Geophysical Laboratory I, academic year: 2022/2023

# Scalar transport and bulk condensation in a synthetic cloud flow

Agnieszka Makulska,
registration number: 417936

tutor: dr Gustavo Abade

date of submission: May 13, 2023

# 1 Introduction

This exercise concerns the numerical calculation of scalar advection in a synthetic cloud flow. Advection is calculated for the potential temperature, water vapor mixing ratio and liquid water mixing ratio in a steady, two-dimensional flow, that mimics the air flow in the Atmospheric Boundary Layer. Condensation and evaporation are implemented using an instantaneous saturation adjustment scheme. The time evolution of scalar fields of potential temperature, water vapor mixing ratio and liquid water mixing ratio is assessed. Vertical profiles of horizontally-averaged potential temperature, water vapor mixing ratio and liquid water mixing ratio are plotted.

# 2 Formulation of the problem

It is assumed that the air is composed of a binary gas phase (dry air and water vapor) and a condensed phase (liquid water). The water content is described by two quantities: water vapor mixing ratio $r_v$ and liquid water mixing ratio $r_l$. The thermodynamic state is described by the potential temperature $\theta$. The goal of this exercise is to assess the time evolution of $r_v$, $r_l$ and $\theta$, given their initial distribution, for a prescribed steady-state velocity field $\boldsymbol{u}(\boldsymbol{r})$. Condensation and evaporation are treated using a saturation adjustment scheme. Equations of momentum balance and continuity for scalar quantities $r_v$, $r_l$ and $\theta$ can be reduced to a single balance equation:

$$\frac{\partial \phi}{\partial t} = -\dot{\nabla}(\boldsymbol{u}\phi) + S_\phi, \tag{1}$$

where $\phi = [r_v, r_l, \theta]$ represents scalar quantities, $\boldsymbol{u}$ is the velocity of the flow, $S_\phi$ represents source terms:

$$S_\phi = \left[-C, C, \frac{L_v}{c_p \Pi}\right], \tag{2}$$

which are sources for $r_v$, $r_l$ and $\theta$ respectively. $C$ is the condensation rate, $L_v$ is the latent heat of vaporization, $c_p$ is the specific heat of dry air and $\Pi$ is the Exner function:

$$\Pi = \left(\frac{p}{p_0}\right)^\kappa, \tag{3}$$

where $p$ is the pressure, $p_0 = 1000$ hPa is the pressure at the reference level, $\kappa = R_d/c_p$ is a constant, where $R_d$ is the specific gas constant for dry air. The profile of pressure $p(z)$ can be calculated, depending on profiles of $\theta(z)$ and $r_v(z)$:

$$p(z) = p_0 \left[\, 1 - \frac{\kappa g (1 + r_v(z))}{\theta(z)(R_d + R_v r_v(z))}\; z \,\right]^{1/\kappa}, \tag{4}$$

where $z$ is the altitude, $g$ is the gravitational acceleration and $R_v$ is the specific gas constant for water vapor.

Variables of the model are calculated on a grid of points throughout the region $0 \le x \le L$, $0 \le y \le H$, including $I$ grid points in the x-direction and $J$ grid-points in the y-direction. The grid spacings are $\Delta x = L/I$ and $\Delta y = H/J$. In this exercise, $L = H = 1.5$ km. A synthetic, steady, two-dimensional velocity field $\boldsymbol{u}(x, y)$ is considered, that mimics the air flow in the Atmospheric Boundary Layer. It is assumed that the flow $\boldsymbol{u}(x, y)$ is incompressible and it may be written in terms of the streamfunction $\psi(x, y)$. The large-scale flow in the Atmospheric Boundary Layer is modeled by a streamfunction of the form:

$$\psi(x, y) = v_{max} \frac{L}{\pi} cos\left(2\pi \frac{x}{L}\right) sin\left(\pi \frac{y}{H}\right), \tag{5}$$

where $v_{max} = 1$ m/s is the maximum value of vertical velocity. Figure 1 shows the velocity field which corresponds to the streamfunction described by equation 5.

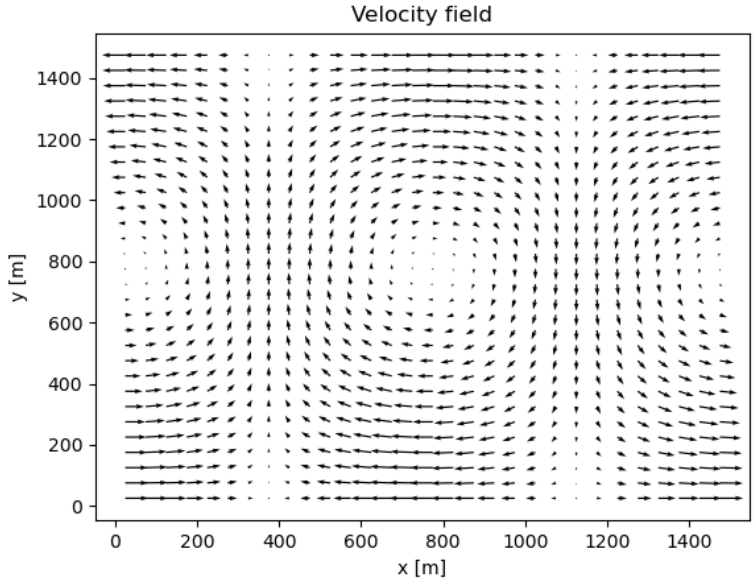


Figure 1: Velocity field considered in the exercise, which mimics the large-scale Atmospheric Boundary Layer flow.

# 3 Advection

Advection is calculated with a donor-cell scheme. During the advection step, scalar quantities satisfy equation 1, with source terms $S_\phi = 0$. In the one-dimensional donor cell scheme,

$$\phi_j^{n+1} = \phi_j^n - [F(\phi_j^n, \phi_{j+1}^n, u_{j+\frac{1}{2}}) - F(\phi_{j-1}^n, \phi_j^n, u_{j-\frac{1}{2}})], \tag{6}$$

where $\phi_j^n$ is the scalar value at the location $j$ in the time step $n$, $\phi_j^{n+1}$ is the scalar value at the location $j$ in the time step $n+1$, $u_{j+\frac{1}{2}}$ is the velocity at the location $j+\frac{1}{2}$, $u_{j-\frac{1}{2}}$ is the velocity at the location $j-\frac{1}{2}$ and $F$ is the flux function defined as:

$$F(\phi_L, \phi_R, u) = \left[\frac{u+|u|}{2}\phi_L + \frac{u-|u|}{2}\phi_R\right]\frac{\Delta t}{\Delta x}, \tag{7}$$

where $\Delta t$ is the time step and $\Delta x$ is the grid spacing. In this exercise, the donor-cell scheme described by equations 6 and 7 is generalised into two dimensions.

# 4 Saturation adjustment scheme

In the saturation adjustment scheme, it is considered that the rate of condensation is such that the air parcel is always at saturation, that is:

$$r_v = r_{vs} = \frac{\epsilon e_s(T)}{p - e_s(T)}, \tag{8}$$

where $r_{vs}$ is the mixing ratio of water vapor for saturated conditions, $e_s(T)$ is the saturation vapor pressure, $p$ is the total pressure and $\epsilon = R_d/R_v$ is the ratio of specific gas constants of dry air and water vapor. The saturation vapor pressure $e_s(T)$ is calculated with the Clausius-Clapeyron equation, with the assumption of a constant latent heat of vaporization $L_v$:

$$e_s(T) = e_0 \; exp\left(-\frac{L_v}{R_v}\left(\frac{1}{T} - \frac{1}{T_0}\right)\right), \tag{9}$$

where $T$ is the temperature and $e_0 = 6.11$ hPa is the saturation vapor pressure at the reference temperature $T_0 = 273.15$ K.

A step of the saturation adjustment scheme is defined as:

$$r_{v,ij}^{n+1} = r_{v,ij}^{*} - \Delta, \tag{10}$$

$$r_{l,ij}^{n+1} = r_{l,ij}^{*} + \Delta, \tag{11}$$

$$\theta_{ij}^{n+1} = \theta_{ij}^{*} + \frac{L_v}{c_p \Pi}\Delta, \tag{12}$$

where $[r_{v,ij}^{n+1}, r_{l,ij}^{n+1}, \theta_{ij}^{n+1}]$ are values of the water vapor mixing ratio, liquid mixing ratio and potential temperature at the location $(i, j)$ in the time step $n+1$, $[r_{v,ij}^{*}, r_{l,ij}^{*}, \theta_{ij}^{*}]$ are provisional values obtained by using the advection algorithm on values from the time step $n$. The value of $\Delta$ is determined from the saturation condition: if $r_{v,ij}^{*} < r_{vs}$, then $\Delta = 0$. Otherwise, the saturation condition must be met:

$$r_{v,ij}^{n+1} = r_{vs}(\theta_{ij}^{n+1}, p_{ij}), \tag{13}$$

which gives an explicit equation for $\Delta$:

$$r_{v,ij}^{*} - \Delta - r_{vs}(\theta_{ij}^{*} + \frac{L_v}{c_p \Pi}\Delta, p_{ij}) = 0. \tag{14}$$

Equation 14 is solved numerically with the Newton-Raphson method.

# 5 Results

It is assumed that initial profiles of $r_v$, $r_l$ and $\theta$ are constant and equal to: $r_{v0} = 17$ g/kg, $r_{l0} = 0$ g/kg and $\theta_0 = 298$ K. The profile of pressure is calculated with equation 4. The maximum vertical velocity is 1 m/s, the grid is a square of the size of 1500 m, with spacing $\Delta x = \Delta y = 50$ m. The time step is equal to 0.1 s.

The time evolution of scalar fields $r_v$, $r_l$ and $\theta$ is assessed. One step of the evolution consists of the advection step and the saturation adjustment step. In the advection step, provisional values of scalar fields are calculated with the donor-cell scheme described by equations 6 and 7. In the saturation adjustment step, scalar fields are updated according to equations 10, 11 and 12, with the value of $\Delta$ assessed numerically from the saturation condition, using the Newton-Raphson method.

Figure 2 shows vertical profiles of horizontally-averaged values of $r_v$, $r_l$ and $\theta$, assessed after 200 steps of evolution (which correspond to 20 seconds).

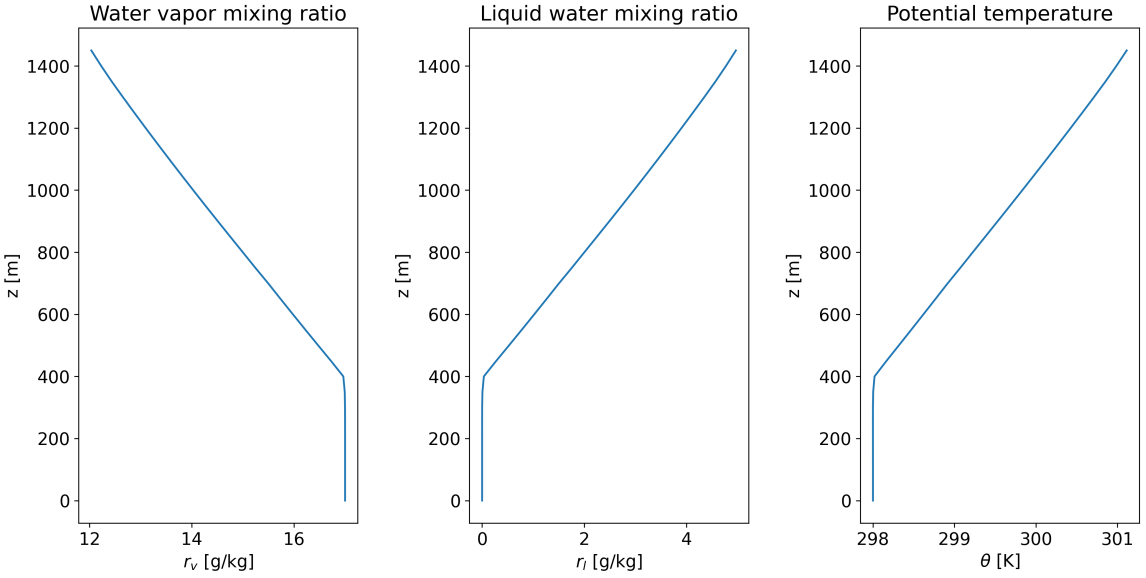


Figure 2: Vertical profiles of horizontally-averaged values of $r_v$, $r_l$ and $\theta$ after 200 steps of evolution.

Below the altitude of roughly 400 m, profiles of average $r_v$, $r_l$ and $\theta$ are constant and equal to their initial values (no liquid water is present). Above the altitude of 400 m, the average mixing ratio of liquid water and average potential temperature increase linearly with height and the average mixing ratio of water vapor decreases linearly with height. Between altitudes of 400 m and 1500 m, the average mixing ratio of liquid water increases by 5 g/kg, as the average mixing ratio of water vapor decreases by 5 g/kg and the average potential temperature increases by 3 K.

Figure 3 shows the field of liquid water mixing ratio assessed after 200 steps (20 seconds) of evolution.

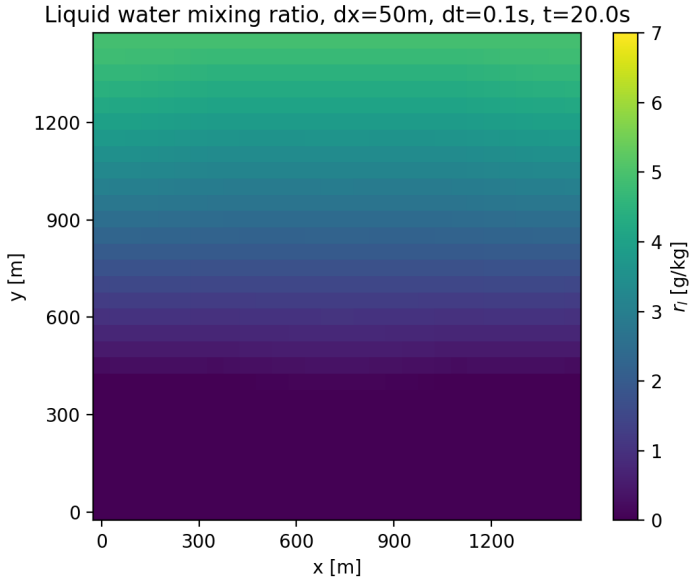


Figure 3: The field of liquid water mixing ratio assessed after 200 steps (20 seconds) of evolution.

The mixing ratio of liquid water does not depend significantly on the horizontal coordinate. Up to the altitude of 400 m it remains constant and above the altitude 400 m it increases with altitude, up to roughly 5 g/kg.

# 6 Conclusions

Scalar advection of potential temperature, water vapor mixing ratio and liquid water mixing ratio was calculated numerically in a steady, synthetic cloud flow, that mimics the air flow in the Atmospheric Boundary Layer. Condensation and evaporation was performed with an instantaneous saturation adjustment scheme. The evolution of scalar fields of potential temperature, water vapor mixing ratio and liquid water mixing ratio was assessed. Profiles of horizontally-averaged potential temperature, water vapor mixing ratio and liquid water mixing ratio were plotted after 200 steps of evolution. It was concluded that below a certain altitude, profiles of potential temperature, water vapor mixing ratio and liquid water mixing ratio are constant (no liquid water is present) and above that altitude profiles of liquid water mixing ratio and potential temperature increase linearly with height, as the profile of water vapor mixing ratio decreases linearly with height.

# 7 Numerical code in Python

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import imageio
v_max=1.
L=H=1500
```

```
dx=dy=50
dt=0.1
I=int(L/dx)
J=int(L/dy)
Lv=2.5*10**6
Rv=461
kappa=0.286
cp=4184
epsilon=0.622
e0=611
T0=273
p0=100000
Rd=287
g=9.81

def pressure(z,rv,theta):
    return p0*(1-kappa*g*(1+rv)*z/(theta*(Rd+Rv*rv)))**(1/kappa)
def exner(p):
    return (p/p0)**kappa
def e_s(theta,p):
    T=np.multiply(exner(p),theta)
    return e0*np.exp(-Lv/Rv *(1/T - 1/T0))
def r_vs(theta,p):
    return epsilon*e_s(theta,p)/(p-e_s(theta,p))
def F(delta,p,rv,theta):    #function used in the Newton method
   return r_vs(theta+ np.multiply(Lv/(cp*exner(p)),delta),p)+delta-rv
def Fp(delta,p,theta):      #defivative of the function used in the Newton method
    theta_p=theta+np.multiply(Lv/(cp*exner(p)),delta)
    T_p=np.multiply(exner(p),theta_p)
    return epsilon*p/(p-e_s(theta_p,p))**2*e0*Lv**2 /(Rv*cp*T_p**2)*np.exp(-Lv/Rv *(1/T-1/T0))+1

def newton_method(delta0,p,rv,theta):
  delta=np.array(np.ones((I,J))*delta0)
  delta=np.reshape(delta,(1,I,J))
  for n in range(10**2):
    h=-F(delta[n],p,rv,theta)/Fp(delta[n],p,theta)
    if np.any(np.isnan(delta[n]+h)):
        break
    delta=np.concatenate((delta,np.reshape(delta[n]+h,(1,I,J))))
  return delta[-1]


def streamfunction(x,y):
    return (v_max*L/np.pi)*np.cos(2*np.pi*x/L-np.pi/2)*np.sin(np.pi*y/H+np.pi/2)
psi=np.ones((I+1,J+1))
psi=np.fromfunction(lambda i,j: streamfunction(i*dx,j*dy), (I+1,I+1))
psi=np.transpose(psi)
u= (np.roll(psi,-1,axis=1)-psi)/dy
v= -(np.roll(psi,-1,axis=0)-psi)/dx
u=u[:-1,:-1]
v=v[:-1,:-1]
#Boundary conditions:
v[0,:]=v[0,:]-v[0,:]
v[-1,:]=v[-1,:]-v[-1,:]
x=[i*dx for i in range(0,I)]
y=[i*dy for i in range(0,J)]
xv,yv=np.meshgrid(x,y)
plt.quiver(xv,yv,u,v)
```

```
plt.title('Velocity field')
plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.show()

def flux(phi_L,phi_R,u):
    return (phi_L*(u+np.abs(u))/2 + phi_R*(u-np.abs(u))/2)*dt/dx
theta=np.ones((I,J))*298.
rv=np.ones((I,J))*0.017
rl=np.zeros((I,J))

T=200
for t in range(T):
    for i in range(1,I-1):
        for j in range(J):
            theta[i][j]= theta[i][j]-(flux(theta[i][j],theta[(i+1)%I][j],u[(i+1)%I][j]) -
                                flux(theta[(i-1)%I][j],theta[i][j],u[i][j]))
            theta[i][j]= theta[i][j]-(flux(theta[i][j],theta[i][(j+1)%J],v[i][(j+1)%J]) -
                                flux(theta[i][(j-1)%J],theta[i][j],v[i][j]))
            rv[i][j]= rv[i][j]-(flux(rv[i][j],rv[(i+1)%I][j],u[(i+1)%I][j]) -
                                flux(rv[(i-1)%I][j],rv[i][j],u[i][j]))
            rv[i][j]= rv[i][j]-(flux(rv[i][j],rv[i][(j+1)%J],v[i][(j+1)%J]) -
                                flux(rv[i][(j-1)%J],rv[i][j],v[i][j]))
            rl[i][j]= rl[i][j]-(flux(rl[i][j],rl[(i+1)%I][j],u[(i+1)%I][j]) -
                                flux(rl[(i-1)%I][j],rl[i][j],u[i][j]))
            rl[i][j]= rl[i][j]-(flux(rl[i][j],rl[i][(j+1)%J],v[i][(j+1)%J]) -
                                flux(rl[i][(j-1)%J],rl[i][j],v[i][j]))

    p=pressure(np.indices((30,30))[0]*dy,rv,theta)
    delta = (rv>=r_vs(theta,p)).astype(int)
    delta=np.multiply(delta,newton_method(0.,p,rv,theta))
    rv=rv-delta
    rl=rl+delta
    theta=theta+ Lv/(cp*exner(p))*delta

    if t%(T/20)==0:
        print(t/(T/20))
        plt.figure()
        plt.imshow(rl*1000,origin="lower",vmin=0., vmax=7.)
        plt.colorbar(label='$r_l$ [g/kg]')
        plt.xticks(ticks=np.arange(0,30,6),labels=np.arange(0,1500,300))
        plt.yticks(ticks=np.arange(0,30,6),labels=np.arange(0,1500,300))
        plt.xlabel('x [m]')
        plt.ylabel('y [m]')
        plt.title('Liquid water mixing ratio, '+'dx='+str(dx)+'m, dt='+str(dt)+'s, t='+
                    str(round(t*dt,1))+'s')
        plt.savefig(fname=str(int(t/(T/20))))
        plt.close()

frames = []
for t in range(20):
    image = imageio.v2.imread(f'./{t}.png')
    frames.append(image)
imageio.mimsave('./source.gif', # output gif
                frames,          # array of input frames
                fps = 2)         # optional: frames per second
```

```
plt.rcParams['font.size'] = 14
fig, ax = plt.subplots(1,3, figsize=(15,7))
ax[0].plot(np.mean(rv,axis=1)*10**3,np.arange(0,H,dy))
ax[1].plot(np.mean(rl,axis=1)*10**3,np.arange(0,H,dy))
ax[2].plot(np.mean(theta,axis=1),np.arange(0,H,dy))
ax[0].set_xlabel('$r_v$ [g/kg]')
ax[1].set_xlabel('$r_l$ [g/kg]')
ax[2].set_xlabel(r'$\theta$ [K]')
for i in range(3):
    ax[i].set_ylabel('z [m]')
ax[0].set_title('Water vapor mixing ratio')
ax[1].set_title('Liquid water mixing ratio')
ax[2].set_title('Potential temperature')
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.4,
                    hspace=0.4)
plt.savefig(fname='profiles.png')
plt.show()
```