AST3310 - Stellar Convection

Project 3

Jakob Borg

May 19, 2019

1 Introduction

In this project we are modeling two dimensional (2D) convection near the surface of a star. To do this we implement an explicit 2D hydrodynamic solver. In this paper we go through how we define our model and the assumptions in use, before we discuss the governing equations and their initial and boundary conditions in section 2. How these equations are solved numerically are laid out in section 3.

Throughout the paper we have used footnotes to mark some important points. To help the reader[1], footnotes are also included to mark where we have meant to answer the different bullet points required of the report in the problem text.

For visualization of results we are using a provided module, designed for this course. The module and how it's used will not be discussed here, as it is well documented in the user guide together with the open code at github repository Frogner (2018). Our code is set up to produce all of the results discussed here by utilizing command line arguments in the terminal when running the code. These commands are also given as footnotes where they are appropriate. Note that this will for some of the commands take some time where the simulation is pushed to run over a long time period[2], in addition to saving the different animations instead of just showing them. There are also some commands not discussed in this paper for loading data etc. The commands can also be mixed to calculate/produce more than one result at a time, but be careful when using this functionality as it is not tested for new users.

1.1 The Model

As this is a 2D model, the volume of our system is in 2D. We will split the volume of the star into cells, where each cell is approximated by one value for each of the variables used in the calculations. For simplicity, we create cubic cells of size $\Delta x \times \Delta y$.

We then choose a computation volume of nx and ny cells in x and y direction respectively. The volume is located with the top at the surface of the star, with y direction radially from the center (vertical) and x along the surface (horizontal). We let x enclose $12\,\mathrm{Mm}$ and y $4\,\mathrm{Mm}$, with $y = 4\,\mathrm{Mm}$ at the surface of the star and $y = 0$ at the bottom of our box.

The computation volume is *much* smaller than the volume of the star and located far from the center, so the horizontal and vertical component are approximated as perpendicular.

Each cell has a position in the volume described as

$$(x, y) = (i\Delta x, j\Delta y) \quad \text{with} \quad \begin{matrix} i \in [0, nx-1] \\ j \in [0, ny-1]. \end{matrix}$$

Due to the way matplotlib orients axes in figures we define our grid in such a way that index $j = 0$ (which is at the top in our 2D matrix/array in python) is at the bottom of the computational volume, and $j = ny - 1$ to be at the surface. A general variable is then expressed using the cells as

$$\phi_{ij}^n = \phi(x_i, y_j, t_n)$$
$$t_n = t_0 + n\Delta t.$$

where we discretize the time by n and Δt from some zero time $t_0 = 0$,

1.2 Assumptions

In the initial implementation of the system we assume that everything is in hydrostatic equilibrium. This way we can set up initial conditions and test our system in a simplified situation, before invoking convection through a Gaussian perturbation in the initial temperature. This will be further elaborated in sections 2.2 and 3.3.2.

We assume the plasma to be an ideal gas with constant mean molecular weight $\mu = 0.61$. Compared to our assumptions from earlier projects the ideal gas assumption is more accurate here as we are so close to the surface. Here the density is much lower, and most of the particles should be mono-atomic hydrogen.

In addition we assume a constant gravitational acceleration over the hole computational volume, pointing towards the center of the star

$$\vec{\mathbf{g}} = -G\frac{M_\odot}{R_\odot^2}\hat{\mathbf{y}} = -g_y$$

where G is the gravitational constant. As the box is so small and far from the center this is a good approximation, but we have to mind the sign. We know the gravitational acceleration from a spherical mass is in the negative radial direction. The upper boundary of y should be at the surface of the star, thus $\hat{\mathbf{y}}$ is pointing in positive radial direction and $\vec{\mathbf{g}}$ is negative. Notice the notation here, we define $g_y = GM_\odot R_\odot^{-2}$ to be the magnitude of the gravitational acceleration, and will deal with the sign appropriately in each equation.

2 The Governing Equations

2.1 The System of Hydrodynamic Differential Equations

Our hydrodynamical system is governed by three main conservation equations. The continuity equation (or mass equation) without sources or sinks, the momentum equation including gravity but not the viscous stress tensor, and the energy equation, given in eqs. (1) to (3) respectively.

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

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

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

We split the governing equations into components in x and y direction. By decomposing the velocity $\vec{\mathbf{u}} = (u, w)$ and

[1] Especially the examiner. Note some of the bullet points are separated over multiple sections.

[2] This can easily be adjusted by changing the *endtime* parameter under the three different simulation configurations in *2Dconvecion.py*.

gradient $\vec{\boldsymbol{\nabla}} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$, we rewrite eqs. (1) to (3) into four equations[3]. First, the continuity equation becomes

$$\frac{\partial \rho}{\partial t} = -\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right) \cdot (\rho u, \rho w) = -\frac{\partial \rho u}{\partial x} - \frac{\partial \rho w}{\partial y}. \tag{4}$$

The momentum equation is split in two, one horizontal and one vertical component. Here Einstein's summation convention is applied to simplify the notation, v_i is the i-th component of the velocity $\vec{\mathbf{u}}$, and $i, j \in [x, y]$.

$$\begin{aligned}
&\frac{\partial \rho v_i}{\partial t} = -\frac{\partial}{\partial x_j}(\rho v_i v_j) - \frac{\partial P}{\partial x_i} + \rho g_i \\
&\underline{\text{for} \quad i = x, \quad j = x, y, \quad g_x = 0} \\
&\Rightarrow \frac{\partial \rho u}{\partial t} = -\frac{\partial \rho u^2}{\partial x} - \frac{\partial \rho u w}{\partial y} - \frac{\partial P}{\partial x} \qquad (5) \\
&\underline{\text{for} \quad i = y, \quad j = x, y, \quad g_y = g_y} \\
&\Rightarrow \frac{\partial \rho w}{\partial t} = -\frac{\partial \rho w^2}{\partial y} - \frac{\partial \rho u w}{\partial x} - \frac{\partial P}{\partial y} - \rho g_y. \qquad (6)
\end{aligned}$$

Where we have used that gravity is only in the vertical direction as discussed in section 1.2. Notice the sign in the last term in eq. (6), as $g_y > 0$.

Similarly for the energy equation, using Einstein's summation convention we rewrite

$$\begin{aligned}
&\frac{\partial e}{\partial t} + \frac{\partial e v_i}{\partial x_i} = -P\frac{\partial v_i}{\partial x_i} \qquad i = x, y \\
&\frac{\partial e}{\partial t} + \frac{\partial e u}{\partial x} + \frac{\partial e w}{\partial y} = -P\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right) \\
&\Rightarrow \frac{\partial e}{\partial t} = -\frac{\partial e u}{\partial x} - \frac{\partial e w}{\partial y} - P\left(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial y}\right). \qquad (7)
\end{aligned}$$

2.2 Initial Values [4]

The initial and boundary conditions in this system are delicate, and we have little information to start from, so we must use some intuition and assumptions. As mentioned we assume hydrostatic equilibrium, expressed as

$$\frac{\partial P}{\partial y} = -\rho g_y \tag{8}$$

where again, notice the sign. Based on this, we know the initial gradients of each parameter to only be in the vertical direction. In addition we know from project 2 the double logarithmic gradient to be

$$\nabla = \frac{\partial \ln T}{\partial \ln P} = \frac{2}{5} + \mathrm{d}\nabla \tag{9}$$

where $\mathrm{d}\nabla$ is some small positive perturbation from the adiabatic gradient of $2/5$. We choose a value $\mathrm{d}\nabla = 1 \times 10^{-4}$, and assume the gradient to be constant.

[3] Bullet point 1; F.14 to F.17 in eqs. (4) to (7).

[4] Bullet point 3; Initial conditions for T and P

Based on this, together with the fact that we know the values of the temperature and pressure from the photosphere of the Sun

$$T_\odot = 5778\,\mathrm{K} \qquad P_\odot = 1.8 \times 10^8\,\mathrm{Pa}$$

for $n = 0$, $i \in [0, nx - 1]$ and $j = ny - 1$, we can find the initial conditions for T and P in the hole box. First we rewrite the logarithmic gradient to find a separable equation for the temperature in the hole box as a function of height y. To do this we also use two expressions for the equation of state of an ideal gas, one relating the pressure and energy and one relating the energy with density and temperature. These expressions are used much in tandem in the following, and are just referred to as the «equations of state».

$$\begin{aligned}
P &= (\gamma - 1)e \\
e &= \frac{1}{(\gamma - 1)}\frac{\rho}{\mu m_u} k_B T
\end{aligned}$$

where m_u is the unit atomic mass in kg and $\gamma = \frac{f+2}{f} = \frac{5}{3}$ is a thermodynamic constant, called the adiabatic index, determined by the degrees of freedom of the particles in the gas, here $f = 3$. Using the product rule and the equations of state we find

$$\begin{aligned}
\nabla &= \frac{P}{T}\frac{\partial y}{\partial P}\frac{\partial T}{\partial y} \quad \text{where} \quad \begin{aligned} P &= \frac{\rho}{\mu m_u} k_B T \\ \frac{\partial y}{\partial P} &= -\frac{1}{\rho g_y} \end{aligned} \\
&= -\frac{k_B}{\mu m_u g_y}\frac{\partial T}{\partial y}
\end{aligned}$$

Now, we can form the initial temperature. From the initial temperature at the surface $y_{\max}$, denoted $T_\odot$, we find the temperature at height y by integrating both sides.

$$\begin{aligned}
&\int_T^{T_\odot} \mathrm{d}T = -\int_y^{y_{\max}} \frac{\mu m_u g_y}{k_B} \nabla \,\mathrm{d}y \\
&\Rightarrow T(y) = T_\odot - \frac{\mu m_u g_y}{k_B} \nabla (y - y_{\max}). \qquad (10)
\end{aligned}$$

With the initial temperature we can find initial pressure by again using the equations of state.

$$\begin{aligned}
P(y) = (\gamma - 1)e = \frac{\rho}{\mu m_u} k_B T(y) \quad &\Big| \cdot \frac{g_y}{g_y} \\
= -\frac{\partial P}{\partial y}\frac{k_B}{\mu m_u g_y}\left(T_\odot - \frac{\mu m_u g_y}{k_B}\nabla(y - y_{\max})\right) \\
= -\frac{\partial P}{\partial y}\left(\frac{k_B T_\odot}{\mu m_u g_y} - \nabla(y - y_{\max})\right). \qquad (11)
\end{aligned}$$

To simplify we make the following substitution

$$\begin{aligned}
\beta(y) &= \frac{k_B T_\odot}{\mu m_u g_y} - \nabla(y - y_{\max}) \\
\frac{\partial \beta}{\partial y} &= -\nabla, \quad \beta(y_{\max}) = \beta_0 = \frac{k_B T_\odot}{\mu m_u g_y}
\end{aligned}$$

Inserting into eq. (11) gives

$$P = -\frac{\partial P}{\partial y}\beta \Rightarrow \frac{\mathrm{d}P}{P} = \frac{\mathrm{d}\beta}{\nabla\beta}.$$

Using the same integration technique we find

$$\begin{aligned}\int_P^{P_\odot} \frac{\mathrm{d}P}{P} &= \int_\beta^{\beta_0} \frac{\mathrm{d}\beta}{\nabla\beta} \\ \ln\left(\frac{P_\odot}{P}\right) &= \frac{1}{\nabla}\ln\left(\frac{\beta_0}{\beta}\right) \\ \Rightarrow P(y) &= P_\odot\left(\frac{\beta}{\beta_0}\right)^{1/\nabla} \end{aligned} \tag{12}$$

where in the last step we used properties of the exponential and logarithmic functions.

With the initial pressure and temperature established we are able to find the initial density and energy[5]. Both are found by again utilizing how coupled the quantities are through the equations of state.

$$e(y) = \frac{P(y)}{(\gamma-1)} \tag{13}$$

$$\rho(y) = e(y)(\gamma-1)\frac{\mu m_u}{k_B T(y)}. \tag{14}$$

2.3 Boundary Conditions[6]

Let us adapt the the following index notation, $[Q]^n_{i,j}$, where n indicate the time step and i, j indicate the position in the grid of the quantity Q.

As our computational volume has discontinuous boundaries we have to apply some smart boundary conditions for our numerical approach to work. First we implement periodic horizontal boundary conditions. This way our volume is continuous in the x-direction. This is simple, and makes sense physically as we pretend there are neighboring «boxes» at either side of the computational volume. This condition is implemented through our numerical differential solvers, which will be discussed in 3.2, and will not be elaborated further as it's quite elementary.

The vertical boundaries are more challenging. As will be discussed in section 3.3, we only need boundary conditions for the four primary variables (ρ, e, u, w). In the following, if not stated otherwise, each vertical boundary are described by indices $i \in [0,\, nx-1]$ and $j = 0$ for bottom and $j = ny-1$ for the surface, and are used for all times n.

The vertical velocity w at the bottom and top of our box should be zero. As we are using the continuity equation (or conservation of mass) without sources or sinks this has to be true to prevent mass from leaking out into the photosphere and in through the bottom of the box.

$$w_{i,j} = 0. \tag{15}$$

Also, the vertical gradient of the horizontal velocity u must be zero at the boundary, for the same reasons as for the vertical velocity. To find an expression for $u_{i,j}$ at the boundary we use the forward and backwards approximations for the partial derivative and solve for $u_{i,j}$ (as the vertical boundaries should not be periodic we cannot use central differentiation here)

$$\begin{aligned}\left[\frac{\partial u}{\partial y}\right]^n_{i,j} &= \frac{-u_{i,j+2} + 4u_{i,j+1} - 3u_{i,j}}{2\Delta y} = 0 \\ \left[\frac{\partial u}{\partial y}\right]^n_{i,j} &= \frac{u_{i,j-2} - 4u_{i,j-1} + 3u_{i,j}}{2\Delta y} = 0,\end{aligned}$$

giving two different expressions for the top and bottom boundary

$$\begin{aligned}u_{i,0} &= \frac{4u_{i,1} - u_{i,2}}{3} \\ u_{i,ny-1} &= \frac{4u_{i,ny-2} - u_{i,ny-3}}{3}.\end{aligned} \tag{16}$$

Lastly we need boundary conditions for the energy and density. These are heavily coupled, seen through the equations of state, and are less intuitive than the flow. Here we utilize that the system must be in hydrostatic equilibrium along the border. Through the equations of state and eq. (8) we find an expression for the vertical gradient of the energy as

$$\begin{aligned}\frac{\partial P}{\partial y} &= (\gamma-1)\frac{\partial e}{\partial y} = -\rho g_y \\ \text{inserting} \quad \rho &= (\gamma-1)\frac{\mu m_u}{k_B T} e \quad \text{gives} \\ \frac{\partial e}{\partial y} &= -g_y \frac{\mu m_u}{k_B T} e.\end{aligned}$$

Now we can use the same technique as we did for the horizontal velocity, but setting the approximation equal the gradient for the energy. Using forward differentiation for $j = 0$ we find

$$\begin{aligned}\frac{\partial e}{\partial y}_{i,0} &= \frac{-e_{i,2} + 4e_{i,1} - 3e_{i,0}}{2\Delta y} = -g_y \frac{\mu m_u}{k_B T_{i,0}} e_{i,0} \\ 4e_{i,1} - e_{i,2} &= e_{i,0}\left(3 - 2\Delta y g_y \frac{\mu m_u}{k_B T_{i,0}}\right)\end{aligned}$$

Similarly, using backwards differentiation for $j = ny-1$, we find the boundary conditions for the energy

$$\begin{aligned}e_{i,0} &= \frac{4e_{i,1} - e_{i,2}}{3 - 2\Delta y g_y \frac{\mu m_u}{k_B T_{i,0}}} \\ e_{i,ny-1} &= \frac{4e_{i,ny-2} - e_{i,ny-3}}{3 + 2\Delta y g_y \frac{\mu m_u}{k_B T_{i,ny-1}}}.\end{aligned} \tag{17}$$

Notice the slight differences in signs in the denominator of the two expressions.

With values for the energy at the boundary, we can again use the equations of state to find the density

$$\begin{aligned}\rho_{i,0} &= (\gamma-1)\frac{\mu m_u}{k_B T_{i,0}} e_{i,0} \\ \rho_{i,ny-1} &= (\gamma-1)\frac{\mu m_u}{k_B T_{i,ny-1}} e_{i,ny-1}.\end{aligned} \tag{18}$$

[5] In the final version we apply the Gaussian perturbation *before* initializing ρ and e.

[6] Bullet point 3; Boundaries for u, ρ and e.

3 Method Description

3.1 Discretizing the Governing Equations

We have to discretize the equations in such a way that we can solve them numerically. From eqs. (4) to (7) we expand the velocity terms using the product rule (as the flow is not constant). Here we list the dicretized equations, these expressions are incorporated into the algorithm in 3.3 after we discuss the numerical methods used to find the differentials in section 3.2.

$$\begin{aligned}\left[\frac{\partial \rho}{\partial t}\right]^n_{i,j} &= -[\rho]^n_{i,j}\left(\left[\frac{\partial u}{\partial x}\right]^n_{i,j} + \left[\frac{\partial w}{\partial y}\right]^n_{i,j}\right) \\ &\quad - [u]^n_{i,j}\left[\frac{\partial \rho}{\partial x}\right]^n_{i,j} - [w]^n_{i,j}\left[\frac{\partial \rho}{\partial y}\right]^n_{i,j} \end{aligned} \tag{19}$$

$$\begin{aligned}\left[\frac{\partial \rho u}{\partial t}\right]^n_{i,j} &= -[\rho u]^n_{i,j}\left(\left[\frac{\partial u}{\partial x}\right]^n_{i,j} + \left[\frac{\partial w}{\partial x}\right]^n_{i,j}\right) \\ &\quad - [u]^n_{i,j}\left[\frac{\partial \rho u}{\partial x}\right]^n_{i,j} - [w]^n_{i,j}\left[\frac{\partial \rho u}{\partial y}\right]^n_{i,j} \\ &\quad - \left[\frac{\partial P}{\partial y}\right]^n_{i,j} \end{aligned} \tag{20}$$

$$\begin{aligned}\left[\frac{\partial \rho w}{\partial t}\right]^n_{i,j} &= -[\rho w]^n_{i,j}\left(\left[\frac{\partial w}{\partial y}\right]^n_{i,j} + \left[\frac{\partial u}{\partial x}\right]^n_{i,j}\right) \\ &\quad - [w]^n_{i,j}\left[\frac{\partial \rho w}{\partial y}\right]^n_{i,j} - [u]^n_{i,j}\left[\frac{\partial \rho w}{\partial x}\right]^n_{i,j} \\ &\quad - \left[\frac{\partial P}{\partial y}\right]^n_{i,j} + [\rho g_y]^n_{i,j}. \end{aligned} \tag{21}$$

$$\begin{aligned}\left[\frac{\partial e}{\partial t}\right]^n_{i,j} &= -[e]^n_{i,j}\left(\left[\frac{\partial u}{\partial x}\right]^n_{i,j} + \left[\frac{\partial w}{\partial y}\right]^n_{i,j}\right) \\ &\quad - [P]^n_{i,j}\left(\left[\frac{\partial u}{\partial x}\right]^n_{i,j} + \left[\frac{\partial w}{\partial y}\right]^n_{i,j}\right) \\ &\quad - [u]^n_{i,j}\left[\frac{\partial e}{\partial x}\right]^n_{i,j} - [w]^n_{i,j}\left[\frac{\partial e}{\partial y}\right]^n_{i,j} \end{aligned} \tag{22}$$

3.2 Numerical schemes

In order to solve our equations in the defined grid of cells we have to approximate the derivatives in the governing equations by finite difference methods. For stability we implement two different explicit methods for the spatial terms. In the following, let Q be any quantity we need to spatially differentiate in eqs. (19) to (22).

First, the Forward Time Centred Space (FTCS) scheme which will be used whenever viable. This is for «non-transport» terms, which means terms not multiplied by the flow (u or w), and is implemented as

$$\begin{aligned}\left[\frac{\partial Q}{\partial x}\right]^n_{i,j} &= \frac{Q^n_{i+1,j} - Q^n_{i-1,j}}{2\Delta x} \\ \left[\frac{\partial Q}{\partial y}\right]^n_{i,j} &= \frac{Q^n_{i,j+1} - Q^n_{i,j-1}}{2\Delta y} \end{aligned} \tag{23}$$

Second, for the «transport» terms, we use the upwind differencing scheme. For example, these are terms like $u\frac{\partial \rho}{\partial x}$, where we multiply by a flow factor u. This method is implemented differently depending on the direction of the flow in the actual cell we are calculating in. With positive flow we use information from the cell before the current cell to find the next value, backward differentiation. This is more accurate as we know the previous cell is moving towards the current cell with positive flow. For negative flow we do the opposite[7]. Terms multiplied by the flow u are calculated as

$$\begin{aligned}\left[\frac{\partial Q}{\partial x}\right]^n_{i,j} &= \begin{cases} \frac{Q^n_{i,j} - Q_{i-1,j}}{\Delta x} & \text{if} \quad [u]^n_{i,j} \geq 0 \\ \frac{Q^n_{i+1,j} - Q_{i,j}}{\Delta x} & \text{if} \quad [u]^n_{i,j} < 0 \end{cases} \\ \left[\frac{\partial Q}{\partial y}\right]^n_{i,j} &= \begin{cases} \frac{Q^n_{i,j} - Q_{i,j-1}}{\Delta y} & \text{if} \quad [u]^n_{i,j} \geq 0 \\ \frac{Q^n_{i,j+1} - Q_{i,j}}{\Delta y} & \text{if} \quad [u]^n_{i,j} < 0 \end{cases} \end{aligned} \tag{24}$$

Terms multiplied by the w flow are done similarly, but where the type of differentiation is determined by the sign of w instead of u in the exact same fashion.

3.2.1 Evolving Forward in Time

Using the numerical methods we discussed we can find values for each time derivative in the discretized governing equations. Based on the largest valued time derivative we find an optimal time step Δt for each integration loop, so that the relative change in the density and energy is below a set tolerance. We do the same for the flow parameters, so that no velocity will move the content of a cell past a hole grid point[8]. The tolerance is determined using parameter $p = 1 \times 10^{-2}$, and then demanding that $\Delta t = \delta \cdot p$, where δ is the largest relative change in any of the primary variables.

To find the new value in the next time step we expand the temporal terms in the equations using forward time, and then solve for $Q^{n+1}_{i,j}$

$$\left[\frac{\partial Q}{\partial t}\right]^n_{i,j} = \frac{Q^{n+1}_{i,j} - Q^n_{i,j}}{\Delta t} \tag{25}$$

$$Q^{n+1}_{i,j} = Q^n_{i,j} + \left[\frac{\partial Q}{\partial t}\right]^n_{i,j}\Delta t \tag{26}$$

given that we have the time derivative at the current time step.

3.3 Solving the Equations Numericaly

With the initial quantities established we are ready to lay out the algorithm for solving our system. Here we will directly follow the code in the *hydro_solver* method in *2Dconvection.py*, which is the numerical solver in use. The goal of this algorithm is to evolve each primary variable one step in time, and use the new values to also update the temperature and pressure in the next time step.

1. We first find the direction of flow for the hole box, which is in use in all the following upwind differentials. We do this once at the start of each integration loop to save computation time.

[7] Bullet point 2; Why terms are calculated using upwind differencing.

[8] This is done in a similar way as in project 1 and 2, and will not be elaborated here as the hole theory is given in the problem text.

2. We then find each of the time derivatives (or right-hand-sides RHS) of eqs. (19) to (22) using the appropriate numerical method from section 3.2 on each term.

3. With the time derivatives we can find the optimal time step to evolve our variables and use eq. (26) to find the value in the next time step for ρ and e. Keep in mind we know the time derivative of ρu and ρw, not u and w them self. But as we have just found the value for ρ^{n+1}, the next flow is found by dividing eq. (26) by ρ^{n+1}

$$w_{i,j}^{n+1} = \frac{[\rho w]_{i,j}^{n} + \left[\frac{\partial \rho w}{\partial t}\right]_{i,j}^{n} \Delta t}{\rho_{i,j}^{n+1}} \tag{27}$$

then inserting eq. (21) for the time derivative.[9] Similarly is done for $u_{i,j}^{n+1}$.

4. After updating each primary variable we apply the boundary conditions. This will reset the new calculated values on the upper and lower horizontal boundaries of the computational volume. It is important that we apply the boundary conditions *before* the next step.

5. Lastly update the pressure and temperature to the new time step, *after* the boundary conditions are applied. This is because we don't have boundary conditions for these quantities explicit, but they are applied through the conditions for ρ and e. To find the correct distribution of T and P we again use the equations of state

$$\begin{aligned} P_{i,j}^{n+1} &= (\gamma - 1) e_{i,j}^{n+1} \\ T_{i,j}^{n+1} &= (\gamma - 1) e_{i,j}^{n+1} \frac{\mu m_u}{k_B \rho_{i,j}^{n+1}}. \end{aligned} \tag{28}$$

3.3.1 Example for e[10]

As an example, we will lay out the algorithm for calculating the energy after initialization.

1. Find flow directions over the hole grid.

2. Find the time-derivative of energy using eq. (22). Here all the terms in the first two lines of the equation are calculated using central differencing, eq. (23), and the two terms in the third line using upwind differencing, eq. (24).

3. Find optimal time step based on all time-derivatives, Δt, and evolve the energy to the next value in time $n+1$ using eq. (26).

3.3.2 Gaussian Perturbation[11]

In the final version of our system, in order to start convection in our hydrostatic equilibrium situation, we perturb the initial temperature with a Gaussian perturbation. This is done in the initialization, before the algorithm discussed in section 3.3 takes place. After calculating the initial temperature and pressure distribution using eqs. (10) and (12), but before initializing the energy and density, we add the perturbation to the values of temperature.

The perturbation, say λ, is created as a 2D Gaussian bell mesh grid

$$\lambda = \exp\left[-\frac{1}{2}\left(\frac{x - x_{avg}}{\sigma_x} \right)^2 - \frac{1}{2}\left(\frac{y - y_{avg}}{\sigma_y} \right)^2 \right] \tag{29}$$

where x_{avg} and y_{avg} marks the center of the perturbation, x and y spans the computational volume and σ_x and σ_y is the full width at half maximum of the perturbation.

Initially we implement a single positive perturbation in the center of the computational box, $x_{avg} = 6\,\mathrm{Mm}$ and $y_{avg} = 2\,\mathrm{Mm}$. We choose equal standard deviations in both directions for a circular bell function, $\sigma_x = \sigma_y = 8 \times 10^5$. Then λ is added to the initial temperature like

$$T_{\text{perturbed}} = T_{\text{inti}} + \alpha T_{\odot} \lambda$$

where α controls the amplitude of the perturbation.

The method is implemented in such a way that we can add any number of perturbations equally spaced along the horizontal direction to see how they influence each other. For cases with multiple perturbations we also chose alternating positive and negative α values. This way every other perturbation create a rising and sinking parcel, or a parcel with higher and lower temperature than the surroundings.

3.4 Benchmarking and Implementation Progress[12]

The progress of putting this code together has been quite involved, and the order of operations are important. In this section we will try to sum up how this progress was dealt with, and how we benchmarked the code along the way.

First we worked on how to implement the initialization of all the variables following the equations established in section 2.2, without perturbation. These equations are highly governed by the equations of state and hydrostatic equilibrium. To benchmark that the initial distributions made sense physically we implemented a simple 1D plot of the vertical component of each variable[13]. This way we could confirm that all the variables were increasing from the surface of the cell down to the bottom.

Before we can do any more benchmarking we have to implement a lot of functionality. All the methods for differenciation were implemented together with the method for finding flow directions. These methods utilize the convenient function *numpy.roll()* from the numpy library (Oliphant, 2019). This way we incorporate periodic boundary conditions in both directions[14] and are able to calculate the differentials over the hole mesh grid in a vectorized fashion for efficiency.

Then the method for applying the boundary conditions were implemented. Here the order of operations are important for the density and energy, while the conditions for the

[9] Bullet point 2; Algorithm for $w_{i,j}^{n+1}$.

[10] Bullet point 2; Algortihm for $e_{i,j}^{n+1}$, with differentiation method. Why answered in section 3.2.

[11] Bullet point 5; Implement Gaussian perturbation, possible to turn off and on.

[12] Bullet point 4; Summary/continuation of code explaination.

[13] Initial conditions test cmds; *sanity init*.

[14] Note that the vertical boundaries are not periodic, but controlled by the boundary conditions.

velocities are independent on other quantities. We first calculate the boundary values for the energy using eq. (17), and then use the new updated values in eq. (18) to set the values for the density.

The last step before we implemented the hydrodynamical solver was to find the optimal time step discussed in section 3.2.1. This is done in a similar way as we did in the previous term projects. For efficiency we included a few contingencies. To remove possibilities of zero divisions we removed all cases of infinity in the calculation of δ, and made it strictly positive. In addition, to prevent the calculation to go too slow we implement a minimum time step of $\Delta t = 1 \times 10^{-7}\,\mathrm{s}$. For the hydrostatic case where the relative change in variables are very low we also force a max time step of $\Delta t = 0.1\,\mathrm{s}$.

With these implementations in place we started working on the solver itself, which is covered in detail in section 3.3. As mentioned, here we really had to mind the order of operations, which is covered in said section. To sum up, it's important that we calculate the energy and density first. Then the flow, using appropriate values for the density in the denominator and numerator as demonstrated in eq. (27). Next we apply the boundary conditions, importantly *before* calculations of temperature and pressure.

With the completed solver we test the code against our main benchmark, hydrostatic equilibrium[15]. Here we run the simulation over $60\,\mathrm{s}$, with unperturbed initial conditions. As we know, the hydrodynamical system should be unchanging when it's in hydrostatic equilibrium, so if we have done everything right we should have a static situation. As the system is so unstable this is close to impossible to get perfect, but we settle with a low tolerance for instability. This is actually easier to control by studying the vertical velocity distribution than the temperature.

We are now finally ready to implement the perturbation in the initial conditions as discussed in 3.3.2, and run the final simulations. In the initial implementation of this code, we only made a single positive perturbation in the middle of the computational volume. This way we could test that the method worked as intended, as well as how to make it better or more interesting, before extending the method to create arbitrary many alternating positive and negative perturbations along the x-direction.

The last addition to the code is animating the convective flux through the volume[16]. This is done using the provided module. We choose to look at the averaged convective flux through each «cross section» of the volume in the vertical direction. This way we can see at which depths, and at what times, most of the energy transportation is happening. This is done as

$$[F_C]_j^n = \frac{\sum_i [e \cdot w]_{i,j}^n}{nx} \tag{30}$$

where we average over all the horizontal cells for each j. We could also look at the cross sections in the horizontal direction, but along the horizontal the system is symmetrical, and so is less interesting in our opinion.

[15]Sanity test cmds; *sanity animate*

[16]Bullet point 7; Convective flux for each depth. Cmds; *basic flux animate* or *snap*.

4 Conclusion and Final Result

The final result from our simulations are movies of the 2D distributions of the different variables, we choose to focus on the temperature. To discuss the results here we include snapshots from the movies at specific times. Note that for most of the following discussion we could have chosen to focus on the pressure and energy, instead of temperature and density, as these quantities are so linearly coupled through the equations of state.

In the movies and snapshots we have tried to reach a middle ground between fontsize and arrow scale/density in order to show the results as clearly as possible at all times through the simulation. We chose to use quite dense and small arrow-scales in order to display the convection cells detailed, while still being able to run the simulation far enough that we could see where it broke down. Unfortunately this causes some of the arrows to be very small, especially in the beginning of the simulations, and it's even harder to see the directions in the still snapshots than in the movie. But with some physical intuition and expectations (and zoom if viewed digitally) there should be no problem interpreting the vector field. Note also that as our numerical approach is so unstable, we are not able to simulate the situation long enough for the distribution in the temperature to change much visibly. Instead we will interpret the results by looking at the induced velocity fields, which eventually would cause the distribution of the other variables to shift around. In the following we will talk about how the parcel moves, as if it moved together with the velocity field.

First, we will discuss the most basic case of perturbation[17], a single positive perturbation at the center of the distribution. We also include a more interesting perturbation[18] in fig. 3, but where all of the physics are the same. We will therefore discuss the simple case in detail, as there are less happening from frame to frame. Snapshots of the time evolution for the basic situation included in figs. 1a to 1d. Here we see the behavior as expected from a hot parcel of gas. At the same time we can study fig. 2 to see how the convective flux is related to the 2D distributions.

In the first figure, 1a, we see the initial distribution after the perturbation is added. The bell shaped increase in the temperature causes the contour-plot (or color-plot) to increase with a smooth bulk in the middle, following a normal distribution in both directions. Here we have close to zero velocity field, from the initialization. This is also evident in fig. 2a, where we see a close to zero flux.

As the initial velocity is set to zero (as seen in 1a), the velocity field takes some time to catch up, shown in 1b is the situation one minute in. We see how the increased temperature infers a positive flow in the vertical direction, consistent with how we know that hot gas rises in a surrounding cooler medium. This can be further explained by looking at the equations of state, where we know the temperature and density to be linearly inverse. The surrounding medium is initialized in hydrostatic equilibrium, so it will not be accelerated, and the velocity field stays zero. The hotter gas will

[17]Movie cmds; *basic animate*. Snapshots cmds; *basic snap*.

[18]Movie cmds; *viz animate*. Snapshots cmds; *viz snap*.

have a lower density, and therefore is pulled down less by the gravitational attraction breaking the hydrostatic equilibrium. The lower gravitational pull causes a net positive acceleration, inducing a positive velocity field, which eventually results in the parcel rising through the medium. Here we are also close to the maximum in convective flux in fig. 2b, as all the velocity is in positive direction. Note that the maximum is located below the middle depth, which is consistent with how the density (and therefore energy from equation of state) is higher at the bottom of the box.

Then, at time $02 : 15$ in 1c, the parcel has reached the surface, which is consistent with hot granules as we observe them on the sun. The hot gas will radiate away energy, and so is cooled off. Here we see how the velocity field start getting a horizontal component near the surface. This is caused by the hot gas still rising below it, pushing the cooled gas to each side. The velocity field is still mostly in the middle and the upper part of the parcel. This is consistent with fig. 2c, where we see the maximum of the vertical flux moving to a higher y-value.

Finally in 1d, the convection circle is completed. Here the cooled off gas falls back down, by the same argument we used for it rising initially, only reversed. The cooler gas has a higher density, and is therefore accelerated down again. Here it is warmed back up to the surrounding temperature at the bottom of the computational volume. The cooled gas still falling above it causes horizontal velocity to each side eventually rejoining the velocity field in the middle which is still in positive vertical direction. The averaged vertical flux is all negative at this point, as there are so many more cells with a negative vertical flow than positive. If we were able to simulate the situation for longer, we would see how the convective flux start changing back to being positive. This process will oscillate back and forth from being negative to positive, as parcels rise and falls back down. From the movie we see how the flux is starting to grow again after the last snapshot.[19]

4.1 Comments, Final Thoughts and Future Perspectives

In this project we have successfully written a 2D hydrodynamics code, able to visualize any of the involved quantities in an informative and beautiful way. We have studied a highly unstable system of partial differential equations, and how to solve them numerically using non-trivial, but still simplistic, methods. We have also gotten new and deeper insight into how coupled all these quantities are through the equations of state. The results has helped us visualize the theory we've covered in lectures, so that we understand the simple form of 2D convection.

We have learned more about boundary conditions, and how important they are for confined systems like this. The process of finding the correct boundary conditions were interesting and challenging, as we had little information to go on. This forced us to do a lot of healthy critical physical thinking. Also being able to test how slightly different *wrong* boundaries changed the behavior of the system was educational.

We've had few problems all in all. That doesn't say the different parts of the project haven't been difficult, but in a good way. Our biggest issue was in the very beginning of the project, whit a lot of information about the system and the problem to absorb. Here we had to figure out how to orient the system and the model in a physically, a numerical and a graphical sense. Through the direction of gravity and the vertical pressure gradient we established how the situation had to behave physically, and then translated that into numerical indexation in Python. Lastly we had to think about how this was handled by the provided visualization module, which flips the vertical axis. After obtaining good understanding of these concepts the rest of the project became much easier. It was then just a matter of writing the code in an ordered fashion, utilizing modules for as much as possible. After the full implementation was complete, following section 3.4, the simulation worked as intended surprisingly fast with little to no bugs.

As mentioned our model is not complete, as we are not able to simulate turbulence. This could be improved by including the viscous stress tensor in our governing equation for the momentum. The stress tensor governs internal friction in the dynamical system. The friction is present between neighboring flow streams of mass with different velocities. We can see how our current simulations struggles with this near the edges where the boundary conditions creates shock waves of unphsycial behavior, and where there are large relative differences in the velocity field over small areas as seen fig. 3d. In addition we could include diffusion, as we know from the density gradient there should be some contribution from diffusion in our dynamical system. These inclusions are not trivial however, and would probably require some higher ordered numerical solvers. We could also have implement higher order solvers, both temporal and spatial, without the tensor or diffusion. If so the simulations would probably stay stable for longer, as especially the central differential method is unconditionally unstable.

Acknowledgments

The author would like to thank Helle Bakke and Boris Vilhelm Gudiksen for an exciting and interesting course, with some inspiring lectures and projects. Thanks to my colleagues for discussions during all the terms projects, in particular Jon Kristian Dahl, Bernhard Nornes Lotsberg, Mats Ola Sand, Nils-Ole Strutzer and Håkon Tansem.

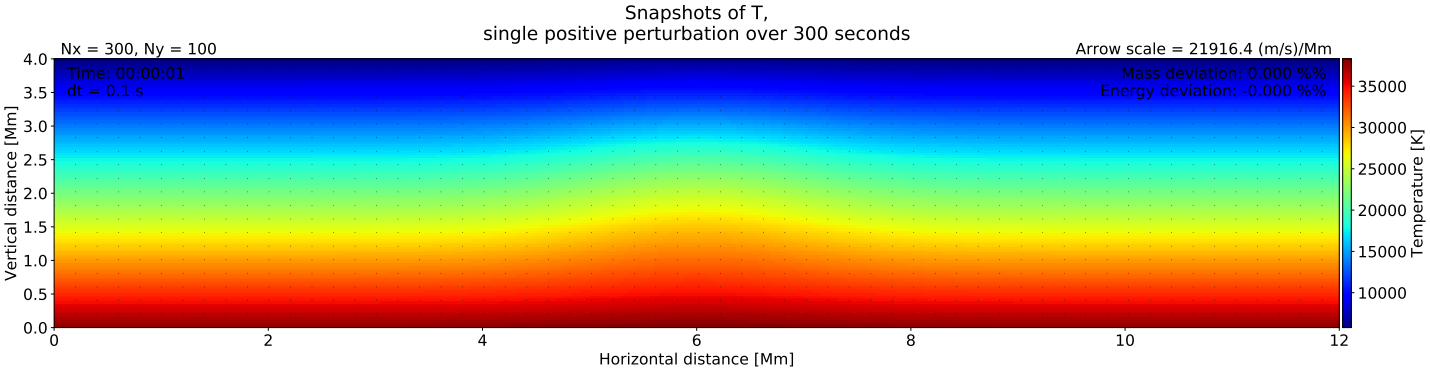
References

[Frogner 2018] FROGNER, Lars: *Github repository for Visualiser module*. May 2018. – URL `https://github.com/lars-frogner/FVis`. – Last checked 13.05.2019

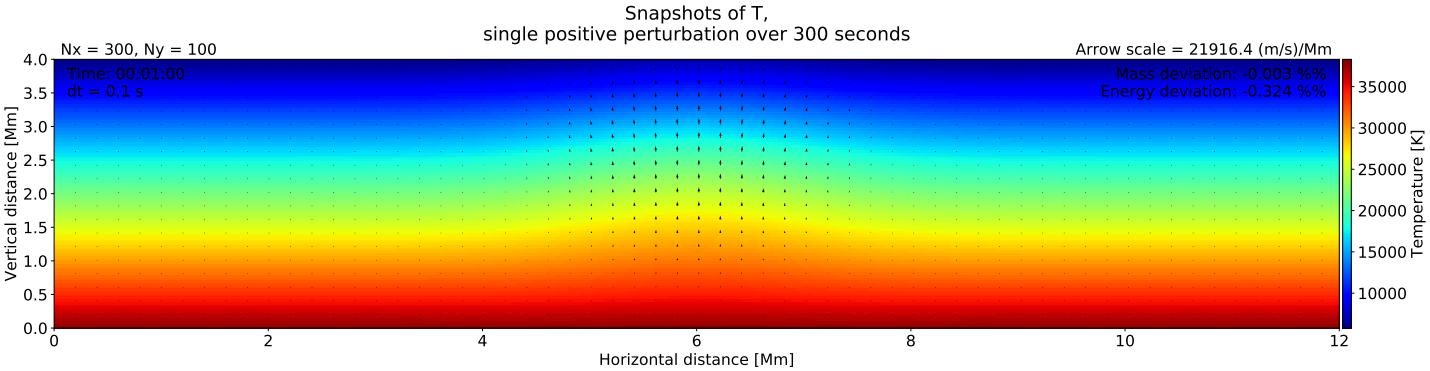
[Oliphant 2019] OLIPHANT, Travis E.: *Numpy.roll() Documentation*. January 2019. – URL `https://docs.scipy.org/doc/numpy/reference/index.html`. – Last checked 13.05.2019

[19] The delivery includes 7 movies; the sanity test (also for w), two 2D animations of the basic perturbation for two endtimes together with corresponding videos of the flux, and last a animation of the case with five perturbations. The shorter videos are discussed in this paper, while in the longer ones the cyclic behavior of the flux is visible as well as how the 2D distribution becomes unstable.

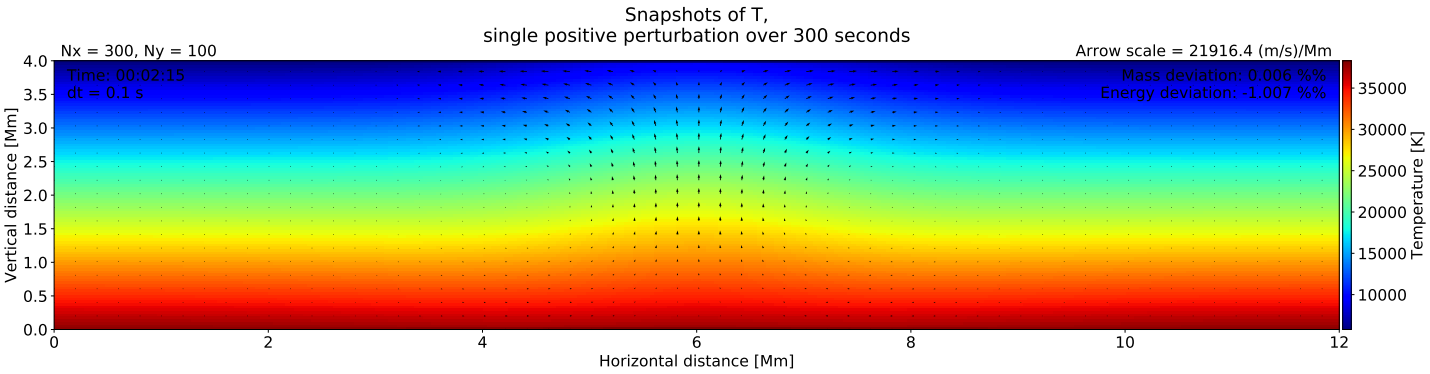
Figure 1: Distribution of temperature shown in colors, velocity vector field shown as arrows. Figures 1a to 1d shows the time evolution of the system. Initial conditions are perturbed with a single positive circular Gaussian perturbation, which acts as a parcel of hot gas rising through the medium. The hot pocket eventually reach the surface and cools down, before it is pushed to the side and falls back down, creating a 2D convection cell. Further discussion of the results in section 4.



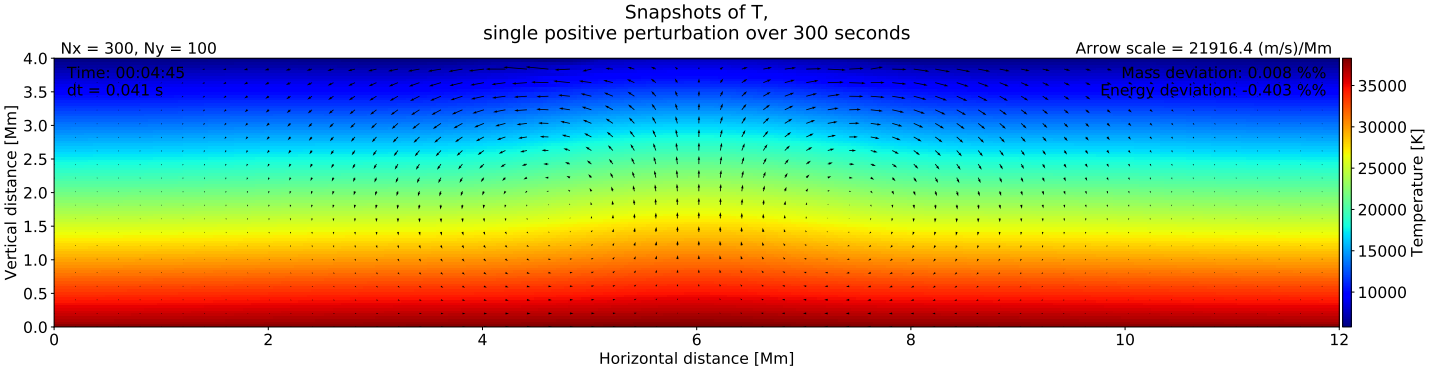
(a) The initial distribution, before the perturbation give non-zero velocity. The increased temperature is visible as the bulk in the y-direction. Notice the zero (or close to zero) value of the velocity field over the hole volume.



(b) One minute into the simulation. Here we see the effects of the hot parcel, giving rise to positive vertical flow in the center of the box pushing the parcel towards the surface.

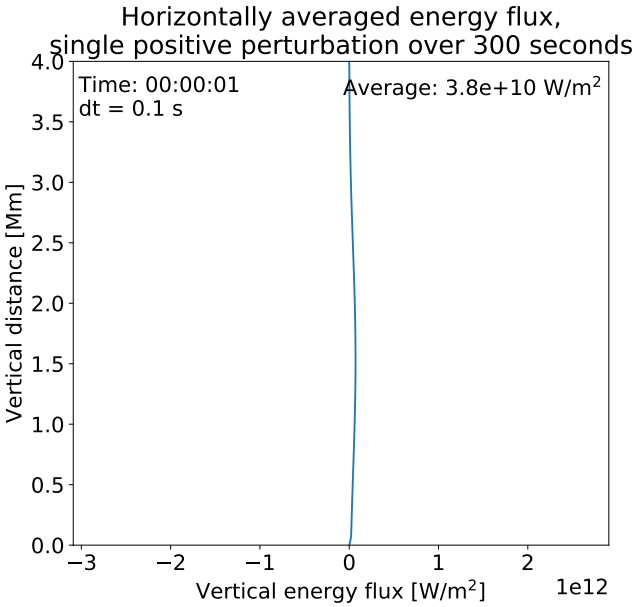


(c) $02:15$ into the simulation and we can see how the medium is pushed out to each side horizontally when the parcel reaches the surface. This is as expected, as we know these can be observed as granules on the Sun.

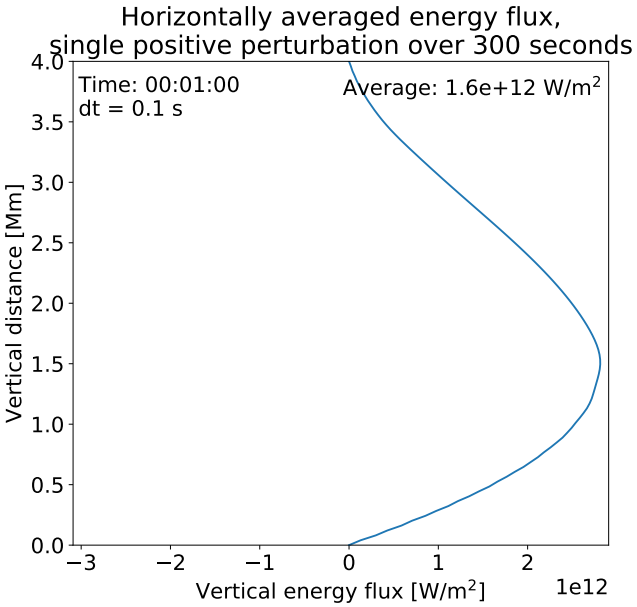


(d) The last snapshot from the simulation, at $04:45$. Here we see how the cooled gas that has been pushed to the sides falls back down. This process concludes the convection «circle», and is visible in the pleasant circular velocity fields at both sides of central hot rising region.

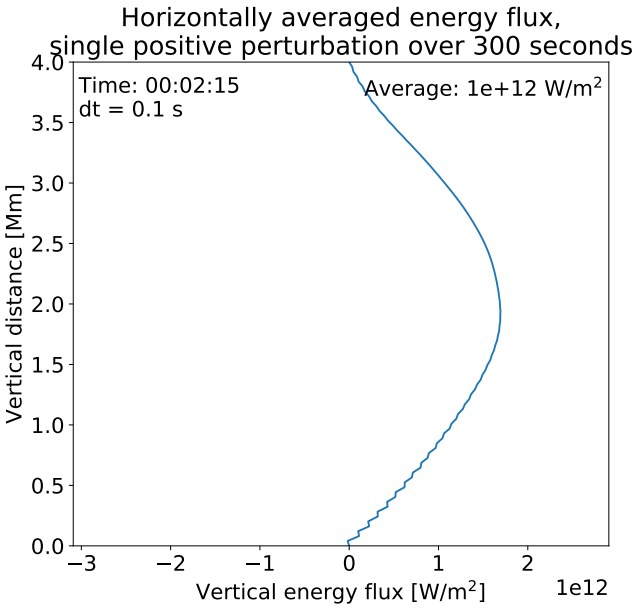
Figure 2: Horizontally averaged vertical energy flux of the basic perturbation situation shown in fig. 1, snapshots of corresponding times in figs. 2a to 2d. Here we see how the vertical energy transportation changes with the changes in velocity fields discussed in section 4. Note the large amplitude of the flux, this is caused by the very large value of e.



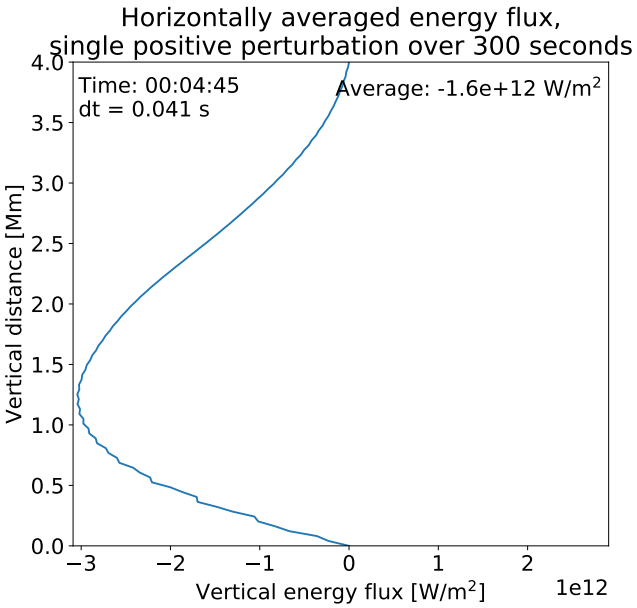
(a) Initial situation, close to hydrostatic equilibrium.



(b) Close to the maximum of the flux, at time $01:00$. Consistent with what we saw in fig. 1b, here we still only have positive vertical flow, causing a large positive vertical convective flux.

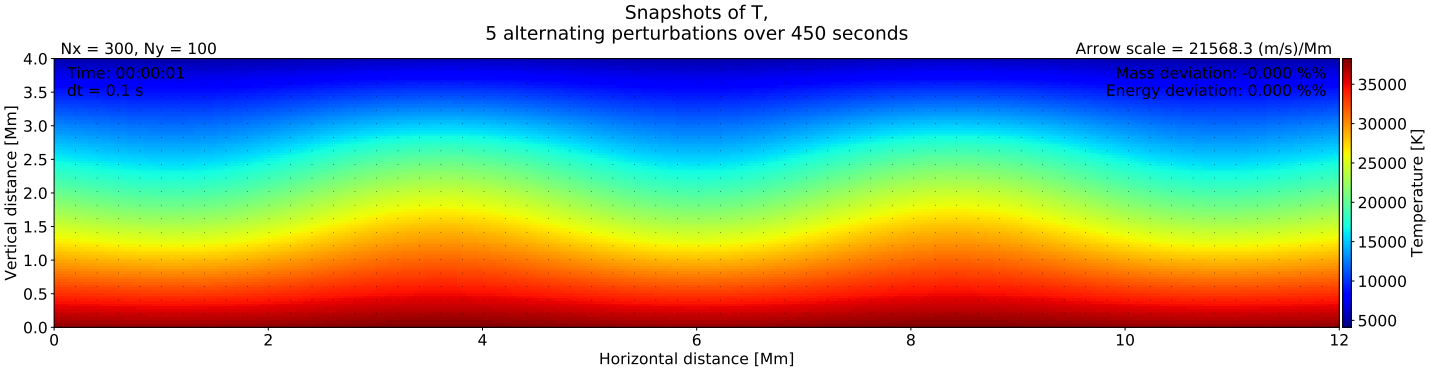


(c) Two minutes and 15 seconds in and we see the maximum of the flux being moved to a higher y-value. Caused by the increase in velocity field, which increases in a circular path before it completes itself, as seen in fig. 1. Also, as the parcel rises through the medium, we get higher values of e closer to the surface.

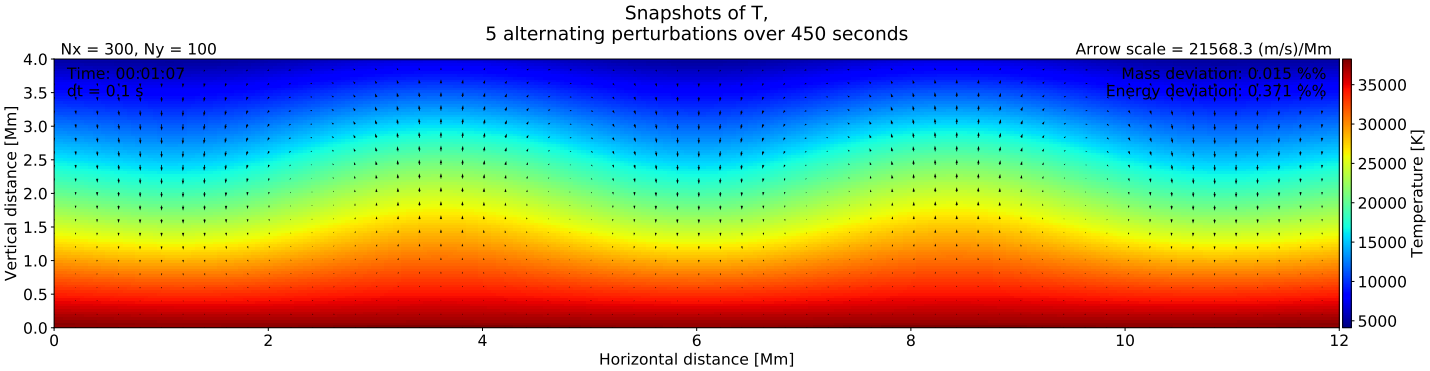


(d) Finally we have a minimum value of the flux, being negative at all depths. This is caused by the parcel falling back down at both sides of the rising pocket. Now the areas where negative flow is happening are so much larger than the positive part, and so the averaged flux across each depth is negative.

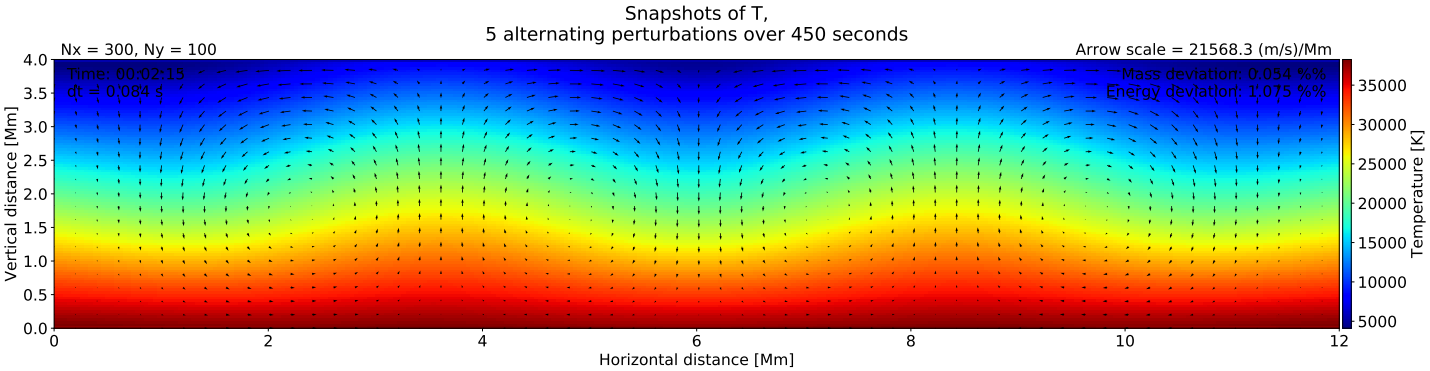
Figure 3: Distribution of temperature shown in colors, velocity vector field shown as arrows. Figures 3a to 3d shows the time evolution of the system. Initial conditions are perturbed with 5 circular Gaussian perturbations equally spaced along x-direction, with altering sign of the perturbation. This creates both parcels rising and falling in the medium, accelerating the process of completing the convection «circles». We refer to the figure captions in fig. 1 and section 4 for a more detailed discussion of the result.



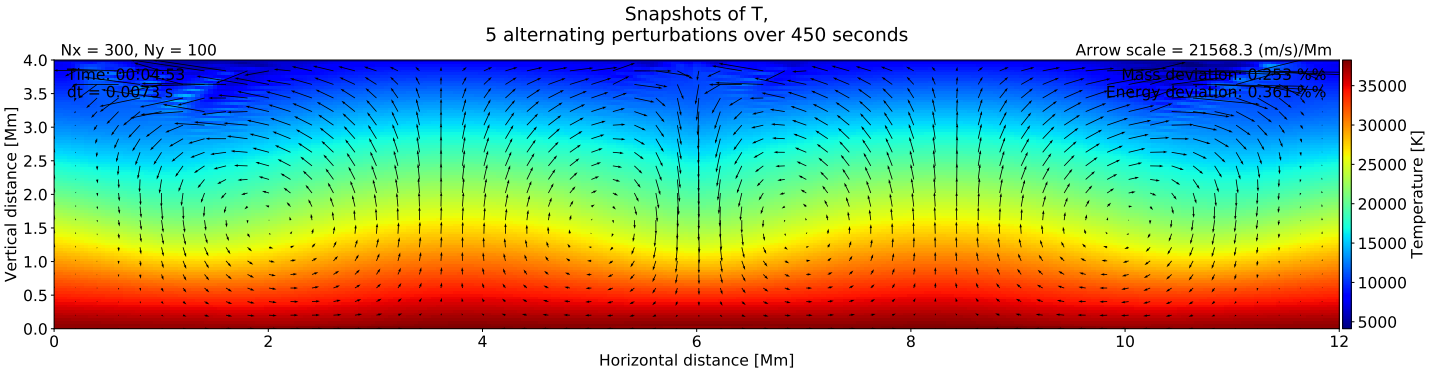
(a) Initial temperature distribution. Compared to fig. 1a the perturbation seems larger, but that is only caused by the neighboring negative perturbation.



(b) One minute and 7 seconds into the simulation and we have five rising and sinking parcels. We can see compared to fig. 1b the process of completing the convection cells are already visible.



(c) Already at two minutes and 15 seconds we can see that the convection cells are fully formed.



(d) Here we show how the simulation starts to break down after a while. Unphysical behavior is present near the edges of our computational volume, especially in each corner and at the top middle area where the two rising parcels meet.