Two-dimensional stellar convection

Project 3, AST3310
Spring 2019

Jowita Borowska

1 Introduction

The following project involves modelling a fluid by solving hydrodynamic equations. Two-dimensional computational grid, representing a collection of fluid-cells, has been created. We track the evolution of prameters like density, velocity and energy density (which are primary variables), as well as temperature and pressure (secondary variables). We apply some numerical differential methods in order to solve necessary derivatives and follow the progress in time of variables assigned to each cell. Thereafter, the time evolution of fluid in the box is animated, using FVis module. The total convective flux for each depth has also been animated, by a separate function applying matplotlib.animation.

2 Method

2.1 Mathematical equations governing convection

The set of hydrodynamic equations that need to be solved involves the continuity equation with no sources or sinks (Eq. 1), the momentum equation, excluding the viscous stress tensor but including gravity (Eq. 2) and the internal energy equation (Eq. 3):

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0, \tag{1}$$

$$\frac{\partial \rho \vec{u}}{\partial t} + \vec{u} \cdot \nabla (\rho \vec{u}) = -\nabla P + \rho \vec{g}, \tag{2}$$

$$\frac{\partial e}{\partial t} + \nabla \cdot (e \vec{u}) = -P \nabla \cdot \vec{u}, \tag{3}$$

where the velocity vector, $\vec{u} = u\hat{\imath} + w\hat{\jmath}$, has a horizontal component, u, and a vertical component, w, and the gravitational acceleration, $\vec{g} = g\hat{\jmath}$, has only a vertical component, working in y-direction (since we will orient the computational box in space with respect to gravity).

We split the above equations into components in horizontal, x, and vertical, y, direction.

In two dimensions the ∇-operator is $\nabla \equiv \hat{i}\frac{\partial}{\partial x} + \hat{j}\frac{\partial}{\partial y}$. The continuity equation then becomes

$$\frac{\partial \rho}{\partial t} + \rho\left(\nabla \cdot \vec{u}\right) + \vec{u} \cdot \left(\nabla \rho\right) = 0,$$

where the divergence of the vector field on the left-hand side of the equation, $\nabla \cdot \vec{u} = \frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}$, is equal zero for the case with constant flow, $\vec{u} = \text{const}$. We have then

$$\frac{\partial \rho}{\partial t} + \vec{u} \cdot \left(\nabla \rho\right) = 0,$$

$$\frac{\partial \rho}{\partial t} + \left(u\hat{i} + w\hat{j}\right) \cdot \left(\frac{\partial \rho}{\partial x}\hat{i} + \frac{\partial \rho}{\partial y}\hat{j}\right) = 0.$$

As the dot product of unit vectors gives $\hat{i} \cdot \hat{i} = \hat{j} \cdot \hat{j} = 1$ and $\hat{i} \cdot \hat{j} = 0$, we get

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho w}{\partial y} = 0 \Rightarrow \frac{\partial \rho}{\partial t} = -\frac{\partial \rho u}{\partial x} - \frac{\partial \rho w}{\partial y},$$

which is the two-dimensional advection equation with the constant flow (Eq.F.14).
We see that the factor $\nabla \cdot (\rho\vec{u})$, equal to the minus right-hand side of the equation above, can also be found in the momentum equation (Eq.2). We can then write Eq.2 as

$$\frac{\partial \rho \vec{u}}{\partial t} + \vec{u} \cdot \left(\frac{\partial \rho u}{\partial x} + \frac{\partial \rho w}{\partial y}\right) = -\nabla P + \rho\vec{g},$$

$$\frac{\partial \rho\left(u\hat{i} + w\hat{j}\right)}{\partial t} + \left(u\hat{i} + w\hat{j}\right) \cdot \left(\frac{\partial \rho u}{\partial x} + \frac{\partial \rho w}{\partial y}\right) = -\frac{\partial P}{\partial x}\hat{i} - \frac{\partial P}{\partial y}\hat{j} + \rho g\hat{j},$$

which now can be multiplied and split into the horizontal momentum equation (F.15) containing the factors with $\hat{i}$ unit vectors:

$$\frac{\partial \rho u}{\partial t} + \frac{\partial \rho u^2}{\partial x} + \frac{\partial \rho w u}{\partial y} = -\frac{\partial P}{\partial x} \Rightarrow \frac{\partial \rho u}{\partial t} = -\frac{\partial \rho u^2}{\partial x} - \frac{\partial \rho w u}{\partial y} - \frac{\partial P}{\partial x},$$

and the vertical momentum equation (F.16) containing the factors with $\hat{j}$ unit vectors:

$$\frac{\partial \rho w}{\partial t} + \frac{\partial \rho u w}{\partial x} + \frac{\partial \rho w^2}{\partial y} = -\frac{\partial P}{\partial y} + \rho g \Rightarrow \frac{\partial \rho w}{\partial t} = -\frac{\partial \rho w^2}{\partial y} - \frac{\partial \rho w u}{\partial x} - \frac{\partial P}{\partial y} + \rho g.$$

In the internal energy equation (Eq.3) we see the factor $\nabla \cdot (e\vec{u})$, which is very similar to the one from first and second equation (but here with e instead of ρ). Using the same approach we can rewrite Eq.3 as

$$\frac{\partial e}{\partial t} + \frac{\partial e u}{\partial x} + \frac{\partial e w}{\partial y} = -P\nabla \cdot \vec{u},$$

$$\frac{\partial e}{\partial t} + \frac{\partial e u}{\partial x} + \frac{\partial e w}{\partial y} = -P\left(\hat{i}\frac{\partial}{\partial x} + \hat{j}\frac{\partial}{\partial y}\right) \cdot \left(u\hat{i} + w\hat{j}\right),$$

$$\frac{\partial e}{\partial t} + \frac{\partial e u}{\partial x} + \frac{\partial e w}{\partial y} = -P\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right) \Rightarrow \frac{\partial e}{\partial t} = -\frac{\partial e u}{\partial x} - \frac{\partial e w}{\partial y} - P\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right),$$

which is Eq.F.17 in the project text. Nevertheless, in our project the flow, $\vec{u} = u\hat{i} + w\hat{j}$, is not assumed to be constant, so that all the equations from this subsection will be split into additional terms before the numerical implementation.

2.2 Numerical algorithms and discretisations

In order to solve the hydrodynamic equations numerically, we need to approximate derivatives involved in these equations. We will implement the algorithms using explicit difference methods, where parameters at the present time, n, are used to calculate the parameters at time $n+1$. Numerical schemes that are going to be applied to discretise the hydrodynamic equations are the Forward Time Centred Space (FTCS) method, as well as the upwind differencing, that, as distinct from FTCS, considers the direction of the flow.

The algorithms with discretisations for the continuity equation and the horizontal momentum equation are thoroughly described in the project text. Taking into account a non-constant flow, the vertical momentum equation (Eq.F.16) is split into additional terms,

$$\frac{\partial \rho w}{\partial t} = -\rho w \left(\frac{\partial w}{\partial y} + \frac{\partial u}{\partial x} \right) - w \frac{\partial \rho w}{\partial y} - u \frac{\partial \rho w}{\partial x} - \frac{\partial P}{\partial y} - \rho g,$$

where the gravitational acceleration component is negative, since the direction of increasing vertical distance in our box is opposite to the gravitational force. This is further

$$\left[\frac{\partial \rho w}{\partial t}\right]^n_{i,j} = -\left[\rho w\right]^n_{i,j} \left(\left[\frac{\partial w}{\partial y}\right]^n_{i,j} + \left[\frac{\partial u}{\partial x}\right]^n_{i,j} \right) - w^n_{i,j} \left[\frac{\partial \rho w}{\partial y}\right]^n_{i,j} - u^n_{i,j} \left[\frac{\partial \rho w}{\partial x}\right]^n_{i,j} - \left[\frac{\partial P}{\partial y}\right]^n_{i,j} - \rho^n_{i,j} g,$$

where n denotes time and i, j denote indices of cells on the computational grid. We use the upwind differencing to approximate all of the spatial derivatives that are multiplied by a factor containing velocity-component, so that only the spatial derivative of pressure is approximated using central differencing. The terms from the right-hand side of the above equation are then discretised as

$$\left[\frac{\partial w}{\partial y}\right]^n_{i,j} \approx \begin{cases} \frac{w^n_{i,j} - w^n_{i,j-1}}{\Delta y} & \text{if } w^n_{i,j} \geq 0 \\ \frac{w^n_{i,j+1} - w^n_{i,j}}{\Delta y} & \text{if } w^n_{i,j} < 0 \end{cases},$$

$$\left[\frac{\partial u}{\partial x}\right]^n_{i,j} \approx \begin{cases} \frac{u^n_{i,j} - u^n_{i-1,j}}{\Delta x} & \text{if } w^n_{i,j} \geq 0 \\ \frac{u^n_{i+1,j} - u^n_{i,j}}{\Delta x} & \text{if } w^n_{i,j} < 0 \end{cases},$$

$$\left[\frac{\partial \rho w}{\partial y}\right]^n_{i,j} \approx \begin{cases} \frac{[\rho w]^n_{i,j} - [\rho w]^n_{i,j-1}}{\Delta y} & \text{if } w^n_{i,j} \geq 0 \\ \frac{[\rho w]^n_{i,j+1} - [\rho w]^n_{i,j}}{\Delta y} & \text{if } w^n_{i,j} < 0 \end{cases},$$

$$\left[\frac{\partial \rho w}{\partial x}\right]^n_{i,j} \approx \begin{cases} \frac{[\rho w]^n_{i,j} - [\rho w]^n_{i-1,j}}{\Delta x} & \text{if } u^n_{i,j} \geq 0 \\ \frac{[\rho w]^n_{i+1,j} - [\rho w]^n_{i,j}}{\Delta x} & \text{if } u^n_{i,j} < 0 \end{cases},$$

$$\left[\frac{\partial P}{\partial y}\right]^n_{i,j} \approx \frac{P^n_{i,j+1} - P^n_{i,j-1}}{2\Delta y}.$$

The left-hand side is discretised using forward time. Moreover, since $\rho^{n+1}_{i,j}$ has already been calculated (using continuity equation), we can advance w in time as well:

$$\left[\frac{\partial \rho w}{\partial t}\right]^n_{i,j} \approx \frac{[\rho w]^{n+1}_{i,j} - [\rho w]^n_{i,j}}{\Delta t} \Rightarrow w^{n+1}_{i,j} = \frac{[\rho w]^n_{i,j} + \left[\frac{\partial \rho w}{\partial t}\right]^n_{i,j} \Delta t}{\rho^{n+1}_{i,j}}.$$

The internal energy equation with non-constant flow is also split into additional terms, which yields

$$\frac{\partial e}{\partial t} = -e\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right) - u\frac{\partial e}{\partial x} - w\frac{\partial e}{\partial y} - P\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right) = -(e+P)\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right) - u\frac{\partial e}{\partial x} - w\frac{\partial e}{\partial y}.$$

This can be expressed as

$$\left[\frac{\partial e}{\partial t}\right]^n_{i,j} = -(e^n_{i,j} + P^n_{i,j})\left(\left[\frac{\partial u}{\partial x}\right]^n_{i,j} + \left[\frac{\partial w}{\partial y}\right]^n_{i,j}\right) - u^n_{i,j}\left[\frac{\partial e}{\partial x}\right]^n_{i,j} - w^n_{i,j}\left[\frac{\partial e}{\partial y}\right]^n_{i,j}.$$

Thereafter, the spatial derivatives of velocity components are discretised using central differencing, and energy spatial derivatives are approximated using upwind differencing:

$$\left[\frac{\partial u}{\partial x}\right]^n_{i,j} \approx \frac{u^n_{i+1,j} - u^n_{i-1,j}}{2\Delta x},$$

$$\left[\frac{\partial w}{\partial y}\right]^n_{i,j} \approx \frac{w^n_{i,j+1} - w^n_{i,j-1}}{2\Delta y},$$

$$\left[\frac{\partial e}{\partial x}\right]^n_{i,j} \approx \begin{cases} \frac{e^n_{i,j} - e^n_{i-1,j}}{\Delta x} & \text{if } u^n_{i,j} \geq 0 \\ \frac{e^n_{i+1,j} - e^n_{i,j}}{\Delta x} & \text{if } u^n_{i,j} < 0 \end{cases},$$

$$\left[\frac{\partial e}{\partial y}\right]^n_{i,j} \approx \begin{cases} \frac{e^n_{i,j} - e^n_{i,j-1}}{\Delta y} & \text{if } w^n_{i,j} \geq 0 \\ \frac{e^n_{i,j+1} - e^n_{i,j}}{\Delta y} & \text{if } w^n_{i,j} < 0 \end{cases}.$$

The left-hand side of the equation is again approximated using forward time, which allows advancing e in time:

$$\left[\frac{\partial e}{\partial t}\right]^n_{i,j} \approx \frac{e^{n+1}_{i,j} - e^n_{i,j}}{\Delta t} \Rightarrow e^{n+1}_{i,j} = e^n_{i,j} + \left[\frac{\partial e}{\partial t}\right]^n_{i,j}\Delta t.$$

It is worth to notice that the first order upwind scheme uses differencing biased in the direction determined by the sign of the speed. For that reason, this is a better method for modelling the transport properties of the system. Therefore, we use the upwind differencing to approximate every derivative-term in the equations that is multiplied by a velocity-component (and consider the given component in the discretization). This approach is applied for the algorithms and discretizations of all the hydrodynamic equations, including the energy equation.

2.3 Initial conditions

At the top row of our computational box, the temperature, T_{top}, and pressure, P_{top}, are in agreement with the values of these parameters at the solar photosphere. We can derive the initial conditions of temperature, pressure, density and energy density for every other point on the grid by following the requirement of hydrostatic equilibrium, appyling the equation of state for an ideal gas, and the fact that

$$\nabla = \frac{\partial \ln T}{\partial \ln P} > \frac{2}{5}.$$

We consider a cell in the computational box, containing the fluid of volume $V = \Delta x \cdot \Delta y$. When the fluid in the box is said to be in hydrostatic equilibrium, the net force acting on each such cell must be equal zero. We have three forces acting in y-direction: the pressure from the fluid above, pushing a given cell downward, $\vec{F}_{top}$, pressure from the fluid below, pushing the cell upward, $\vec{F}_{bottom}$, as well as the gravity force due to the weight of the fluid-cell, $\vec{F}_g$. The hydrostatic equilibrium condition gives

$$0 = \vec{F}_{top} + \vec{F}_{bottom} + \vec{F}_g = -P_{top}\Delta x \hat{j} + P_{bottom}\Delta x \hat{j} - \rho g \Delta x \Delta y \hat{j},$$

$$\Rightarrow P_{top} - P_{bottom} = -\rho g \Delta y \Rightarrow \frac{dP}{dy} = -\rho g,$$

where the last part follows for infinitesimally small changes in pressure, P, and height, y. The sign before g follows from the direction of increasing y-axis with respect to gravity (and we will use 'minus' here, throughout the whole project). From the Eq. F.29 and F.30 in the project description, we derive the density expression:

$$\rho = \frac{(\gamma - 1)\mu m_u}{k_B}\frac{e}{T} = \frac{(\gamma - 1)\mu m_u}{k_B}\frac{\frac{P}{(\gamma-1)}}{T} = \frac{\mu m_u}{k_B}\frac{P}{T}. \tag{4}$$

Furthermore, we can use the double-logarithmic gradient (which is a constant, $C > 2/5$):

$$C = \frac{\partial \ln T}{\partial \ln P} = \frac{\frac{1}{T}dT}{\frac{1}{P}dP} = \frac{P}{T}\frac{dT}{dP} \Rightarrow C\frac{1}{P}dP = \frac{1}{T}dT.$$

Dividing both sides of the equation by dy, applying the hydrostatic equilibrium equation and density-expression yields

$$C\frac{1}{P}\frac{dP}{dy} = \frac{1}{T}\frac{dT}{dy} \Rightarrow C\frac{1}{P}(-\rho g) = \frac{1}{T}\frac{dT}{dy} \Rightarrow C\frac{1}{P}\left(-\frac{\mu m_u g}{k_B}\frac{P}{T}\right) = \frac{1}{T}\frac{dT}{dy} \Rightarrow -C\frac{\mu m_u g}{k_B} = \frac{dT}{dy},$$

$$dT = -C\frac{\mu m_u g}{k_B}dy \Rightarrow \int_{T'}^{T_{top}} dT = -C\frac{\mu m_u g}{k_B}\int_{y'}^{Y} dy \Rightarrow T_{top} - T' = -C\frac{\mu m_u g}{k_B}(Y - y'),$$

$$T' = T_{top} + C\frac{\mu m_u g}{k_B}(Y - y'),$$

where Y is the position of the top-row in the box (equal to the height of the box minus Δy) and T' is the initial temperature at the level y', so that there is the highest temperature at the bottom of the box, where $y' = 0$, and $T' = T_{top}$ at the top, where $y' = Y$. Moreover, using the expression derived from the double logarithmic gradient allows calculating the initial pressure in the box

$$C\frac{1}{P}dP = \frac{1}{T}dT \Rightarrow C\int_{P'}^{P_{top}} \frac{1}{P}dP = \int_{T'}^{T_{top}} \frac{1}{T}dT,$$

$$C\left[\ln(P_{top}) - \ln(P')\right] = \ln(T_{top}) - \ln(T') \Rightarrow \ln(P') = \ln(P_{top}) - \frac{1}{C}\ln\left(\frac{T_{top}}{T'}\right),$$

$$\ln(P') = \ln(P_{top}) - \ln\left(\frac{T_{top}}{T'}\right)^{1/C} \Rightarrow \ln(P') = \ln\left(P_{top} \cdot \left(\frac{T'}{T_{top}}\right)^{1/C}\right),$$

and exponentiating both sides of the above equation gives

$$P' = P_{top} \cdot \left(\frac{T'}{T_{top}}\right)^{1/C} .$$

Now, we can use the previously derived expression for density (Eq.4) and Eq. F.29. to initialize consecutively the density and internal energy density for the entire box. In addition, we set the initial velocity equal zero everywhere.

2.4 Boundary conditions

We cannot solve the discretised hydrodynamic equations at the end points of the numerical grid (indices 0 and −1 in both directions). Therefore, the *boundary conditions* must be applied for these regions, in order to update the values of variables with each time-step for the entire box. Horizontally, every variable is periodic. When it comes to the vertical boundary conditions, we need to consider some requirements. First of all, we set the vertical component of the velocity equal zero at the upper and lower boundary, $w^n_{i,0} = w^n_{i,-1} = 0$. Then, the gradient of the horizontal component of the velocity, $\frac{\partial u}{\partial y}$, should be zero at the boundaries, which gives the lower boundary

$$0 = \left[\frac{\partial u}{\partial y}\right]^n_{i,0} = \frac{-u^n_{i,2} + 4u^n_{i,1} - 3u^n_{i,0}}{2\Delta y} \Rightarrow u^n_{i,0} = \frac{4u^n_{i,1} - u^n_{i,2}}{3},$$

where the second order upwind-scheme has been used - here, forward difference approximation (Eq. F.32.). The upper boundary is

$$0 = \left[\frac{\partial u}{\partial y}\right]^n_{i,-1} = \frac{3u^n_{i,-1} - 4u^n_{i,-2} + u^n_{i,-3}}{2\Delta y} \Rightarrow u^n_{i,-1} = \frac{4u^n_{i,-2} - u^n_{i,-3}}{3},$$

where backward difference approximation (Eq. F.33.) has been applied. Further, we use the fact that the hydrostatic equilibrium equation must hold at the boundaries. We insert the pressure from the equation of state for an ideal gas (Eq. F.30) and density from Eq.4:

$$\frac{\partial P}{\partial y} = \frac{\partial(\gamma-1)e}{\partial y} = (\gamma-1)\frac{\partial e}{\partial y} = -\rho g = -\frac{(\gamma-1)\mu m_u g}{k_B}\frac{e}{T} \Rightarrow \frac{\partial e}{\partial y} = -\frac{\mu m_u g}{k_B}\frac{e}{T},$$

which can be expressed as

$$\left[\frac{\partial e}{\partial y}\right]^n_{i,j} = -\frac{\mu m_u g}{k_B}\frac{e^n_{i,j}}{T^n_{i,j}}.$$

Nevertheless, we can only use the temperature value from the previous time-step (more about that in the *Program description*). Now, applying Eq.32. allows finding the lower boundary condition:

$$\left[\frac{\partial e}{\partial y}\right]^n_{i,0} = \frac{-e^n_{i,2} + 4e^n_{i,1} - 3e^n_{i,0}}{2\Delta y} = -\frac{\mu m_u g}{k_B}\frac{e^n_{i,0}}{T^{n-1}_{i,0}} \Rightarrow e^n_{i,0} = \frac{4e^n_{i,1} - e^n_{i,2}}{\left(3 - \frac{2\Delta y \mu m_u g}{k_B T^{n-1}_{i,0}}\right)},$$

and applying Eq.33. - upper boundary condition:

$$\left[\frac{\partial e}{\partial y}\right]^n_{i,-1} = \frac{3e^n_{i,-1} - 4e^n_{i,-2} + e^n_{i,-3}}{2\Delta y} = -\frac{\mu m_u g}{k_B}\frac{e^n_{i,-1}}{T^{n-1}_{i,-1}} \Rightarrow e^n_{i,-1} = \frac{e^n_{i,-3} - 4e^n_{i,-2}}{-\left(\frac{2\Delta y \mu m_u g}{k_B T^{n-1}_{i,-1}} + 3\right)}.$$

The boundary conditions for energy density and density are coupled. The values of density at the boundary points can be found by using Eq.4 and the already computed energy density in these regions, which gives:

$$\rho^n_{i,0} = \frac{(\gamma-1)\mu m_u}{k_B}\frac{e^n_{i,0}}{T^{n-1}_{i,0}} \quad \text{and} \quad \rho^n_{i,-1} = \frac{(\gamma-1)\mu m_u}{k_B}\frac{e^n_{i,-1}}{T^{n-1}_{i,-1}}.$$

2.5 Program description

The program consists of a class called twoD_convection and a short main-body program, that creates an instance of the visualisation class (FVis.FluidVisualiser), as well as instances of the self 2D convection-clss (for both the sanity check - which is optional to run - and the relevant convection program). Here, we also call the functions needed to perform the simulation and create the animations.

Starting from the top of the code, after some import-statements, we define the class, that will be simulating the movement of fluid in our two-dimensional computational box. We collect all of the data attributes in the constructor, __init__. Some of the variables here are simply constants, used throughout the whole program. Other, like two-dimensional arrays assigned to different parameters or an empty list for storing the convective flux along y-direction, are going to be updated and advanced in time, as the program is being run.

The next method, initialize, takes one argument, perturbation, which is by default set to 'True'. Here, the mathematical equations derived in subsection 2.3 (*Initial conditions*) are implemented (at first for the temperature and pressure). If the perturbation-argument is not set to 'False' or not given, we create a Gaussian-like two-dimensional array matching the shape of our grid, following the expression

$$f(x,y) = A\exp\left(-\frac{(x-x_{mid})^2+(y-y_{mid})^2}{2\sigma^2}\right),$$

where x and y are the position arrays of cells in consecutively horizontal and vertical direction, x_{mid} is the width of the box divided by 2, y_{mid} is half of the height. We set the amplitude, $A = 6000$ K (which is a little bit higher than the temperature at the top of the box), and the standard devatiation, $\sigma = 1$ Mm (which constitutes a 1/4 of the height). We add this perturbation to the previously computed initial temperature. Thereafter, the density and the internal energy density can also be initialized.

Next method, that has been defined, is timestep, which calculates and returns a suitable time-step for a given iteration. This method takes in four arguments: the present values of density and energy density time-derivatives, as well as horizontal and vertical velocity-components of the points that are going to be updated by the difference methods, using Δt (so the ones with indices 1 to -2). The method essentially uses the approach that is described in the project text, with the simple if-statement at the end, that prevents from a very big time-step or division by zero.

The next method, boundary_conditions, applies the expressions shown in the previous subsection. It is going to advance in time the boundary rows of cells, since these cannot be updated by the difference methods. We update horizontal boundaries (points with indices $i = 0$ and $i = -1$) and vertical boundaries (indices $j = 0$ and $j = -1$) of the arrays corresponding to primary variables (u, w, e, ρ).

The following four methods apply the previously explained numerical difference schemes and return spatial derivatives of a given function for every inner point on the grid (i.e. points with indices 1 to -2). We have central_x and central_y, that compute derivatives consecutively in x- and y-direction, based on the central differencing scheme. Next, we find the definitions of upwind_x and upwind_y, that apply upwind differencing and calculate derivatives based on the direction of the velocity-component, given as an argument.

Finally, the most important method, hydro_solver, that advances the simulation by one time-step, updates the content of arrays, as well as computes and collects convective flux values and time-steps needed for the convective flux animation. First, the time-derivatives of primary variables are calculated (for the inner points on the grid), using the algorithms and discretisations presented in subsection 2.2. Here, previously defined difference methods are called, wherever it is needed. Next, we find a suitable time-step by calling timestep method, and then use it to advance the inner points of the primary variables in time. After the indices 1 to -2 (inner points) of ρ, u, w and e have been updated, we have to call the boundary_conditions function in order to update the boundary points as well. Here, we see that in some expressions involving temperature, the value of the temperature from the previous time-step has to be used. Next, with the entire grid of updated primary variables, we can calculate the secondary variables, P and T, using Eq. F.29 and F.30. The last part of the method consists of calculating the convective flux, F_C, as a function of vertical distance, y, by summing the internal energy times velocity, $F_{C_j} = \sum_i e_{i,j}\sqrt{u_{i,j}^2 + w_{i,j}^2}$, on each y-level (for each j index). Then, we append both the convective flux array and the given time-step to the separate lists, that wil be used to create the flux-animation.

Last but not least, the method animate_convective_flux has been written. It can be called at the very end of the program, after we initialize the simulation and completely advance it in time by calling FVis.FluidVisualiser.save_data prior to the animate_convective_flux. It uses matplotlib.animation module in order to create a movie of the time evolution of the total convective flux in our box, as a function of vertical distance, y.

3 Results

Together with this report (*my_report.pdf*), code of the written program (*my_code.py*) and *FVis2.py* file with FluidVisualiser module for Python 2.x, the following files are attached: the folder called *Sanity_check_data*, containing the data saved from the run of the program by choosing the 'sanity_check = True' option, with corresponding *Sanity_check.mp4* movie; the folder called *Results_data*, containing the data saved from the run of the program with temperature perturbation switched on, with corresponding *Convection_movie.mp4*; as well as the movie showing evolution of the convective flux in time, called *Flux_movie.mp4*.

4 Discussion and conclusions

4.1 Analysis of results - 2D convection simulation

First, we perform the sanity check, with the temperature perturbation switched off. On the attached movie, *Sanity_check.mp4* (where the 60 seconds long visualisation of the temperature parameter has been shown), it can be seen that there are no changes in the computational box, which means that the system is in hydrostatic equilibrium.

After we verified that the code is stable in hydrostatic equilibrium, we run the simulation with the temperature perturbation switched on, provoking the fluid in the computational box to become convectively unstable. The movie of the two-dimensional convection for 250 seconds of simulation can be found as *Convection_movie.mp4*, and shows the time evolution of temperature (in color) and velocity (as vectors). The snapshots of the simulation for some points in time (1 s, 45 s, 105 s, 180 s) are shown on the Figure 1.

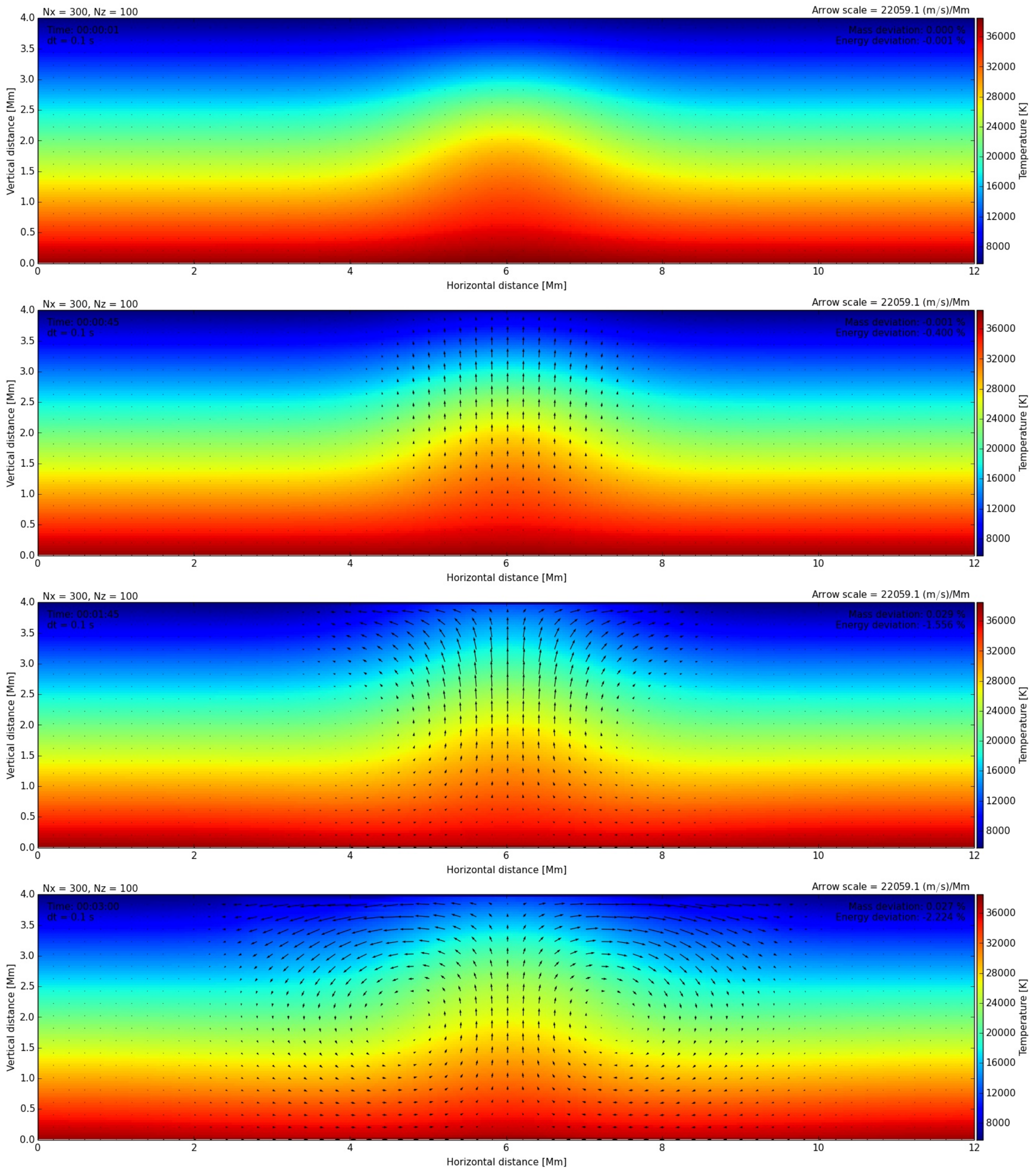

Figure 1: Snapshots from the animation of the temperature (shown in color) and velocity (shown as vectors) at different runtimes, corresponding to (from the top down) 1st, 45th, 105th and 180th second of the simulation.

Initially (top snapshot on Figure 1), we can clearly see the Gaussian-like top in the middle of the box, representing the region with higher temperature than its surroundings. Animating density (which is not included here, but can easily be run using *Results_data* folder) shows that the region of increased temperature corresponds to the region with lower density. This is of course in agreement with Eq.4., as we calculate the initial density based on the perturbed temperature. The gas becomes then convectively unstable and begins to rise through the computational box towards the top (see snapshot from 45th second). When the circumstances change and gas reaches the top-region with lower temperature (and higher density), it starts to fall down (this process begins approximately after 105th second of the simulation, as can be seen on the third and fourth snapshot). Gas continues to move in the convective motion, having an impact on the time evolution of temperature (and remaining variables) in the box, as it interchanges energy. Nevertheless, more or less after 190 seconds, the simulation becomes unstable, as a consequence of the numerical approach.

4.2 Analysis of results - total convective flux

The animation of the time-evolution of the total convective flux at each y-level shows that at the beginning, F_C at each level is equal zero. This is obvious, since the initial values of velocities are zero. It starts to rise very slowly - first in the middle of the box (since it is here, we start to see the motion), then more at the bottom of the box. The velocity-pattern in the box is rather vertically-symmetric, but magnitudes of the velocity components are bigger at the top. For instance, u in the upper regions increases up to $\approx |6000|$ m/s and in the lower regions to $\approx |2000|$ m/s. However, this is balanced out by the most significant impact on F_C, the internal energy density, e, which is much bigger at the bottom ($Y = 0$) than at the top of the box throughout the whole simulation. That is why there are no major differences between the total convective flux at each y-level, but the total convective flux tends to be a bit larger at the lower levels.

4.3 Summary - what I have learned

I have learned a lot in the process of completing this project! First time I have created a long program in Python using classes. This gave a nice, modular structure to my code (and from now on I will definitely be using this approach). On the way, I have discovered that *self.*-prefix needs to be added whenever I want to update the whole attributes (lack of this can lead to some problems while initializng P, ρ and e arrays). Moreover, I have learned some new numerical difference methods, how to make movies using matplotlib.animation and how to discover a bug in the program by printing almost every bit of it :)

The most significant problem, I have encountered, was the way of understanding the point of boundary conditions. At first, I treated boundaries as the points needed only to calculate the derivatives at the last rows of cells inside the box. That is why I created the set of larger arrays, that included both the box, with 300 x 100 cells, and the outer boundary-points (so their shape was 302 x 102). However, this approach requires updating in time the two sets of arrays (which I clearly forgot about) and results in more definitions, hence the longer code than the one I finally decided to use. Last but not least, a big issue that arose at the end was the fact that initializing the pressure after implementing the temperature-perturbation does not produce a convective motion. This was a subtle change, but not so easy to track down.