CALC-LES: A Fortran Code for LES and Hybrid LES-RANS

Lars Davidson
Div.. of Fluid Dynamics
Dept. of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

June 26, 2020

Contents

1 Introduction

2 Geometrical Details of the Grid

2.1 Grid

[13]

Python and Matlab scripts for synthetic fluctuations The coordinates of the corners (X_C, Y_C, Z_C) of each control volume should be specified by the user, i.e. the grid must be generated by the user. The nodes of the control volume (X_P, Y_P, Z_P) are placed at the center of their control volumes. The control volume adjacent to the boundaries have two nodes, one in the center and one at the boundary, see Fig. 2.1. In any coordinate direction, lets say ξ, there are NI nodes, NI-1 control volume faces, and NI-2 control volumes. The nodes are numbered (from low ξ to high ξ) from 1 to NI , the control volume faces are numbered from 1 to NI-1, and the control volumes from 2 to NI-1. This is the same for η and ζ directions. Note that (ξ, η, ζ) must form a right-hand coordinate system.

CALC-LES employs curvilinear grids but the code can also be used with Cartesian as well as cylindrical coordinate systems.

2.1.1 Nomenclature for the Grid

A schematic control volume grid is shown in Fig. 2.2. Single capital letters define nodes [E(ast), S(outh), etc.], and single small letters define faces of the control volumes. When a location can not be referred to by a single character, combination of letters are used. The order in which the characters appear is: first east-west, then north-south, and finally high-low.

2.1.2 Area Calculation of Control Volume Faces

The area of the control volume faces are calculated as the sum of two triangles. The x-coordinates of the corners of the east face are, for example, X_C(I,J,K), X_C(I,J-1,K), X_C(I,J,K-1) and X_C(I,J-1,K-1); the Y and Z- coordinates are Y_C and Z_C with the same indices, see Fig. 2.3. The area of the two triangles, $A1$, $A2$, is calculated as the cross product

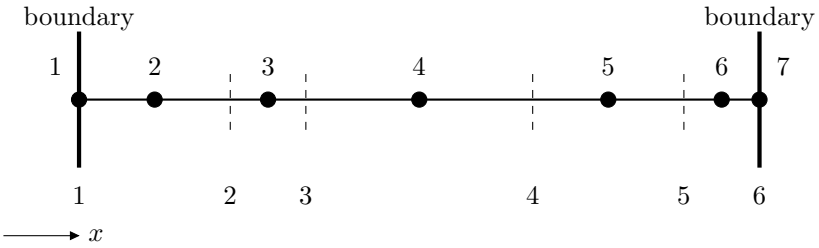

Figure 2.1: 1D grid. $NI = 7$. The bullets denote cell centers which are labeled 1–7. Dashed lines denote control volume faces labeled 1–6. The number of interior control volumes is 5.

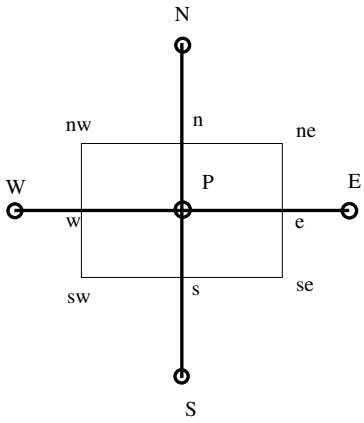

Figure 2.2: Control volume

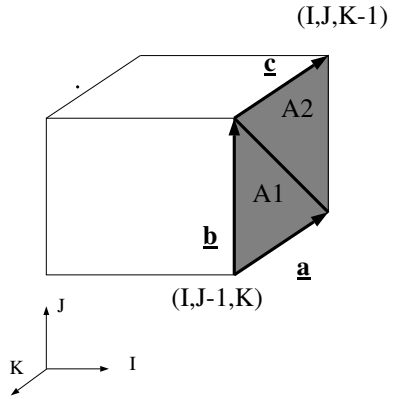

Figure 2.3: Calculation of control volume faces

$$A1 = \frac{1}{2}|\vec{a} \times \vec{b}|; \qquad A2 = \frac{1}{2}|-\vec{b} \times \vec{c}| \tag{2.1}$$

Above is has been assumed that the fourth edge [from (I,J-1,K-1) to (I,J,K-1)] of the east face (shaded area in Fig. 2.3) is approximately equal to $\vec{b}$. The vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$ for faces in Fig. 2.3) are set in a manner that the normal vectors $\vec{n1}$ and $\vec{n2}$ are pointed outwards. For the east face, for example, they are defined as

$\vec{a}$: from corner (I,J-1,K) to (I,J-1,K-1)
$\vec{b}$: from corner (I,J-1,K) to (I,J,K)
$\vec{c}$: from corner (I,J,K) to (I,J,K-1)

The Cartesian components of $\vec{a}$ are thus

$$\begin{aligned} a_x &= X(I, J-1, K-1) - X(I, J-1, K) \\ a_y &= Y(I, J-1, K-1) - Y(I, J-1, K) \\ a_z &= Z(I, J-1, K-1) - Z(I, J-1, K) \end{aligned} \tag{2.2}$$

The total area for the east face is obtained as

$$|\vec{A}|_e = |\vec{A1}|_e + |\vec{A2}|_e \tag{2.3}$$

The normal vector of the vector area is obtained as the average of the normal vectors of the two triangles

$$\vec{n} = \frac{1}{2}\left(\vec{a} \times \vec{b} - \vec{b} \times \vec{c}\right) \tag{2.4}$$

The Cartesian areas are calculated as

$$\begin{aligned} \vec{A}_{ex} &= |\vec{A}|_e \vec{n} \cdot \vec{e}_x \\ \vec{A}_{ey} &= |\vec{A}|_e \vec{n} \cdot \vec{e}_y \\ \vec{A}_{ez} &= |\vec{A}|_e \vec{n} \cdot \vec{e}_z \end{aligned} \tag{2.5}$$

where $\vec{e}_x$, $\vec{e}_y$ and $\vec{e}_z$ are the Cartesian unit base vector

The areas and the normal vectors of the north and high control volumes faces are calculated in exactly the same way. For the north face the vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$ are defined as

$\vec{a}$: from corner (I-1,J,K) to (I,J,K)
$\vec{b}$: from corner (I-1,J,K) to (I-1,J,K-1)
$\vec{c}$: from corner (I-1,J,K-1) to (I,J,K-1)

For the high faces the vectors a, b and c are defined as

$\vec{a}$: from corner (I-1,J,K) to (I-1,J-1,K)
$\vec{b}$: from corner (I-1,J,K) to (I,J,K)
$\vec{c}$: from corner (I,J,K) to (I,J-1,K)

In **CALC-LES** the Cartesian areas for each face (east, north and high) are calculated only once and stored in three dimensional arrays.

2.1.3 Volume Calculation of Control Volume

The volume is calculated using Gauss' law, see Burns and Wilkes [4]. Gauss' law for a vector field $\vec{B}$ reads

$$\int_V \nabla \cdot \vec{B} dV = \int_A \vec{B} \cdot d\vec{A} \tag{2.6}$$

setting $\vec{B} = \vec{x}$ gives

$$\delta V = \int_V dV = \frac{1}{3} \int_A \vec{x} \cdot d\vec{A} \tag{2.7}$$

In **CALC-LES** the volume is calculated once only and stored in a three dimensional array.

2.1.4 Interpolation

The nodes where all variables are stored are situated in the center of the control volume. When a variable is needed at a control volume face, linear interpolation is used. The value of the variable ϕ at the east face is

$$\phi_e = f_x \phi_E + (1 - f_x)\phi_P \tag{2.8}$$

where

$$f_x = \frac{|\vec{Pe}|}{|\vec{Pe}| + |\vec{eE}|} \tag{2.9}$$

where $|\vec{Pe}|$ is the distance from P (the node) to e (the east face). In **CALC-LES** the interpolation factors (f_x, f_y, f_z) are calculated once only and stored in three-dimensional arrays.

2.2 Gradient

The derivatives of ϕ ($\partial\phi/\partial x_i$) at the cell center are in **CALC-LES** computed as follows. We apply Green's formula to the control volume, i.e.

$$\frac{\partial \Phi}{\partial x} = \frac{1}{V} \int_A \Phi n_x dA, \quad \frac{\partial \Phi}{\partial y} = \frac{1}{V} \int_A \Phi n_y dA, \quad \frac{\partial \Phi}{\partial z} = \frac{1}{V} \int_A \Phi n_z dA$$

where A is the surface enclosing the volume V. For the x component, for example, we get

$$\frac{\partial \Phi}{\partial x} = \frac{1}{V} \left(\Phi_e A_{ex} - \Phi_w A_{wx} + \Phi_n A_{nx} - \Phi_s A_{sx} + \Phi_h A_{hx} - \Phi_l A_{lx} \right) \tag{2.10}$$

where index w, e, s, n, l, h denotes east $(I+1/2)$, west $(I-1/2)$, north $(J+1/2)$, south $(J-1/2)$, high $(K+1/2)$ and low $(K-1/2)$. The derivative $\partial\Phi/\partial x$ is computed in the function `dphidx`.

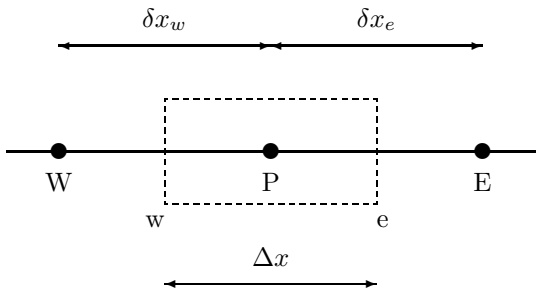

Figure 3.1: 1D control volume. Node P located in the middle of the control volume.

3 Diffusion

We start by looking at 1D diffusion, e.g. the 1D heat conduction equation

$$\frac{d}{dx}\left(k\frac{dT}{dx}\right) + S = 0.$$

To discretize (i.e. to go from a *continuous* differential equation to an algebraic *discrete* equation) this equation is integrated over a control volume (C.V.), see Fig. 3.1.

$$\int_w^e \left[\frac{d}{dx}\left(k\frac{dT}{dx}\right) + S\right] dx = \left(k\frac{dT}{dx}\right)_e - \left(k\frac{dT}{dx}\right)_w + \bar{S}\Delta x = 0 \tag{3.1}$$

where (see Fig. 3.1):

P: an arbitrary node

E, W: its east and west neighbor node, respectively

e, w: the control volume's east and west face, respectively

$\bar{S}$: volume average of S

The temperature T and the coefficient of heat conductivity k are stored at the nodes W, P and E. Now we need the derivatives dT/dx at the faces w and e. These are estimated from a straight line connecting the two adjacent nodes, i.e.

$$\left(\frac{dT}{dx}\right)_e \simeq \frac{T_E - T_P}{\delta x_e}, \quad \left(\frac{dT}{dx}\right)_w \simeq \frac{T_P - T_W}{\delta x_w}. \tag{3.2}$$

The heat conductivity k is also needed at the faces. It is estimated by linear interpolation between the adjacent nodes. For the east face, for example, we obtain

$$k_e = f_x k_E + (1 - f_x)k_P, \; f_x = \frac{0.5\Delta x}{\delta x_e}. \tag{3.3}$$

For an equidistant mesh (constant $\Delta x \Rightarrow \Delta x = \delta x_w = \delta x_e$) $f_x = 0.5$.

Insertion of Eq. 3.2 into Eq. 3.1 gives

$$\begin{aligned} a_P T_P &= a_E T_E + a_W T_W + S_U \\ a_E &= \frac{k_e}{\delta x_e},\ a_W = \frac{k_w}{\delta x_w},\ S_U = \bar{S}\Delta x,\ a_P = a_E + a_W \end{aligned} \tag{3.4}$$

3.1 Convergence Criteria

Compute the residual for Eq. 3.4

$$R = \sum_{\text{all cells}} |a_E T_E + a_W T_W + S_U - a_P T_P|$$

Since we want Eq. 3.4 to be satisfied, the difference of the right-hand side and the left-hand side is a good measure of how well the equation is satisfied. Note that R has the units of the integrated differential equation. Thus, in the present case R has the same dimension as heat transfer rate, i.e. Joule per second $[J/s] = [W]$. If $R = 1[W]$, it means that the residual for the computation is 1. This does not tell us anything, since it is problem dependent. We can have a problem where the total heat transfer rate is $1000[W]$, and a another where it is only $1[W]$. In the former case $R = 1$ means that the solutions can be considered converged, but in the latter case this is not true at all. We realize that we must normalize the residual to be able to judge whether the equation system has converged or not. The criterion for convergence is then

$$\frac{R}{F} \leq \varepsilon$$

where $0.0001 < \varepsilon < 0.01$, and F represents the total flux of T, i.e. total heat transfer rate.

Regardless if we solve the continuity equation, the Navier-Stokes equation or the energy equation, the procedure is the same: F should represent the total flux of the dependent variable.

- Continuity equation. F is here the total incoming mass flux $\dot{m}$.
- Navier-Stokes equation. The unit is that of a force, i.e. Newton. A suitable value of F is obtained from $F = \dot{m}U$ at the inlet.
- Energy equation. F should be the total incoming heat flux. In a convection-diffusion problem we can take the convective flux at the inlet i.e. $F = \dot{m}c_p T$. In a conduction problem we can integrate the boundary flux, taking the absolute value at each cell, since the sum will be zero in case of internal source. If there are large heat sources in the computational domain, F could be taken as the sum of all heat sources.

3.2 2D Diffusion

The two-dimensional heat conduction equation reads

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) + S = 0. \tag{3.5}$$

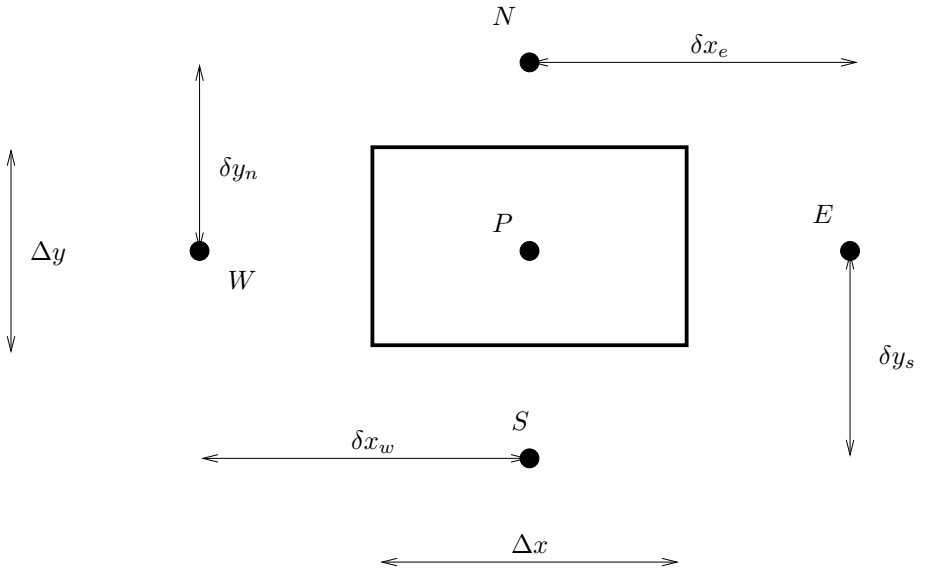

Figure 3.2: 2D control volume.

In the same way as we did for the 1D case, we integrate over our control volume, but now it's in 2D (see Fig. 3.2, i.e.

$$\int_w^e \int_s^n \left[\frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(k \frac{\partial T}{\partial y} \right) + S \right] dxdy = 0.$$

We start by the first term. The integration in x direction is carried out in exactly the same way as in 1D, i.e.

$$\begin{aligned} & \int_w^e \int_s^n \left[\frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) \right] dxdy = \int_s^n \left[\left(k \frac{\partial T}{\partial x} \right)_e - \left(k \frac{\partial T}{\partial x} \right)_w \right] dy \\ = & \int_s^n \left(k_e \frac{T_E - T_P}{\delta x_e} - k_w \frac{T_P - T_W}{\delta x_w} \right) dy \end{aligned}$$

Now integrate in the y direction. We do this by estimating the integral

$$\int_s^n f(y) dy = f_P \Delta y + \mathcal{O}\left((\Delta y)^2 \right)$$

(i.e. f is taken at the mid-point P) which is second order accurate, since it is exact if f is a linear function. For our equation we get

$$\begin{aligned} & \int_s^n \left(k_e \frac{T_E - T_P}{\delta x_e} - k_w \frac{T_P - T_W}{\delta x_w} \right) dy \\ & = \left(k_e \frac{T_E - T_P}{\delta x_e} - k_w \frac{T_P - T_W}{\delta x_w} \right) \Delta y \end{aligned}$$

Doing the same for the diffusion term in the y direction in Eq. 3.5 gives

$$\begin{aligned} & \left(k_e \frac{T_E - T_P}{\delta x_e} - k_w \frac{T_P - T_W}{\delta x_w} \right) \quad \Delta y \\ + & \left(k_n \frac{T_N - T_P}{\delta y_n} - k_s \frac{T_P - T_S}{\delta y_s} \right) \quad \Delta x + \bar{S} \Delta x \Delta y = 0 \end{aligned}$$

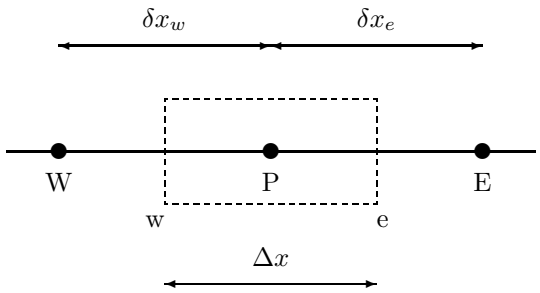

Figure 4.1: 1D control volume. Node P located in the middle of the control volume.

Rewriting it as an algebraic equation for T_P, we get

$$\begin{aligned} a_P T_P &= a_E T_E + a_W T_W + a_N T_N + a_S T_S + S_U \\ a_E &= \tfrac{k_e \Delta y}{\delta x_e},\ a_W = \tfrac{k_w \Delta y}{\delta x_w},\ a_N = \tfrac{k_n \Delta x}{\delta y_n},\ a_S = \tfrac{k_s \Delta x}{\delta y_s} \\ S_U &= \bar{S} \Delta x \Delta y,\ a_P = a_E + a_W + a_N + a_S - S_P. \end{aligned} \tag{3.6}$$

In this 2D equation we have introduced the general form of the source term; this could also be done in the 1D equation (Eq. 3.4).

For more detail on diffusion, see
http://www.tfd.chalmers.se/~lada/comp_fluid_dynamics/lecture_notes.html

4 Convection – Diffusion

The 1D convection-diffusion equation reads

$$\frac{d}{dx}(\rho U T) = \frac{d}{dx}\left(\Gamma \frac{dT}{dx}\right) + S,\ \Gamma = \frac{k}{c_p}$$

We discretize this equation in the same way as the diffusion equation. We start by integrating over the control volume (see Fig. 4.1).

$$\int_w^e \frac{d}{dx}(\rho U T)\, dx = \int_w^e \left[\frac{d}{dx}\left(\Gamma \frac{dT}{dx}\right) + S\right] dx. \tag{4.1}$$

We start by the convective term (the left-hand side)

$$\int_w^e \frac{d}{dx}(\rho U T)\, dx = (\rho U T)_e - (\rho U T)_w .$$

We assume the velocity U to be known, or, rather, obtained from the solution of the Navier-Stokes equation.

4.1 Central Differencing Scheme (CDS)

How to estimate T_e and T_w? The most natural way is to use linear interpolation (central differencing); for the east face this gives

$$(\rho U T)_e = (\rho U)_e T_e$$

where the convecting part, ρU, is taken by central differencing, and the convected part, T, is estimated with different differencing schemes. We start by using central differencing for T so that

$$(\rho UT)_e = (\rho U)_e T_e, \quad \text{where } T_e = f_x T_E + (1 - f_x) T_P$$

where f_x is the interpolation function (see Eq. 3.3, p. 10), and for constant mesh spacing $f_x = 0.5$. Assuming constant equidistant mesh (i.e. $\delta x_w = \delta x_e = \Delta x$) so that $f_x = 0.5$, inserting the discretized diffusion and the convection terms into Eq. 4.1 we obtain

$$\begin{aligned} (\rho U)_e \frac{T_E + T_P}{2} - (\rho U)_w \frac{T_P + T_W}{2} = \\ = \frac{\Gamma_e (T_E - T_P)}{\delta x_e} - \frac{\Gamma_w (T_P - T_W)}{\delta x_w} + \bar{S} \Delta x \end{aligned}$$

which can be rearranged as

$$\begin{aligned} a_P T_P &= a_E T_E + a_W T_W + S_U \\ a_E &= \tfrac{\Gamma_e}{\delta x_e} - \tfrac{1}{2}(\rho U)_e, \; a_W = \tfrac{\Gamma_w}{\delta x_w} + \tfrac{1}{2}(\rho U)_w \\ S_U &= \bar{S} \Delta x, \; a_P = \tfrac{\Gamma_e}{\delta x_e} + \tfrac{1}{2}(\rho U)_e + \tfrac{\Gamma_w}{\delta x_w} - \tfrac{1}{2}(\rho U)_w \end{aligned}$$

We want to compute a_P as the sum of its neighbor coefficients to ensure that $a_P \geq a_E + a_W$ which is the requirement to make sure that the iterative solver converges. We can add

$$(\rho U)_w - (\rho U)_e = 0$$

(the continuity equation) to a_P so that

$$a_P = a_E + a_W.$$

Central differencing is second-order accurate (easily verified by Taylor expansion), i.e. the error is proportional to $(\Delta x)^2$. This is very important. If the number of cells in one direction is doubled, the error is reduced by a factor of four. By doubling the number of cells, we can verify that the discretization error is small, i.e. the difference between our algebraic, numerical solution and the exact solution of the differential equation.

Central differencing gives negative coefficients when $|Pe| > 2$; this condition is unfortunately satisfied in most of the computational domain in practice. The result is that it is difficult to obtain a convergent solution in steady flow. However, in LES this does usually not pose any problems.

4.2 First-Order Upwind Scheme

For turbulent quantities upwind schemes must usually be used in order stabilize the numerical procedure. Furthermore, the source terms in these equations are usually very large which means that an accurate estimation of the convection term is less critical.

In this scheme the face value is estimated as

$$T_e = \begin{cases} T_P & \text{if } U_e \geq 0 \\ T_E & \text{otherwise} \end{cases}$$

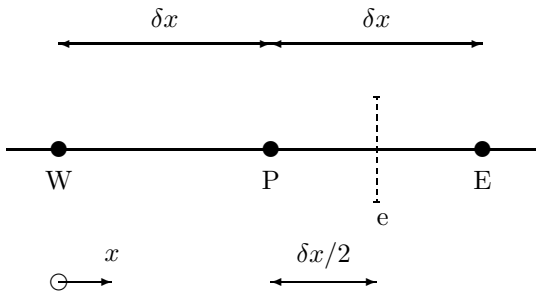

Figure 4.2: Constant mesh spacing. $U > 0$.

- first-order accurate
- bounded

The large drawback with this scheme is that it is inaccurate.

4.3 Hybrid Scheme

This scheme is a blend of the central differencing scheme and the first-order upwind scheme. We learned that the central scheme is accurate and stable for $|Pe| \leq 2$. In the Hybrid scheme, the central scheme is used for $|Pe| \leq 2$; otherwise the first-order upwind scheme is used. This scheme is only marginally better than the first-order upwind scheme, as normally $|Pe| > 2$. It should be considered as a first-order scheme.

4.4 Second-Order Upwind Scheme

We use two nodes upstream and assume that the derivative between W and P is equal to that between P and e, i.e. (see Fig. 4.2)

$$\frac{T_P - T_W}{\delta x} = \frac{T_e - T_P}{\frac{1}{2}\delta x} \Rightarrow T_e \simeq \frac{3}{2}T_P - \frac{1}{2}T_W \tag{4.2}$$

- second-order accurate
- unbounded (negative coefficients), i.e. $T_e < T_W$, $T_e < T_P$ or $T_e < T_E$ (see Fig. 4.3), or vice versa.

4.5 Bounded Second-Order Upwind Scheme

Often, bounded second-order upwind schemes are used. One example is the Van Leer scheme [34]. This scheme reads as follows ($U_e > 0$ assumed):

$$T_e = \begin{cases} T_P + \frac{T_E - T_P}{T_E - T_W}(T_P - T_W) & \text{if } |T_E - 2T_P + T_W| \leq |T_E - T_W| \\ T_P & \text{otherwise} \end{cases} \tag{4.3}$$

see Fig. 4.4, If the variation of T is smooth then

$$\frac{T_E - T_P}{T_E - T_W} \simeq \frac{1}{2},$$

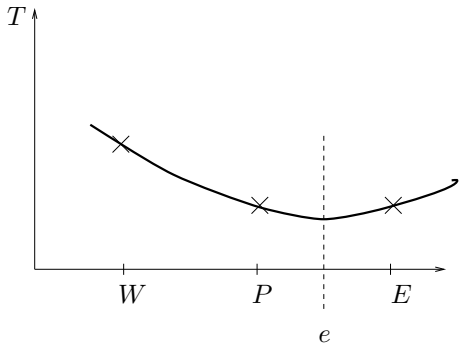

Figure 4.3: Constant mesh spacing. $U > 0$.

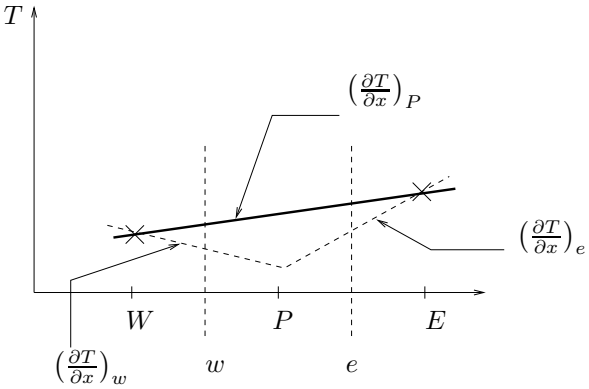

Figure 4.4: Van Leer scheme.

and we find that van Leer scheme gives $T_e = 1.5T_P - 0.5T_W$, i.e. it returns to the second-order upwind scheme (see p. 15).

Now let's illustrate what happens if T has a minimum at node P (dashed line in the Fig. 4.4). We want to show that in this case 1st order upwind is used in the van Leer scheme. When T has a minimum at node P the expression $T_E - 2T_P + T_W$ $[= (\Delta x)^2 (d^2T/dx^2)_P^{CD}]$ is larger than $T_E - T_W$ $(= \Delta x (dT/dx)_P^{CD}$. This is seen by rewriting the first expression as $T_E - 2T_P + T_W = T_E - T_W + 2(T_W - T_P)$ and noting that for the dashed line in Fig. 4.4 $T_W - T_P >$. When so, the condition on the first line of Eq. 4.3 is not satisfied, and thus the second line of of Eq. 4.3 is used, i.e. the first-order upwind scheme is used.

The van Leer scheme

- is second-order accurate, except at local minima and maxima where is only first-order accurate. It can be regarded as a second-order scheme.
- is bounded

MUSCL [35], which is an improved Van Leer scheme is also available in CALC-LES.

5 The Fractional-step method

A numerical method based on an implicit, finite volume method with collocated grid arrangement, central differencing in space, and Crank-Nicolson ($\alpha = 0.5$) in time is briefly described below. An implicit, two-step time-advancement methods is used [5]. The Navier-Stokes equation for the $\bar{u}_i$ velocity reads

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j}\left(\bar{u}_i \bar{u}_j\right) = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial^2 \bar{u}_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}}{\partial x_j} \tag{5.1}$$

The discretized momentum equations read

$$\begin{aligned} \bar{v}_i^{n+1/2} &= \bar{v}_i^n + \Delta t H\left(\bar{v}^n, \bar{v}_i^{n+1/2}\right) \\ &-\alpha \Delta t \frac{\partial \bar{p}^{n+1/2}}{\partial x_i} - (1-\alpha)\Delta t \frac{\partial \bar{p}^n}{\partial x_i} \end{aligned} \tag{5.2}$$

where H includes convective, viscous and SGS terms. In SIMPLE notation this equation reads

$$a_P \bar{v}_i^{n+1/2} = \sum_{nb} a_{nb} \bar{v}^{n+1/2} + S_U - \alpha \frac{\partial \bar{p}^{n+1/2}}{\partial x_i} \Delta V$$

where S_U includes the explicit pressure gradient. The face velocities $\bar{v}_{f,i}^{n+1/2} = 0.5(\bar{v}_{i,J}^{n+1/2} + \bar{v}_{i,J-1}^{n+1/2})$ (note that J denotes node number and i is a tensor index) do not satisfy continuity. Create an intermediate velocity field by subtracting the implicit pressure gradient from Eq. 5.2, i.e.

$$\begin{aligned} \bar{v}_i^* &= \bar{v}_i^n + \Delta t H\left(\bar{v}^n, \bar{v}_i^{n+1/2}\right) - (1-\alpha)\Delta t \frac{\partial \bar{p}^n}{\partial x_i} \\ \Rightarrow \bar{v}_i^* &= \bar{v}_i^{n+1/2} + \alpha \Delta t \frac{\partial \bar{p}^{n+1/2}}{\partial x_i} \end{aligned} \tag{5.3}$$

Take the divergence of Eq. 5.3b and require that $\partial \bar{v}_{f,i}^{n+1/2}/\partial x_i = 0$ so that

$$\frac{\partial^2 \bar{p}^{n+1}}{\partial x_i \partial x_i} = \frac{1}{\Delta t \alpha}\frac{\partial \bar{v}_{f,i}^*}{\partial x_i} \tag{5.4}$$

The Poisson equation for $\bar{p}^{n+1}$ is solved with an efficient multigrid method [21]. In the 3D MG we use a plane-by-plane 2D MG. The face velocities are corrected as

$$\bar{v}_{f,i}^{n+1} = \bar{v}_{f,i}^* - \alpha \Delta t \frac{\partial \bar{p}^{n+1}}{\partial x_i} \tag{5.5}$$

A few iterations (typically two) solving the momentum equations and the Poisson pressure equation are required each time step to obtain convergence. More details can be found [18].

1. Solve the discretized filtered Navier-Stokes equation, Eq. 5.3a, for $\bar{v}_1$, $\bar{v}_2$ and $\bar{v}_3$.
2. Create an intermediate velocity field $\bar{v}_i^*$ from Eq. 5.3b.

	RANS	LES
Domain	2D or 3D	always 3D
Time domain	steady or unsteady	always unsteady
Space discretization	2nd order upwind	central differencing
Time discretization	1st order	2nd order (e.g. C-N)
Turbulence model	more than two-equations	zero- or one-equation

Table 5.1: Differences between a finite volume RANS and LES code.

3. Use linear interpolation to obtain the intermediate velocity field, $\bar{v}_{f,i}$, at the face
4. The Poisson equation (Eq. 5.4) is solved with an efficient multigrid method [21].
5. Compute the face velocities (which satisfy continuity) from the pressure and the intermediate face velocity from Eq. 5.5
6. Step 1 to 4 is performed till convergence (normally two or three iterations) is reached.
7. The turbulent viscosity is computed.
8. Next time step.

Since the Poisson solver in [21] is a nested MG solver, it is difficult to parallelize with MPI (Message Passing Interface) on large Linux clusters.

The discretized equation for $\bar{p}$ is assembled in subroutine `calcpe`. It calls the MG solver `peter_multi`. The MG solver includes a number of subroutines:

- `key`
- `key2`
- `mg_2d`
- `peter_init.f` (called once from `main`)
- `peter_2d_cyclic`
- `peter_2d_relax` `peter_cyclic`
- `peter_relax`

It is a stand alone solver; it does not use the usual COMMON block but all information is passed from `main` and `calcpe` via arguments in the `call` statement. The MG subroutines use their own COMMON blocks in `PETER_COMMON`.

6 Boundary Conditions

6.1 Inlet

The velocity component normal to the inlet (e.g. U if inlet is a west boundary) is usually, as well other scalar variables such as temperature and concentration. Turbulent quantities, such as k and ε are normally not known, but they must

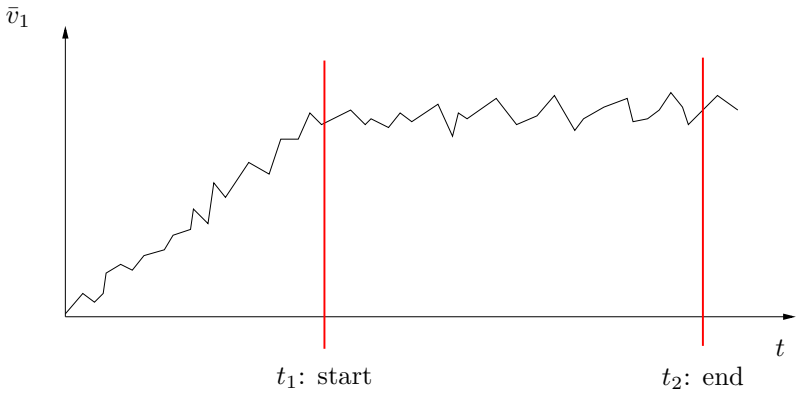

Figure 5.1: Time averaging in LES.

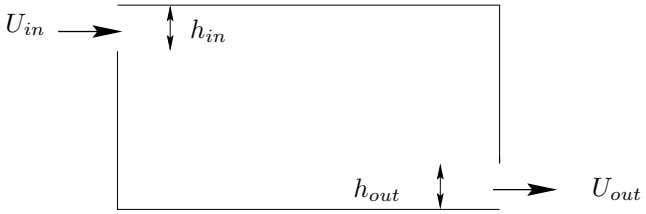

Figure 6.1: Outlet boundary condition. Small outlet

be estimated. Usually k is set to $(\gamma U)^2$, where $0.01 \lesssim \gamma \lesssim 0.1$. The dissipation is set from

$$\varepsilon_{in} = c\frac{k_{in}^{3/2}}{L},$$

where $c = 0.54$, and $L = 0.1h$, where h denotes height of inlet. Note that the expressions for k and ε only are guide lines.

6.2 Exit velocity

For *small* outlets the exit velocity can be determined from global continuity. As the inlet is small a constant velocity over the whole outlet can be used. The exit velocity is set as (see Fig. 6.1)

$$U_{in}h_{in} = U_{out}h_{out} \;\Rightarrow U_{out} = U_{in}h_{in}/h_{out}$$

For *large* outlets the exit velocity must be allowed to vary over the outlet. The proper boundary condition in this case is $\partial U/\partial x = 0$. Hence it is important that the flow near the exit is fully developed, so that this boundary condition corresponds to the flow conditions. The best way to ensure this is to locate the exit boundary sufficiently far downstream. If we have a recirculation region in the domain (see Fig. 6.2), the exit should be located sufficiently far downstream of this region so that $\partial U/\partial x \simeq 0$.

The exit boundary condition is implemented as follows (see Fig. 6.2)

1. Set $U_e = U_w$ for all nodes (i.e. for $j = 2$ to 6, see Fig. 6.2);

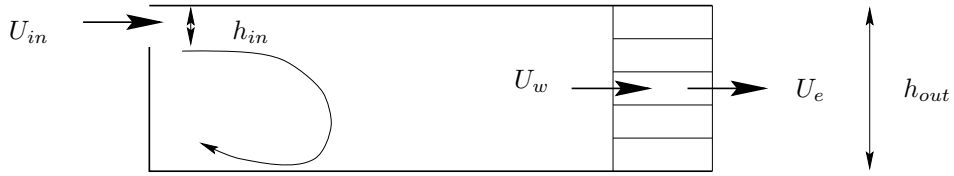

Figure 6.2: Outlet boundary condition. Large outlet.

2. In order to speed up convergence, enforce global continuity.
   - Inlet mass flow: $\dot{m}_{in} = \rho \sum_{inlet} U_{in} \Delta y$
   - Outlet mass flow: $\dot{m}_{out} = \rho \sum_{outlet} U_{out} \Delta y$
   - Compute correction velocity: $U_{corr} = (\dot{m}_{in} - \dot{m}_{out})/(\rho A_{out})$, where $A_{out} = \sum_{outlet} \Delta y$.
   - Correct U_e so that global continuity (i.e. $\dot{m}_{in} = \dot{m}_{out}$) is satisfied: $U_e^{new} = U_e + U_{corr}$

This boundary condition is implemented in subroutine `mod.f, entry modcon` for the hump flow.

6.3 Remaining variables

Set $\partial\Phi/\partial x = 0$, and implement it through $\Phi_{ni} = \Phi_{ni-1}$ each iteration.

6.4 Interior boundary conditions

Sometimes we want to prescribe a fixed value on a variable in an interior cell. A typical example is turbulent quantities. The wall boundary condition for dissipation, ε, and the specified dissipation, ω, are usually fixed at wall-adjacent nodes as

$$\begin{aligned} \varepsilon_w &= \frac{2\nu k}{y_w^2} \\ \omega_w &= \frac{6\nu}{\beta y_w^2} \end{aligned} \tag{6.1}$$

where y_w is the distance from the cell center to the wall and $\beta = 0.075$. These interior boundary conditions are conveniently prescribed using source terms. The 1D discretized equation with the general source term reads

$$a_P \Phi_P = a_E \Phi_E + a_W \Phi_W + S_U, \; a_P = a_E + a_W - S_P \tag{6.2}$$

with, for example, $\Phi = \varepsilon$. Now we prescribe ε at a wall-adjacent cell with source terms as

$$s_P = -10^{20}, \quad s_U = 10^{20} \varepsilon_w \tag{6.3}$$

Insert Eq. 6.3 into Eq. 6.2 gives

$$10^{20} \Phi_P \simeq 10^{20} \varepsilon_w \tag{6.4}$$

since $a_W \ll 10^{20}$ and $a_E \ll 10^{20}$ which gives $\Phi_P = \varepsilon_w$ as intended.

This boundary condition is implemented in subroutine `mod.f, entry moded`.

7 The Smagorinsky Model

subroutine: `vist_les`

The simplest model is the Smagorinsky model [33]:

$$\begin{aligned} \tau_{ij} - \frac{1}{3}\delta_{ij}\tau_{kk} &= -\nu_{sgs}\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right) = -2\nu_{sgs}\bar{s}_{ij} \\ \nu_{sgs} &= (C_S\Delta)^2\sqrt{2\bar{s}_{ij}\bar{s}_{ij}} \equiv (C_S\Delta)^2|\bar{s}| \end{aligned} \tag{7.1}$$

and the filter-width is taken as the local grid size

$$\Delta = (\Delta V_{IJK})^{1/3} \tag{7.2}$$

Near the wall, the SGS viscosity becomes quite large since the velocity gradient is very large at the wall. However, because the SGS turbulent fluctuations near a wall go to zero, so must the SGS viscosity. A damping function f_μ is added to ensure this

$$f_\mu = 1 - \exp(-x_2^+/26) \tag{7.3}$$

A more convenient way to dampen the SGS viscosity near the wall is simply to use the RANS length scale as an upper limit, i.e.

$$\Delta = \min\left\{(\Delta V_{IJK})^{1/3}, \kappa n\right\} \tag{7.4}$$

where n is the distance to the nearest wall. C_S is set to 0.1.

8 The WALE model

subroutine: `vist_wale`

The WALE model by [30] reads

$$\begin{aligned} g_{ij} &= \frac{\partial \bar{v}_i}{\partial x_j},\ g_{ij}^2 = g_{ik}g_{kj} \\ \bar{s}_{ij}^d &= \frac{1}{2}\left(g_{ij}^2 + g_{ji}^2\right) - \frac{1}{3}\delta_{ij}g_{kk}^2 \\ \nu_{sgs} &= (C_m\Delta)^2\frac{\left(\bar{s}_{ij}^d\bar{s}_{ij}^d\right)^{3/2}}{\left(\bar{s}_{ij}\bar{s}_{ij}\right)^{5/2} + \left(\bar{s}_{ij}^d\bar{s}_{ij}^d\right)^{5/4}} \end{aligned} \tag{8.1}$$

with $C_m = 0.325$ which corresponds to $C_s = 0.1$.

9 The PANS Model

subroutines: `calcte_pans, calced_pans, vist_pans`

The low-Reynolds number partially averaged Navier-Stokes (LRN PANS) turbulence model reads [26]

$$\begin{aligned} \frac{Dk}{Dt} &= \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_{ku}}\right)\frac{\partial k}{\partial x_j}\right] + P_k + P_{k_{tr}} - \varepsilon \\ \frac{D\varepsilon}{Dt} &= \frac{\partial}{\partial x_j}\left[\left(\nu + \frac{\nu_t}{\sigma_{\varepsilon u}}\right)\frac{\partial \varepsilon}{\partial x_j}\right] + C_{\varepsilon 1}P_k\frac{\varepsilon}{k} - C_{\varepsilon 2}^*\frac{\varepsilon^2}{k} \\ \nu_t &= C_\mu f_\mu\frac{k^2}{\varepsilon}, P_k = 2\nu_t\bar{s}_{ij}\bar{s}_{ij}, \bar{s}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i}\right) \\ C_{\varepsilon 2}^* &= C_{\varepsilon 1} + \frac{f_k}{f_\varepsilon}(C_{\varepsilon 2}f_2 - C_{\varepsilon 1}), \sigma_{ku} \equiv \sigma_k\frac{f_k^2}{f_\varepsilon}, \sigma_{\varepsilon u} \equiv \sigma_\varepsilon\frac{f_k^2}{f_\varepsilon} \\ \sigma_k &= 1.4, \sigma_\varepsilon = 1.4, C_{\varepsilon 1} = 1.5, C_{\varepsilon 2} = 1.9, C_\mu = 0.09, f_\varepsilon = 1 \end{aligned} \tag{9.1}$$

where $D/Dt = \partial/\partial t + \bar{v}_j \partial/\partial x_j$ denotes the material derivative. The damping functions are defined as

$$\begin{aligned} f_2 &= \left[1 - \exp\left(-\tfrac{y^*}{3.1}\right)\right]^2 \left\{1 - 0.3 \exp\left[-\left(\tfrac{R_t}{6.5}\right)^2\right]\right\} \\ f_\mu &= \left[1 - \exp\left(-\tfrac{y^*}{14}\right)\right]^2 \left\{1 + \tfrac{5}{R_t^{3/4}} \exp\left[-\left(\tfrac{R_t}{200}\right)^2\right]\right\} \\ R_t &= \tfrac{k^2}{\nu\varepsilon}, \quad y^* = \tfrac{U_\varepsilon y}{\nu}, \quad U_\varepsilon = (\varepsilon\nu)^{1/4} \end{aligned} \tag{9.2}$$

The term $P_{k_{tr}}$ in Eq. 9.1 is an additional term which is non-zero in the interface region because $Df_k/Dt \neq 0$. This is used at the inlet in the hump flow (see subroutine `mod.f, entry modte`).

The function f_ε, the ratio of the modeled to the total dissipation, is set to one since the turbulent Reynolds number is high. f_k is set to 1 in the RANS region and to 0.4 in the LES region.

It may also be computed and then the interface is chosen automatically [17], see Sections 19.

10 The $k-\omega$ Model

subroutines: `calcte_kom, calcom, vist_kom`

The Wilcox $k-\omega$ turbulence model reads [38]

$$\begin{aligned} \frac{\partial k}{\partial t} + \frac{\partial \bar{v}_i k}{\partial x_i} &= P^k - c_\mu k\omega + \tfrac{\partial}{\partial x_j}\left[\left(\nu + \tfrac{\nu_t}{\sigma_k}\right)\tfrac{\partial k}{\partial x_j}\right] \\ \frac{\partial \omega}{\partial t} + \frac{\partial \bar{v}_i \omega}{\partial x_i} &= C_{\omega_1}\tfrac{\omega}{k}P^k - C_{\omega 2}\omega^2 + \tfrac{\partial}{\partial x_j}\left[\left(\nu + \tfrac{\nu_t}{\sigma_\omega}\right)\tfrac{\partial \omega}{\partial x_j}\right] \\ \nu_t &= \tfrac{k}{\omega} \end{aligned} \tag{10.1}$$

where $c_\mu = 0.09$, $c_{\omega_1} = 5/9$, $c_{\omega_2} = 3/40$, $\sigma_k = 0.5 = \sigma_\omega = 2.0$.

11 The IDDES Model

This model will be implemented by the course participants during the workshop, see Section 25.6.

We start by defining – as in DES and DDES – a RANS and LES lengthscale as

$$\begin{aligned} L_{LES} &= C_{DES}\Delta_{dw} \\ L_{RANS} &= \tfrac{k^{3/2}}{\varepsilon} \\ \Delta_{dw} &= \min\left(\max\left[C_{dw}d_w, C_w\Delta_{max}, \Delta_{nstep}\right], \Delta_{max}\right) \end{aligned} \tag{11.1}$$

where d_w denotes the distance to the nearest wall and Δ_{step} is the grid step size in the wall-normal direction. The final lengthscale is blended

$$L_t = f_d\left(1 + f_e\right)L_{:ES} + \left(1 - f_d\right)L_{RANS}, \tag{11.2}$$

In DES, ψ is computed from Eq. 25.1. In IDDES, it is computed as

$$\psi = \max\left(1, \frac{L_{RANS}}{L_t}\right\} \tag{11.3}$$

The blending functions in Eq. 11.2 read

$$f_d = \max\left\{\left(1 - f_{dt}\right), f_B\right\} \tag{11.4}$$

$$f_e = \max\left\{\left(f_{e1} - 1\right), 0\right\} f_{e2}, \tag{11.5}$$

where f_{dt} and f_B in Eq. 11.4 read

$$f_{dt} = 1 - \tanh\left[\left(8 r_{dt}\right)^3\right] \tag{11.6}$$

$$f_B = \min\left\{2\exp\left(-9\alpha^2\right), 1\right\} \tag{11.7}$$

with

$$\alpha = 0.25 - d_w / h_{max} \tag{11.8}$$

The functions f_{e1} and f_{e2} in Eq. 11.5 read

$$\begin{aligned} f_{e1} &= 2\exp\left(-11.09\alpha^2\right) \quad \text{if } \alpha \geq 0 \\ f_{e1} &= 2\exp\left(-9\alpha^2\right) \quad \text{if } \alpha < 0 \end{aligned} \tag{11.9}$$

and

$$f_{e2} = 1 - \max\left\{f_t; f_l\right\} \ , \tag{11.10}$$

where the functions f_t and f_l are given by

$$\begin{aligned} f_t &= \tanh\left[\left(c_t^2 r_{dt}\right)^3\right] \\ f_l &= \tanh\left[\left(c_l^2 r_{dl}\right)^{10}\right] \end{aligned} \tag{11.11}$$

The constants c_t and c_l above, depend on the background RANS model. They were originally tuned in [32] for the SA model, and later in [22] for the $k-\omega$ SST model. The chosen values are $c_t = 1.87$ and $c_l = 5$. The quantities r_{dt} and r_{dl} in Eqs. 11.6 and 11.11 are defined as

$$\begin{aligned} r_{dt} &= \frac{\nu_t}{\kappa^2 d_w^2 \max\{\mathcal{S}, 10^{-10}\}} \\ r_{dl} &= \frac{\nu}{\kappa^2 d_w^2 \max\{\mathcal{S}, 10^{-10}\}} \end{aligned} \tag{11.12}$$

where

$$\mathcal{S} = \sqrt{\left(\frac{\partial u_i}{\partial x_j}\frac{\partial u_i}{\partial x_j}\right)^2} \tag{11.13}$$

12 Inlet boundary conditions

In RANS it is sufficient to supply profiles of the mean quantities such as velocity and temperature plus the turbulent quantities (e.g. k and ε). However, in unsteady simulations (LES, URANS, DES ...) the time history of the velocity and temperature need to be prescribed; the time history corresponds to turbulent, resolved fluctuations. In some flows it is critical to prescribe reasonable turbulent fluctuations, but in many flows it seems to be sufficient to prescribe constant (in time) profiles [9, 10].

There are different ways to create turbulent inlet boundary conditions. One way is to use a pre-cursor DNS or well resolved LES of channel flow. This method is limited to fairly low Reynolds numbers and it is difficult (or impossible) to re-scale the DNS fluctuations to higher Reynolds numbers.

Another method based partly on synthesized fluctuations is the vortex method [25]. It is based on a superposition of coherent eddies where each eddy is described by a shape function that is localized in space. The eddies are generated randomly in the inflow plane and then convected through it. The method is able to reproduce first and second-order statistics as well as two-point correlations.

A third method is to take resolved fluctuations at a plane downstream of the inlet plane, re-scale them and use them as inlet fluctuations.

Below we present a method of generating synthesized inlet fluctuations.

12.1 Synthesized turbulence

The method described below was developed in [2, 3] for creating turbulence for generating noise. It was later further developed for inlet boundary conditions [15, 6, 8, 7].

A turbulent fluctuating velocity fluctuating field (whose average is zero) can be expressed using a Fourier series, see [12]. Let us re-write this formula as

$$\begin{aligned} a_n \cos(nx) + b_n \sin(nx) &= \\ c_n \cos(\alpha_n)\cos(nx) + c_n \sin(\alpha_n)\sin(nx) &= c_n \cos(nx - \alpha_n) \end{aligned} \tag{12.1}$$

where $a_n = c_n \cos(\alpha)$, $b_n = c_n \sin(\alpha_n)$. The new coefficient, c_n, and the phase angle, α_n, are related to a_n and b_n as

$$c_n = \left(a_n^2 + b_n^2\right)^{1/2}, \quad \alpha_n = \arctan\left(\frac{b_n}{a_n}\right) \tag{12.2}$$

A general form for a turbulent velocity field can thus be written as

$$\mathbf{v}'(\mathbf{x}) = 2\sum_{n=1}^{N} \hat{u}^n \cos(\boldsymbol{\kappa}^n \cdot \mathbf{x} + \psi^n)\boldsymbol{\sigma}^n \tag{12.3}$$

where $\hat{u}^n$, ψ^n and σ_i^n are the amplitude, phase and direction of Fourier mode n. The synthesized turbulence at one time step is generated as follows.

12.2 Random angles

The angles φ^n and θ^n determine the direction of the wavenumber vector $\boldsymbol{\kappa}$, see Eq. 12.3 and Eq. 12.1; α^n denotes the direction of the velocity vector, $\mathbf{v}'$. For more details, see [12].

It is implemented in function `random` in subroutine `synt_generate`.

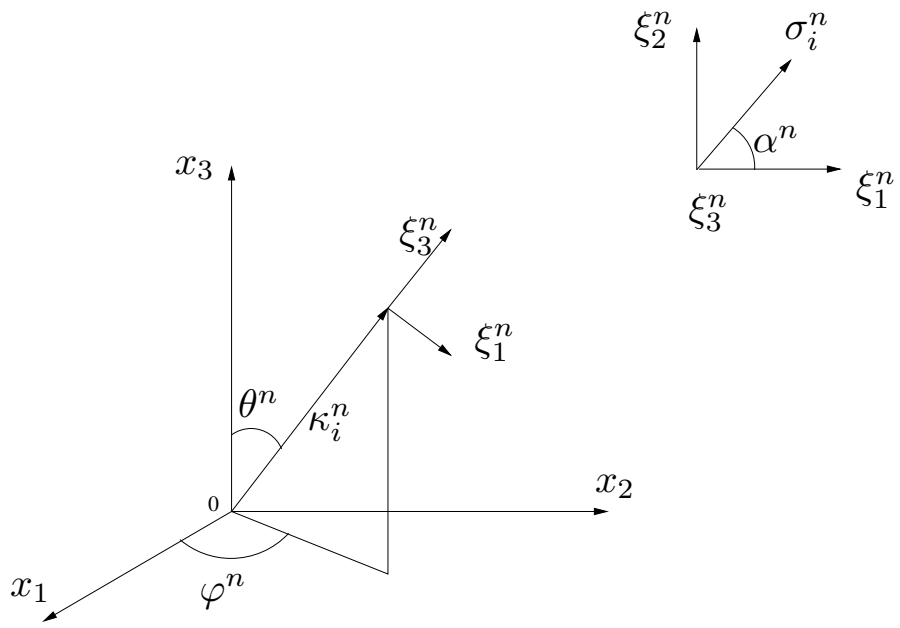

Figure 12.1: The wave-number vector, κ_i^n, and the velocity unit vector, σ_i^n, are orthogonal (in physical space) for each wave number n.

12.3 Highest wave number

Define the highest wave number based on mesh resolution $\kappa_{max} = 2\pi/(2\Delta)$ (see [12]), where Δ is the grid spacing. Often the smallest grid spacing near the wall is too small, and then a slightly larger values may be chosen. The fluctuations are generated on a grid with equidistant spacing (or on a weakly stretched mesh), $\Delta\eta = x_{2,max}/N_2$, $\Delta x_3 = x_{3,max}/N_3$, where η denotes the wall-normal direction and N_2 and N_3 denote the number of cells in the x_2 and x_3 direction, respectively. The fluctuations are set to zero at the wall and are then interpolated to the inlet plane of the CFD grid (the $x_2 - x_3$ plane).

This is implemented in subroutine `synt_init`.

12.4 Smallest wave number

Define the smallest wave number from $\kappa_1 = \kappa_e/p$, where $\kappa_e = \alpha 9\pi/(55L_t)$, $\alpha = 1.453$. The turbulent length scale, L_t, may be estimated in the same way as in RANS simulations, i.e. $L_t \propto \delta$ where δ denotes the inlet boundary layer thickness. In [6, 8, 7] it was found that $L_t \simeq 0.1\delta_{in}$ is suitable.

Factor p should be larger than one to make the largest scales larger than those corresponding to κ_e. A value $p = 2$ is suitable.

This is implemented in subroutine `synt_init`.

12.5 Divide the wave number range

Divide the wavenumber space, $\kappa_{max} - \kappa_1$, into N modes, equally large, of size $\Delta\kappa$.

This is implemented in subroutine `synt_init`.

12.6 von Kármán spectrum

A modified von Kármán spectrum is chosen, see Eq. 12.4 and Fig. 12.2. The amplitude $\hat{u}^n$ of each mode in Eq. 12.3 is then obtained from

$$\begin{aligned} \hat{u}^n &= (E(\kappa)\Delta\kappa)^{1/2} \\ E(\kappa) &= c_E \frac{u_{rms}^2}{\kappa_e} \frac{(\kappa/\kappa_e)^4}{[1+(\kappa/\kappa_e)^2]^{17/6}} e^{[-2(\kappa/\kappa_\eta)^2]} \\ \kappa &= (\kappa_i\kappa_i)^{1/2}, \quad \kappa_\eta = \varepsilon^{1/4}\nu^{-3/4} \end{aligned} \tag{12.4}$$

The coefficient c_E is obtained by integrating the energy spectrum over all wavenumbers to get the turbulent kinetic energy, i.e.

$$k = \int_0^\infty E(\kappa) d\kappa \tag{12.5}$$

which gives [23]

$$c_E = \frac{4}{\sqrt{\pi}} \frac{\Gamma(17/6)}{\Gamma(1/3)} \simeq 1.453 \tag{12.6}$$

where

$$\Gamma(z) = \int_0^\infty e^{-z'} x^{z-1} dz' \tag{12.7}$$

This is implemented in subroutine `synt_generate`.

12.7 Computing the fluctuations

Having $\hat{u}^n$, κ_j^n, σ_i^n and ψ^n, allows the expression in Eq. 12.3 to be computed, i.e.

$$\begin{aligned} v_1' &= 2\sum_{n=1}^N \hat{u}^n \cos(\beta^n)\sigma_1 \\ v_2' &= 2\sum_{n=1}^N \hat{u}^n \cos(\beta^n)\sigma_2 \\ v_3' &= 2\sum_{n=1}^N \hat{u}^n \cos(\beta^n)\sigma_3 \\ \beta^n &= k_1^n x_1 + k_2^n x_2 + k_3^n x_3 + \psi^n \end{aligned} \tag{12.8}$$

where $\hat{u}^n$ is computed from Eq. 12.4.

In this way inlet fluctuating velocity fields (v_1', v_2', v_3') are created at the inlet $x_2 - x_3$ plane.

This is implemented in subroutine `synt_generate`.

12.8 Introducing time correlation

A fluctuating velocity field is generated each time step as described above. They are independent of each other and their time correlation will thus be zero. This is nonphysical. To create correlation in time, new fluctuating velocity fields, $\mathcal{V}_1'$, $\mathcal{V}_2'$, $\mathcal{V}_3'$, are computed based on an asymmetric time filter

$$\begin{aligned} (\mathcal{V}_1')_m &= a(\mathcal{V}_1')_{m-1} + b(v_1')_m \\ (\mathcal{V}_2')_m &= a(\mathcal{V}_2')_{m-1} + b(v_2')_m \\ (\mathcal{V}_3')_m &= a(\mathcal{V}_3')_{m-1} + b(v_3')_m \end{aligned} \tag{12.9}$$

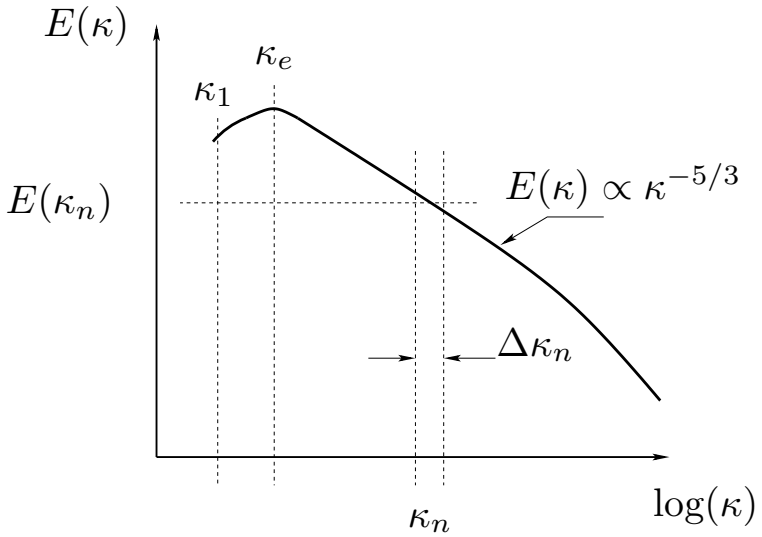

Figure 12.2: Modified von Kármán spectrum

where m denotes the time step number and

$$a = \exp(-\Delta t/T_{int}) \tag{12.10}$$

where Δt and T_{int} denote the computational time step and the integral time scale, respectively. The second coefficient is taken as

$$b = (1 - a^2)^{0.5} \tag{12.11}$$

which ensures that $\langle \mathcal{V}_1'^2 \rangle = \langle v_1'^2 \rangle$ ($\langle \cdot \rangle$ denotes averaging). The time correlation of will be equal to

$$\exp(-\hat{t}/T_{int}) \tag{12.12}$$

where $\hat{t}$ is the time separation and thus Eq. 12.9 is a convenient way to prescribe the turbulent time scale of the fluctuations. For more detail, see Section 12.8. The inlet boundary conditions are prescribed as (we assume that the inlet is located at $x_1 = 0$ and that the mean velocity is constant in the spanwise direction, x_3)

$$\begin{aligned} \bar{v}_1(0, x_2, x_3, t) &= V_{1,in}(x_2) + u'_{1,in}(x_2, x_3, t) \\ \bar{v}_2(0, x_2, x_3, t) &= V_{2,in}(x_2) + v'_{2,in}(x_2, x_3, t) \\ \bar{v}_3(0, x_2, x_3, t) &= V_{3,in}(x_2) + v'_{3,in}(x_2, x_3, t) \end{aligned} \tag{12.13}$$

where $v'_{1,in} = (\mathcal{V}'_1)_m$, $v'_{2,in} = (\mathcal{V}'_2)_m$ and $v'_{3,in} = (\mathcal{V}'_3)_m$ (see Eq. 12.9). The mean inlet profiles, $V_{1,in}$, $V_{2,in}$, $V_{3,in}$, are either taken from experimental data, a RANS solution or from the law of the wall; for example, if $V_{2,in} = V_{3,in} = 0$ we can estimate $V_{1,in}$ as [37]

$$V^+_{1,in} = \begin{cases} x_2^+ & x_2^+ \leq 5 \\ -3.05 + 5\ln(x_2^+) & 5 < x_2^+ < 30 \\ \frac{1}{\kappa}\ln(x_2^+) + B & x_2^+ \geq 30 \end{cases} \tag{12.14}$$

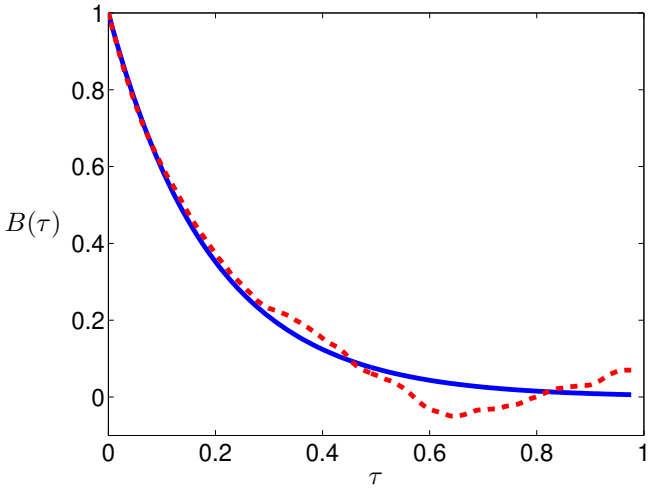

Figure 12.3: Auto correlation, $B(\tau) = \langle v_1'(t)v_1'(t-\tau)_t$ (averaged over time, t). —: Eq. 12.12; - - -: computed from synthetic data, $(\mathcal{V}_1')^m$, see Eq. 12.9.

where $\kappa = 0.4$ and $B = 5.2$.

The method to prescribed fluctuating inlet boundary conditions have been used for channel flow [7], for diffusor flow [10] as well as for the flow over a bump and an axisymmetric hill [11].

This is implemented in subroutine `synt_generate`.

13 Procedure to generate anisotropic synthetic fluctuations

The methodology is as follows:

1. A pre-cursor RANS simulation is made using a RANS model. For fully-developed channel flow, this may be done with the Python or Matlab script `rans` which can be found at [13].

2. After having carried out the pre-cursor RANS simulation, the Reynolds stress tensor is computed using the EARSM model [36]. This is done with the Python/Matlab script `synt_main_earsm` at [13].

3. The Reynolds stress tensor is used as input for generating the anisotropic synthetic fluctuations. The integral length scale, L_{int}, need to be prescribed; it can be set to $0.1\delta < L_{int} < 0.3\delta$, where δ denotes half-channel width.

4. Since the method of synthetic turbulence fluctuations assumes homogeneous turbulence, we can only use the Reynolds stress tensor in one point. We need to choose a relevant location for the Reynolds stress tensor. In a turbulent boundary layer, the Reynolds shear stress is by far the most important stress component. Hence, the Reynolds stress tensor is taken at the location where the magnitude of the turbulent shear stress is largest.

5. Finally, the synthetic fluctuations are scaled with $\left(|\overline{u'v'}|/|\overline{u'v'}|_{max}\right)^{1/2}_{RANS}$, which is taken from the 1D RANS simulation.
   This is done in subroutine `mod` entry `modu` in the hump flow.

The only constant used when generating these synthetic simulations is the prescribed integral length scale.

Eigenvalues and eigenvectors are computed in the Python/Matlab script `synt_main_earsm` which can be found in [13]. In that script, the eigenvalues are denoted by $a11$, $a22$, $a33$ and the eigenvectors by $r11$, $r12$, $\ldots r33$; they are stored in the files `a_synt_inlet.dat` and `R_synt_inlet.dat`, respectively, read by the subroutine `synt_init`.

14 Post-processing

The processing during the Workshop is made by Python and Matlab/Octave scripts. CALC-LES generates output files on TEcplot fprmat which can be read by – among others Tecplot (of course) and the open-source package Paraview Paraview. To generate a TEcplot file set `tecplot=.true.`.

15 Flow Chart

Go in to the directory `calc-les-flowchart/html` and open the file `index.html` with your Internet browser (in Firefox you should type `file://` in the address field). Or you can simply type `firefox index.html` at the prompter.

To see the flowchart, click on `main.f`.

16 Subroutines

calced, calced_pans, calcpe, calcph, calcte, calcte_pans, calcte_kom, calcu, calcv, calcw, calcom compute source terms etc for ε, ε (PANS), p, Φ, k, k (PANS),k $(k-\omega)$, $\bar{v}$, $\bar{v}$, $\bar{w}$ and ω)

calcte_keqabl one-equation turbulence model (only used for ABL (atmospheric boundary layer))

coeff: compute discretized coefficients for different discretization schemes

conv: compute mass flux through control volume face

ctdma: cyclic TDMA

dphidx, dphidxo, dphidy, dphidyo, dphidz, dphidzo: $\dfrac{\partial\Phi}{\partial x}$, $\dfrac{\partial\Phi^o}{\partial x}$ (old time step), $\dfrac{\partial\Phi}{\partial y}$, $\ldots$, $\dfrac{\partial\Phi^o}{\partial z}$ (old time step)

echo1: echoes input data

forest: computes the drag force in the forest (only used for ABL)

forest_init: calculates the LAD (Leaf-Area Density) of a given forest (only used for ABL)

forest_heat: calculates the forest heat flux as a distributed source term (only used for ABL)

init: computes geometrical quantities, initialization, ...

init_turb: sets an initial flow field by superimposing syntgetic flucutatins on a RANS flow field

inter_pol: interpolates the RANS inlet profiles to a new grid

key, key2: pointers used in MG solver for $\bar{p}$

main: main

mg_2d: 2D MG solver for $\bar{p}$

muscle: MUSCL discretization scheme

peter_2d_relax, peter_cyclic, peter_init, peter_multi, peter_relax: subroutines used in MG solver for $\bar{p}$

restr1: if `restrt.eq.true`, reads binary file `savres`

save1: `save.eq.true`, (over)writes binary file `savres`

solvt: TDMA solver

statisz: time averaging for post-processing

synt_generate, synt_init: files for generating synthetic inlet fluctuations

update: update `phi=phio`

van_leer: van Leer scheme

vist, vist_les, vist_pans, vist_wale, vis_kom: computes turbulent viscosity for the $k-\varepsilon$, Smagorinsky, PANS, WALE and $k-\omega$ model

vist_smagabl, vist_keqabl computes turbulent viscosity the Smagorinsky and the one-equation turbulence model (only used for ABL (atmospheric boundary layer))

17 Fully-developed channel flow

DNS of Fully-developed channel flow at $Re_\tau = 500$. The case is defined in subroutines `setup` and `mod`.

17.1 Setup

17.1.1 Section 1

`nphmax=6` Number of variables in `phi`. If you want to solve for more variables, increase this.

17.1.2 Section 2

Default values.

17.1.3 Section 3

`urfvis=0.5` Under-relaxation for turbulent viscosity (see `vist_wale`).

17.1.4 Section 4

`maxit=10` Maximum number of outer iterations.

17.1.5 Section 5

`sormax =1.e-3` Convergence limit at each time step.

17.1.6 Section 6

`densit =1.` density

`prt_lam(t)=0.72` viscous Prandtl number

`re =500.` Reynolds number

`viscos =1./re` viscosity

17.1.7 Section 7

`nsweep(u)=1` Number of sweeps in the TDMA solver for $\bar{v}$ (see `solvt`).

`nsweep(v)=1` Number of sweeps in the TDMA solver for $\bar{v}$

`nsweep(w)=1` Number of sweeps in the TDMA solver for $\bar{w}$

`nsweep(p)=5` Number of sweeps in the MB solver for $\bar{p}$

`isol(2)=1` This tells the multigrid solver (`peter_multi`) to solve the $\bar{p}$ equations at all MG levels by using a TDMA along x grid lines. If these variable are not set, `peter_init` tries to figure out the best option. If the grid is highly anisotropic in a plane, a 2D multigrid is used in this plane. The convergence history of the MG solver is monitored; an example of the print-out in the output file is

```
MG solver: iteration:    1                residual:  2.082E-01
MG solver: iteration:    2                residual:  3.590E-02
MG solver: iteration:    3                residual:  1.204E-02
MG solver: iteration:    4                residual:  5.089E-03
MG solver: iteration:    5                residual:  2.397E-03
```

To extract this convergence history, type `grep MG out`.

`isol(3)=1` Same as above, but TDMA along y grid lines

17.1.8 Section 8

`restrt=.true.` Read initial flow fields from binary file `savres`

`les=.false.` Not Smagorinsky turbulence model

`wale=.false.` Not WALE turbulence model

`pans=.false.` Not PANS turbulence model

`save=.true.` (over)writes flow fields in binary file `savres`

`cycli=.true.` Use cyclic boundary condition in x direction

`cyclk=.true.` Use cyclic boundary condition in z direction

`steady=.false.` Not steady

`echo=.true.` Echoes input data

17.1.9 Section 9

`scheme='c'` Use central scheme

`acrank_conv=0.51` Use Crank-Nicolson for convection-diffusion (see `solvt`)

`acrank=0.6` Use slightly more implicit than Crank-Nicolson for the pressure gradient (see, for instance, `calcu`)

`schtur='h'` Use hybrid central/upwinding for turbulent quantities (not relevant since we are doing DNS)

17.1.10 Section 10

`betap=1.` Driving pressure gradient (see `mod`)

17.1.11 Section 11

`solve(u)=.true.` Solve for $\bar{v}$

`solve(v)=.true.` Solve for $\bar{v}$

`solve(w)=.true.` Solve for $\bar{w}$

`solve(te)=.false.` Don't solve for k

`solve(ed)=.false.` Don't solve for ε

`solve(p)=.true.` Solve for $\bar{p}$

17.1.12 Section 12

`ni=98` Number of cells in x direction (including boundary nodes)

`nj=98` Number of cells in y direction (including boundary nodes)

`nk=98` Number of cells in z direction (including boundary nodes)

Note that the MG solver for pressure wants as many grid levels as possible. This means that we want to be able to divide the number of cells in each direction with 2^{m-1} where m is as large as possible. In this case $m = 6$ (i.e. 6 MG levels) so that the number of cells on the coarsest grid level is 3 ($96/2^5$). The number of MG levels are computed in `peter_init` and written to standard output

```
NUMBER OF LEVELS= 6
```

`xmax=2.*3.14` The domain in x direction is $2 \cdot 3.14$ (equidistant)

`zmax=1.*3.14` The domain in z direction is 3.14 (equidistant)

`yfac=1.065` Stretching away from the walls. Constant cells are used near walls and in the center

`uin=20` Approximate bulk velocity

`ntstep=20000` number of time steps

`iprint_start=10000` start time averaging from this time step

`dt(i)=0.5*xc(2,2,2)/uin` time step (should give a CFL number of approximately 0.4)

17.1.13 Section 13

`reref(p)=ymax*zmax*uin` Scale residuals of pressure (continuity equation)

`reref(u)=ymax*zmax*uin**2` Scale residuals of $\bar{v}$ (Navier-Stokes)

`reref(v)=ymax*zmax*uin**2` Scale residuals of $\bar{v}$ (Navier-Stokes)

`reref(w)=ymax*zmax*uin**2` Scale residuals of $\bar{w}$ (Navier-Stokes)

17.1.14 Section 14

`imon, jmon, kmon` Print time history of field variables for this cell

17.1.15 Section 15-17

Not relevant since we are doing DNS

17.2 mod

17.2.1 entry modini

It is called from `main`.

Compute `delta`.

17.2.2 entry modpro

It is called from `vist_les`, `vist_wale` and `vist_pans`.

Set cyclic boundary conditions on `vis`.

17.2.3 entry modcon

It is called from `conv`.

Set cyclic or zero boundary conditions on `conve`,`convn` and `convh`. Compute CFL.

17.2.4 entry modu

It is called from `calcu`.

Set the driving pressure gradient. It is either set to 1 which gives $\tau_w = 1$ (i.e. Re_τ is prescribed). Or it is set by setting the bulk velocity, U_b (i.e. Re_b is prescribed).

17.2.5 entry modv

It is called from `calcv`.

17.2.6 entry modw

It is called from `calcw`.

17.2.7 entry modpp

It is called from `calcpe`.

Set cyclic or zero boundary conditions on `p`.

17.2.8 entry modte

It is called from `calcte`.

17.2.9 entry moded

It is called from `calced`.

17.2.10 entry modphi

It is called from `calcph`.

17.3 Run the code

Go into the main directory `calc-les-202?` and continue down into the directory `channel-500-DNS`. The first time, remove all object files because the code has been compiled on my Linux machine with a different compiler. Do this by

```
rm -f *.o
rm -f ../*.o
rm -f ../*/*.o
```

When you run the code, the post-processing file, `vectz.dat`, will be over-written. Also the re-start file `savres`, will be over-written. Hence copy these two files to other names, e.g.

```
cp vectz.dat vectz_org.dat
cp savres savres_org.dat
```

Compile the code by typing

```
make
```

The default compiler is `gfortran`. The gfortan compiler is used In the file `makefile`. If you want to change compiler, remember to first remove all object files (*.o).

Let's run 100 time steps. Change `ntstep` and `iprint_start` in `setup` to

```
ntstep=100
iprint_start=1
```

The code will run 100 time steps and start time-averaging from the first time step. Run the code by typing

```
nice -n 19 ./calc-les > out&
```

The `nice -19` sets the priority of the execution as low as possible. That means that your interactive use of the lap-top is not affected by the fact that CALC-LES is running (usually you can run two runs at the same time; if you run more than two jobs they usually slow down each other).

Look at the file `out`. For example, you will find the line

```
NUMBER OF LEVELS= 6.
```

This show that the MG solver in `peter_init` has created six MG levels.

18 Fully-developed channel flow without re-start

In Section 17 we re-start the simulations from a previous simulation (i.e. `restrt=.true.`). Here we will start from scratch, i.e. we create a realistic initial flow field.

18.1 Setup

18.1.1 Section 8

`restrt=.false.` No re-start

`nfiles_restart=4`: save four fields $(\bar{v}, \bar{v}, \bar{w}, \bar{p})$ in `save1`

18.2 mod

18.2.1 entry modini

A RANS velocity and shear stress are read. These can be created with the RANS solver `rans.py` (Python) or `rans.m` (Matlab) [13]. The RANS field is interpolated to a new grid using the subroutine `inter_pol` from e.g. `uinit` to `uinit_)new`. In this case the grid is not modified, and hence the interpolation does not modify the initial profiles. Synthetic turbulence is superimposed to the RANS field. This is done plane-by-plane ($y-z$ planes) in exactly the same way as inlet synthetic fluctuations (see Section 12.1). In Section 12.1 many inlet planes are created and correlation in time is achieved by Eq. 12.9. Here we create many $y-z$ planes and correlation is introduced using the same formula

but the timestep is taken as $\Delta t = \Delta x/U_{bulk}$; Δt is the time it takes to convect the turbulence from x to $x + \Delta x$.

19 Fully-developed channel flow at $Re_\tau = 5\,200$ using PANS

Here we repeat the simulations carried out in [17]. The PANS model is used. The mesh is $34\times98\times34$ nodes and the Reynolds number is $Re_\tau = u_\tau h/\nu = 5\,200$. `setup` and `mod` are modified using the corresponding files in Section 18. Go to the directory `channel-5200-pans-32-96-20000-40000/`.

19.1 setup.f

19.1.1 Section 8

`pans=.true.` use the PANS model

19.2 mod.f

19.2.1 entry modini

Fields from a 1D RANS solution is read from file `y_u_uv_k_eps_pans-akn-re-5200.dat`. Synthetic fluctuations are superinmposed as in Section 18. Initial LES values are also given to k and ε as

```
phi(i,j,k,te)=rkinit(j)*0.4 !  PANS with fk=0.4
```

```
phi(i,j,k,ed)=epsinit(j)
```

The value 0.4 is chosen since this is good value when PANS is used in LES mode [14]. The LES dissipation is the same as RANS dissipation.

19.2.2 entry modu

At the end of this entry, the f_k coefficient is computed using Eq. 16 in [17], i.e.

```
term1=1.-(psi-1.)/(c2-c1)
```

Below that code, f_k is set to 0.4 and 1.0 in the LES and RANS region, respectively, during the 100 first timesteps in order to increase numerical stability. This is actually not needed in this case.

19.2.3 entry moded

The wall boundary conditions for ε is set by prescribing it as

$$\varepsilon_w = 2\nu k/d^2 \tag{19.1}$$

in the wall-adjacent cells. d is the distance from the cell center to the wall. The code for the south wall reads

`phi(i,2,k,ed)=2.*viscos*phi(i,2,k,te)/dist` Set the value

`j=2` south wall

`su(i,j,k)=great*phi(i,j,k,ed)` Use large su

`sp(i,j,k)=-great` and sp to force ε according to Eq. 19.1

20 Hill flow

LES of periodic hill flow using the PANS model. at $Re_b = 10\,595$. The case is defined in subroutines `setup` and `mod`. Here we will comment the sections that differ from those for the channel flow (see Section 17). We re-start the simulations from a previous simulations (i.e. `restrt=.true.`).

Go to the directory `hill-pans`.

20.1 Setup

20.1.1 Section 8

`pans=.true.` Use PANS turbulence model

20.1.2 Section 8

`pans=.true.` Use PANS turbulence model

20.1.3 Section 9

`acrank=0.8` Use more implicit than Crank-Nicolson for the pressure gradient (see, for instance, `calcu`)

20.1.4 Section 10

`betap=1.` Not used. It is recomputed in `mod`, `entry modu`

20.1.5 Section 11

`solve(te)=.true.` Solve for k

`solve(ed)=.true.` Solve for ε

20.1.6 Section 12

`open(unit=11,file='hill_161_81.dat',status='unknown')` Read grid

20.1.7 Section 16

`c1=1.5` $C_{\varepsilon 1}$ in ε equation

`c2=1.9` $C_{\varepsilon 2}$ in ε equation

20.1.8 Section 16

`prt(te)=-1.4` in k equation; it is re-computed in `coeff`

`prt(ed)=-1.4` in ε equation; it is re-computed in `coeff`

20.2 mod

20.2.1 entry modini

Compute `dist2d` which is the distance to the nearest wall. It is used in `vist_pans` and `calced_pans`.

20.2.2 entry modu

Compute the driving pressure gradient from a balance of all forces on the surfaces, i.e. wall shear stresses and pressure force. For more details, see Section 3.5 in Irannezhad [24].

20.2.3 entry moded

Sets wall boundary conditions

$$\varepsilon_w = 2\nu k/d^2$$

where d is the distance from the cell center to the wall.

21 Hump flow with re-start

LES of hump flow using the PANS model. at $Re_b = 9.36 \cdot 10^5$. This work is presented in [17].

The case is defined in subroutines `setup` and `mod`. Here we will comment the sections that differ from those for the hill flow. We re-start the simulations from a previous simulations (i.e. `restrt=.true.`).

Go to the directory `hump-computed-fk`.

21.1 Setup

21.1.1 Section 8

`cycli=.false.` No cyclic boundary condition in x direction

21.1.2 Section 16

`jl0=21` Defines the RAMS region at the inlet for the commutation term at the inlet (see `entry modte` below)

21.2 mod

21.2.1 entry modini

`fk(i,j,k)=0.4` Sets, initially, $f_k = 1$ near the wall and to 0.4 away from the wall.

`read(84,*)y2,umean_in(j),vmean_in(j),rkmean_in(j),epsmean_in(j)`

`dummy_uv,dummy_p,uv_rans(j)` Reads inlet b.c.

`phi(1,j,k,u)=umean_in(j)` Sets initial conditions from inlet b.c.

`nstep=500`

`do n=1,nstep` create 500 planes of synthetic fluctuations in order to compute the synthetic maximum shear stress `uv_synt` (it should be homogeneous)

`ss_u=(tauw/uv_synt)**0.5` scaling factor to get correct (i.e. same as RANS) peak of shear stress for the synthetic inlet fluctuations

21.2.2 entry modcon

Sets outlet b.c. according to Fig. 6.2.

21.2.3 entry modu

Below, we compute time-averaged normal Reynolds stresses near the inlet (to be used when prescribing additional commutation source term in k equation for inlet boundary conditions, see `modte`)

```
fnk=float(nk-2)
do k=2,nkm1
do j=1,njm1
do i=1,5
   umeanv(i,j)=umeanv(i,j)+phi(i,j,k,u)/fnk
   vmeanv(i,j)=vmeanv(i,j)+phi(i,j,k,v)/fnk
   umean2v(i,j)=umean2v(i,j)+phi(i,j,k,u)**2/fnk
   vmean2v(i,j)=vmean2v(i,j)+phi(i,j,k,v)**2/fnk
   wmean2v(i,j)=wmean2v(i,j)+phi(i,j,k,w)**2/fnk
end do
end do
end do
```

Recall to initialize these variables to zero in `entry modini`,

`an(i,njm1,k)=0.` Neumann b.c. on north boundary.

`term1=1.-(psi-1.)/(c2-c1)` Compute f_k using Eq. 16 in [17]

`rl_in=0.2*0.015` Prescribe length scale of synthetic inlet fluctuations 0.015 is the boundary layer width

`call synt_init` Compute initial arrays etc for the synthetic fluctuations

`a=exp(-dt(itstep)/tturb)` a and b from Eqs. 12.10 and 12.11.

`call synt_generate` Create synthetic fluctuations

`ss1=(abs(uv_rans(j))/uvmax)**0.5` make a non-dimensional $\overline{v_1' v_2'}$ profile

`ss2=(uvmax/uvmax_synt)` correct the amplitude

`ss=ss1*ss2` scaling factor

`uprim=ss*utemp(j,k)` scale the inlet fluctuations

`ufluct_inlet(j,k)=a*ufluct_inlet(j,k)+b*uprim` see Eq. 12.9.

`phi(1,j,k,u)=phi(1,j,k,u)-uinc` This make sure that the integral of `ufluct_inlet(j,k)` over the inlet is zero. This may help convergence.

21.2.4 entry modte

`rk_source=(0.5*(vprim2+uprim2+wprim2)+phi(i,j,k,te))*dfk_dt` This is the source term due to the commutation error, see *2.2.1. Interface Model 1* and Eq. 19 in [14]. The inlet boundary condition is set to RANS values, i.e. `rkmean_in]` see `modini`.

`gente(i,njm1,k)=0.` This is done to fix convergence problems which sometime appear due to the irregular grid near the upper boundary (recall that it is a symmetry boundary and no turbulence production should occur there)

21.2.5 entry moded

`epsmean_in` The inlet boundary condition is set to RANS values, i.e. `epsmean_in`, see `modini`.

22 Hump flow without re-start

When the simulations are not re-started from an earlier simulations, different tricks must often be used to make the simulations stable. There are two standard options during the first couple of 100 timesteps:

- use a smaller timestep
- use a dissipative first-order upwind scheme (the hybrid scheme).

Here, it is sufficient to use the first option to make the simulations stable.

22.1 setup

22.1.1 Section 8

`restrt=.false.` No re-start

`nfiles_restart=4:` save four fields $(\bar{v}, \bar{v}, \bar{w}, \bar{p})$ in `save1`

22.1.2 Section 12

`if (.not.restrt.and.i.le.200) dt(i)=0.0002.` To increase numerical stability, use smaller timestep the first 200 timesteps

22.2 mod

22.2.1 entry modini

Synthetic initial fluctuations are created in the same way as in Section 18.2.1. The difference is here that the flow is not homogeneous in x direction. Furthermore, a 2D RANS field must be created, see Section 23.

`open(unit=17,file='savres_rans',form='unformatted',status='old')` The 2D RANS field is read and used as initial mean field.

`call init_turb` create synthetic initial fluctuations

22.2.2 entry modu

`scheme=scheme_org` To increase numerical stability, use the first-order hybrid central/upwind discretization scheme the first `iadd_source` timesteps.

23 Hump flow, 2D RANS

CALC-LES cannot run 2D flow because the multi-grid solver for pressure is a 3D solver.

23.1 setup

23.1.1 Section 9

`scheme='h'` Choose hybrid central/upwind scheme to make the simulations more stable.

23.1.2 Section 12

`nkm1=9` We get eight cells in z direction. That should give three MG-levels

When you look in `out` you will find the line
`NUMBER OF LEVELS= 3`.
This shows that the MG solver in `peter_init` has created three MG levels.

23.1.3 Section 14

`imon =ni-5`

`jmon =3` It is difficult to know when the 2D RANS flow field has reached a steady state. Here we look at the monitor point at the far end of the domain close to the wall. Note that it is not critical that the flow reaches a fully steady state; the object is only to use these results as initial conditions for the PANS simulations later on.

23.1.4 Section 18

`fk(i,j,k)=1.0` This turns the PANS model into the RANS AKN model [1].

When the simulations have been finished, copy the `savres` file to the PANS simulations, i.e.

```
cp savres ../hump-computed-fk-no-restart/savres_rans
```

24 Atmospheric boundary layer in a forest

The application of this case is windpower in forests. The work is presented in [27, 28, 29].

24.1 setup

24.1.1 Section 2a

Details on the forest and the ABL (Atmospheric Boundary Layer) are set here.

24.1.2 Section 8

`keq_abl=.true.` The one-equation turbulence model is used

24.1.3 Section 11

`if (stability) solve(t)=.true.` The temperature equation is solved

24.1.4 Section 12

`xmax=1000.` x domain extent is $1000m$ (streamwise)

`ymax=1000.` y domain extent is $1000m$ (vertical)

`zmax=1000.` z domain extent is $1000m$ (lateral)

`dly=2.0` the cells in the forest is set to $2m$

`if (y(j).gt.210.)` `yfac=1.105` stretch the cells by 1.105% above $210m$

`dly=min(dly,10.)` don't let the cells be larger than $10m$

24.1.5 Section 13

`dt(i)=0.2` Timestep is $0.2s$

24.2 mod

24.2.1 entry modini

`call forest_init` compute LAD

`deltaIDDES=min(max(h1,h2,h3),del_max)` compute Δ

`if (.not.restrt) then` if no re-start, initialize the flow

24.2.2 entry modpro

`vist=(0.41/term1)**2*yp1*upar` the wall-function boundary condition is implemented by setting the wall turbulent viscosity, see Section 3.4.1 in [16].

24.2.3 entry modu

`deltapx = alpha_geo*(udes-uhub)*apmean/volmean` the pressure gradient is adjusted so that the $\bar{v}$ should be `udes`.

`su(i,j,k)=su(i,j,k)-fcori*phi(i,j,k,w)*vol(i,j,k)` add the Coriolis force

`sp(i,j,k)=sp(i,j,k)+forceu(i,j,k)*vol(i,j,k)` add the drag force due to the forest

24.2.4 entry modv

`sp(i,j,k)=sp(i,j,k)+forcev(i,j,k)*vol(i,j,k)` add the drag force due to the forest

`su(i,j,k)=su(i,j,k)+grav*volextcoef*(phi(i,j,k,t)-tplane)*vol(i,j,k)` add the buoyancy force

24.2.5 entry modw

`deltapz = alpha_geo*(wdes-whub)*apmean/volmean` the pressure gradient is adjusted so that the $\bar{w}$ should be `wdes`.

`sp(i,j,k)=sp(i,j,k)+forcew(i,j,k)*vol(i,j,k)` add the drag force due to the forest

`su(i,j,k)=su(i,j,k)+fcori*phi(i,j,k,u)*vol(i,j,k)` add the Coriolis force

24.2.6 entry modphi(nphi)

`su(i,j,k) = su(i,j,k) + dqdy(j)*vol(i,j,k)` add the heat sink due to the forest

25 Workshop

25.1 Fully-developed channel flow using PANS

In the workshop we will try to reduce the CPU time as much as we can. We start from case in Section 19. Hence, we make some simplifications.

- Reduce the number of nodes in the y direction from 98 to 60.
- Reduce the minimum iterations at each timestep from two to one.
- Reduce the number of timesteps

Go to the main directory `calc-les-202?` and create a new directory for this case, enter the directory and copy files, i.e.

```
mkdir channel-pans-workshop
cd channel-pans-workshop
cp ../channel-5200-pans-32-96-10000-20000/* .
```

25.1.1 setup

Section 4

`minit=1` In main, the minimum number of iterations is by default set to two. In many applications, however, it works fine to set it to one. Note that the flow equations still much satisfy the convergence criteria (defined by `sormax`).

Section 12 Use only 60 grid nodes in the y direction and increase the stretching to 1.24. The grid is created as

```
nj=60
yfac=1.24
dy=0.1
y(1)=0.
do j=2,nj/2
   y(j)=y(j-1)+dy
   if (j.ge.26) yfac=1.
   dy=yfac*dy
end do
ymax=y(nj/2)
do j=2,nj/2
   y(j)=y(j)/ymax
   y(nj+1-j)=2.-y(j-1)
end do
```

Note that the stretching is omitted after grid node 26 to reduce Δy near the center.

Figure 5.1 shows where the time-averaging begins (t_1) and ends (t_2). In order to reduce the CPU time, we reduce these numbers as

`iprint_start=3000` set t_1

`ntstep=6000` set t_2

Now you will compile and run the code.

`make` Compile the code

`nice -n 19 ./calc-les > out&` If no error messages appear, run the code

`tail -1000 out` view the last, say, 1 000 lines of the output

`grep 'max res' out` monitor the convergence

`grep MG out` monitor the convergence of the multigrid solver (i.e. the pressure solver)

`grep CFL out` monitor the maximum CFD number in the x, y, z directions

`grep CPU out` monitor the CPU time time per iteration

25.2 Fully-developed half-width channel flow using PANS

Here we will make simulations in only the lower half of the channel. At he north boundary, we change the boundary condition from wall to symmetry. We start from case in Section 25.1. We will compute the flow only in the lower half of the channel. This reduces the number of nodes from 60 to 34. Note that if we use 30 nodes, we get very poor results, presumably because of the symmetry boundary condition at the north boundary reduces the resolved the turbulence.

Go to the main directory `calc-les-202?` and create a new directory for this case, enter the directory and copy files, i.e.

```
mkdir half-channel-pans-workshop
cd half-channel-pans-workshop
cp ../channel-pans-workshop/* .
```

25.2.1 setup.f

Section 12 Use only 34 grid nodes in the y direction in lower half of the channel. The grid is created as

```
nj=34
yfac=1.24
dy=0.1
y(1)=0.
do j=2,njm1
   y(j)=y(j-1)+dy
   if (j.ge.26) yfac=1.
   dy=yfac*dy
end do
ymax=y(njm1)
do j=2,njm1
   y(j)=y(j)/ymax
end do
```

25.2.2 mod.f

entry modini Note that, although the domain now consists only of the lower part of the channel, the interpolation to a new grid (using the subroutine `inter_pol`) is still valid.

In this Section the initial condition for $\bar{v}_2$ is set so as to make is anti-symmetric (i.e. $\bar{v}_2 = -\bar{v}_2$ in the upper half of the channel.

`if (j.ge.nj/2) vprim=-vprim` switch sign in the middle of the channel to make the shear stress symmetric

Since we here compute the flow only in the lower half of the channel, we delete this line. N.B.: this line appears twice, for $i = 2$ and for all i (delete both).

entry modpro Set symmetry boundary condition of for `vis` at the north boundary

```
c symmetry
      do k=1,nk
      do i=1,ni
         vis(i,nj,k)=vis(i,njm1,k)
      end do
      end do
```

entry modu Set symmetry boundary condition of for $\bar{v}$ at the north boundary

```
c symmetry
      do k=1,nk
      do i=1,ni
         phi(i,nj,k,u)=phi(i,njm1,k,u)
         an(i,njm1,k)=0.
      end do
      end do
```

entry modv No change is made here because for $\bar{v}$ the boundary condition is $\bar{v} = 0$ (both for a wall and for a symmetry boundary).

entry modw Set symmetry boundary condition of for $\bar{w}$ at the north boundary

```
c symmetry
      do k=1,nk
      do i=1,ni
         phi(i,nj,k,w)=phi(i,njm1,k,w)
         an(i,njm1,k)=0.
      end do
      end do
```

entry modte Set symmetry boundary condition of for k at the north boundary

```
c symmetry
      do k=1,nk
      do i=1,ni
         phi(i,nj,k,te)=phi(i,njm1,k,te)
         an(i,njm1,k)=0.
      end do
      end do
```

entry moded Delete the wall-boundary condition and set symmetry boundary condition of for ε at the north boundary

```
c symmetry
      do k=1,nk
      do i=1,ni
         phi(i,nj,k,ed)=phi(i,njm1,k,ed)
         an(i,njm1,k)=0.
      end do
      end do
```

Compile and run the code and plot the results.

25.3 Half-width channel flow, with inlet-outlet, using PANS

Here we will compute the flow in a half-width channel with inlet and outlet. We start by copying `setup.f` and `mod.f` (and all other files) from Section 25.2. Go to the main directory `calc-les-202?` and create a new directory for this case, enter the directory and copy files, i.e.

```
mkdir half-channel-pans-inlet-outlet-workshop
cd half-channel-pans-inlet-outlet-workshop
cp ../half-channel-pans-workshop/* .
```

25.3.1 setup.f

Section 8

`cycli=.false.` The flow is not periodic in the streamwise direction

25.3.2 mod.f

In Section 21 the hump flow with inlet and outlet is setup. We can copy much of the coding in `mod.f` from that case.

entry modini The 1D variables below will be used for inlet boundary conditions.

```
uinit(j)=uinit_new(j)
uvinit(j)=uvinit_new(j)
rkinit(j)=rkinit_new(j)
epsinit(j)=epsinit_new(j)
```

These variables are defined as local variables. This means that when the execution enters `mod.f` next time (e.g into `entry modu`), these arrays will reset to zero (or some arbitrary value). We fix this by defining the arrays in a COMMON block at the top of `mod.f` , e.g.

```
common/init_com/uinit(jt)
```

(the name `init_com` can be any name, provided it is not use elsewhere). Now `uinit` can be used anywhere in `mod.f`, but not in any other subroutine (if you would like to use `uinit` in other subroutines, you should add it in the file `COMMON`).

Add also the variable `rl_in` (the prescribed turbulent lengthscale for initial and inlet synthetic fluctuations) and `uv_synt_1d` to the COMMON block.

A commutation term is added in `entry modte` in order to reduce the RANS k values prescribed at the inlet. The commutation term is based on $\partial f_x/\partial x$ at the inlet. Hence, f_k must be set at the inlet.

```
c set fk at inlet; this creates dfk/dx at inlet
      do k=2,nkm1
      do j=2,njm1
         fk(1,j,k)=1.0
      end do
```

```
end do

jl1=nj-jl0+1
do k=2,nkm1
do j=2,jl0
do i=2,nim1
   fk(i,j,k)=1.0
end do
end do
end do

do k=2,nkm1
do j=jl1,njm1
do i=2,nim1
   fk(i,j,k)=1.0
end do
end do
end do
```

entry modu Copy the code from the hump flow between

```
if (.not.cycli.and.iter.eq.1) then

            ⋮

end if
```

This coding adds inlet synthetic fluctuations superimposed on the inlet 1D RANS field. Note that the RANS inlet profiles in the hump flow (`umean_in`, ...) are different from those used in this case (`uinit`, ...).

Try to compile the code by typing `make`. We get a number of compilation errors because we forgot to dimension new arrays. Dimension, for example, `ufluct_inlet(j,k)` by adding the line at the top of `mod.f`

```
common/init_com_real/uinit(jt),ufluct_inlet(jt,kt)
```

When we try to compile again, we find that the arrays
`umeanv, umean2v,vmean2v,wmean2v`
are un-defined. This is because we need to add the commutation term near the inlet, see *2.2.1. Interface Model 1* and Eq. 19 in [14]. Copy relevant code from `mod.f` of the hump flow, i.e.

1. additional source term at $i = 2$ in `entry modte`
2. compute `umeanv, umean2v` ... in `entry modu`
3. define `umeanv, umean2v` ... in COMMON blocks at the top of `mod.f`
4. initialize `umeanv, umean2v` ... (i.e. set to zero) in `entry modu`

Now we have added rather much new code. It is easy to make mistakes and it is often a good idea yo re-compile the entire code letting the compiler do some additional checks. Add the compilation option `-fbounds-check` in the `makefile`, i.e.

```
gfortran -O   -fbounds-check -Wuninitialized  -fdefault-real-8
-Ofast  -mcmodel=medium  -o $*.o -c $*.f
```

Furthermore, look carefully for warnings. The option `-Wuninitialized` is particularly useful since it warns for uninitialized variables.

When all compilations are fixed, run the code
`nice -n 19 ./calc-les > out&`

Look at the CPU time by typing `grep CPU out`. You'll find that the code is running much slower than the channel flow simulations in Section 25.2. The reason is that it takes much time to generate the synthetic inlet fluctuations (partly because we have few cells in the x direction and because we run only one iterations, i.e. `minit=1`).

It turns out that the `gfortran` compiler is much slower than `ifort` for trigonometric functions. Hence, two new functions – `rcos` and `rsin`– are written where *cos* and *sin* are approximated by the first terms in Taylor expansion (they are included at the end of `synt_generate`).

Now plot the result using the Python file `pl_uuvvww_2d_channel.py` or the MAtlab/Octave file `pl_uuvvww_2d_channel.m` (they are located in `calc-les-202?`).

Now you may change some parameters in the synthetic inlet fluctuations in order to investigate the sensitivity. What happens if you

- change the number of Fourier modes. This is set with the variable `nmodes` in subroutine `synt_init`.
- change the integral length scale for defining the location of the peak of the energy spectrum. This is set with the variable `rl_in` in subroutine `mod` in `entry modini`.
- change the integral time scale. This is set with the variable `tturb` in subroutine `mod` in `entry modu`.
- change the anisotropy. The eigenvalues and eigenvectors set in subroutine `synt_init` were obtained from the EARSM model using the Python or Matlab/Octave script `synt_main_earsm` which can be downloaded at [13]. Instead of taking the Reynolds stress tensor from EARSM, you can take it from DNS as in Section III in [19]. Use then the DNS data file `LM_Channel_5200_mean_prof.dat` (choose, for example, the Reynolds stress tensor where $|\overline{v_1' v_2'}|$ is largest).

25.4 Half-width channel flow: a hybrid one-equation turbulence model

Here we will implement a hybrid one-equation turbulence model. The turbulence model is described in Chapter *The one-equation hybrid LES-RANS model* in [12].

Start by creating a new directory and copy all files used in Section 25.2.

25.4.1 setup.f

Section 8 We will not use the PANS model. Hence we set

```
pans=.false.
```

At the same time, we should define a new logical variable for the new one-equation model. Let's call it `keq`. We active the new turbulence model with

```
keq=.true.
```

We want the new logical variable to be global. Hence we must add it in COMMON at two places

```
1/alll/save,restrt,cycli,cyclk,steady,echo,les,wale,pans,kom, 2
les_abl,stability,keq_abl,keq
logical pans,wale,les_abl,stability,keq_abl,keq
```

Section 11 The k equation is solved in this model.

```
solve(te)=.true.
```

Section 12 The one-equation model is a hybrid model (i.e. a combination of RANS and LES). Here we will specify at which cell number (in the wall-normal direction, i.e. `j`) the model switches from RANS to LES. We do this as

```
jl0=20
```

Note that the variable `jl0` is already included in the COMMON file (is it global, accessible in all subroutines). Setting `jl0=20` means that for cell 2-20, RANS is used and for cell 21-njm1, LES is employed.

25.4.2 mod.f

entry modu Delete the code where f_k is computed. This coding part starts with

```
      cdes=0.67
      if (iter.eq.1) then
         do k=2,nkm1
         do j=2,njm1
         do i=2,nim1
C
            rk=phi(i,j,k,te)
            diss1=phi(i,j,k,ed)
```

entry modte We will solve the k equation. The dissipation, `phi(i,j,k,ed)`, is added as a sink term in subroutine `calcte`. This means that we must compute the dissipation and put it in `phi(i,j,k,ed)`. We do that in `modte`. Table 25.1 gives the expression for the dissiption. It can be implemented as

```
c sink term
      cl=cappa*cmu**(-.75)
      amu=70.
      aeps=2.*cl

      do k=2,nkm1
```

	URANS region	LES region
ℓ	$\kappa c_\mu^{-3/4} n[1-\exp(-0.2k^{1/2}n/\nu)]$	$\ell = \Delta$
ν_T	$\kappa c_\mu^{1/4} k^{1/2} n[1-\exp(-0.014k^{1/2}n/\nu)]$	$0.07k^{1/2}\ell$
C_ε	1.0	1.05

Table 25.1: Turbulent viscosity and turbulent length scales in the URANS and LES regions. n and κ denote the distance to the nearest wall and von Kármán constant ($= 0.41$), respectively. $\Delta = (\delta V)^{1/3}$ (taken from [12])

.

```
      do j=2,jl0
      do i=2,nim1
c in calcte: a sink (dissipation) term is added as phi(i,j,k,ed)
c set ed=k**1.5/delta
         xy=yp(i,j,k)
         rn=max(sqrt(phi(i,j,k,te))*densit*xy/viscos,1.e-5)

c calc lmu & leps
         arg=rn/aeps
         arg=min(60.,arg)
         rleps=cl*xy*(1.-exp(-arg))

c calc eps
         phi(i,j,k,ed)=phi(i,j,k,te)**1.5/rleps

      end do
      end do
      end do

      do k=2,nkm1
      do j=jl0+1,njm1
      do i=2,nim1
         delt1=vol(i,j,k)**0.33333
         phi(i,j,k,ed)=1.05*phi(i,j,k,te)**1.5/delt1
      end do
      end do
      end do
```

25.4.3 vist_keq.f

The turbulent viscosity should be computed, see Table 25.1. To do this, copy – for example – the file `vist.f` i.s.

```
cp vist.f vist_keq.f
```

Change the name of the subroutine to `vist_keq` so that

```
subroutine vist_keq
```

Then do the coding for the expressions of the turbulence viscosity, see Table 25.1.

25.4.4 main.f

Next step is to make sure that the `vist_keq` is called in `main.f`. Insert the code line after, for example, the call to `vist_pans`, i.e.

```
if (pans) call vist_pans
if (keq) call vist_keq
```

25.4.5 makefile

The last step is to add the new routine to the `makefile`. This reads

```
../coeff.o ../peter_relax.o ../conv.o ../filt_time.o ../vist_keq.o
```

Now you should compile the code. But since you have changed the COMMON file which is included in many subroutine when compiling, you must first remove all object files. Do this with the command

```
rm *.o; rm *../*.o
```

Now compile the code by typing `make` and execute the code.

Plot the results. You find that the interface at `j=jl0` is visible in the velocity profile. Also you can see that the turbulent viscosity decreases rapidly outside the interface.

Change `jl0`, re-compile and run the code and see what happens.

25.5 Half-width channel flow: a DES $k-\varepsilon$ turbulence model

We will implement a $k-\varepsilon$ model DES based on the AKN model which is the basis of the PANS model used in Section 25.2.

When implementing a DES $k-\varepsilon$ turbulence model, we will modify the dissipation term in the k equation so that $-\varepsilon$ is replaced with $-\psi\varepsilon$ where (see Eq. 11 in [17])

$$\psi = \max\left(1, \frac{k^{3/2}/\varepsilon}{C_{DES}\Delta_{max}}\right), \quad \Delta_{max} = \max(\Delta x_1, \Delta x_2, \Delta x_3) \tag{25.1}$$

Start by creating a new directory and copy all files used in Section 25.2.

25.5.1 setup.f

We will not use the PANS model. Hence we set

Section 8

```
pans=.false.
```

At the same time, we should define a new logical variable for the new DES $k-\varepsilon$ model. Let's call it `des`. We active the new turbulence model with

```
des=.true.
```

We want the new logical variable to be global. Hence we must add it in COMMON at two places

```
1/alll/save,restrt,cycli,cyclk,steady,echo,les,wale,pans,kom, 2
les_abl,stability,keq_abl,keq,des
logical pans,wale,les_abl,stability,keq_abl,des
```

Section 11 The k and ε equations are solved in this model.

```
if (des) then
   solve(te)=.true.
   solve(ed)=.true.
end if
```

Section 16 We use the same turbulent constants as as in the PANS model. We insert the coding

```
if (des) then
   c1=1.5
   c2=1.9
end if
```

Section 17 We use the same turbulent Prandtl numbers as in the PANS model. So insert this code at the end of the section:

```
if (des) then
   prt(te)=1.4
   prt(ed)=1.4
end if
```

25.5.2 mod.f

entry modu Delete the code where f_k is computed. This coding part starts with

```
cdes=0.67
if (iter.eq.1) then
   do k=2,nkm1
   do j=2,njm1
   do i=2,nim1
C
      rk=phi(i,j,k,te)
      diss1=phi(i,j,k,ed)
```

25.5.3 calcte_des.f

Copy the file `calcte.f` to `calcte_des.f`, i.e.

```
cp  calcte.f calcte_des.f
```

Now we should insert ψ in Eq. 25.1. This is done with the coding

```
          ymin=yp(i,j,k)
          rl_dist=0.15*ymin
          dy=yc(i,j,k)-yc(i,j-1,k)
          dx=xc(i,j,k)-xc(i-1,j,k)
          dz=zc(i,j,k)-zc(i,j,k-1)

          delta_max=max(dx,dz,dy)

          rk=phi(i,j,k,te)
          diss=phi(i,j,k,ed)

          rl=rk**1.5/diss

          dmax=0.67*delta_max
          psi=rl/dmax  !DES
          psi=max(1.,psi)

c store it in fk for post-processing
          fk(i,j,k)=psi


c---------dissipation term
          sp(i,j,k)=sp(i,j,k)-densit*psi*phi(i,j,k,ed)/phi(i,j,k,te)
     &              *vol(i,j,k)
```

25.5.4 calced_des.f

Copy the file `calced_pans.f` to `calced_des.f`. Remove the f_k function in the destruction tern, i.e.

```
c--------dissipation term
         sp(i,j,k)=sp(i,j,k)-fdampf2*c2*densit*phi(i,j,k,ed)
     &             *vol(i,j,k)/phi(i,j,k,te)
```

25.5.5 main.f

Next step is to make sure that the `calcte_des, calced_des, vist_keq` are called in `main.f`. Insert the code line after, for example, the call to `calcte_keqabl`, i.e.

```
                elseif (keq_abl) then
                   call calcte_keqabl
                elseif (des) then
                    call calcte_des
```

and

```
               elseif (des) then
                  call calced_des
```

and

```
if (pans.or.des) call vist_pans
```

25.5.6 makefile

The last step is to add the new routine to the `makefile`. This reads

```
../coeff.o ../peter_relax.o ../conv.o ../filt_time.o ../vist_keq.o
../calcte_des.o  ../calced_des.o
```

Now you should compile the code. But since you have changed the COMMON file which is included in many subroutine when compiling, you must first remove all object files. Do this with the command

```
rm *.o; rm *../*.o
```

Now compile the code by typing `make` and execute the code.
Plot the results. Plot also f_k (it it loaded as `fk2d`).

25.6 Half-width channel flow: a IDDES $k-\varepsilon$ turbulence model

Here we will implement a IDDES $k-\varepsilon$ turbulence model.

Repeat the procedure in Section 25.5, but now for IDDES. The formulas to be used are given in Section 11.

25.7 Heat transfer in half-width channel flow with inlet-outlet

Here we will compute heat transfer and the flow in a half-width channel with inlet and outlet. We use PANS as a turbulence model. We start by copying `setup.f` and `mod.f` (and all other files) from Section 25.3.

25.7.1 setup.f

Section 11 We will solve for temperature. Hence set

```
solve(t)=.true.
```

You find in the beginning `main` that the variable `t` is already defined (it is set to `t=9`). Hence you must increase the variable `nphmax` in Section 1 to 9, i.e.

```
nphmax=9
```

Check also in `COMMON` to verify that the dimension of `phi, solve ...` is sufficient. You find that

```
parameter(it=454,jt=200,kt=102,nphit=9,nphito=9)
```

which is large enough (`nphit=9,nphito=9`).

Section 6 Set the laminar Prandtl number to 0.72, i.e.

```
prt_lam(t)=0.72
```

Section 11

```
solve(t)=.true.
```

For the momentum equations we use central differencing (`scheme='c'`) and for the turbulent quantities we use first-order hybrid/upwind (`scheme='h'`). For scalar equations (like temperature) we usually need to use a bounded scheme. Here we choose the MUSCL scheme

```
schemet='m'
```

When using the MUSCL scheme, we can choose to use full MUSCL or a linear combination of MUSCL and central differencing. This is determined by the variable `blend`. If it is equal to one, it is full central differencing (deferred correction). If it is zero, it is full MUSCL (which we choose). Hence

```
blend=0.
```

Section 17 The turbulent Prandtl number σ_c must be set. We choose

```
prt(t)=0.8
```

25.7.2 main.f

Look at this routine. Go to the `110 loop`, i.e.

```
do 110 nphi=9,nphmax
```

You find that `calcph` is solved if solve(t)=.true. and that the discretize scheme is determined by the variable `schemet`.

25.7.3 mod.f

entry modph We will set inlet boundary conditions $T = 0$ for the temperature. All variables are set to zero by default at all boundaries. Hence we don't need to do anything.

Let us set a constant heat transfer at the lower boundary. We do that in `modph` which is called by `calcph`.

```
do i=2,nim1
do k=2,nkm1
    j=2
    su(i,j,k)=su(i,j,k)+1.*areany(i,j-1,k)
    as(i,j,k)=0.
    j=njm1
    an(i,j,k)=0.
end do
end do
```

Note that the usual heat transfer via the diffusion is cut-off be setting $a_S = 0$.

We also define the upper surface as an adiabatic surface.

Now run and plot. Check in file `out` that you see the temperature variable, for example

```
************************************************
 ---------------------- Section 11 ---------------------

 The following variables are solved:

 U

 V

 W

 P

 TE

 ED

 T
```

You can plot skinfriction and Nusselt number as well as resolve temperature fluctuations at the wall.

You can also plot a contour of the temperature as by inserting the following lines in `pl_uuvvww_2d_channel.py`

```
x2d=np.zeros((ni,nj))
y2d=np.zeros((ni,nj))
for i in range(0,ni):
   for j in range(0,nj):
      x2d[i,j]=x[i]
      y2d[i,j]=y[j]

fig10 = plt.figure()
plt.xlabel("x")
plt.ylabel("y")
plt.contourf(x2d,y2d,t2d, 100)
plt.clim(0,1)
plt.title("temperature")
plt.colorbar()
plt.savefig('t_contour.eps')
```

25.8 Dispersion of passive pollution source

Here we will simulate dispersion of passive pollution source with heat transfer in a half-width channel with inlet and outlet. We will use PANS as a turbulence model. We start by copying `setup.f` and `mod.f` (and all other files) from Section 25.7

25.8.1 main.f

You find in the beginning of `main` that the there is no variable for a concentration. Let's denote the concentration `conc` (note that you cannot, for example,

call it `c1` since that is already used as a constant in the $k - \varepsilon$ turbulence model). We define `conc` as solution variable number 10, i.e.

```
conc=10
```

Check also in `COMMON` to verify that the dimension of `phi, solve ...` is sufficient. You find that

```
parameter(it=454,jt=200,kt=102,nphit=9,nphito=9)
```

which is not large enough. Set

```
parameter(it=454,jt=200,kt=102,nphit=10,nphito=10)
```

We must also define the variable `conc` as an integer

```
integer u,v,w,p,pp,m,te,ed,t,f1,f2,f3,om,conc
```

and define it as a global variable

```
1/point/u,v,w,p,pp,m,te,ed,t,f1,f2,f3,om,conc
```

25.8.2 echo1.f

We need to define the name of the concentration. Look for the line `if (nphmax.ge.9) name(t)='T'`. Insert

```
if (nphmax.ge.9) name(t)='T'
if (nphmax.ge.10) name(conc)='C'
```

25.8.3 setup.f

We must increase `nphmax` to 10, i.e.

```
nphmax=10
```

Section 6 Set the laminar Prandtl number to 0.72, i.e.

```
prt_lam(conc)=0.72
```

Section 11

```
solve(conc)=.true.
```

The same discretization scheme as the temperature is used.

Section 17 The turbulent Prandtl number σ_t must be set. We choose

```
prt(conc)=0.8
```

25.8.4 mod.f

entry modph When we solved for the temperature in the Section 25.7 is was done by calling `calcph`. This will also be the case for the concentration and it is called automatically in `main.f`. The boundary conditions for concentration will be set in `entry modph` both for temperature and concentration. Hence, the boundary conditions set for temperature in Section must in enclosed in an IF statement as

```
if (nphi.eq.t) then
   do i=2,nim1
   do k=2,nkm1
       j=2
       su(i,j,k)=su(i,j,k)+1.*areany(i,j-1,k)
       as(i,j,k)=0.
       j=njm1
       an(i,j,k)=0.
   end do
   end do
end if
```

We will set inlet boundary conditions $C = 0$ for the temperature. All variables are set to zero by default at all boundaries. Hence we don't need to do anything.

Solving for C

Now we will introduce a pollution source at $x \simeq 1$ and for $y < 0.1$. Look at x and y coordinates in the file `out` in the directory used in Section 25.7. We find that we should set the source for cells $i = 12$ and $j = 2 - 19$. We set the source over the entire z direction. We choose the strength of the source as 0.001 per unit volume. Hence

```
if (nphi.eq.conc) then
   i=10
   do k=2,nkm1
   do j=2,21
       su(i,j,k)=su(i,j,k)+vol(i,j,k)*0.001
   end do
   end do
endif
```

We also define the lower and upper surface as zero-flux boundaries

```
do i=2,nim1
do k=2,nkm1
    j=2
    as(i,j,k)=0.
    j=njm1
    an(i,j,k)=0.
end do
end do
```

We need to time and spanwise average the concentration. Open the `statisz.f` file in the main directory. We could define a new variable `cvec`. But it is easier to instead put the concentration in the `wvec` variable (which anyway is zero).

Hence re-write the line as

```
vvec(i,j) =vvec(i,j) +phi(i,j,k,v)/fnk
wvec(i,j) =wvec(i,j) +phi(i,j,k,conc)/fnk
```

Compile and run. Since you have made many changes in `COMMON`. it may be a good idea to compile with the option `-fbounds-check` and run a couple of time steps. When it works, recompile without this option (is slows down the code).

Check in file `out` that you see the concentration variable, for example

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
 time step:     1      dt:  1.227E-03     time:  1.227E-03
                  variable:  U         residual:  7.125E-05
                  variable:  V         residual:  2.526E-05
                  variable:  W         residual:  3.414E-05
                  variable:  P         residual:  1.375E-03
                  variable:  TE        residual:  7.353E-04
                  variable:  ED        residual:  2.758E+08
                  variable:  T         residual:  8.582E-04
                  variable:  C         residual:  1.186E-11
                  max residual:  1.375E-03 at iteration    1 time step     1
```

You can now add line plots of the concentration in `pl_uuvvww_2d_channel.py`. Recall that it is stored in `statisz.f` as `w`. So type

```
w=vectz[iw]/ntstep
conc2d=np.reshape(w,(ni,nj))
```

Not you can make line plots in the same way as you plot viscosity or temperature. You can also make contour plots of concentration (cf. end of Section 25.7)

```
fig11 = plt.figure()
plt.xlabel("x")
plt.ylabel("y")
plt.contourf(x2d,y2d,conc2d, 100)
plt.clim(0,0.02)
plt.title("concentration")
plt.colorbar()
plt.savefig('c_contour.eps')
```

25.9 Heat transfer with buoyancy

Here we will simulate heat transfer with buoyancy. We start by copying `setup.f` and `mod.f` (and all other files) from Section 25.7

The only thing we need to change is to add a buoyancy term in the vertical momentum equation. We assume that the channel walls are aligned with the $x - z$ plane, and hence the vertical direction is y.

25.9.1 mod.f

entry modv Here we add the buoyancy source term (cf. `entry modv` in Section 24)

```
      ! add bouyancy term to the v-eq
tref=0.
grav       = 9.81        ! gravitation
tnoll      = 300.0       ! reference temperature (T_0 = 300K)
volextcoef = 1./tnoll    ! volumetric expansion coefficient of air (1/T_0)

do k=2,nkm1
do j=2,njm1
do i=2,nim1
   su(i,j,k)=su(i,j,k)+grav*volextcoef*
&                        (phi(i,j,k,t)-tref)*vol(i,j,k)
end do
end do
end do
```

You may notice that whereas in Section 24 we set the reference temperature to the plane-averaged temperature, here we use a constant value. We can use a plane-averaged temperature in the former case since we have periodic boundaries.

26 COMMON blocks

26.1 COMMON

Make sure that the dimension of the vectors is sufficient.

`it` Must be larger than `ni`

`jt` Must be larger than `nj`

`kt` Must be larger than `nk`

`nphit, nphito` Must be larger than `numph`

If you change anything in this file, you must re-compile the entire code. This is done by deleting all object files *.o and ../*.o and then type `make`.

26.2 PETER_COMMON

`iomax` Must be larger than `1.2 * ni * nj * nk`

`io2dmax` Must be larger than `1.5 * n1 * n2` where `n1` is the largest of (`ni, nj, nk`) and `n2` is the second largest

If you change anything in this file, you must re-compile the entire code. This is done by deleting all object files *.o and ../*.o and then type `make`.

26.3 Variables in COMMON blocks in file `COMMON`

`acrank`: time integration scheme for pressure (1: fully implicit)

`acrank_conv`: time integration scheme for convection and diffusion (1: fully implicit)

`alpha_geo`: some kind of under-relaxation factor on the pressure gradient (only for ABL)

`areaex,areaey,areaez`: the x, y and z component of the area of east face

`areahx,areahy,areahz`: the x, y and z component of the area of high face

`areanx,areany,areanz`: the x, y and z component of the area of north face

`aw,ae,as,an,al,ah,ap`: discretization coefficients, a_W, a_W, a_S, a_N, a_L, a_H, a_P

`betap`: driving pressure gradient

`c1,c2,cappa`: turbulence model constants, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, κ

`cdfor`: drag value of forest [31] (only for ABL)

`cdiss`: dissipation parameter [20] (only for ABL[1])

`cmu,cd,cmucd`: turbulence model constants, C_μ, $C_d = 1$, $cmucd = cmu * cd$

`conv_blend`: blending factor between CDS and MUSCL (1: fully CDS)

`conve,convn,convh`: convection through east, north and high face

`cycli,cyclk`: if true, cyclic in i and k direction

`delta_leaf`: leaf size, lf [31] (only for ABL)

`densit`: density, ρ

`dist2d`: distance to the nearest wall (used in PANS)

`dqdy`: y derivative of heat flux in forest (only for ABL)

`dt`: time step

`echo`: if true, echoes settings

`elog,pfun`: constants in RANS wall-function, E, P

`extinct`: extinction coefficient in forest (only for ABL)

`f_lad`: Leaf-Area Density (LAD) (only for ABL)

`fcori`: Coriolis coefficient (only for ABL)

[1] Atmospheric boundary layers

`fk,fe:` f_k, f_ε, used in PANS

`flowangle:` the direction of the wind [degrees] (only for ABL)

`fmax_leaf_area_dens:` max value for the leaf area density (only for ABL)

`forceu,forcev,forcew:` Drag force due to forest in u, v and w momentum equations (only for ABL)

`forheight:` forest height (only for ABL)

`formaxdens,:` height at which the forest is most dense (only for ABL)

`fx,fy,fz:` f_x, f_y, f_y, the interpolation function in I, J and K direction

`gente:` velocity derivatives in P^k in the k and ε equations

`grav:` acceleration gravity coefficient (9.81) (only for ABL)

`great:` 1e20

`head(nphi):` name of variable `nphi`

`heat_forest:` total heat source in forest (only for ABL)

`hWind:` the height at which the wind should be specified [m] (only for ABL)

`imon,jmon,kmon:` print time history of `phi` for this node

`iprint_start:` timestep at which time averaging begins

`isol(7):` Governs the way the $\bar{p}$ equation is solved in the MG solver (see `peter_init`.)

`it,jt,kt:` dimension of the arrays (i,j,k direction)

`iter:` current global iteration

`itstep:` current timestep

`jformax:` number of cells in forest (only for ABL)

`jhub:` hub height, index j (only for ABL)

`jl0:` interface between RANS and LES in PANS

`keq_abl:` one-equation SGS model for atmospheric boundary layers (only for ABL)

`kom:` if true, $k-\omega$ model

`les:` if true, LES (Smagorinsky)

`les_abl:` Smagorinsky model (only for ABL)

`maxit:` maximum number of global iterations at each timestep

`minit:` minimum number of global iterations at each timestep

`name(nphi):` short name of variable `nphi`

`nfiles_restart:` number of variables in restart file

`ni,nj,nk:` number of cell centers (including boundaries) in i, j and k direction

`nim1,njm1,nkm1:` ni-1, nj-1, nk-1

`nphit,nphito:` size of `phi` and `phio` arrays

`nphmax:` number of nphi

`nsweep(nphi):` number of sweeps in TDMA solver and MG solver

`ntstep:` number of timesteps

`pans:` if true, PANS model

`phi(nphi):` dependent variable ($\bar{v}$, $\bar{v}$, ...)

`phio(nphi):` dependent variable ($\bar{v}$, $\bar{v}$, ...) at old time step

`prt(nphi):` σ_Φ, turbulent Prandtl number

`prt_lam(nphi):` σ_Φ, viscous Prandtl number

`qfluxh:` constant heat flux at canopy top $[Km/s]$ (only for ABL)

`resor(nphi):` scaling of residuals

`restrt:` if true, read the flow field from file 'savres'

`rrl:` used to compute `cdiss` (only for ABL)

`save:` if true, save the flow field to file 'savres' (it overwrites old file)

`scheme:` discretization scheme for $\bar{v}$, $\bar{v}$, $\bar{w}$, $\bar{\theta}$ ('c', CDS; 'h', hybrid; 'v', van Leer; 'm', MUSCL)

`schemet:` discretization scheme for temperature

`schtur:` discretization scheme for k, ε, ω

`small:` 1e-20

`solve(nphi):` true if `phi` is solved for

`sormax:` global convergence limit in each time step

`sp,su:` discretization source terms, S_p, S_U

`spvw:` discretization S_P source terms in $\bar{v}$ and $\bar{w}$ equations (used for van Leer and MUSCL)

`stability:` is set to true if the temperature equation should be solved for (only for ABL)

`steady:` if true, steady

`suv,suw:` discretization S_U source terms in $\bar{v}$ and $\bar{w}$ equations (used for van Leer and MUSCL)

`tecplot:` generate a Tecplot data file or not

`time:` current time

`tnoll:` reference temperature ($T_0 = 300K$) (only for ABL)

`u,v,w,p,pp,te,ed,om:` integers used in `phi`, `prt`, ... ($\bar{v}$, $\bar{v}$, $\bar{w}$, $\bar{p}$, p', k, ε, ω)

`u_geo,w_geo:` the desired wind speed magnitude in u and w (only for ABL)

`urfvis:` under-relaxation factor for turbulent viscosity

`ustarv:` friction velocity, u_τ

`vis:` effective dynamic viscosity, $\mu_{eff} = \mu + \mu_t$:

`viscos:` dynamic viscosity, μ

`vol:` volume of cell

`volextcoef:` volumetric expansion coefficient of air ($1/T_0$) (only for ABL)

`wale:` if true, WALE model

`xc,yc,zc:` X_C, Y_C, Z_C, coordinates of east-north-high corner of cell

`xp,yp,zp:` X_P, Y_P, Z_P, cell center coordinates in x, y and z direction

`ynoll:` surface roughness at ground [31] (only for ABL)

References

[1] K. Abe, T. Kondoh, and Y. Nagano. A new turbulence model for predicting fluid flow and heat transfer in separating and reattaching flows - 1. Flow field calculations. *Int. J. Heat Mass Transfer*, 37(1):139–151, 1994.

[2] M. Billson. *Computational Techniques for Turbulence Generated Noise*. PhD thesis, Dept. of Thermo and Fluid Dynamics, Chalmers University of Technology, Göteborg, Sweden, 2004.

[3] M. Billson, L.-E. Eriksson, and L. Davidson. Jet noise prediction using stochastic turbulence modeling. AIAA paper 2003-3282, 9th AIAA/CEAS Aeroacoustics Conference, 2003.

[4] A. D. Burns and N. S. Wilkes. A finite difference method for the computation of fluid flow in complex three-dimensional geometries. Aere r 12342, Harwell Laboratory, U.K., 1986.

[5] L. Davidson. LES of recirculating flow without any homogeneous direction: A dynamic one-equation subgrid model. In K. Hanjalić and T. W. J. Peeters, editors, *2nd Int. Symp. on Turbulence Heat and Mass Transfer*, pages 481–490, Delft, 1997. Delft University Press.

[6] L. Davidson. Hybrid LES-RANS: Inlet boundary conditions. In B. Skallerud and H. I. Andersson, editors, *3rd National Conference on Computational Mechanics – MekIT'05 (invited paper)*, pages 7–22, Trondheim, Norway, 2005.

[7] L. Davidson. Using isotropic synthetic fluctuations as inlet boundary conditions for unsteady simulations. *Advances and Applications in Fluid Mechanics*, 1(1):1–35, 2007.

[8] L. Davidson. Hybrid LES-RANS: Inlet boundary conditions for flows with recirculation. In *Second Symposium on Hybrid RANS-LES Methods*, Corfu island, Greece, 2007.

[9] L. Davidson. Inlet boundary conditions for embedded LES. In *First CEAS European Air and Space Conference*, 10-13 September, Berlin, 2007.

[10] L. Davidson. Hybrid LES-RANS: Inlet boundary conditions for flows including recirculation. In *5th International Symposium on Turbulence and Shear Flow Phenomena*, volume 2, pages 689–694, 27-29 August, Munich, Germany, 2007.

[11] L. Davidson. HYBRID LES-RANS: Inlet boundary conditions for flows with recirculation. In *Advances in Hybrid RANS-LES Modelling*, volume 97 of *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 55–66. Springer Verlag, 2008.

[12] L. Davidson. Fluid mechanics, turbulent flow and turbulence modeling. eBook, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology, Gothenburg, http://www.tfd.chalmers.se/~lada/postscript_files/solids-and-fluids_turbulent-flow_turbulence-modelling.pdf, 2014.

[13] L. Davidson. Python and matlab scripts for synthetic fluctuations, 2014. URL http://www.tfd.chalmers.se/~lada/projects/inlet-boundary-conditions/proright.html.

[14] L. Davidson. Zonal PANS: evaluation of different treatments of the RANS-LES interface. *Journal of Turbulence*, 17(3):274–307, 2016.

[15] L. Davidson and M. Billson. Hybrid LES/RANS using synthesized turbulent fluctuations for forcing in the interface region. *International Journal of Heat and Fluid Flow*, 27(6):1028–1042, 2006.

[16] L. Davidson and B. Farhanieh. CALC-BFC: A finite-volume code employing collocated variable arrangement and cartesian velocity components for computation of fluid flow and heat transfer in complex three-dimensional geometries. Rept. 95/11, Dept. of Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 1995.

[17] L. Davidson and C. Friess. A new formulation of f_k for the PANS model. *Journal of Turbulence*, pages 1–15, 2019. doi: 10.1080/14685248.2019.1641605. URL http://dx.doi.org/10.1080/14685248.2019.1641605.

[18] L. Davidson and S.-H. Peng. Hybrid LES-RANS: A one-equation SGS model combined with a $k-\omega$ model for predicting recirculating flows. *International Journal for Numerical Methods in Fluids*, 43:1003–1018, 2003.

[19] L. Davidson and S.-H. Peng. Embedded large-eddy simulation using the partially averaged Navier-Stokes model. *AIAA Journal*, 51(5):1066–1079, 2013. doi: doi:10.2514/1.J051864. URL http://dx.doi.org/10.2514/1.J051864.

[20] J. W. Deardorff. Stratocumulus-capped mixed layers derived from a three-dimensional model. *Boundary-Layer Meteorology*, 18(4):495–527, 1980.

[21] P. Emvin. *The Full Multigrid Method Applied to Turbulent Flow in Ventilated Enclosures Using Structured and Unstructured Grids*. PhD thesis, Dept. of Thermo and Fluid Dynamics, Chalmers University of Technology, Göteborg, 1997.

[22] M. Gritskevich, A. Garbaruk, J. Schütze, and F. R. Menter. Development of DDES and IDDES formulations for the $k - \omega$ shear stress transport model. *Flow, Turbulence and Combustion*, 88:431–449, 2012.

[23] J. O. Hinze. *Turbulence*. McGraw-Hill, New York, 2nd edition, 1975.

[24] M. Irannezhad. DNS of channel flow with finite difference method on a staggered grid. Msc thesis, Division of Fluid Dynamics, Department of Applied Mechanics, Chalmers University of Technology, Göteborg, Sweden, 2006.

[25] N. Jarrin, S. Benhamadouche, D. Laurence, and R. Prosser. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *International Journal of Heat and Fluid Flow*, 27(4):585–593, 2006.

[26] J. Ma, S.-H. Peng, L. Davidson, and F. Wang. A low Reynolds number variant of Partially-Averaged Navier-Stokes model for turbulence. *International Journal of Heat and Fluid Flow*, 32(3): 652–669, 2011. doi: 10.1016/j.ijheatfluidflow.2011.02.001. URL http://dx.doi.org/10.1016/j.ijheatfluidflow.2011.02.001. 10.1016/j.ijheatfluidflow.2011.02.001.

[27] B. Nebenführ and L. Davidson. INFLUENCE OF A FOREST CANOPY ON THE NEUTRAL ATMOSPHERIC BOUNDARY LAYER - A LES STUDY. In *Proceedings of 10th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements (ETMM10), Marbella, Spain, 17-19 September*, 2014.

[28] B. Nebenführ and L. Davidson. Large-eddy simulation study of thermally stratified canopy flow. *Boundary-Layer Meteorology*, pages 1–24, 2015. ISSN 0006-8314. doi: 10.1007/s10546-015-0025-9. URL http://dx.doi.org/10.1007/s10546-015-0025-9.

[29] B. Nebenführ and L. Davidson. Prediction of wind-turbine fatigue loads in forest regions based on turbulent les inflow fields. *Wind Energy*, 2016. ISSN 1099-1824. doi: 10.1002/we.2076. URL http://dx.doi.org/10.1002/we.2076. we.2076.

[30] F. Nicoud and F. Ducros. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. *Flow, Turbulence and Combustion*, 62(3):183–200, 1999.

[31] R. H. Shaw and U. Schumann. Large-eddy simulation of turbulent flow above and within a forest. *Boundary-Layer Meteorology*, 61:47 – 64, 1992.

[32] M. L. Shur, P. R. Spalart, M. K. Strelets, and A. K. Travin. A hybrid RANS-LES approach with delayed-DES and wall-modelled LES capabilities. *International Journal of Heat and Fluid Flow*, 29:1638–1649, 2008.

[33] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91:99–165, 1963.

[34] B. van Leer. Towards the ultimate conservative difference scheme. Monotonicity and conservation combined in a second order scheme. *Journal of Computational Physics*, 14(4):361–370, 1974.

[35] B. van Leer. Towards the ultimate conservative difference scheme. V. A second-order sequel to godonov's method. *Journal of Computational Physics*, 32:101–136, 1979.

[36] S. Wallin and A. V. Johansson. A new explicit algebraic Reynolds stress model for incompressible and compressible turbulent flows. *Journal of Fluid Mechanics*, 403:89–132, 2000.

[37] J. R. Welty, C. E. Wicks, and R. E. Wilson. *Fundamentals of Momentum, Heat, and Mass Transfer*. John Wiley & Sons, New York, 3 edition, 1984.

[38] D. C. Wilcox. Reassessment of the scale-determining equation. *AIAA Journal*, 26(11):1299–1310, 1988.