the trees, profile plots are made at street level ($z/H = 0.1$), halfway in the canyon ($z/H = 0.5$) and just above the buildings ($z/H = 1.1$). These plots only display the values inside the street canyon ($9 < x/H < 10$).



(a) (b) (c)

Figure 6.17: Profile plots of the temperature at street level, halfway the canyon and just above the canyon

From the profile plots in Figure 6.17 can be seen that the green facade and the trees influence the temperature on all three heights.
However, as already concluded from the contour plots, the drop in temperature at street level for the street canyon with green facades is significant. Especially close to the facades, the temperature is $4°C$ lower than without vegetation and with trees in the street canyon. For the quality of life of citizens, the temperature at street level is an important parameter.

Halfway the canyon, the temperature drop of the street canyon with trees is the highest. Inside

the tree canopy the lowest probed temperature is $12.5°C$. This temperature drop is as expected, since at this height, the temperatures in the middle of the canyon are extracted from inside the tree canopy. Although this drop is also expected in experiments, the specific strength of this drop in the simulation is much stronger than that.

Not only the green facades cause temperature drops halfway the canyon, also the green facades cause a temperature drop. Although, the drop is less than at the street level, the temperature dropped at least $1°C$.
Just above the canyon ($z/H = 1.1$), the trees in the canyon and the green facades also generate a drop in temperature. The drop is the highest for the street canyon with trees. For both the green facades and the trees, the drop is the highest at the left side of the canyon. At the right side of the canyon both temperature distributions approach the temperature of the empty street canyon.

Earlier research by Gromke et al. [5] reported different conclusions regarding the temperature effects of these greening options. Their research noticed that the cooling effect of green facades on temperature in the street canyon was bounded to only the near surroundings of the walls. However, this research modelled the green facades differently. They took into account that the facades are only partly available for greening. Because of windows, balconies and other openings in the facade, the average leaf area indices of the facade are modelled lower. They assumed a coverage fraction of 50%, where the set up of the simulations of this research assumed full coverage. They used a different leaf area density of the green facades (leaf area indices between 0.25 and $1.5 m^2/m^2$). Finally, the research of Gromke [5] used a volumetric cooling power for the green facades of $187.5 W/m^3$ (against $250 W/m^3$ in this research). In conclusion, Gromke found a smaller cooling effect of the green facades than in this research.
Gromke also simulated a set up with a avenue-tree row. Although, the set up differs, the results can be compared roughly. Gromke et al. noticed a cooling effect due to the trees at the entire pedestrian level.

7

Implementation of the Van Leer limiter

From the theory from Chapter 2 about modelling airflow and Chapter 3 about temperature distributions in urban areas and Chapter 4 about numerical methods, a model can be developed. However, building such a model up from scratch is very complicated and time consuming. To gain insight in the discussed concepts, a simplified case is studied and implemented in Python in this section.

7.1. Simple Advection Equation in Python

In the simplified case studied in this section, the behaviour of the Van Leer limiter can be analysed when using a High Resolution method. To illustrate the behaviour of the limiter, first order upwind and the Beam-Warming method are also implemented. Comparing these three yields more understanding of the functioning of the limiter. The equations 2.3 and 3.3 can be reduced to a basic advection equation. This simplified equation is the starting point:

$$\frac{\partial q}{\partial t} + \bar{u}\frac{\partial q}{\partial x} = 0 \text{ with } \bar{u} = 1 \text{ and initial condition } q(x,0) = f(x) \tag{7.1}$$

For this problem we take periodic boundary conditions in space and $x \in [-5,5]$. Furthermore, we have $t \in [0,T]$, where T is chosen equal to 5 periods.

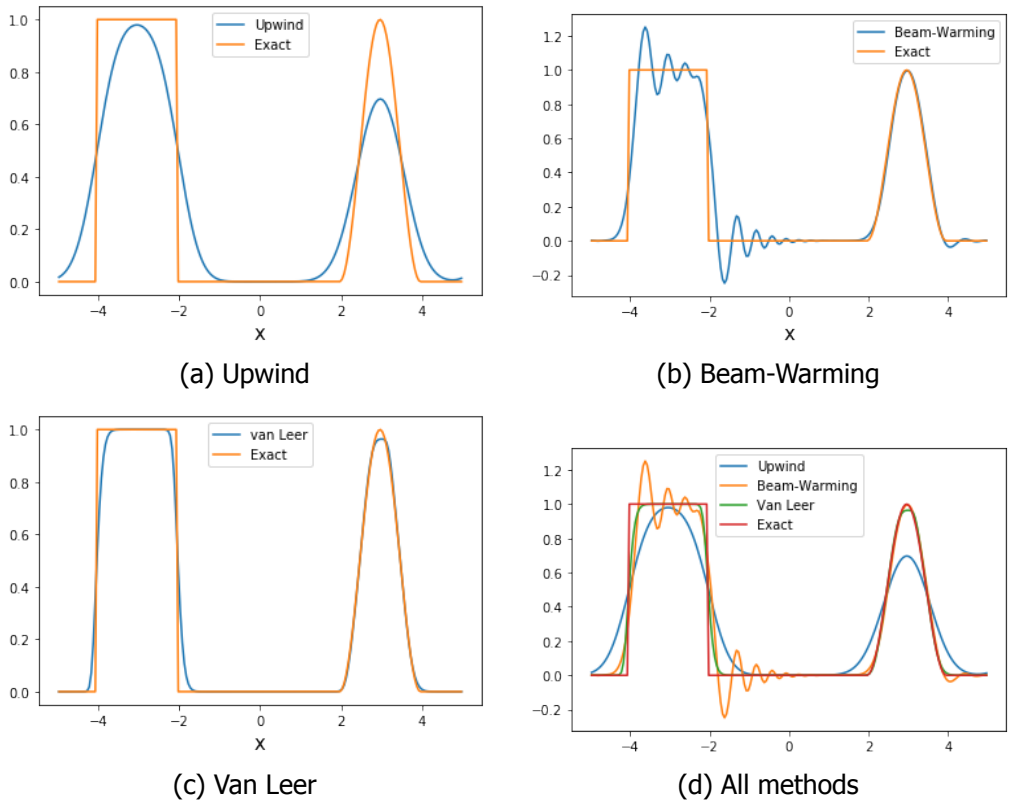
First order upwind is the simplest upwind scheme and is given by:

$$\begin{cases} \frac{u_i^{n+1}-u_i^n}{\Delta t} + \bar{u}\frac{u_i^n - u_{i-1}^n}{\Delta x} = f(x) \text{ for } \bar{u} > 0 \\ \frac{u_i^{n+1}-u_i^n}{\Delta t} + \bar{u}\frac{u_{i+1}^n - u_i^n}{\Delta x} = f(x) \text{ for } \bar{u} < 0 \end{cases} \tag{7.2}$$

Beam-Warming is a second order method. The Beam-Warming second-order upwind method discretises the advection equation in the following way:

$$\begin{cases} \frac{u_i^{n+1}-u_i^n}{\Delta t} + \bar{u}\frac{3u_i^n - 4u_{i-1}^n + u_{i-2}^n}{2\Delta x} = \frac{\bar{u}^2\Delta t}{2}\frac{u_i^n - 2u_{i-1}^n + u_{i-2}^n}{\Delta x^2} \text{ for } \bar{u} > 0 \\ \frac{u_i^{n+1}-u_i^n}{\Delta t} - \bar{u}\frac{3u_i^n - 4u_{i+1}^n + u_{i+2}^n}{2\Delta x} = \frac{\bar{u}^2\Delta t}{2}\frac{u_i^n - 2u_{i+1}^n + u_{i+2}^n}{\Delta x^2} \text{ for } \bar{u} < 0 \end{cases} \tag{7.3}$$

In the Appendix A the Python code is attached.



(a) Upwind (b) Beam-Warming

(c) Van Leer (d) All methods

Figure 7.1: Results from different numerical methods solving a simple advection equation

In Figure 7.1 the results for the different high resolution methods are plotted. In Subfigure (d) all different methods are plotted in one figure.

In Subfigure (a) it can be seen that the first order method does not give very accurate results. Especially at the boundaries of the step functions, the difference between the exact solution is significant. In Subfigure (b), the Beam-Warming method already gives better results at the boundaries of the step functions. However, the Van Leer limiter has the closest approach to the exact solution with sharp gradients at the boundaries of the step function.

7.2. Implementing the Van Leer limiter in the FORTRAN code

In Chapter 4 Total Variation Diminishing was discussed. One of the main purposes of this research, is adding the Van Leer limiter to the numerical methods. In order to do so, the subroutine CELLUVW is modified in the FORTRAN code. In this subroutine the new cell face values of different parameters, such as the velocity component in the x, y and z-direction, are calculated. The new part of this subroutine is appended in Appendix B.

In the research of Manickathan et al. [12] the Linear Upwind Difference Scheme is used to model the convection terms in the RANS equations. However, since the LUDS scheme is highly numerically diffusive, this could have an impact of the performance of our simulations. In the thesis of Espen Tielrolff [10] the QUICK and CDS is implemented in the FORTRAN code. As expected, the used differencing scheme had some effect on the obtained flow field.

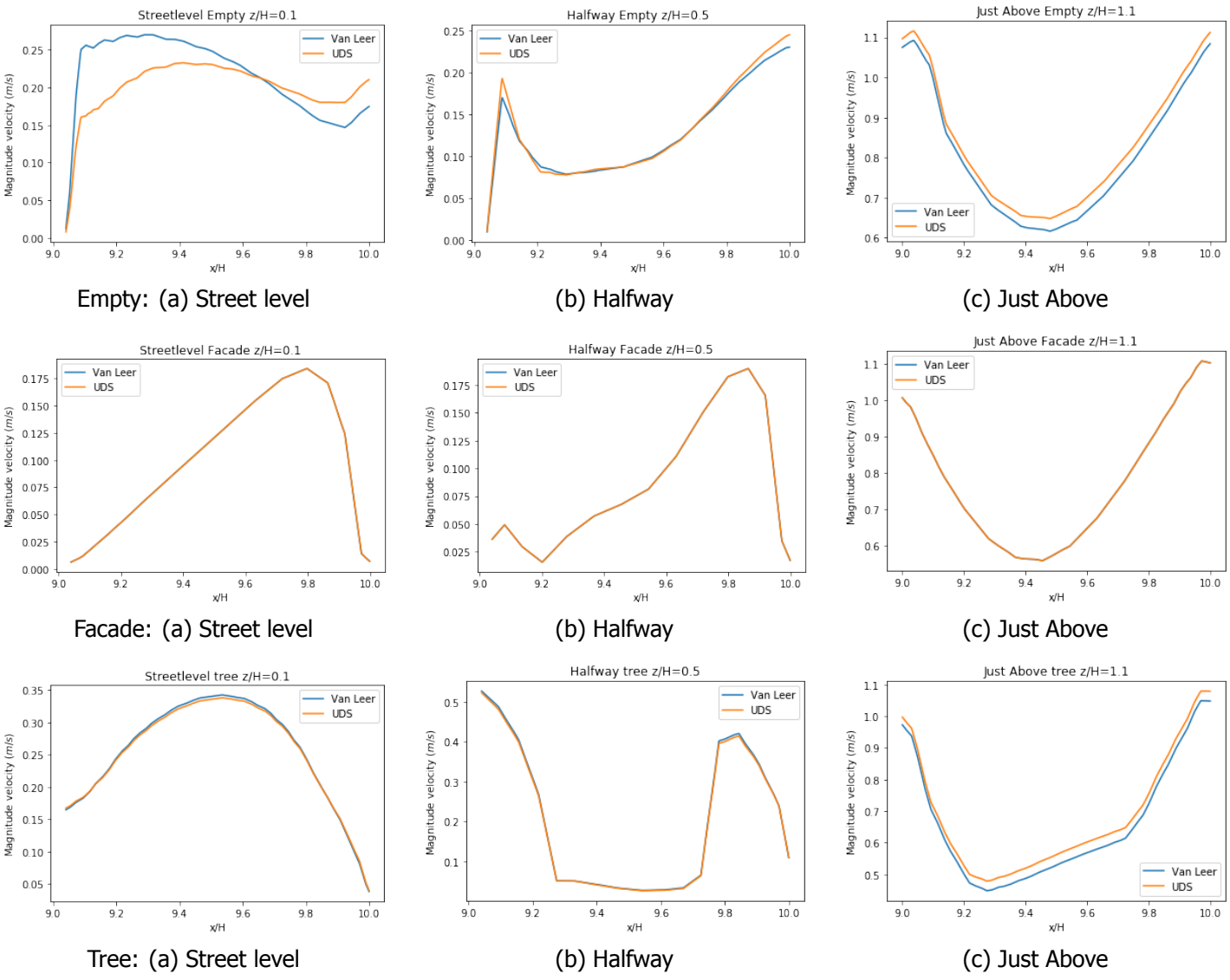
With implementing a high resolution method in the FORTRAN, the goal is to achieve a better accuracy in smooth flow and better resolution of sharp gradients.

7.3. Results: Comparison UDS and Van Leer limiter

To analyse the implemented Van Leer limiter, both UDS and Van Leer limiter are used to calculate the momentum terms. The same procedure is used as in Chapter 5 and Chapter 6 with the only difference

that 100% Van Leer limiter is used in stead of UDS. This is done for the empty street canyon, the street canyon with green facades and the street canyon with trees in the middle. The simulations yielded good convergence of the residuals. To further extend the analysis, the simulations are also executed using different percentages of QUDS. Unfortunately, these simulations yielded unstable results and the residuals did not meet the convergence criterion.

The velocity profiles are extracted and displayed in Figure 7.2.



Empty: (a) Street level (b) Halfway (c) Just Above

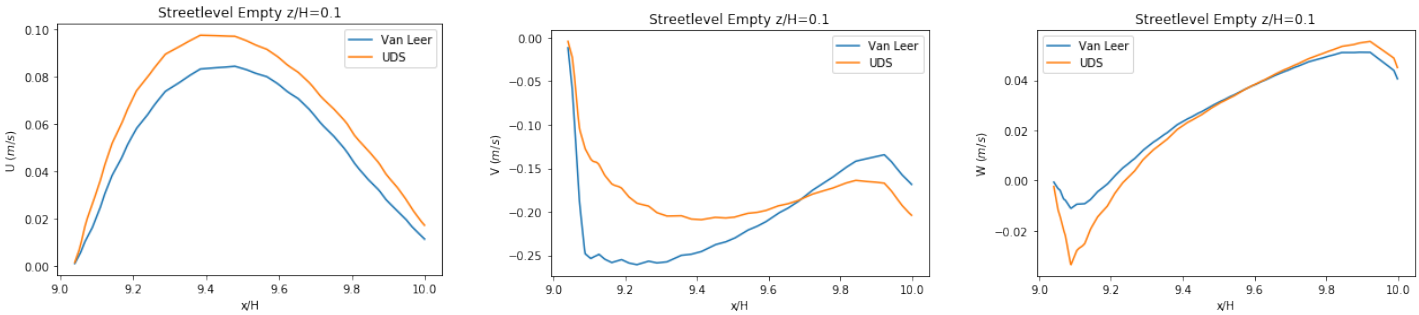
Facade: (a) Street level (b) Halfway (c) Just Above

Tree: (a) Street level (b) Halfway (c) Just Above

Figure 7.2: Magnitude of the total velocity: Van Leer limiter and UDS results compared for the empty street canyon, the street canyon with green facades and the street canyon with trees

The results for the empty canyon show that the Van Leer and UDS yield different numerical values. Especially at street level (Figure 7.2: Empty (a)), the TVD and UDS simulations differ quite much. The shape of the Van Leer and UDS lines are really distinct. This is discussed below in more detail.
On the other hand, the shape of the simulations of the halfway and just above figure for the empty street canyon are qualitatively seen similar.

On the contrary, the Van Leer and UDS simulations with vegetation resemble each other much better. The results of the green facade yields almost no differences between UDS and Van Leer. Lastly, the results of the tree option give some small differences between the Van Leer limiter and Van Leer.



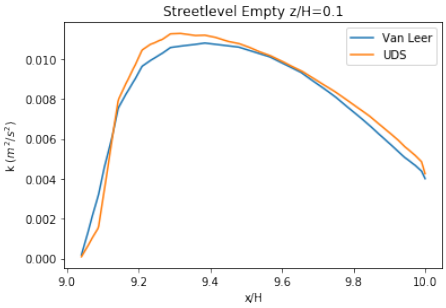
Street level Empty: (a) U (b) V (c) W

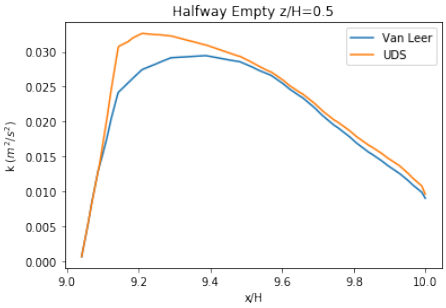
Figure 7.3: Profile plots of the three velocity components at street level: Van Leer limiter and UDS results of the empty street canyon

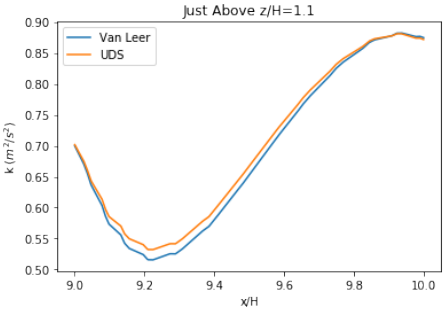
In Figure 7.3 the results of the Van Leer and UDS simulations of the three velocity components are showed. U is the velocity component in the x-direction, V the component in the y-direction and W the component in the z-direction. It is visible that the difference between Van Leer and UDS for V contributes the most to the shape difference of the magnitude of the total velocity component in Figure 7.2. A possible explanation is that the method is very sensitive low to the ground.

The turbulent kinetic energy profiles are extracted and displayed in Figure 7.4.

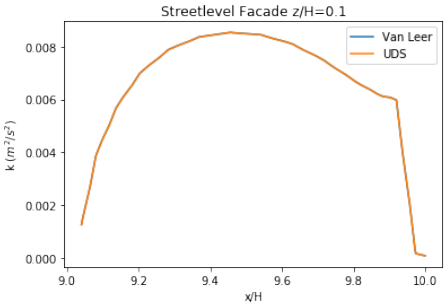
In general, the differences between Van Leer and UDS for the turbulent kinetic energy are smaller than in the velocity profiles. Just as with the velocity profiles, the profiles of empty street canyon differ the most. The results of the street canyon with green facades are almost identical for Van Leer and UDS. The results of the street canyon with trees show some minor differences.

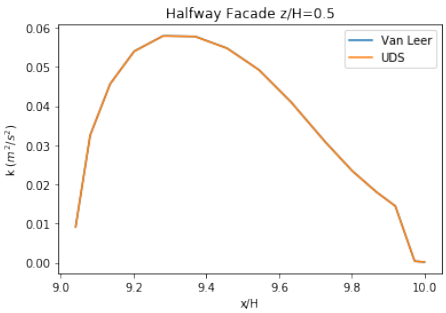


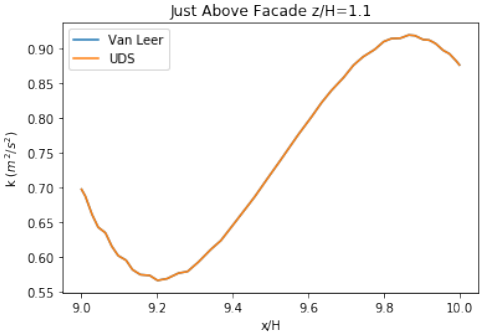




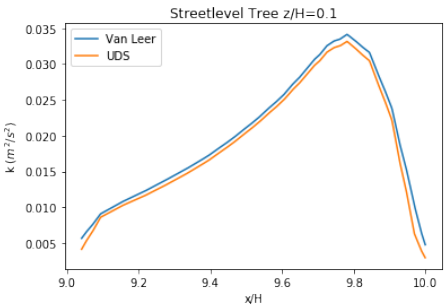
Empty: (a) Streetlevel (b) Halfway (c) Just Above

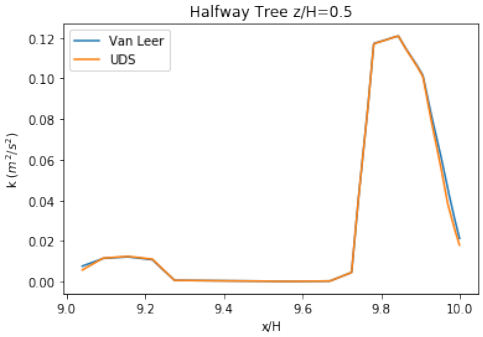


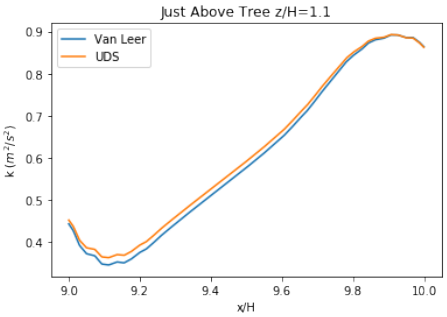




Facade: (a) Streetlevel (b) Halfway (c) Just Above







Tree: (a) Street level (b) Halfway (c) Just Above

Figure 7.4: Turbulent kinetic energy: Van Leer limiter and UDS results compared for the empty street canyon, the street canyon with green facades and the street canyon with trees

8

Conclusions and Recommendations

8.1. Conclusions

Simulations are performed to obtain knowledge about the influence of vegetation in a street canyon and to validate the implementation of numerical methods that are used to calculate the influence of vegetation on the transport of heat. To this end, the wind tunnel experiments of Gromke [5] are simulated.

The results regarding the velocities in the street canyon are in line with the expectations. From the simulations, the conclusion can be drawn that trees in a street canyon have a bigger influence on the vertical velocity than the green facade. The tree option causes higher values of the vertical velocity close to the wall. In the results of the simulations of the street canyon with green facades and the empty street canyon, some unexpected behaviour is detected: only positive values are seen in the street canyon, whereas both positive and negative values are expected, which causes a natural circulation flow. The latter is also seen in the experiments and simulation by Gromke [20]. A possible explanation is that one of the simulation settings is not in line with Gromke's research.

From the profile plots, it can be concluded that the street canyon with trees has the highest magnitude velocity. This is probably due to the circulation of the air under the canopy of the trees. The turbulent kinetic energy is also influenced by the vegetation. The green facades have little influence on the TKE, whereas the integration of trees in the street canyon influence the TKE much more.

The simulations involving temperature showed significant effects of the vegetation in the street canyon. At street level, the green facades yielded a stronger cooling in the canyon than trees. However, the cooling effect of trees is stronger halfway the canyon and just above the canyon in comparison with green facades. The contour and profile plots of the street canyon with trees suggests that the cooling power is stronger than one would expect in a real street canyon experiment. The defined sink strength

Although this drop is also expected in experiments, the specific strength of this drop in the simulation is much stronger than that.

The last research question concerned the implementation of a new numerical method to he Computational Fluid Dynamics model, written in FORTRAN: a Higher Order Scheme with the Van Leer limiter. The simulations with this new numerical method yielded very similar results compared with the UDS and QUDS simulations. One outlier is noticed in the velocity component of the vertical direction for the empty street canyon. A possible explanation could be the sensitivity of the method near the ground.

8.2. Recommendations

On the basis of this research and conclusions some recommendations can be formulated for further research.

First of all, it is suggested to improve the set up of the street canyon. By doing this, the behaviour of the vertical velocity component is likely to behave more naturally. A suggestion to improve the set

up is to expand the set up with a total of 10 sequential buildings. If the results in the last street canyon are analysed, the situation is probably more in line with a real life scenario.

As discussed in Section 8.1, the cooling power of the trees is stronger than one would expect in an experiment. Therefore, it is recommended to test different cooling powers of the vegetation. The predefined sink-strength can be fine-tuned.

In the temperature simulations, the buildings are modelled as solid blocks. However, this model assumption can be argued. In modern cities, buildings can be seen as a heat source term. For example, air conditioners can raise the outside temperature.

The study of the effect of vegetation in a street canyon could be extended in the future by comparing combinations of vegetation and adjusting the parameters of the green facades. In this research a full coverage of the facades is treated, but a coverage of 50% is more reasonable.

The results in Chapter 7 are made with the first simulations of the implementation of the Van Leer limiter. In the future, more simulations need to be made to test the implementation more thoroughly. Based on the performed simulations in this research, several recommendations are made for further research. A first suggestion is to extend this limiter, so that it can solve the temperature part. Now only the momentum equations are solved. Because the temperature gives sharper gradients, the differences between the Van Leer simulations and the UDS and QUDS simulations are presumably larger. Furthermore, the numerical methods could be extended by adding another limiter. In Appendix B, the Koren scheme is already added to the code. This scheme can be tested as well. Furthermore, it is recommended to compare the results of the Van Leer limiter to the results using QUDS.

Acknowledgements

I would like to express my great appreciation to professor Kenjeres and professor Vuik for the weekly meetings and sharing their valuable knowledge. Even during the summer, they were very flexible and available to supervise me. Although, I had many struggles working with the new software, the new programming language and remote working, this project helped me with my development as a researcher.

Furthermore, I would like to thank Nikola Mirkov for his enormous insights and help with the implementation of the Van Leer limiter in FORTRAN.

I would also like to thank Jacob van der Woude and Jos Thijssen for their time to read and evaluate this thesis.

Tot slot wil ik mijn familie, Yuran en studievrienden Esmee, Sam en Stephan bedanken voor de steun bij dit project. Zowel voor een luisterend oor wanneer het even tegenzat, als voor de inhoudelijke tips en adviezen.

Ilse Nijensteen
Delft, August 2020

Bibliography

[1] R. Vautard and G. J. Van Oldenborgh, *Human contribution to the record-breaking July 2019 heat-wave in Western Europe,* (2019).

[2] N. E. Theeuwes, G.-J. Steeneveld, R. R. J., and A. A. Holtslag, *A diagnostic equation for the daily maximum urban heat island effect for cities in northwestern europe,* International journal of climatology **37**, 443 (2017).

[3] P. Gkatsopoulos, *A Methodology for Calculating Cooling from Vegetation Evapotranspiration for Use in Urban Space Microclimate Simulations,* Procedia Environmental Sciences **38**, 477 (2017).

[4] M. Stuiver, J. Spijker, S. de Vries, R. Snep, and C. Jacobs, *Zeven redenen om te investeren in een groene stad,* (2017).

[5] C. Gromke, B. Blocken, W. Janssen, B. Merema, T. van Hooff, and H. Timmermans, *CFD analysis of transpirational cooling by vegetation : Case study for specific meteorological conditions during a heat wave in Arnhem, Netherlands,* Building and Environment **83**, 11 (2015).

[6] B. ter Kuile, *Numerical simulations of pollutants in urban areas with vegetation using a modified k-ϵ model*, Master's thesis, Delft University of Technology (2009).

[7] H. v. Akker and R. Mudde, *Fysische transportverschijnselen: Denken in balansen* (Delft Academic Press, Delft, 2014).

[8] A. Filipovic, *Design and implementation of a sink term for the thermal energy equation pertaining to transpiration from vegetation*, Master's thesis, Delft University of Technology (2016).

[9] R. Narasimha, S. R. Kumar, A. Prabhu, and S. Kailas, *Turbulent flux events in a nearly neutral atmospheric boundary layer,* :Philosophical Transactions of the Royal Society **365**, 841 (2007).

[10] E. Tierolff, *Extending a RANS Solver with heat and pollution modules for dispersion models in urban areas with vegetation*, Master's thesis, Delft University of Technology (2018).

[11] M. van Soest, *Effect of Vegetation on Airflow and Turbulence in the Built Environment*, Master's thesis, Delft University of Technology (2019).

[12] L. Manickathan, T. Defraeye, J. Allegrini, D. Derome, and J. Carmeliet, *Transpirative cooling potential of vegetation in urban environment using coupled CFD and leaf energy balance model,* Proceedings of the 15th IBPSA Conference (2017).

[13] R. Hagenzieker, *Numerical simulations of turbulent flows over hills and complex urban areas with dispersion of pollutants*, Master's thesis, Delft University of Technology (2006).

[14] K. Hanjalic, S. Kenjeres, M. Tummers, and H. Jonker, *Analysis and Modelling of Physical Transport Phenomena* (VSSD, Delft, 2009).

[15] S. Kenjereš, *Numerical Modelling of Complex Buoyancy-Driven Flows*, Ph.D. thesis, Delft University of Technology (1999).

[16] L. Zondag, *Parallelization of RANS solver with obstacles using MPI*, Master's thesis, Delft University of Technology (2018).

[17] J. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics* (Springer Verlag, Berlin, Heidelberg, NewYork, 2002).

[18] R. J. Leveque, *Finite-Volume Methodes for Hyperbolic Problems* (Cambridge University Press, Cambridge, 2004).

[19] N. Waterson and H. Deconinck, *Design principles for bounded higher-order convection schemes – a unified approach,* Journal of Computational Physics (2007).

[20] C. Gromke, R. Buccolieri, S. D. Sabatino, and B. Ruck, *Dispersion study in a street canyon with tree planting by means of wind tunnel and numerical investigations – evaluation of cfd data with experimental data,* Atmospheric Environment **42**, 11 (2008).

Appendices

A

Python code from Chapter 6

```
import scipy.sparse
import scipy as sc
import matplotlib.pyplot as plt
from scipy.linalg import norm
import numpy as np

#Initialization

fancy_plots = False

Nx = 200
dx = 10/Nx
xfaces = np.linspace(-5,5,Nx+1)
xcen = np.zeros(Nx)

for i in range(len(xfaces)-1):
    xcen[i]=(xfaces[i]+xfaces[i+1])/2

T = 15
Nt = 400
dt = T/Nt
u = 1
CFL = u*dt/dx
#CFL is the convergence condition by --CourantFriedrichsLewy

print("The CFL number is equal to %.3f." %CFL)

q_upw = np.zeros((Nx,Nt))
q_beam = np.zeros((Nx,Nt))
q_vanLeer = np.zeros((Nx,Nt))

solution = np.zeros((Nx,Nt))

for j in range(Nt):
    for i in range(len(xcen)):
        if -3<=(xcen[i]-u*dt*j+5)%10-5<=-1:
            solution[i,j]=(1-np.cos(np.pi*((xcen[i]-u*dt*j+5)%10-5-1)))/2
        elif 1<=(xcen[i]-u*dt*j+5)%10-5<=3:
            solution[i,j]=1
```

```
        else:
            solution[i,j]=0

q_upw[:,0]= solution[:,0]
q_beam[:,0]= solution[:,0]
q_vanLeer[:,0]= solution[:,0]

# Upwind

def solutionve_upw(q_upw,CFL):
    if CFL>0:
        maindiagional=(1-CFL)*np.ones(Nx)
        lowerdiagonal = CFL*np.ones(Nx)
        A = sc.sparse.diags([maindiagonal, lowerdiagonal], [0, -1], format="lil")
        A[0,Nx-1]=CFL
    elif CFL<0:
        maindiagonal = (1+CFL)*np.ones(Nx)
        upperdiagonal = -CFL*np.ones(Nx)
        A = sc.sparse.diags([maindiagnonal, upperdiagnonal], [0, 1], format="lil")
        A[Nx-1,0]=-CFL

    A = sc.sparse.csr_matrix(A)

    for j in range(1,Nt):
        q_upw[:,j]=A.dot(q_upw[:,j-1])

    return q_upw

q_upw = solutionve_upw(q_upw,CFL)

num,= plt.plot(xcen,q_upw[:,Nt-1],label='Upwind')
exact, = plt.plot(xcen,solution[:,Nt-1],label='Exact')
plt.xlabel('x',fontsize=15)
plt.legend(handles=[num,exact])
Error_upw=solution[:,Nt-1]-q_upw[:,Nt-1]
normError_upw=norm(Error_upw)

#Beam-Warming

def solve_beam(q_beam,CFL):
    if CFL>0:
        maindiagnonal = (1-CFL*3/2+(CFL**2)/2)*np.ones(Nx)
        lowerdiagnonal = (2*CFL-CFL**2)*np.ones(Nx)
        lowlowerdiagnonal = (-CFL/2+(CFL**2)/2)*np.ones(Nx)
        A = sc.sparse.diags([maindiagnonal, lowerdiagnonal, lowlowerdiagnonal], [0,-1
        A[0,Nx-1]=2*CFL-CFL**2
        A[0,Nx-2]=-CFL/2+(CFL**2)/2
        A[1,Nx-1]=-CFL/2+(CFL**2)/2
    elif CFL<0:
        maindiagnonal = (1+CFL*3/2+(CFL**2)/2)*np.ones(Nx)
        upperdiagnonal = (-2*CFL-CFL**2)*np.ones(Nx)
        upupperdiagnonal = (CFL/2+(CFL**2)/2)*np.ones(Nx)
        A = sc.sparse.diags([maindiagnonal,upperdiagnonal,upupperdiagnonal],[0,1,2],f
        A[Nx-1,0]=-2*CFL-CFL**2
        A[Nx-1,1]=CFL/2+(CFL**2)/2
        A[Nx-2,0]=CFL/2+(CFL**2)/2
```

```
    A = sc.sparse.csr_matrix(A)
    # This is a Compressed Sparse Row matrix

    for j in range(1,Nt):
        q_beam[:,j]=A.dot(q_beam[:,j-1])

    return q_beam

q_beam = solve_beam(q_beam,CFL)
num,= plt.plot(xcen,q_beam[:,Nt-1],label='Beam-Warming')
exact, = plt.plot(xcen,solution[:,Nt-1],label='Exact')
plt.xlabel('x',fontsize=15)
plt.legend(handles=[num,exact])

Error_beam=solution[:,Nt-1]-q_beam[:,Nt-1]
normError_beam = norm(Error_beam)

#Define High Order with van Leer limiter as a function

def solve_vanLeer(q_vanLeer,CFL,u):
    theta = -5.0*np.ones(Nx+1)
    phi = -2000

    def ev_theta(q,u):
        if u>0:
            #boundary values at left
            if (q[0]-q[Nx-1]) != 0:
                theta[0]=(q[Nx-1]-q[Nx-2])/(q[0]-q[Nx-1])
            else:
                theta[0]=2000
            if (q[1]-q[0]) != 0:
                theta[1]=(q[0]-q[Nx-1])/(q[1]-q[0])
            else:
                theta[1] = 2000
            #internal values
            for i in range(2,len(theta)-1):
                if (q[i]-q[i-1]) != 0:
                    theta[i]=(q[i-1]-q[i-2])/(q[i]-q[i-1])
                else:
                    theta[i]=2000
            #boundary values at right
                theta[Nx]=theta[0]
            else:
                theta[Nx]=2000
        elif u<0:
            #boundary values at right
            if (q[0]-q[Nx-1]) != 0:
                theta[Nx]=(q[1]-q[0])/(q[0]-q[Nx-1])
            else:
                theta[Nx]=2000
            if (q[Nx-1]-q[Nx-2]) != 0:
                theta[Nx-1]= (q[0]-q[Nx-1])/(q[Nx-1]-q[Nx-2])
            else:
                theta[Nx-1]=2000
            #internal values
            for i in range(1,len(theta)-3):
```

```
                if (q[i]-q[i-1]) != 0:
                    theta[i]=(q[i+1]-q[i])/(q[i]-q[i-1])
                else:
                    theta[i]=2000
            theta[0]=theta[Nx]
        else:
            print("PLEASE␣CHECK!␣u=0?␣Is␣that␣correct?")
        return theta

    def ev_phi(theta):
        phi=(theta+np.absolute(theta))/(1+np.absolute(theta))
        return phi

    A = np.zeros((Nx,Nx))

    for j in range(1,Nt):
        theta = ev_theta(q_vanLeer[:,j-1],u)
        #assemble coefficient matrix at each time step
        if u>0:
            for ind in range(Nx):
                A[ind,ind]=1-CFL+0.5*CFL*(1-CFL)*(ev_phi(theta[ind+1])+ev_phi(theta[i
                if ind>0:
                    A[ind,ind-1]=CFL-0.5*CFL*(1-CFL)*ev_phi(theta[ind])
                if ind<Nx-1:
                    A[ind,ind+1]=-0.5*CFL*(1-CFL)*ev_phi(theta[ind+1])
            A[0,Nx-1]=CFL-0.5*CFL*(1-CFL)*ev_phi(theta[0])
            A[Nx-1,0]=-0.5*CFL*(1-CFL)*ev_phi(theta[Nx])
            As = sc.sparse.csr_matrix(A)
            q_vanLeer[:,j]=As.dot(q_vanLeer[:,j-1])
        elif u<0:
            for ind in range(Nx):
                A[ind,ind]=1+CFL-0.5*CFL(1+CFL)*(ev_phi(theta[ind+1])+ev_phi(theta[in
                if ind>0:
                    A[ind,ind-1]=0.5*CFL*(1+CFL)*ev_phi(theta[ind])
                if ind<Nx-1:
                    A[ind,ind+1]=-CFL+0.5*CFL*(1+CFL)*ev_phi(theta[ind+1])
            A[0,Nx-1]= 0.5*CFL*(1+CFL)*ev_phi(theta[0])
            A[Nx-1,0]=-CFL+0.5*CFL*(1+CFL)*ev_phi(theta[Nx])
            As = sc.sparse.csr_matrix(A)
            q_vanLeer[:,j]=As.dot(q_vanLeer[:,j-1])
        else:
            print("PLEASE␣CHECK!␣u=0?␣Is␣that␣correct?")

    return q_vanLeer

q_vanLeer = solve_vanLeer(q_vanLeer,CFL,u)

num,= plt.plot(xcen,q_vanLeer[:,Nt-1],label='van␣Leer')
exact, = plt.plot(xcen,solution[:,Nt-1],label='Exact')
plt.xlabel('x',fontsize=15)
plt.legend(handles=[num,exact])

Error_vanLeer=solution[:,Nt-1]-q_vanLeer[:,Nt-1]
normError_vanLeer = norm(Error_vanLeer)

# Plot all methods in one figure
```

```
plt.plot(xcen,q_upw[:,Nt-1])
plt.plot(xcen,q_beam[:,Nt-1])
plt.plot(xcen,q_vanLeer[:,Nt-1])
plt.plot(xcen,solution[:,Nt-1])
plt.legend(('Upwind','Beam-Warming", "Van␣Leer", "Exact"))
```

B

TVD implementation in Fortran

```
SUBROUTINE CELUVW(NIE,NJE,NKE,IDEW,IDNS,IDTB, &
           FIF,FIS,FIT,ACFE,ACFW,FCF)
!#########################################################

! Interpolate velocity from center of gridbox to grid-faces

!---------------------------------------------------------------------
! Initialize subroutine

! Open modules
use INC_PAR
use INC_BBB
use INC_BUOY
use INC_COEFB
use INC_GEO
use INC_INDEX
use INC_OBSTACLE
use INC_TIME
use INC_UVW
use INC_COEF

! Don't allow implicit declaration of variables
implicit none

! Declare variables used in this subroutine
integer :: NIE,NJE,NKE
integer :: IDEW,IDNS,IDTB
real, dimension(NXYZA) :: FIF
real, dimension(NXYZA) :: FIS
real, dimension(NXYZA) :: FIT
real, dimension(NXYZA) :: ACFE
real, dimension(NXYZA) :: ACFW
real, dimension(NXYZA) :: FCF
real :: AE1
real :: ARE
real :: AW1
real :: CE,CW
real :: CH2,CH3
real :: DE
```

```
real :: FLCF
real :: FXE,FXW,FYN,FYS,FZT,FZB
real :: g11,g12,g21,g22
real :: GAM
real :: GAME
integer :: INB,INT,INN,INS,INE,INW
integer :: INBE,INTE,INNE,INSE,INEE,INWE
integer :: INBS,INTS,INNS,INSS,INES,INWS
integer :: INBW
integer :: INP
integer :: LIK
integer :: LKK
real :: SHIGH1,SHIGH2,SHIGH3
real :: SUEU,SUEV,SUEW
real :: UB,UE,UN,US,UT
real :: VB,VE,VN,VS,VT
real :: VOLE
real :: WB,WE,WN,WS,WT
real :: r1,r2,r3,r4,r5,r6                          ! TVD related
real :: PSIE1,PSIE2,PSIE3,PSIW1,PSIW2,PSIW3        !
real :: FUUDS,FVUDS,FWUDS,FUHIGH,FVHIGH,FWHIGH     !
!-------------------------------------------------------------------------------
GAM=GDS(IU)

!.....CALCULATE EAST,TOP,NORTH  CELL FACE
grid_K : DO K=2,NKE
  grid_I : DO I=2,NIE
    grid_J : DO J=2,NJE
      INP=LK(K)+LI(I)+J
      INE=INP+IDEW
      INW=INP-IDEW
      INN=INP+IDNS
      INS=INP-IDNS
      INB=INP-IDTB
      INT=INP+IDTB

      INBS=INB-IDNS

      INSE=INE-IDNS
      INBE=INE-IDTB

      INBW=INW-IDTB

      INTE=INT+IDEW
      INNE=INN+IDEW

!.....INTERPOLATION FACTORS IN FIRST,SECOND AND THIRD WAY
      FXE=FIF(INP)
      FXW=1.-FXE
      FYN=FIS(INP)
      FYS=1.-FYN
      FZT=FIT(INP)
      FZB=1.-FZT
!.....COMPONENTS OF THREE VECTORS
!.....FIRST
      DXKS=XC(INE)-XC(INP)
```

```
      DYKS=YC(INE)-YC(INP)
      DZKS=ZC(INE)-ZC(INP)
!.....SECOND
      DXET=.5*(X(INP)-X(INS)+X(INB)-X(INBS))
      DYET=.5*(Y(INP)-Y(INS)+Y(INB)-Y(INBS))
      DZET=.5*(Z(INP)-Z(INS)+Z(INB)-Z(INBS))
!.....THIRD
      DXZD=.5*(X(INP)-X(INB)+X(INS)-X(INBS))
      DYZD=.5*(Y(INP)-Y(INB)+Y(INS)-Y(INBS))
      DZZD=.5*(Z(INP)-Z(INB)+Z(INS)-Z(INBS))
!.....SECOND X THIRD DEFINE  ALWAYS CF AREA
      B11=DYET*DZZD-DYZD*DZET
      B12=DXZD*DZET-DXET*DZZD
      B13=DXET*DYZD-DYET*DXZD
      B21=DZKS*DYZD-DZZD*DYKS
      B22=DXKS*DZZD-DXZD*DZKS
      B23=DXZD*DYKS-DXKS*DYZD
      B31=DYKS*DZET-DYET*DZKS
      B32=DZKS*DXET-DZET*DXKS
      B33=DXKS*DYET-DXET*DYKS

      ARE=B11**2+B12**2+B13**2

!.....FIRST .(SECOND X THIRD) = VOL
      VOLE=DXKS*B11+DYKS*B12+DZKS*B13
!.....CELL FACE
      GAME=(VIS(INP)*FXW+VIS(INE)*FXE)/VOLE

!.....EXPLICIT PART OF DIFFUSION FLUXES
      UE=U(INP)*FXW+U(INE)*FXE
      VE=V(INP)*FXW+V(INE)*FXE
      WE=W(INP)*FXW+W(INE)*FXE
      UB=(U(INB)*FXW+U(INBE)*FXE)*(1.-FIT(INB))+UE*FIT(INB)
      VB=(V(INB)*FXW+V(INBE)*FXE)*(1.-FIT(INB))+VE*FIT(INB)
      WB=(W(INB)*FXW+W(INBE)*FXE)*(1.-FIT(INB))+WE*FIT(INB)
      UT=(U(INT)*FXW+U(INTE)*FXE)*FZT+UE*FZB
      VT=(V(INT)*FXW+V(INTE)*FXE)*FZT+VE*FZB
      WT=(W(INT)*FXW+W(INTE)*FXE)*FZT+WE*FZB
      US=(U(INS)*FXW+U(INSE)*FXE)*(1.-FIS(INS))+UE*FIS(INS)
      VS=(V(INS)*FXW+V(INSE)*FXE)*(1.-FIS(INS))+VE*FIS(INS)
      WS=(W(INS)*FXW+W(INSE)*FXE)*(1.-FIS(INS))+WE*FIS(INS)
      UN=(U(INN)*FXW+U(INNE)*FXE)*FYN+UE*FYS
      VN=(V(INN)*FXW+V(INNE)*FXE)*FYN+VE*FYS
      WN=(W(INN)*FXW+W(INNE)*FXE)*FYN+WE*FYS
!.....GRAD FI +(GRAD FI)*I    ;I=1,2,3
!.....HELP COEFICIENTS FOR ALL THREE VELOCITIES (GRAD FI IS THE SAME)
      CH2=B11*B21+B12*B22+B13*B23
      CH3=B31*B11+B32*B12+B33*B13

      SUEU=GAME*(CH2*(UN-US)+     &
                 CH3*(UT-UB)+     &
           (B11*(U(INE)-U(INP))+B21*(UN-US)+B31*(UT-UB))*B11+ &
           (B11*(V(INE)-V(INP))+B21*(VN-VS)+B31*(VT-VB))*B12+  &
           (B11*(W(INE)-W(INP))+B21*(WN-WS)+B31*(WT-WB))*B13)
      SUEV=GAME*(CH2*(VN-VS)+ &
                 CH3*(VT-VB)+ &
```

```
          (B12*(U(INE)-U(INP))+B22*(UN-US)+B32*(UT-UB))*B11+  &
          (B12*(V(INE)-V(INP))+B22*(VN-VS)+B32*(VT-VB))*B12+  &
          (B12*(W(INE)-W(INP))+B22*(WN-WS)+B32*(WT-WB))*B13)
      SUEW=GAME*(CH2*(WN-WS)+ &
                 CH3*(WT-WB)+ &
          (B13*(U(INE)-U(INP))+B23*(UN-US)+B33*(UT-UB))*B11+  &
          (B13*(V(INE)-V(INP))+B23*(VN-VS)+B33*(VT-VB))*B12+  &
          (B13*(W(INE)-W(INP))+B23*(WN-WS)+B33*(WT-WB))*B13)

! .....DIFUSION COEFFICIENT
! .....CONVECTION FLUXES - UDS
! .....EXPLICIT PART OF CONVECTION FLUXES - GAMA*(CDS-UDS)

      DE=GAME*ARE
      FLCF=FCF(INP)
      CE=MAX(-FLCF,ZERO)
      CW=MAX( FLCF,ZERO)
      ACFE(INP)=DE+CE
      ACFW(INE)=DE+CW
!-----------------------------------------------------
!     [ CENTRAL DIFFERENCING SCHEME (CDS) ]
!-----------------------------------------------------
      CDS : IF(LCDS.EQ.1) THEN
      AE1=-(CE+FLCF*FXE)*GAM
      AW1=-(CW-FLCF*FXW)*GAM
      SHIGH1=0.
      SHIGH2=0.
      SHIGH3=0.
      END IF CDS
!-----------------------------------------------------------
!     [ LINEAR UPWIND DIFFERENCING SCHEME (LUDS) ]
!-----------------------------------------------------------
      LUDS : IF(LLUDS.EQ.1) THEN
        SHIGH1=0.
        SHIGH2=0.
        SHIGH3=0.
        AW1=0.
        AE1=0.

     IF(K.GT.2.AND.K.LT.NK-1.AND. &
           J.GT.2.AND.J.LT.NJ-1.AND. &
           I.GT.2.AND.I.LT.NI-1) THEN
!
! sasa's fix - lluds: 11.07.2016 / only flow in x-direction
!
!     IF(K.GT.5.AND.K.LT.NK-4.AND. &
!        J.GT.5.AND.J.LT.NJ-4.AND. &
!        I.GT.5.AND.I.LT.NI-4) THEN

          AE1=-(CE-CE*(1+FIF(INP+IDEW)))*GAM
          AW1=-(CW-CW*(2-FIF(INP-IDEW)))*GAM

          SHIGH1=-GAM*(CE*U(INP+2*IDEW)*FIF(INP+IDEW) &
                      -CW*U(INP-IDEW)*(1.-FIF(INP-IDEW)))
          SHIGH2=-GAM*(CE*V(INP+2*IDEW)*FIF(INP+IDEW) &
                      -CW*V(INP-IDEW)*(1.-FIF(INP-IDEW)))
```

```
          SHIGH3=-GAM*(CE*W(INP+2*IDEW)*FIF(INP+IDEW) &
                      -CW*W(INP-IDEW)*(1.-FIF(INP-IDEW)))
        END IF
      END IF LUDS
!-------------------------------------------------------------
!     [ QUADRATIC UPWIND DIFFERENCING SCHEME (QUDS) ]
!-------------------------------------------------------------
      QUDS : IF(LQUDS.EQ.1) THEN
        SHIGH1=0.
        SHIGH2=0.
        SHIGH3=0.
        AW1=0.
        AE1=0.

        IF(K.GT.2.AND.K.LT.NK-1.AND. &
           J.GT.2.AND.J.LT.NJ-1.AND. &
           I.GT.2.AND.I.LT.NI-1) THEN

          g11=((2-FIF(INP-IDEW))*FIF(INP)**2)/ &
              (1+FIF(INP)-FIF(INP-IDEW))
          g12=((1-FIF(INP))*(1-FIF(INP-IDEW))**2)/ &
              (1+FIF(INP)-FIF(INP-IDEW))
          g21=((1+FIF(INP+IDEW))*(1-FIF(INP))**2)/ &
              (1+FIF(INP+IDEW)-FIF(INP))
          g22=(FIF(INP+IDEW)**2*FIF(INP))/ &
              (1+FIF(INP+IDEW)-FIF(INP))

          AE1=-(CE+CW*g11-CE*(1-g21+g22))*GAM
          AW1=-(CW+CE*g21-CW*(1-g11+g12))*GAM

          SHIGH1=-GAM*(CE*g22*U(INP+2*IDEW)-CW*g12*U(INP-IDEW))
          SHIGH2=-GAM*(CE*g22*V(INP+2*IDEW)-CW*g12*V(INP-IDEW))
          SHIGH3=-GAM*(CE*g22*W(INP+2*IDEW)-CW*g12*W(INP-IDEW))

        END IF
      END IF QUDS

!-------------------------------------------------------------
!     [ TOTAL VARIATION DIMINISHING SCHEME (TVD) ]
!-------------------------------------------------------------
! Here we implement BOUNDED HIGHER-ORDER CONVECTIVE SCHEMES.
! based on presentation in the following reference:
!   Waterson & Deconinck Journal of Computational Physics 224 (2007) pp. 182-207
! There they give generalized representation of many bounded schemes in flux limiter
! form. Among those are also Total Variation Diminishing Schemes (TVD), TVD being the
! stronger condition then just 'bounded'. For scheme to be TVD necessary condition is
! that the flux limiter function is enclosed within a small region of the Sweby diagram.
! Check out the paper for more details.

      TVD : IF(LTVD.EQ.1) THEN

!+++++Find 'r'. This is universal for all TVD schemes. ++++++
!
!     NOTE: We assume NON-UNIFORM RECTILINEAR GRID.
!     If the grid is uniform, expressions are the same but can be simplified.
!     If the grid is NON-OTHOGONAL we need to change the code a bit.
```

```
!
!     The 'r' is shorten for 'ratio' and what is meant is ratio of gradients on two f
!     These two faces are present one 'e' (that changes too, if ID East-West is IDEW=
!     then our face is 'e', in other cases if IDEW=1, then present face is North face
!     if IDEW=NIJ, present face is Top face 't'. But lets keep things simple and exp
!     it doesn't change for North and Top cell face.
!
!     To understand this lets study fluid flow trough control volumes:
!  _____________________________________________________________________
! |           |           |           |           |           |           |
! |           ww          w           e           ee          |
! |     o WW ===>   o W   ===>   o P   ===>   o E   ===>   o  EE  |
! |           |           |           |           |           |
! |           |           |           |           |           |
! |___________|___________|___________|___________|___________|
!
!     Here direction is from present cell 'P' to East cell 'E' just for clarity.
!     'INP' means - the value IN cell center P,
!     'INE' means - the value IN cell center E.
!     Gradient of U in x-axis direction, at cell face center 'e'
!     is the value difference  (U(INE)-U(INP)) divided by
!     the distance between cell centers (XC(INE)-XC(INP))
!     Similar is for cell face center 'w' between 'P' and 'W' cells..
!     This allows us to define
!.....If flow goes from P to E, i.e. the upwind cell of 'e' is P.
      r1 = (U(INE)-U(INP))*(XC(INP)-XC(INW))/((U(INP)-U(INW))*(XC(INE)-XC(INP)))
      r2 = (V(INE)-V(INP))*(YC(INP)-YC(INW))/((V(INP)-V(INW))*(YC(INE)-YC(INP)))
      r3 = (W(INE)-W(INP))*(ZC(INP)-ZC(INW))/((W(INP)-W(INW))*(ZC(INE)-ZC(INP)))
!
!     We anticipate also the situation where flow goes from E to P,
!     and the upwind cell is E, while the upwind face is 'ee'
!.....If flow goes from E to P, i.e. the upwind cell of 'e' is E.
      r4 = (U(INP)-U(INE))*(XC(INE)-XC(INEE))/((U(INE)-U(INEE))*(XC(INP)-XC(INE)))
      r5 = (V(INP)-V(INE))*(YC(INE)-YC(INEE))/((V(INE)-V(INEE))*(YC(INP)-YC(INE)))
      r6 = (W(INP)-W(INE))*(ZC(INE)-ZC(INEE))/((W(INE)-W(INEE))*(ZC(INP)-ZC(INE)))

!     Having the gradient ratio 'r' between gradients at present face 'e'
!     and an upwind face (either 'w' or 'ee' depending on a flow direction),
!     we can now define flux limiter function Psi = Psi(r).
!     References:
!     B. van Leer, Towards the ultimate conservative difference scheme. IV. A new app
!     N.P.Waterson & H.Deconinck, Design principles for bounded higher-order convecti

!.....PSI for MUSCL scheme:
!.....If flow goes from P to E
      PSIW1 = max(0., min(2.*r1, 0.5*r1+0.5, 2.))
      PSIW2 = max(0., min(2.*r2, 0.5*r2+0.5, 2.))
      PSIW3 = max(0., min(2.*r3, 0.5*r3+0.5, 2.))
!.....If flow goes from E to P
      PSIE1 = max(0., min(2.*r4, 0.5*r4+0.5, 2.))
      PSIE2 = max(0., min(2.*r5, 0.5*r5+0.5, 2.))
      PSIE3 = max(0., min(2.*r6, 0.5*r6+0.5, 2.))

!     Koren scheme!
!     References:
!     B. Koren, Upwind discretization of the steady -NavierStokes equations, Int. J.
```

```
!     B. Koren, A robust upwind discretization method for advection, diffusion and source
AdvectionDiffusion Problems, Vieweg, Braunschweig, 1993, p. 117.
!.....PSI for Koren scheme:
!      twothirds = 2./3.
!      onethird  = 1./3.
!.....If flow goes from P to E
!      PSIW1 = max(0., min(2.*r1, twothirds*r1+onethird, 2.))
!      PSIW2 = max(0., min(2.*r2, twothirds*r2+onethird, 2.))
!      PSIW3 = max(0., min(2.*r3, twothirds*r3+onethird, 2.))
!.....If flow goes from E to P
!      PSIE1 = max(0., min(2.*r4, twothirds*r4+onethird, 2.))
!      PSIE2 = max(0., min(2.*r5, twothirds*r5+onethird, 2.))
!      PSIE3 = max(0., min(2.*r6, twothirds*r6+onethird, 2.))

!.....EXPLICIT CONVECTIVE FLUXES FOR HIGH ORDER BOUNDED SCHEMES
!     $Flux_high_order_scheme = mass_flow_rate_trough_cell_face_e * Phi_e$
!     Phi_e is found by extrapolation from upwind nodes, see eq. (3.29) in Thesis professo
!     Additional multiplication with PSI is application of flux limiters,
!     see eq. (10) in Waterson&Deconinck paper.


      FUHIGH = CW*(U(INP) + FXW*PSIW1*(U(INE)-U(INP))) + &
               -CE*(U(INE) + FXE*PSIE1*(U(INP)-U(INE)))
!        mass flux| bounded interpolation of velocity to face |

      FVHIGH = CW*(V(INP) + FXW*PSIW2*(V(INE)-V(INP))) + &
               -CE*(V(INE) + FXE*PSIE2*(V(INP)-V(INE)))

      FWHIGH = CW*(W(INP) + FXW*PSIW3*(W(INE)-W(INP))) + &
               -CE*(W(INE) + FXE*PSIE3*(W(INP)-W(INE)))

!
!.....EXPLICIT CONVECTIVE FLUXES FOR UDS
      FUUDS=CW*U(INP)-CE*U(INE)
      FVUDS=CW*V(INP)-CE*V(INE)
      FWUDS=CW*W(INP)-CE*W(INE)

!
!.....UPDATE SOURCE VECTORS WITH EXPLICIT PART OF DIFFUSION FLUXES (SUEU, SUEV, SUEW)
!     AND SOURCES DUE TO DEFFERED CORRECTION FOR TVD SCHEME.

!.....To put everything into existing structure with SHIGH1, SHIGH2, SHIGH3, AW1, AE1, we
      AW1=0.0
      AE1=0.0
      SHIGH1 = GAM*(FUUDS-FUHIGH)
      SHIGH2 = GAM*(FVUDS-FVHIGH)
      SHIGH3 = GAM*(FWUDS-FWHIGH)

!     Here we used deffered correction approach for explicit part of convection terms.
!     DEFERRED CORRECTION: UDS part is treated implicitly and difference between UDS and H
!     GAM is gamma - a deferred correction parameter which controls how much of high order
!     is added to source. If GAM=0, we don't add any, and we have pure UDS scheme, even if
!     in the input file.

!
```

```
      END IF TVD

!
!.....EXPLICIT PART OF DIFFUSION FLUXES (SUEU,SUEV,SUEW) AND SOURCES DUE TO DEFERRED
!
      SU(INP)=SU(INP)+AE1*(U(INE)-U(INP))+SUEU+SHIGH1
      SV(INP)=SV(INP)+AE1*(V(INE)-V(INP))+SUEV+SHIGH2
      SW(INP)=SW(INP)+AE1*(W(INE)-W(INP))+SUEW+SHIGH3
!-----------------------------------------------------------------
!......[Vectorization procedure:]
!-----------------------------------------------------------------
      BP(INE)=AW1*(U(INP)-U(INE))-SUEU-SHIGH1
      BT(INE)=AW1*(V(INP)-V(INE))-SUEV-SHIGH2
      BB(INE)=AW1*(W(INP)-W(INE))-SUEW-SHIGH3
!-----------------------------------------------------------------


    END DO grid_J
  END DO grid_I
END DO grid_K

DO K=2,NKE
  LKK=LK(K)
  DO I=2,NIE
    LIK=LKK+LI(I)
    DO J=2,NJE
      INP=LIK+J+IDEW

      SU(INP)=SU(INP)+BP(INP)
      SV(INP)=SV(INP)+BT(INP)
      SW(INP)=SW(INP)+BB(INP)

    END DO !J-loop
  END DO !I-loop
END DO !K-loop

DO IJK=ICST,ICEN
  BP(IJK)=0.
  BT(IJK)=0.
  BB(IJK)=0.
END DO

RETURN

END SUBROUTINE CELUVW
```