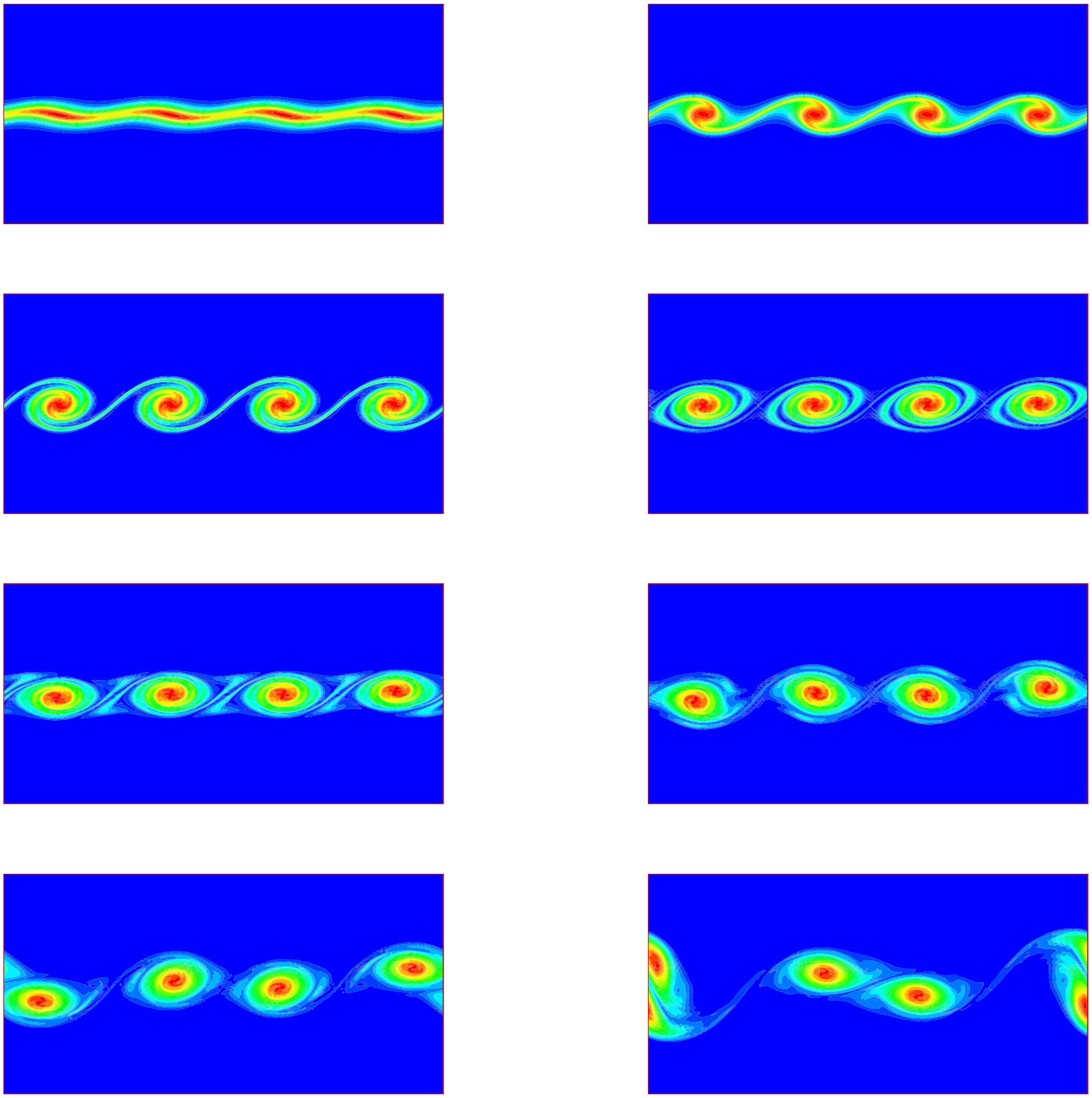
An Introduction to LES/DNS simulation



Contents

1 Introduction

In this report we will describe a simple FORTRAN program which calculates three dimensional turbulent flow in simple (Cartesian) geometries, using LES. A Smagorinsky subgrid model is used to model the scales which are smaller than a grid volume. This report/program gives only an outline of the solution method used in LES/DNS-simulations, we will not give the theory behind the methods because several good textbooks exist on this subject.

2 The equations of motion

In this section we will give the governing equations for an incompressible viscous flow, i.e. the continuity and Navier-Stokes equations. The continuity equation reads:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0, \tag{1}$$

where u is the velocity in the x-direction, v the velocity in the y-direction and w the velocity in the z-direction. The Navier-Stokes equation in the x-direction reads:

$$\frac{\partial u}{\partial t}=-\frac{\partial uu}{\partial x}-\frac{\partial vu}{\partial y}-\frac{\partial wu}{\partial z}-\frac{1}{\rho}\frac{\partial P}{\partial x}+\frac{\partial}{\partial x}\left[2\nu\frac{\partial u}{\partial x}\right]+\frac{\partial}{\partial y}\left[\nu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]+\frac{\partial}{\partial z}\left[\nu\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right)\right]-g_x, \tag{2}$$

where ν is the (non-constant) kinematic viscosity, P the pressure, and g_x a volume force in the x-direction (for instance gravity). The Navier-Stokes equation in the y-direction reads:

$$\frac{\partial v}{\partial t}=-\frac{\partial uv}{\partial x}-\frac{\partial vv}{\partial y}-\frac{\partial wv}{\partial z}-\frac{1}{\rho}\frac{\partial P}{\partial y}+\frac{\partial}{\partial x}\left[\nu\left(\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}\right)\right]+\frac{\partial}{\partial y}\left[2\nu\frac{\partial v}{\partial y}\right]+\frac{\partial}{\partial z}\left[\nu\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right)\right]-g_y. \tag{3}$$

The Navier-Stokes equation in the z-direction reads:

$$\frac{\partial w}{\partial t}=-\frac{\partial uw}{\partial x}-\frac{\partial vw}{\partial y}-\frac{\partial ww}{\partial z}-\frac{1}{\rho}\frac{\partial P}{\partial z}+\frac{\partial}{\partial x}\left[\nu\left(\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}\right)\right]+\frac{\partial}{\partial y}\left[\nu\left(\frac{\partial w}{\partial y}+\frac{\partial v}{\partial z}\right)\right]+\frac{\partial}{\partial z}\left[2\nu\frac{\partial w}{\partial z}\right]-g_z. \tag{4}$$

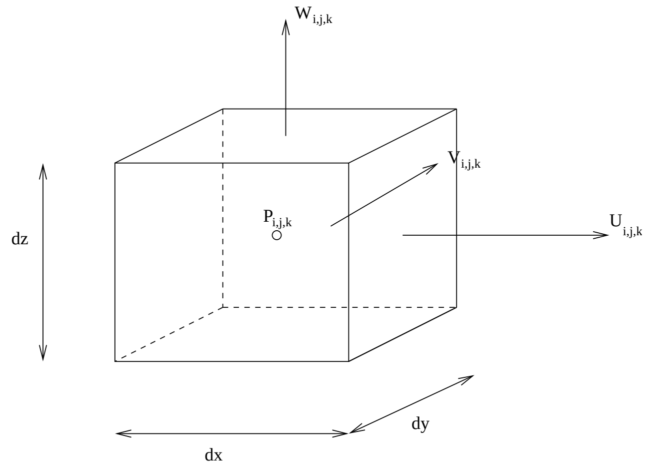


Figure 1: A grid cell of the staggered grid

For convenience we will write the equations above also as:

$$\frac{\partial \underline{u}}{\partial t} = -ADV - \nabla P + DIFF - G, \tag{5}$$

where ADV stands for the advection, $DIFF$ for the diffusion, P for the pressure, and G for the acceleration due to the gravity. In the numerical model the turbulence is modelled with a Smagorinsky eddy viscosity model, i.e.

$$\nu_t = l_{mix}^2 \sqrt{\frac{1}{2} S_{ij} S_{ij}}, \tag{6}$$

where $S_{ij} = S_{ji}$ is the strain rate tensor,

$$\begin{aligned} S_{11} &= 2\frac{\partial u}{\partial x}, \\ S_{22} &= 2\frac{\partial v}{\partial y}, \\ S_{33} &= 2\frac{\partial w}{\partial z}, \\ S_{12} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}, \\ S_{13} &= \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}, \\ S_{23} &= \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}, \end{aligned}$$

The mixing length l_{mix} is related to the grid spacing Δ as following

$$l_{mix} = C_s \Delta, \tag{7}$$

where C_s is the Smagorinsky constant ($C_S \approx 0.1$). In the following we denote the sum of the kinematic and turbulent eddy viscosity by ν, i.e.

$$\nu = \nu_{molc} + \nu_t \tag{8}$$

3 Discretisation

In this section we will approximate the governing equations on a staggered grid. In Figure 2 we have sketched a grid cell. The pressure point is located at the centre of the cell and the velocity components at the cell sides. (The discretisation used here is a straightforward extension of the 2D discretisation given by Patankar [?]) Firstly we will discretize the continuity equation around the pressure point, with central differences:

$$\begin{aligned}\frac{\partial u}{\partial x} &= \frac{1}{dx}(U_{i,j,k} - U_{i-1,j,k})\\ \frac{\partial v}{\partial y} &= \frac{1}{dy}(V_{i,j,k} - V_{i,j-1,k})\\ \frac{\partial w}{\partial z} &= \frac{1}{dz}(W_{i,j,k} - W_{i,j,k-1})\end{aligned}$$

Thus the continuity equation reads:

$$div(u,v,w) = \frac{1}{dx}(U_{i,j,k} - U_{i-1,j,k}) + \frac{1}{dy}(V_{i,j,k} - V_{i,j-1,k}) + \frac{1}{dz}(W_{i,j,k} - W_{i,j,k-1}). \quad (9)$$

The discretisation given above is second order accurate in space. We can derive equation (9) also in a different way. Consider the grid volume given in Figure 2, the mass balance over this volume reads

$$-\rho W_{i,j,k}dxdy + \rho W_{i,j,k-1}dxdy - \rho V_{i,j,k}dxdz + \rho V_{i,j-1,k}dxdz - \rho U_{i,j,k}dydz + \rho V_{i-1,j,k}dydz = 0 \quad (10)$$

dividing the equation above by $\rho dxdydz$ gives exactly equation (9). The Navier-Stokes equation in the x-direction will be approximated around the U-velocity point.

$$\frac{\partial uu}{\partial x} = \frac{1}{dx}\left(\left[\frac{(U_{i+1,j,k} + U_{i,j,k})}{2}\right]^2 - \left[\frac{(U_{i,j,k} + U_{i-1,j,k})}{2}\right]^2\right), \quad (11)$$

$$\frac{\partial uv}{\partial y} = 0.25\frac{1}{dy}\left(\left[(U_{i,j,k} + U_{i,j+1,k})(V_{i,j,k} + V_{i+1,j,k})\right] - \left[(U_{i,j,k} + U_{i,j-1,k})(V_{i,j-1,k} + V_{i+1,j-1,k})\right]\right), \quad (12)$$

$$\frac{\partial uw}{\partial z} = 0.25\frac{1}{dz}\left(\left[(U_{i,j,k} + U_{i,j,k+1})(W_{i,j,k} + W_{i+1,j,k})\right] - \left[(U_{i,j,k} + U_{i,j,k-1})(W_{i,j,k-1} + W_{i+1,j,k-1})\right]\right), \quad (13)$$

$$\frac{\partial p}{\partial x} = \frac{1}{dx}(P_{i+1,j,k} - P_{i,j,k}) \quad (14)$$

$$\frac{\partial}{\partial x}\left[2\nu\frac{\partial u}{\partial x}\right] = \frac{1}{dx}\left[2\nu\left(\frac{U_{i+1,j,k} - U_{i,j,k}}{dx}\right) - 2\nu\left(\frac{U_{i,j,k} - U_{i-1,j,k}}{dx}\right)\right] \quad (15)$$

$$\frac{\partial}{\partial y}\left[\nu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] = \frac{1}{dy}\left[\nu\left(\frac{U_{i,j+1,k} - U_{i,j,k}}{dy} + \frac{V_{i+1,j,k} - V_{i,j,k}}{dx}\right) - \nu\left(\frac{V_{i,j,k} - V_{i,j-1,k}}{dy} + \frac{V_{i+1,j-1,K} - V_{i,j-1,k}}{dx}\right)\right] (16)$$

$$\frac{\partial}{\partial z}\left[\nu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)\right] = \frac{1}{dz}\left[\nu\left(\frac{U_{i,j,k+1} - U_{i,j,k}}{dz} + \frac{W_{i+1,j,k} - W_{i,j,k}}{dx}\right) - \nu\left(\frac{U_{i,j,k} - U_{i,j,k-1}}{dz} + \frac{W_{i+1,j,k-1} - W_{i,j,k-1}}{dx}\right)\right] (17)$$

where $\nu = \nu(x,y,z)$ is a function of the spatial coordinates (x,y,z). The Navier-Stokes equation in the y direction will be approximated around the V velocity point,

$$\frac{\partial uv}{\partial x} = 0.25\frac{1}{dx}[(U_{i,j,k} + U_{i,j+1,k})(V_{i+1,j,k} + V_{i,j,k}) - (U_{i-1,j,k} + U_{i-1,j+1,k})(V_{i,j,k} + V_{i-1,j,k})] \quad (18)$$

$$\frac{\partial vv}{\partial x} = 0.25\frac{1}{dx}[(V_{i,j+1,k} + V_{i,j,k})^2 - (V_{i,j,k} + V_{i,j-1,k})^2] \quad (19)$$

$$\frac{\partial wv}{\partial x} = 0.25\frac{1}{dx}[(V_{i,j,k} + V_{i,j,k+1})(W_{i,j,k} + W_{i,j+1,k}) - (V_{i,j,k} + V_{i,j,k-1})(W_{i,j,k-1} + W_{i,j+1,k-1})] \quad (20)$$

$$\frac{\partial p}{\partial y} = \frac{1}{dy}(P_{i,j+1,k} - P_{i,j,k}) \quad (21)$$

$$\frac{\partial}{\partial x}\left[\nu\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)\right] = \frac{1}{dx}\left[\nu\left(\frac{V_{i+1,j,k} - V_{i,j,k}}{dx} + \frac{U_{i,j+1,k} - U_{i,j,k}}{dx}\right) - \nu\left(\frac{V_{i,j,k} - V_{i-1,j,k}}{dx} + \frac{U_{i-1,j+1+k} - U_{i-1,j,k}}{dx}\right)\right] (22)$$

$$\frac{\partial}{\partial y}\left[2\nu\frac{\partial v}{\partial y}\right] = \frac{1}{dy}\left[2\nu\left(\frac{V_{i,j+1,k} - V_{i,j,k}}{dy}\right) - 2\nu\left(\frac{V_{i,j,k} - V_{i,j-1,k}}{dy}\right)\right] \quad (23)$$

$$\frac{\partial}{\partial z}\left[\nu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)\right] = \frac{1}{dz}\left[\nu\left(\frac{V_{i,j,k+1} - V_{i,j,k}}{dz} + \frac{W_{i,j+1,k} - W_{i,j,k}}{dy}\right) - \nu\left(\frac{V_{i,j,k} - V_{i-1,j,k}}{dz} + \frac{W_{i,j+1,k-1} - W_{i,j,k-1}}{dy}\right)\right] (24)$$

The Navier-Stokes equation in the z-direction will be approximated around the W velocity point.

$$\frac{\partial uw}{\partial x} = 0.25\frac{1}{dx}[(U_{i,j,k} + U_{i,j,k+1})(W_{i+1,j,k} + W_{i,j,k}) - (U_{i-1,j,k} + U_{i-1,j,k+1})(W_{i,j,k} + W_{i-1,j,k})] \quad (25)$$

$$\frac{\partial vw}{\partial y} = 0.25\frac{1}{dy}[(V_{i,j,k} + V_{i,j,k+1})(W_{i,j,k} + W_{i,j+1,k}) - (V_{i,j-1,k} + V_{i,j-1,k+1})(W_{i,j,k} + W_{i,j-1,k})] \tag{26}$$

$$\frac{\partial ww}{\partial z} = 0.25\frac{1}{dz}[(W_{i,j,k+1} + W_{i,j,k})^2 - (W_{i,j,k} + W_{i,j,k-1})^2] \tag{27}$$

$$\frac{\partial p}{\partial z} = \frac{1}{dz}(P_{i,j,k+1} - P_{i,j,k}) \tag{28}$$

$$\frac{\partial}{\partial x}\left[\nu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)\right] = \frac{1}{dx}\left[\nu\left(\frac{W_{i+1,j,k} - W_{i,j,k}}{dx} + \frac{U_{i,j,k+1} - U_{i,j,k}}{dz}\right) - \nu\left(\frac{W_{i,j,k} - W_{i-1,j,k}}{dx} + \frac{U_{i-1,j,k+1} - U_{i-1,j,k}}{dz}\right)\right] \tag{29}$$

$$\frac{\partial}{\partial y}\left[\nu\left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial}\right)\right] = \frac{1}{dy}\left[\nu\left(\frac{W_{i,j+1,k} - W_{i,j,k}}{dy} + \frac{V_{i,j,k+1} - V_{i,j,k}}{dz}\right) - \nu\left(\frac{W_{i,j,k} - W_{i,j-1,k}}{dy} + \frac{V_{i,j-1,k+1} - V_{i,j-1,k}}{dz}\right)\right] \tag{30}$$

$$\frac{\partial}{\partial z}\left[2\nu\frac{\partial w}{\partial z}\right] = \frac{1}{dz}\left[2\nu\left(\frac{W_{i,j,k+1} - W_{i,j,k}}{dz}\right) - 2\nu\left(\frac{W_{i,j,k} - W_{i-1,j,k}}{dz}\right)\right] \tag{31}$$

The discretisation of all the terms of the Navier-Stokes equations is second order accurate in space. The discretisation of the subgrid model is given in section 6.

4 Time integration

In this section we will deal with the time integration. In the literature about Computational Fluid Dynamics several explicit and implicit schemes are used. The maximum allowed time step dt is for implicit methods in general much larger than for explicit methods. However, we think that for large 3D applications like LES/DNS implicit schemes are too expensive, and that explicit schemes even with a small time step are cheaper and have a better accuracy. Here we will use a second-order explicit Adams-Bashfort scheme which can easily be derived from Taylor expansions. Consider the following two-level scheme for the equation $y' = f$,

$$y^{n+1} - y^n = h(af^n + bf^{n-1}) \tag{32}$$

Taylor expansions give

$$\begin{aligned} y^{n+1} &= y(x+h) = y(x) + hf(x) + \frac{h^2}{2}f'(x) + O(h^3) \\ y^n &= y(x) \\ af^n &= af(x) \\ bf^{n-1} &= bf(x-h) = bf(x) - bhf'(x) + b\frac{h^2}{2}f''(x) + O(h^3) \end{aligned} \tag{33}$$

Substituting the expansions given by equation (33) in equation (32) and collecting terms gives

$$hf(x) + \frac{h^2}{2}f'(x) = h(a+b)f(x) - h^2bf'(x) + O(h^3) \tag{34}$$

The method is second order accurate for $(y^{n+1} - y^n)/h$ when $a + b = 1$ and $b = -0.5$ which gives the second order Adams-Bashfort scheme. For the Navier-Stokes equation we use this method for the time derivatives of the velocity, but we can not use it for the pressure because no time derivative exists. A possible way to circumvent problems with the integration of the pressure is as follows, integrate the Navier-Stokes equations in time without the pressure gradient terms, and use the continuity equation to find an equation for the pressure. In formula we should solve

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t[1.5(-ADV + DIFF)^n - 0.5(-ADV + DIFF)^{n-1} - \nabla P^{n+1}], \tag{35}$$

note that the pressure is evaluated at $n + 1$. We can write equation (35) as

$$\mathbf{u}^{n+1} + \Delta t\nabla P^{n+1} = \mathbf{u}^* = \mathbf{u}^n + \Delta t[1.5(-ADV + DIFF)^n - 0.5(-ADV + DIFF)^{n-1}], \tag{36}$$

Taking the divergence of the equation above gives:

$$\Delta t\nabla^2 P^{n+1} = div(\mathbf{u}^*) \tag{37}$$

where we have forced the divergence at $n+1$ to zero. The discretisation of this equation is very simple

$$\frac{P_{i+1,j,k}-2P_{i,j,k}+P_{i-1,j,k}}{dx^2}+\frac{P_{i,j+1,k}-2P_{i,j,k}+P_{i,j-1,k}}{dy^2}+\frac{P_{i,j,k+1}-2P_{i,j,k}+P_{i,j,k-1}}{dz^2}= \frac{1}{dt}\left(\frac{u^*_{i,j,k}-u^*_{i-1,j,k}}{dx}+\frac{v^*_{i,j,k}-v^*_{i,j-1,k}}{dy}+\frac{w^*_{i,j,k}-w^*_{i,j,k-1}}{dz}\right) \tag{38}$$

We solve the discrete Poisson equation (38) with a direct solver (**pois3d**) from the FISHPAK package. When the solution P is obtained we can calculate $\mathbf{u}^{n+1}$ from:

$$\mathbf{u}^{n+1}=\mathbf{u}^*-\Delta t\nabla P^{n+1}. \tag{39}$$

or

$$\begin{aligned} U_{i,j,k} &= U^*_{i,j,k}-dt\left(\frac{P_{i+1,j,k}-P_{i,j,k}}{dx}\right)\\ V_{i,j,k} &= V^*_{i,j,k}-dt\left(\frac{P_{i,j+1,k}-P_{i,j,k}}{dy}\right)\\ W_{i,j,k} &= W^*_{i,j,k}-dt\left(\frac{P_{i,j,k+1}-P_{i,j,k}}{dz}\right) \end{aligned} \tag{40}$$

5 No-slip and Free-slip boundary conditions

In this section we deal with boundary conditions at solid walls and free slip boundaries. At solid walls no-slip or free-slip boundary conditions can be implemented in the following way. No-slip conditions:

$$W_{i,j,0}=0. \tag{41}$$

$$V_{i,j,0}=-V_{i,j,1} \tag{42}$$

$$U_{i,j,0}=-U_{i,j,1} \tag{43}$$

Free-slip conditions:

$$W_{i,j,0}=0. \tag{44}$$

$$V_{i,j,0}=V_{i,j,1} \tag{45}$$

$$U_{i,j,0}=U_{i,j,1} \tag{46}$$

5.1 *Boundary* conditions for the pressure

Finally we will say something about the boundary conditions for the pressure field. Theoretically we may not specify boundary conditions for the pressure because they are already fixed by the velocity boundary conditions. However, the FISHPAK solver needs boundary conditions for the pressure. These boundary conditions have to be chosen in such a way that the don't alter the velocity boundary conditions and therefore Neumann boundary conditions **must** be used for the pressure.

5.2 Periodic Boundary Conditions

A special type of boundary conditions which are frequently used in LES/DNS are the so-called periodic boundary conditions. The implementation of these boundary conditions is extremely simple, for instance in the y-direction

$$U_{i,jmax+1,k}=U_{i,1,k} \tag{47}$$

$$V_{i,jmax+1,k}=V_{i,1,k} \tag{48}$$

$$W_{i,jmax+1,k}=W_{i,1,k} \tag{49}$$

$$U_{i,0,k}=U_{i,jmax,k} \tag{50}$$

$$V_{i,0,k}=V_{i,jmax,k} \tag{51}$$

$$W_{i,0,k}=W_{i,jmax,k} \tag{52}$$

$$\tag{53}$$

In this special case the pressure boundary conditions should also be periodic.

6 Subgrid model

In this section we will discretize the Smagorinsky subgrid model. The eddy viscosity will be calculated at the pressure point. The eddy viscosity ν_t is given by the following relation

$$\nu_t = (C_s \Delta)^2 \sqrt{\frac{1}{2} S_{ij} S_{ij}} \tag{54}$$

where $\Delta = (dx \cdot dy \cdot dz)^{1/3}$

The discretisation of the rate of strain tensor $S_{ij}S_{ij}$ around the pressure point reads:

$$\begin{aligned}
S_{ij}S_{ij} = {} & 4\left(\frac{U_{i,j,k} - U_{i-1,j,k}}{dx}\right)^2 + 4\left(\frac{V_{i,j,k} - V_{i,j-1,k}}{dy}\right)^2 + 4\left(\frac{W_{i,j,k} - W_{i,j,k-1}}{dz}\right)^2 + \\
& 0.5\left[\left(\frac{U_{i,j+1,k} - U_{i,j,k}}{dy} + \frac{V_{i+1,j,k} - V_{i,j,k}}{dx}\right)^2 + \left(\frac{U_{i,j,k} - U_{i,j-1,k}}{dy} + \frac{V_{i+1,j-1,k} - V_{i,j-1,k}}{dx}\right)^2 + \right. \\
& \left. \left(\frac{U_{i-1,j+1,k} - U_{i-1,j,k}}{dy} + \frac{V_{i,j,k} - V_{i-1,j,k}}{dx}\right)^2 + \left(\frac{U_{i-1,j,k} - U_{i-1,j-1,k}}{dy} + \frac{V_{i,j-1,k} - V_{i-1,j-1,k}}{dx}\right)^2\right] + \\
& 0.5\left[\left(\frac{U_{i,j,k+1} - U_{i,j,k}}{dz} + \frac{W_{i+1,j,k} - W_{i,j,k}}{dx}\right)^2 + \left(\frac{U_{i,j,k} - U_{i,j,k-1}}{dz} + \frac{W_{i+1,j,k-1} - W_{i,j,k-1}}{dx}\right)^2 + \right. \\
& \left. \left(\frac{U_{i-1,j,k+1} - U_{i-1,j,k}}{dz} + \frac{W_{i,j,k} - W_{i-1,j,k}}{dx}\right)^2 + \left(\frac{U_{i-1,j,k} - Ui-1,j,k-1}{dz} + \frac{W_{i,j,k-1} - W_{i-1,j,k-1}}{dx}\right)^2\right] + \\
& 0.5\left[\left(\frac{V_{i,j,k+1} - V_{i,j,k}}{dz} + \frac{W_{i,j+1,k} - W_{i,j,k}}{dy}\right)^2 + \left(\frac{V_{i,j,k} - V_{i,j,k-1}}{dz} + \frac{W_{i,j+1,k-1} - W_{i,j,k-1}}{dy}\right)^2 + \right. \\
& \left. \left(\frac{V_{i,j-1,k+1} - V_{i,j-1,k}}{dx} + \frac{W_{i,j,k} - W_{i,j-1,k}}{dy}\right)^2 + \left(\frac{V_{i,j-1,k} - V_{i,j-1,k-1}}{dx} + \frac{W_{i,j,k-1} - W_{i,j-1,k-1}}{dy}\right)^2\right]
\end{aligned} \tag{55}$$

The viscosity can now be written as

$$\nu = \nu_{molc} + (C_s \Delta)^2 \sqrt{\frac{1}{2} S_{ij} S_{ij}} \tag{56}$$

Note that the viscosity is calculated at the pressure point while most of the viscous terms are evaluated at the cell sides, and a spatial interpolation of the viscosity is necessary (see program).

Figure 2: Vorticity patterns in a temporal mixing layer at T=0.02

Figure 3: Vorticity patterns in a temporal mixing layer at T=0.04

7 Examples

In this section we will give two simple examples.

7.1 Example 1

First we will calculate a temporal mixing layer (see Lesieur *et. al.* J. Fluid Mech. **192**). Consider a plane Mixing Layer in the x-z-plane in a box $0 < y, z < 1$ and $0 < x < 2$. At $T = 0$ the velocity field is given by the following relation

$$u(z) = U \tanh\left[\frac{2(z-0.5)}{\delta}\right] \tag{57}$$

where $\delta = 1/28$ for a so-called 4-eddy simulation of the temporal mixing layer. Upon $u(z)$ a perturbation is superposed. This perturbation is defined by a stream function

$$\psi = 0.1U \exp[-(z/\delta)^2] \exp[i\alpha x], \tag{58}$$

where α is the most amplified mode, which can be found from a stability analysis. Such an analysis gives a value

$$\alpha = 0.44/\delta \tag{59}$$

The initial conditions given above are implemented in the subroutine **init**, free slip boundary conditions are used at $z = 0$ and $z = 1$ and periodic boundary conditions in the x and y direction. In the following Figures we have plotted the vorticity at $T = 0.02, 0.04, 0.06, 0.08, 0.1, 0.12$, using a grid of 180x20x80 point (x,y,z) ($Re = 10000$, $U = 28$ and $C_s = 0.1$)

7.2 Example 2

Here we will give an example of a flow which is frequently encountered in lubrication theory. Consider two flat plates in the $x - y$-plane, with a distance of $\Delta z = 1$, between this plates a very viscous fluid (oil) is present. The lower plate is in rest and the upper plate is moving with a constant velocity $U = 10$ in the x-direction. Furthermore a constant pressure gradient $\frac{\partial P}{\partial x} = 1$ is present (see Figure 8). What is the steady state velocity at $z = 0.5$ for a fluid with a viscosity of 0.01

The problem can be solved in the following way: Take the boundary conditions for a solid wall at $z = 0$ and the following boundary conditions at $z = 1$ (moving wall)

$$\begin{aligned} W_{i,j,kmax} &= 0. \\ V_{i,j,kmax+1} &= -V_{i,j,kmax} \\ U_{i,j,kmax+1} &= 2 \times 10 - U_{i,j,kmax} \end{aligned} \tag{60}$$

The constant pressure gradient is incorporated in the diffusion term in the x direction, as following

$$DIFF_x = DIFF_x + 1 \tag{61}$$

In Figure 9 we have plotted the steady state solution, obtained with the program (dots) , and the analytic solution (lines).

Figure 4: Vorticity patterns in a temporal mixing layer at T=0.06

Figure 5: Vorticity patterns in a temporal mixing layer at T=0.08

Figure 6: Vorticity patterns in a temporal mixing layer at T=0.10

A Program

```
C     parameter list:
C     imax : number of gridpoints in the x-direction
C     jmax : number of gridpoints in the y-direction
C     kmax : number of gridpoints in the z-direction
C     lx   : length of the computational domain in the x-dir.
C     ly   : length of the computational domain in the y-dir.
C     lz   :    ...................................
C     Cs   : Smagorinsky constant
C     dxi=1/dx  := imax / lx
C     dyi=1/dy  := jmax / ly
C     dzi=1/dz  := kmax / lx
C     Rei   : 1 / Re  where Re is the Reynolds number
C     nstap : Number of timesteps
C     Uold(0:imax+1,0:jmax+1,0:kmax+1) velocity in
C                  x-direction at the oldest timestep
C     Vold(0:imax+1,0:jmax+1,0:kmax+1) velocity in
C                  y-direction at the oldest timestep
C     Wold(0:imax+1,0:jmax+1,0:kmax+1) velocity in
C                  z-direction at the oldest timestep
C     Unew(.....)  velocity at the present timestep in x-direction
C     Vnew(.....)  velocity at the present timestep in y-direction
C     Wnew(.....)  velocity at the present timestep in z-direc.
C
C     dUdt(.....),dVdt(...),dWdt(....) predicted velocity at
C     the new timestep
C
C     P(imax,jmax,kmax) : Pressure array
C     visc(0:imax+1,0:jmax+1,0:kmax+1)  : viscosity array used
C                                          in LES.
C*****************************************************************
C*****************************************************************
C*****************************************************************
C*****************************************************************
C  Subroutines ::::
C  subroutine init    : set U,V,W to some initial value
C  subroutine chkdt   : set timestep   dt
C  subrouitne chkdiv  : checkes the divergence
C  subroutine momx    : integrates the momentum equation in x-dir
C  subroutine momy    : integrates the momentum equation in y-dir
C  subroutine momz    : integrates the momentum equation in z-dir
C  subroutine adams   : integrates the momentum equation in time
C  subroutine fillps  : fills the right hand side for the Poisson solver
C  subroutine correc  : corrects the velocity such that div(u,v,w)==0
C  subroutine bound   : set boundary conditions
C  subroutine solver  : Calls a Fishpak routine to solve Poisson equation
C  subroutine submod  : subgrid model
C**********************************************************************
C
C     Describtion of the model.
C
C     In this model the Navier-Stokes equations are integrated
C     in space with a Finite Volume Technique on a staggerd grid.
C
C                         W          V
C                         ^         ^
C                         |        /
C                     ----|-----------
C                   /|    |    /      /
```

Figure 7: Vorticity patterns in a temporal mixing layer at T=0.12

Figure 8: Geometry used in Example 2

Figure 9: Poisseule/Couette flow, the dots denote the numerical simulation and the line the analytic solution

```
C                  /  |    |  /      / |
C                /    |     /      /   |
C              /      |         /      |
C             ------------*-----   -------> U
C             |       |________|______|
C          dz |     /          |      /
C             |   /            |    /
C             | /              |  / dy
C             |                |/
C             ------------------
C                   dx
C
C     U : velocity in x-direction    [ U(i,j,k) ]
C     V : velocity in y-direction    [ V(i,j,k) ]
C     W : velocity in z-direction    [ W(i,j,k) ]
C     * : Pressure point
C     dx,dy,dz  dimensions of the grid cell
C
C     For instance:
C
C     d u u    (U(i,j,k)+U(i+1,j,k))**2 - (U(i,j,k)+U(i-1,j,k))**2
C     ----- =  ---------------------------------------------------
C      d x                       4 dx
C
C       etc.
C*********************************************************************
C
C
C        ADV  = ADVECTION
C        DIFF = DIFFUSION
C        PRES = PRESSURE (GRADIENT)
C
C
C      Time integration with Adams-Bashfort / pressure correction
C
C
C       dUdt = 1.5(-ADV+DIFF)_new -0.5(-ADV+DIFF)_old
C       div grad (P) =div (dudt,dvdt,dwdt)
C       Unew = dUdt - dt grad(P)
C*********************************************************************
      implicit none
      include 'param.txt'
      include 'common.txt'
      real    lx,ly,lz,Re
      integer istap,nstap
      real    time,utime
      read(5,*) Ufree
      read(5,*) cs
      read(5,*) lx
      read(5,*) ly
      read(5,*) lz
      read(5,*) Re
      read(5,*) nstap
********
      dxi = imax / lx
      dyi = jmax / ly
      dzi = kmax / lz
      Rei = 1.   / Re
      call init
      call loadd(0)
      call bound(Uold,Vold,Wold)
      call bound(Unew,Vnew,Wnew)
      call chkdt
*******
```

```
      time  =0.
      utime =0.
C  main loop starts here

      do istap = 1,nstap
      time = time + dt
      utime=utime + dt
      write(6,*) time,dt
      call submod
      call adamsb('adams')
      call bound   (Unew,Vnew,Wnew)
      call bound   (dUdt,dVdt,dWdt)
      call fillps  (istap)
      call solver  (p,dxi,dyi,dzi)
      call correc
      call bound   (Uold,Vold,Wold)
      call bound   (Unew,Vnew,Wnew)
      if (mod(istap,20).eq.0) then
      call chkdt
      call chkdiv
      endif
      if ( utime .gt. 0.02) then
      utime = 0.
      call avs
      call loadd(1)
      stop
      endif
      enddo
      stop
      end

      subroutine adamsb(string)
C
C
C  Integrate in time with second order Adams-Bashfort (or Euler forward)
C  dold(...),dnew(....) are dummy arrays
C
      implicit none
      include 'param.txt'
      include 'common.txt'
      real   dold(0:i1,0:j1,0:k1),dnew(0:i1,0:j1,0:k1)
      real   cof1,cof2
      character*5 string
*     Adams-Bashfort  cof1=1.5  , cof2 = -0.5
*     Euler-Forward   cof1=  1  , cof2 =    0
      if (string .eq. 'euler') then
      cof1=1.
      cof2=0.
      endif
      if (string .eq. 'adams') then
      cof1= 1.5
      cof2=-0.5
      endif
c
c   integrate momentum equations at old and new timelevel
c
      call momx(dold,uold,vold,wold,dxi,dyi,dzi,
     &          1,imax,1,jmax,1,kmax,i1,j1,k1,visc)
      call momx(dnew,unew,vnew,wnew,dxi,dyi,dzi,
     &          1,imax,1,jmax,1,kmax,i1,j1,k1,visc)
c
c   calculate predicted velocity
c
      do k=1,kmax
        do j=1,jmax
          do i=1,imax
          dudt(i,j,k)=Unew(i,j,k) +
     1    dt * ( cof1 * dnew(i,j,k) + cof2 * dold(i,j,k) )
          enddo
        enddo
      enddo
      call momy(dold,uold,vold,wold,dxi,dyi,dzi,
     &          1,imax,1,jmax,1,kmax,i1,j1,k1,visc)
      call momy(dnew,unew,vnew,wnew,dxi,dyi,dzi,
     &          1,imax,1,jmax,1,kmax,i1,j1,k1,visc)

      do k=1,kmax
        do j=1,jmax
          do i=1,imax
          dvdt(i,j,k)=Vnew(i,j,k)+
     1    dt * ( cof1 * dnew(i,j,k) + cof2 * dold(i,j,k) )
          enddo
        enddo
      enddo
      call momz(dold,uold,vold,wold,dxi,dyi,dzi,
     &          1,imax,1,jmax,1,kmax,i1,j1,k1,visc)
      call momz(dnew,unew,vnew,wnew,dxi,dyi,dzi,
     &          1,imax,1,jmax,1,kmax,i1,j1,k1,visc)
      do k=1,kmax
        do j=1,jmax
          do i=1,imax
          dwdt(i,j,k)=Wnew(i,j,k)+
     1    dt * ( cof1 * dnew(i,j,k) + cof2 * dold(i,j,k) )
          enddo
        enddo
      enddo
      return
      end

      subroutine fillps
      implicit none
      include 'param.txt'
      include 'common.txt'
      real   dti
      dti =1./dt
c
c
c     fill right hand side of the poisson equation
c
c     discrete divergence is
c   w(i,j,k)-w(i,j,k-1)  v(i,j,k)-v(i,j-1,k)  u(i,j,k)-u(i-1,j,k)
c   -------------------  -------------------  -------------------  = div
c          d z                  d y                  d x
c
c  note that p is not yet the pressure
c  but only the rhs of the poisson equation
c
      do k=1,kmax
       do j=1,jmax
  do i=1,imax
      p(i,j,k)= dti*(
     1   ( dWdt(i,j,k)-dWdt(i,j,k-1))*dzi+
     2   ( dVdt(i,j,k)-dVdt(i,j-1,k))*dyi+
     3   ( dUdt(i,j,k)-dUdt(i-1,j,k))*dxi
     e              )
          enddo
       enddo
      enddo
      return
      end


      subroutine correc
c
c
c   Corrects the velocity in such a way that div(u,v,w) is exactly zero
c
c
      implicit none
      include 'param.txt'
      include 'common.txt'
      do k=1,kmax
      do j=1,jmax
      do i=1,imax-1
       dudt(i,j,k)=
     1  dUdt(i,j,k) - dt * dxi * ( P(i+1,j,k)-P(i,j,k) )
      enddo
      enddo
      enddo
      do k=1,kmax
      do j=1,jmax
      dudt(imax,j,k)=
     1  dudt(imax,j,k) -dt*  dxi *(P(1,j,k)-P(imax,j,k))
      enddo
      enddo

      do k=1,kmax
      do j=1,jmax-1
      do i=1,imax
       dvdt(i,j,k)=
     1  dvdt(i,j,k) - dt*dyi * ( P(i,j+1,k)-P(i,j,k) )
      enddo
      enddo
      enddo
      do k=1,kmax
      do i=1,imax
      dvdt(i,jmax,k)=
     1  dvdt(i,jmax,k)-dt*dyi*(P(i,1,k)-P(i,jmax,k))
      enddo
      enddo
      do k=1,kmax-1
      do j=1,jmax
      do i=1,imax
       dwdt(i,j,k)=
     1 dwdt(i,j,k) - dt*dzi * ( P(i,j,k+1)-P(i,j,k) )
      enddo
      enddo
      enddo
c
c
c    Update the velocity fields
c
c
      do k=1,kmax
         do j=1,jmax
            do i=1,imax
            Uold(i,j,k)=Unew(i,j,k)
            Unew(i,j,k)=dUdt(i,j,k)
            Vold(i,j,k)=Vnew(i,j,k)
            Vnew(i,j,k)=dVdt(i,j,k)
            Wold(i,j,k)=Wnew(i,j,k)
            Wnew(i,j,k)=dWdt(i,j,k)
            enddo
        enddo
      enddo
      return
      end

      subroutine avs
c
c     output routine for AVS
c
      include 'param.txt'
      include 'common.txt'
      real vor(imax,kmax)
      open(20,file='avsU')
      open(21,file='avsV')
      open(22,file='avsW')
      open(23,file='avsP')
      open(30,file='vort')
      j=2
      do i=1,imax
        do k=1,kmax
```

```
write(20,*) Unew(i,j,k)
        write(21,*) Vnew(i,j,k)
        write(22,*) Wnew(i,j,k)
        write(23,*) p   (i,j,k)
        vor(i,k)= (Unew(i,j,k+1)-Unew(i,j,k))*dzi  -
     ^              (Wnew(i+1,j,k)-Wnew(i,j,k))*dxi
        write(30,*) vor(i,k)
        enddo
      enddo

      close(30)
      close(20)
      close(21)
      close(22)
      close(23)
c
c output for Gnuplot
c
      open (24,file='congnu')
      do k=1,kmax
do i=1,imax
  write(24,111) 1.*i/dxi,1.*k/dzi,vor(i,k)
        enddo
      enddo
111   FORMAT(3F12.6)
      close(24)
      Return
      end
c
c
      subroutine chkdiv
c
c checks the discrete divergence (should be of order of machine prec.)
c
      implicit none
      include 'param.txt'
      include 'common.txt'
      real div,divmax,divtot
      divmax = -9999.9999
      divtot = 0.
      do k=1,kmax
         do j=1,jmax
            do i=1,imax
            div = (
     1   ( Wnew(i,j,k)-Wnew(i,j,k-1) ) * dzi +
     2   ( Vnew(i,j,k)-Vnew(i,j-1,k) ) * dyi +
     3   ( Unew(i,j,k)-Unew(i-1,j,k) ) * dxi )
           divtot = divtot + div
           divmax = max ( div,divmax)
c      write(6,*) i,j,k,div
           enddo
         enddo
      enddo
      write(6,111) divtot,divmax
111   FORMAT('Mass loss/gain : Tot =',e13.6,' Max = ',e13.6)
      if (divtot .gt. 10e-3) stop 'Fatal Error **DIVERGENCE TO LARGE**'
      return
      end

      subroutine chkdt
c
c     Calculates the timestep dt.
c
      implicit none
      include 'param.txt'
      include 'common.txt'
      real    Courant,dtemp,dtmax
      Courant  = .35
      dtmax = -9999.9999
      do k=1,kmax
         do j=1,jmax
```

```
            do i=1,imax
            dtemp =
     1      abs(Unew(i,j,k) * dxi) +
     2      abs(Vnew(i,j,k) * dyi) +
     3      abs(Wnew(i,j,k) * dzi) +
     4       visc(i,j,k) * (dxi*dxi + dyi*dyi + dzi*dzi)
            dtmax = max ( dtmax , dtemp )
            enddo
         enddo
      enddo
      dt = Courant / dtmax
      return
      end

      subroutine bound(U,V,W)
      include 'param.txt'
      include 'common.txt'
      real U(0:i1,0:j1,0:k1),V(0:i1,0:j1,0:k1),
     $     W(0:i1,0:j1,0:k1)
c
c   boundary conditions at z=zmin and z =zmax
c
      do i=0,i1
        do j=0,j1
         W(i,j,0    ) =      0.
         W(i,j,kmax ) =      0.
         V(i,j,0    ) =  V(i,j,1   )
         V(i,j,k1   ) =  V(i,j,kmax)
         U(i,j,0    ) =  U(i,j,1   )
         U(i,j,k1   ) =  U(i,j,kmax)
        enddo
      enddo
c
c  periodic boundary conditions in the x direction
c
      do k=0,k1
        do j=0,j1
          U(0 ,j,k)=U(imax,j,k)
          V(0 ,j,k)=V(imax,j,k)
          W(0 ,j,k)=W(imax,j,k)
          U(i1,j,k)=U(1   ,j,k)
          V(i1,j,k)=V(1   ,j,k)
          W(i1,j,k)=W(1   ,j,k)
         enddo
       enddo
c
c periodic boundary conditions in the y-direction
c
       do k=0,k1
         do i=0,i1
           U(i,0,k )=U(i,jmax,k)
           V(i,0,k )=V(i,jmax,k)
           W(i,0,k )=W(i,jmax,k)
           U(i,j1,k)=U(i,1   ,k)
           V(i,j1,k)=V(i,1   ,k)
           W(i,j1,k)=W(i,1   ,k)
         enddo
       enddo
       return
       end


      subroutine momx(dudt,u,v,w,dxi,dyi,dzi,ib,ie,jb,je,kb,ke,i1,j1,k1,
     $                visc)
      implicit none
      integer ib,ie,jb,je,kb,ke,i1,j1,k1
      integer ip,im,jp,jm,kp,km,i,j,k
      real dudt(0:i1,0:j1,0:k1),
     $     u   (0:i1,0:j1,0:k1),
     $     v   (0:i1,0:j1,0:k1),
     $     w   (0:i1,0:j1,0:k1),
     $    dxi,dyi,dzi,visc(0:i1,0:j1,0:k1)
      real uuip ,uuim ,uvjp ,uvjm ,uwkp ,uwkm,vzp,vzm,vyp,vym
      real du1ip,du1im,du1jp,du1jm,du1kp,du1km
      do k=kb,ke
        do j=jb,je
          do i=ib,ie
*
          ip = i + 1
          jp = j + 1
          kp = k + 1
          im = i - 1
          jm = j - 1
          km = k - 1
          uuip  = 0.25 * ( U(ip,j,k)+U(i,j,k) )*( U(ip,j,k)+U(i,j,k)  )
          uuim  = 0.25 * ( U(im,j,k)+U(i,j,k) )*( U(im,j,k)+U(i,j,k)  )
          uvjp  = 0.25 * ( U(i,jp,k)+U(i,j,k) )*( V(ip,j,k)+V(i,j,k)  )
          uvjm  = 0.25 * ( U(i,jm,k)+U(i,j,k) )*( V(ip,jm,k)+V(i,jm,k))
          uwkp  = 0.25 * ( U(i,j,kp)+U(i,j,k) )*( W(ip,j,k) +W(i,j,k) )
          uwkm  = 0.25 * ( U(i,j,km)+U(i,j,k) )*( W(ip,j,km)+W(i,j,km))
          du1ip = 2.*( dxi * (U(ip,j,k)-U(i ,j,k) ))
          du1im = 2.*( dxi * (U(i ,j,k)-U(im,j,k) ))
          du1jp = dyi*(U(i,jp,k)-U(i, j,k))+dxi*(V(ip,j, k)-V(i ,j ,k))
          du1jm = dyi*(U(i,j ,k)-U(i,jm,k))+dxi*(V(ip,jm,k)-V(i ,jm,k))
          du1kp = dzi*(U(i,j,kp)-U(i ,j,k))+dxi*(W(ip,j, k)-W(i ,j ,k))
          du1km = dzi*(U(i,j ,k)-U(i,j,km))+dxi*(W(ip,j,km)-W(i ,j,km))
      vzp =
     &   0.25*(visc(i,j,k)+visc(ip,j,k)+visc(ip,j,kp)+visc(i,j,kp))
      vzm =
     &   0.25*(visc(i,j,k)+visc(ip,j,k)+visc(ip,j,km)+visc(i,j,km))
      vyp =
     &   0.25*(visc(i,j,k)+visc(ip,j,k)+visc(ip,jp,k)+visc(i,jp,k))
      vym =
     &   0.25*(visc(i,j,k)+visc(ip,j,k)+visc(ip,jm,k)+visc(i,jm,k))

          dudt(i,j,k) =
     1   dxi*( -uuip + visc(ip,j,k)*du1ip+uuim - visc(i,j,k)*du1im )+
     2   dyi*( -uvjp + vyp*du1jp+uvjm - vym*du1jm )+
     3   dzi*( -uwkp + vzp*du1kp+uwkm - vzm*du1km )
            enddo
         enddo
      enddo
      return
      end

      subroutinemomy(dvdt,u,v,w,dxi,dyi,dzi,ib,ie,jb,je,kb,ke,i1,j1,k1,
     $                visc)
      implicit none
      integer ib,ie,jb,je,kb,ke,i1,j1,k1
      integer im,ip,jm,jp,km,kp,i,j,k
      real dvdt(0:i1,0:j1,0:k1),
     $     u   (0:i1,0:j1,0:k1),
     $     v   (0:i1,0:j1,0:k1),
     $     w   (0:i1,0:j1,0:k1),
     $    dxi,dyi,dzi,visc(0:i1,0:j1,0:k1)
      real uvip ,uvim ,vvjp ,vvjm ,wvkp ,wvkm
      real dv1ip,dv1im,dv1jp,dv1jm,dv1kp,dv1km
      real vxp,vxm,vzp,vzm
      do k=kb,ke
        do j=jb,je
          do i=ib,ie
*
          ip = i + 1
          jp = j + 1
          kp = k + 1
          im = i - 1
          jm = j - 1
          km = k - 1
          uvip  = 0.25 * (U(i ,j,k)+U(i ,jp,k))*( V(i,j,k)+V(ip,j,k) )
          uvim  = 0.25 * (U(im,j,k)+U(im,jp,k))*( V(i,j,k)+V(im,j,k) )
          vvjp  = 0.25 * (V(i,j,k )+V(i,jp,k) )*( V(i,j,k)+V(i,jp,k) )
          vvjm  = 0.25 * (V(i,j,k )+V(i,jm,k) )*( V(i,j,k)+V(i,jm,k) )
          wvkp  = 0.25 * (W(i,j,k )+W(i,jp,k) )*( V(i,j,kp)+V(i,j,k) )
          wvkm  = 0.25 * (W(i,j,km)+W(i,jp,km))*( V(i,j,km)+V(i,j,k) )

          dv1ip = dxi*(V(ip,j,k)-V(i ,j,k))+dyi*(U(i,jp ,k)-U(i ,j,k))
          dv1im = dxi*(V(i ,j,k)-V(im,j,k))+dyi*(U(im,jp,k)-U(im,j,k))
          dv1jp = 2.*dyi*(V(i,jp,k)-V(i,j ,k))
          dv1jm = 2.*dyi*(V(i,j ,k)-V(i,jm,k))
          dv1kp = dzi*(V(i,j,kp)-V(i, j,k))+dyi*(W(i,jp,k )-W(i ,j,k))
          dv1km = dzi*(V(i,j,k )-V(i,j,km))+dyi*(W(i,jp,km)-W(i,j,km))
      vxp =
     & 0.25*(visc(i,j,k)+visc(ip,j,k)+visc(ip,jp,k)+visc(i,jp,k))
      vxm =
     & 0.25*(visc(i,j,k)+visc(im,j,k)+visc(im,jp,k)+visc(i,jp,k))
      vzp =
     & 0.25*(visc(i,j,k)+visc(i,jp,k)+visc(i,jp,kp)+visc(i,j,kp))
      vzm =
     & 0.25*(visc(i,j,k)+visc(i,jp,k)+visc(i,jp,km)+visc(i,j,km))

          dvdt(i,j,k)=
     1 dxi*((-uvip + vxp         *dv1ip)-(-uvim + vxm        *dv1im))+
     2 dyi*((-vvjp + visc(i,jp,k)*dv1jp)-(-vvjm + visc(i,j,k)*dv1jm))+
     3 dzi*((-wvkp + vzp         *dv1kp)-(-wvkm + vzm        *dv1km))
          enddo
        enddo
      enddo
      return
      end
      subroutinemomz(dwdt,u,v,w,dxi,dyi,dzi,ib,ie,jb,je,kb,ke,i1,j1,k1,
     $                visc)
      implicit none
      integer ib,ie,jb,je,kb,ke,i1,j1,k1
      integer im,jm,km,ip,jp,kp,i,j,k
      real dwdt(0:i1,0:j1,0:k1),
     $     u   (0:i1,0:j1,0:k1),
     $     v   (0:i1,0:j1,0:k1),
     $     w   (0:i1,0:j1,0:k1),
     $    dxi,dyi,dzi,visc(0:i1,0:j1,0:k1)
      real uwip ,uwim ,vwjp ,vwjm ,wwkp ,wwkm
      real dw1ip,dw1im,dw1jp,dw1jm,dw1kp,dw1km
      real  vxm,vym,vxp,vyp
      do k=kb,ke
        do j=jb,je
          do i=ib,ie
*
          ip = i + 1
          jp = j + 1
          kp = k + 1
          im = i - 1
          jm = j - 1
          km = k - 1
          uwip  = 0.25 * ( W(i,j,k)+W(ip,j,k))*(U(i ,j,k)+U(i ,j,kp) )
          uwim  = 0.25 * ( W(i,j,k)+W(im,j,k))*(U(im,j,k)+U(im,j,kp) )
          vwjp  = 0.25 * ( W(i,j,k)+W(i,jp,k))*(V(i ,j,k)+V(i,j ,kp) )
          vwjm  = 0.25 * ( W(i,j,k)+W(i,jm,k))*(V(i,jm,k)+V(i,jm,kp) )
          wwkp  = 0.25 * ( W(i,j,k)+W(i,j,kp))*(W(i ,j,k)+W(i,j,kp ) )
          wwkm  = 0.25 * ( W(i,j,k)+W(i,j,km))*(W(i ,j,k)+W(i,j,km ) )

          dw1ip = dxi*(W(ip,j,k)-W(i ,j,k))+dzi*(U(i ,j,kp)-U(i ,j,k))
          dw1im = dxi*(W(i ,j,k)-W(im,j,k))+dzi*(U(im,j,kp)-U(im,j,k))
          dw1jp = dyi*(W(i,jp,k)-W(i, j,k))+dzi*(V(i ,j,kp)-V(i ,j,k))
          dw1jm = dyi*(W(i,j ,k)-W(i,jm,k))+dzi*(V(i,jm,kp)-V(i,jm,k))
          dw1kp = 2*dzi*(W(i,j,kp)-W(i,j ,k))
          dw1km = 2*dzi*(W(i,j,k )-W(i,j,km))
          vxp =
     & 0.25*(visc(i,j,k)+visc(ip,j,k)+visc(ip,j,kp)+visc(i,j,kp))
          vxm =
     & 0.25*(visc(i,j,k)+visc(im,j,k)+visc(im,j,kp)+visc(i,j,kp))
          vyp =
     & 0.25*(visc(i,j,k)+visc(i,jp,k)+visc(i,jp,kp)+visc(i,j,kp))
          vym =
     & 0.25*(visc(i,j,k)+visc(i,jm,k)+visc(i,jm,kp)+visc(i,j,kp))
```

```
          dwdt(i,j,k) =
     1dxi*((-uwip + vxp         *dw1ip)-(-uwim + vxm        *dw1im))+
     2dyi*((-vwjp + vyp         *dw1jp)-(-vwjm + vym        *dw1jm))+
     3dzi*((-wwkp + visc(i,j,kp)*dw1kp)-(-wwkm + visc(i,j,k)*dw1km))
            enddo
         enddo
      enddo
      return
      end


      subroutine init
      include 'param.txt'
      include 'common.txt'
      real delta,z,x
      delta = 1/28.
      do k=1,kmax
         do j=1,jmax
           do i=1,imax
           Uold(i,j,k)=0.
           Vold(i,j,k)=0.
           Wold(i,j,k)=0.
           Unew(i,j,k)=Ufree*tanh(28.* ((1.*k/kmax) -.5  ))
           Uold(i,j,k)=Unew(i,j,k)
           Vnew(i,j,k)=0.
           Wnew(i,j,k)=0.
           dUdt(i,j,k)=0.
           dVdt(i,j,k)=0.
           dWdt(i,j,k)=0.
           enddo
         enddo
      enddo
      do k=1,kmax
        z= (1.*k/kmax)-0.5
do j=1,jmax
           do i=1,imax
           x = 2.*i/imax
     Unew(i,j,k)=Unew(i,j,k)+0.1*Ufree*(-z/delta)*
     &    exp( -(z/delta)**2)*cos(0.44/delta*x)
             Wnew(i,j,k)=Wnew(i,j,k)+0.1*Ufree*
     &    exp( -(z/delta)**2)*sin(0.44/delta*x)
           enddo
        enddo
      enddo

      return
      end
      subroutine loadd(in)
      implicit none
      include 'param.txt'
      include 'common.txt'
      integer reclengte,in
      reclengte=8
      if(in.eq.0) then
      open(15,file='start',access='direct',recl=6*(imax+2)*(jmax+2)*
     ^     reclengte)
      do 100 k=0,k1
      read(15,rec=(k+1))
     &  ( (Unew(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Vnew(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Wnew(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Uold(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Vold(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Wold(i,j,k) , j=0,j1) , i=0,i1)
100   continue
      close(15)
      endif
      if(in.eq.1) then
      open(25,file='start',access='direct',recl=6*(imax+2)*(jmax+2)*
     ^      reclengte)
      do 200 k=0,k1
      write(25,rec=(k+1))
     &  ( (Unew(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Vnew(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Wnew(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Uold(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Vold(i,j,k) , j=0,j1) , i=0,i1),
     &  ( (Wold(i,j,k) , j=0,j1) , i=0,i1)
200   continue
      close(25)
      endif
      return
      end

      subroutine submod
      include 'param.txt'
      include 'common.txt'
      real shear,delta
      delta = ((1./dxi)*(1./dyi)*(1./dzi))**(1./3.)
      shear =0.
      do k=1,kmax
do j=1,jmax
  do i=1,imax
  shear = 2.0*(
     1  (dxi*( Unew(i,j,k)-Unew(i-1,j,k) ) )**2 +
     1  (dyi*( Vnew(i,j,k)-Vnew(i,j-1,k) ) )**2 +
     1  (dzi*( Wnew(i,j,k)-Wnew(i,j,k-1) ) )**2
     a                  )

       shear = shear + 0.25*(
     1 (  (dyi*( Unew(i  ,j+1,k)-Unew(i  ,j  ,k))) +
     2    (dxi*( Vnew(i+1,j  ,k)-Vnew(i  ,j  ,k)))   )**2 +
     1 (  (dyi*( Unew(i  ,j  ,k)-Unew(i  ,j-1,k))) +
     2    (dxi*( Vnew(i+1,j-1,k)-Vnew(i  ,j-1,k)))   )**2 +
     1 (  (dyi*( Unew(i-1,j+1,k)-Unew(i-1,j  ,k))) +
     2    (dxi*( Vnew(i  ,j  ,k)-Vnew(i-1,j  ,k)))   )**2 +
     1 (  (dyi*( Unew(i-1,j  ,k)-Unew(i-1,j-1,k))) +
     2    (dxi*( Vnew(i  ,j-1,k)-Vnew(i-1,j-1,k)))   )**2
     e                      )

      shear = shear + 0.25*(
     1 (  (dzi*(Unew(i  ,j  ,k+1)-Unew(i  ,j  ,k  ))) +
     2    (dxi*(Wnew(i+1,j  ,k  )-Wnew(i  ,j  ,k  )))   )**2 +
     1 (  (dzi*(Unew(i  ,j  ,k  )-Unew(i  ,j  ,k-1))) +
     2    (dxi*(Wnew(i+1,j  ,k-1)-Wnew(i  ,j  ,k-1)))   )**2 +
     1 (  (dzi*(Unew(i-1,j  ,k+1)-Unew(i-1,j  ,k  ))) +
     2    (dxi*(Wnew(i  ,j  ,k  )-Wnew(i-1,j  ,k  )))   )**2 +
     1 (  (dzi*(Unew(i-1,j  ,k  )-Unew(i-1,j  ,k-1))) +
     2    (dxi*(Wnew(i  ,j  ,k-1)-Wnew(i-1,j  ,k-1)))   )**2
     e                      )

      shear = shear + 0.25*(
     1 (   (dzi*(Vnew(i  ,j  ,k+1)-Vnew(i  ,j  ,k  ))) +
     2     (dyi*(Wnew(i  ,j+1,k  )-Wnew(i  ,j  ,k  )))  )**2 +
     1 (   (dzi*(Vnew(i  ,j  ,k  )-Vnew(i  ,j  ,k-1))) +
     2     (dyi*(Wnew(i  ,j+1,k-1)-Wnew(i  ,j  ,k-1)))  )**2 +
     1 (   (dzi*(Vnew(i  ,j-1,k+1)-Vnew(i  ,j-1,k  ))) +
     2     (dyi*(Wnew(i  ,j  ,k  )-Wnew(i  ,j-1,k  )))  )**2 +
     1 (   (dzi*(Vnew(i  ,j-1,k  )-Vnew(i  ,j-1,k-1))) +
     2     (dyi*(Wnew(i  ,j  ,k-1)-Wnew(i  ,j-1,k-1)))  )**2
     e                      )
      visc(i,j,k) = (cs*delta)**2 * sqrt(shear)
          enddo
        enddo
      enddo
      do k=0,k1
        do j=0,j1
         visc(0 ,j,k)=visc(imax,j,k)
         visc(i1,j,k)=visc(1   ,j,k)
        enddo
      enddo
      do i=0,i1
```

```
         do k=0,k1
         visc(i,0 ,k)=visc(i,jmax,k)
         visc(i,j1,k)=visc(i,1   ,k)
         enddo
      enddo
      do j=0,j1
        do i=0,i1
         visc(i,j,k1)=visc(i,j,kmax)
         visc(i,j,0 )=visc(i,j,1   )
        enddo
      enddo
c
c  Add molecular viscosity
c
      do k=0,k1
         do j=0,j1
            do i=0,i1
             visc(i,j,k)=visc(i,j,k)+Rei
            enddo
         enddo
      enddo
      return
      end

      subroutine solver(p,dxi,dyi,dzi)
      implicit none
      include 'param.txt'
      integer  ier
      real     p(imax,jmax,kmax)
      real     dxi,dyi,dzi,dzi2,c1,c2,a(kmax),b(kmax),c(kmax)
      real     rhs(kmax,jmax,imax),d(kmax)
      real     save(k1*jmax*imax+3*kmax+3*jmax+4*imax+1000)
      real     w(i1*j1*k1)
      c1   = dxi*dxi
      c2   = dyi*dyi
      dzi2 = dzi*dzi
      do k=1,kmax
      a(k)=dzi2
      c(k)=dzi2
      b(k)=-(a(k)+c(k))
      d(k)=c2
      enddo
      b(1)=b(1)+a(1)
      a(1)=0.
      b(kmax)=b(kmax)+c(kmax)
      c(kmax)=0.
      callpois3d(0,imax,c1,0,jmax,c2,1,kmax,a,b,c,imax,jmax,p,ier,w)
      if (ier .ne. 0) write(6,*) ier
      If (ier .ne. 0) stop 'Fatal Error **Poisson Solver**'
      return
      end
```

B include files

(param.txt):

```
      integer imax,jmax,kmax,i,j,k
      integer i1  ,j1  ,k1
      parameter ( imax = 60)
      parameter ( jmax = 60)
      parameter ( kmax = 60)
      parameter ( i1 =imax+1)
      parameter ( j1 =jmax+1)
      parameter ( k1 =kmax+1)
```

(common.txt)

```
      realUold(0:i1,0:j1,0:k1),Vold(0:i1,0:j1,0:k1),Wold(0:i1,0:j1,0:k1)
      common /old/Uold,Vold,Wold
      save   /old/

      realUnew(0:i1,0:j1,0:k1),Vnew(0:i1,0:j1,0:k1),Wnew(0:i1,0:j1,0:k1)
      common /new/Unew,Vnew,Wnew
      save   /new/

      realdUdt(0:i1,0:j1,0:k1),dVdt(0:i1,0:j1,0:k1),dWdt(0:i1,0:j1,0:k1)
      common /deriv/dUdt,dVdt,dWdt
      save   /deriv/

      real  p(imax,jmax,kmax)
      common /pressure/p
      save   /pressure/

      real dxi,dyi,dzi,Rei,dt,Ufree,cs
      common /parm/dxi,dyi,dzi,Rei,dt,Ufree,cs
      save   /parm/

      real   visc(0:i1,0:j1,0:k1)
      common /viscosity/visc
      save   /viscosity/
```

C FISHPAK pois3d documentation

```
      SUBROUTINE POIS3D (LPEROD,L,C1,MPEROD,M,C2,NPEROD,N,A,B,C,LDIMF,
     1                   MDIMF,F,IERROR,W)
C
C
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *                                                               *
C     *                        F I S H P A K                          *
C     *                                                               *
C     *                                                               *
C     *     A PACKAGE OF FORTRAN SUBPROGRAMS FOR THE SOLUTION OF      *
C     *                                                               *
C     *      SEPARABLE ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS        *
C     *                                                               *
C     *                  (VERSION 3.1 , OCTOBER 1980)                  *
C     *                                                               *
C     *                             BY                                *
C     *                                                               *
C     *        JOHN ADAMS, PAUL SWARZTRAUBER AND ROLAND SWEET         *
C     *                                                               *
C     *                             OF                                *
C     *                                                               *
C     *         THE NATIONAL CENTER FOR ATMOSPHERIC RESEARCH          *
C     *                                                               *
C     *                BOULDER, COLORADO  (80307)  U.S.A.             *
C     *                                                               *
C     *                   WHICH IS SPONSORED BY                       *
C     *                                                               *
C     *              THE NATIONAL SCIENCE FOUNDATION                  *
C     *                                                               *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C
C    * * * * * * * * *  PURPOSE    * * * * * * * * * * * * * * * * * *
C
C     SUBROUTINE POIS3D SOLVES THE LINEAR SYSTEM OF EQUATIONS
C
C       C1*(X(I-1,J,K)-2.*X(I,J,K)+X(I+1,J,K))
C     + C2*(X(I,J-1,K)-2.*X(I,J,K)+X(I,J+1,K))
C     + A(K)*X(I,J,K-1)+B(K)*X(I,J,K)+C(K)*X(I,J,K+1) = F(I,J,K)
C
C     FOR  I=1,2,...,L , J=1,2,...,M , AND K=1,2,...,N .
C
C     THE INDICES K-1 AND K+1 ARE EVALUATED MODULO N, I.E.
C     X(I,J,0) = X(I,J,N) AND X(I,J,N+1) = X(I,J,1). THE UNKNOWNS
C     X(0,J,K), X(L+1,J,K), X(I,0,K), AND X(I,M+1,K) ARE ASSUMED TO TAKE
C     ON CERTAIN PRESCRIBED VALUES DESCRIBED BELOW.
```

```
C
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C
C    * * * * * * * *    PARAMETER DESCRIPTION     * * * * * * * * * *
C
C
C            * * * * * *   ON INPUT    * * * * * *
C
C     LPEROD   INDICATES THE VALUES THAT X(0,J,K) AND X(L+1,J,K) ARE
C              ASSUMED TO HAVE.
C
C              = 0  IF X(0,J,K) = X(L,J,K) AND X(L+1,J,K) = X(1,J,K).
C              = 1  IF X(0,J,K) = X(L+1,J,K) = 0.
C              = 2  IF X(0,J,K) = 0  AND X(L+1,J,K) = X(L-1,J,K).
C              = 3  IF X(0,J,K) = X(2,J,K) AND X(L+1,J,K) = X(L-1,J,K).
C              = 4  IF X(0,J,K) = X(2,J,K) AND X(L+1,J,K) = 0.
C
C     L        THE NUMBER OF UNKNOWNS IN THE I-DIRECTION. L MUST BE AT
C              LEAST 3.
C
C     C1       THE REAL CONSTANT THAT APPEARS IN THE ABOVE EQUATION.
C
C     MPEROD   INDICATES THE VALUES THAT X(I,0,K) AND X(I,M+1,K) ARE
C              ASSUMED TO HAVE.
C
C              = 0  IF X(I,0,K) = X(I,M,K) AND X(I,M+1,K) = X(I,1,K).
C              = 1  IF X(I,0,K) = X(I,M+1,K) = 0.
C              = 2  IF X(I,0,K) = 0 AND X(I,M+1,K) = X(I,M-1,K).
C              = 3  IF X(I,0,K) = X(I,2,K) AND X(I,M+1,K) = X(I,M-1,K).
C              = 4  IF X(I,0,K) = X(I,2,K) AND X(I,M+1,K) = 0.
C
C     M        THE NUMBER OF UNKNOWNS IN THE J-DIRECTION. M MUST BE AT
C              LEAST 3.
C
C     C2       THE REAL CONSTANT WHICH APPEARS IN THE ABOVE EQUATION.
C
C     NPEROD   = 0  IF A(1) AND C(N) ARE NOT ZERO.
C              = 1  IF A(1) = C(N) = 0.
C
C     N        THE NUMBER OF UNKNOWNS IN THE K-DIRECTION. N MUST BE AT
C              LEAST 3.
C
C
C     A,B,C    ONE-DIMENSIONAL ARRAYS OF LENGTH N THAT SPECIFY THE
C              COEFFICIENTS IN THE LINEAR EQUATIONS GIVEN ABOVE.
C
C              IF NPEROD = 0 THE ARRAY ELEMENTS MUST NOT DEPEND UPON THE
C              INDEX K, BUT MUST BE CONSTANT.  SPECIFICALLY,THE
C              SUBROUTINE CHECKS THE FOLLOWING CONDITION
C
C                          A(K) = C(1)
C                          C(K) = C(1)
C                          B(K) = B(1)
C
C                  FOR K=1,2,...,N.
C
C     LDIMF    THE ROW (OR FIRST) DIMENSION OF THE THREE-DIMENSIONAL
C              ARRAY F AS IT APPEARS IN THE PROGRAM CALLING POIS3D.
C              THIS PARAMETER IS USED TO SPECIFY THE VARIABLE DIMENSION
C              OF F.  LDIMF MUST BE AT LEAST L.
C
C     MDIMF    THE COLUMN (OR SECOND) DIMENSION OF THE THREE-DIMENSIONAL
C              ARRAY F AS IT APPEARS IN THE PROGRAM CALLING POIS3D.
C              THIS PARAMETER IS USED TO SPECIFY THE VARIABLE DIMENSION
C              OF F.  MDIMF MUST BE AT LEAST M.
C
C     F        A THREE-DIMENSIONAL ARRAY THAT SPECIFIES THE VALUES OF
C              THE RIGHT SIDE OF THE LINEAR SYSTEM OF EQUATIONS GIVEN
C              ABOVE.  F MUST BE DIMENSIONED AT LEAST L X M X N.
C
```

```
C     W        A ONE-DIMENSIONAL ARRAY THAT MUST BE PROVIDED BY THE
C              USER FOR WORK SPACE.  THE LENGTH OF W MUST BE AT LEAST
C              30 + L + M + 2*N + MAX(L,M,N) +
C              7*(INT((L+1)/2) + INT((M+1)/2)).
C
C
C            * * * * * *   ON OUTPUT   * * * * * *
C
C     F        CONTAINS THE SOLUTION X.
C
C     IERROR   AN ERROR FLAG THAT INDICATES INVALID INPUT PARAMETERS.
C              EXCEPT FOR NUMBER ZERO, A SOLUTION IS NOT ATTEMPTED.
C              = 0  NO ERROR
C              = 1  IF LPEROD .LT. 0 OR .GT. 4
C              = 2  IF L .LT. 3
C              = 3  IF MPEROD .LT. 0 OR .GT. 4
C              = 4  IF M .LT. 3
C              = 5  IF NPEROD .LT. 0 OR .GT. 1
C              = 6  IF N .LT. 3
C              = 7  IF LDIMF .LT. L
C              = 8  IF MDIMF .LT. M
C              = 9  IF A(K) .NE. C(1) OR C(K) .NE. C(1) OR B(I) .NE.B(1)
C                     FOR SOME K=1,2,...,N.
C              = 10 IF NPEROD = 1 AND A(1) .NE. 0 OR C(N) .NE. 0
C
C              SINCE THIS IS THE ONLY MEANS OF INDICATING A POSSIBLY
C              INCORRECT CALL TO POIS3D, THE USER SHOULD TEST IERROR
C              AFTER THE CALL.
C
C
C    * * * * * * *   PROGRAM SPECIFICATIONS    * * * * * * * * * * * *
C
C     DIMENSION OF   A(N),B(N),C(N),F(LDIMF,MDIMF,N),
C     ARGUMENTS      W(SEE ARGUMENT LIST)
C
C     LATEST         DECEMBER 1, 1978
C     REVISION
C
C     SUBPROGRAMS    POIS3D,POS3D1,TRID,RFFTI,RFFTF,RFFTF1,RFFTB,
C     REQUIRED       RFFTB1,COSTI,COST,SINTI,SINT,COSQI,COSQF,COSQF1
C                    COSQB,COSQB1,SINQI,SINQF,SINQB,CFFTI,CFFTI1,
C                    CFFTB,CFFTB1,PASSB2,PASSB3,PASSB4,PASSB,CFFTF,
C                    CFFTF1,PASSF1,PASSF2,PASSF3,PASSF4,PASSF,PIMACH,
C
C     SPECIAL        NONE
C     CONDITIONS
C
C     COMMON         VALUE
C     BLOCKS
C
C     I/O            NONE
C
C     PRECISION      SINGLE
C
C     SPECIALIST     ROLAND SWEET
C
C     LANGUAGE       FORTRAN
C
C     HISTORY        WRITTEN BY ROLAND SWEET AT NCAR IN JULY,1977
C
C     ALGORITHM      THIS SUBROUTINE SOLVES THREE-DIMENSIONAL BLOCK
C                    TRIDIAGONAL LINEAR SYSTEMS ARISING FROM FINITE
C                    DIFFERENCE APPROXIMATIONS TO THREE-DIMENSIONAL
C                    POISSON EQUATIONS USING THE FOURIER TRANSFORM
C                    PACKAGE SCLRFFTPAK WRITTEN BY PAUL SWARZTRAUBER.
C
C     SPACE          6561(DECIMAL) = 14641(OCTAL) LOCATIONS ON THE
C     REQUIRED       NCAR CONTROL DATA 7600
C
C     TIMING AND        THE EXECUTION TIME T ON THE NCAR CONTROL DATA
C     ACCURACY       7600 FOR SUBROUTINE POIS3D IS ROUGHLY PROPORTIONAL
```

```
C                   TO L*M*N*(LOG2(L)+LOG2(M)+5), BUT ALSO DEPENDS ON
C                   INPUT PARAMETERS LPEROD AND MPEROD.  SOME TYPICAL
C                   VALUES ARE LISTED IN THE TABLE BELOW WHEN NPEROD=0.
C                      TO MEASURE THE ACCURACY OF THE ALGORITHM A
C                   UNIFORM RANDOM NUMBER GENERATOR WAS USED TO CREATE
C                   A SOLUTION ARRAY X FOR THE SYSTEM GIVEN IN THE
C                   'PURPOSE' WITH
C
C                      A(K) = C(K) = -0.5*B(K) = 1,       K=1,2,...,N
C
C                   AND, WHEN NPEROD = 1
C
C                      A(1) = C(N) = 0
C                      A(N) = C(1) = 2.
C
C                   THE SOLUTION X WAS SUBSTITUTED INTO THE GIVEN SYS-
C                   TEM AND, USING DOUBLE PRECISION, A RIGHT SIDE Y WAS
C                   COMPUTED.  USING THIS ARRAY Y SUBROUTINE POIS WAS
C                   CALLED TO PRODUCE AN APPROXIMATE SOLUTION Z.  THEN
C                   THE RELATIVE ERROR, DEFINED AS
C
C                   E = MAX(ABS(Z(I,J,K)-X(I,J,K)))/MAX(ABS(X(I,J,K)))
C
C                   WHERE THE TWO MAXIMA ARE TAKEN OVER I=1,2,...,L,
C                   J=1,2,...,M AND K=1,2,...,N, WAS COMPUTED.  THE
C                   VALUE OF E IS GIVEN IN THE TABLE BELOW FOR SOME
C                   TYPICAL VALUES OF L,M AND N.
C
C
C                      L(=M=N)   LPEROD    MPEROD    T(MSECS)    E
C                      ------    ------    ------    --------  ------
C
C                        16        0         0         272     1.E-13
C                        15        1         1         287     4.E-13
C                        17        3         3         338     2.E-13
C                        32        0         0        1755     2.E-13
C                        31        1         1        1894     2.E-12
C                        33        3         3        2042     7.E-13
C
C
C    PORTABILITY    AMERICAN NATIONAL STANDARDS INSTITUTE FORTRAN.
C                   THE MACHINE DEPENDENT CONSTANT PI IS DEFINED IN
C                   FUNCTION PIMACH.
C
C    REQUIRED       COS,SIN,ATAN
C    RESIDENT
C    ROUTINES
C
C    REFERENCE      NONE
C
C   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```