Tesis de Maestría

David Peña Peralta

Invalid Date

Table of contents

1 Introducción

En este libro veremos la continuación del modelo visto en la Tesis de Licenciatura.

$$
\begin{aligned}
\mathrm{d}z &= \omega \mathrm{d}t \\
\mathrm{d}\omega &= \left(b\left(z\right) - \frac{\omega}{\tau_w} \right) \mathrm{d}t + b_w \mathrm{d}W
\end{aligned}
$$

Donde vimos que, por ejemplo $q_t = q_v + q_r$. En este modelo consideraremos quitaremos tal restricción, y agregaremos la dependencia temporal, es decir $q_v(z) \to q_v(z,t)$. Esto convierte el sistema de ecuaciones diferenciales ordinarias en un sistema de ecuaciones diferenciales parciales.

2 Teoría Fisica

3 Teoria Númerica

3.1 Adimensionalizacion

Al momento de trabajar con modelos físicos, en este caso atmosféricos, es importante tener cuidado al trabajar las unidades para evitar errores. Uno de los métodos más usados es la eliminación de las unidades.

3.2 Discretizacion

3.2.1 Malla (Regiones Rectangulares)

En el modelo trabajamos dentro de la tropopausa, por lo tanto $Z = [0, 15]$ y consideremos $T = [0, t], t \in \mathbb{R}$. Por lo tanto, nuestra zona de trabajo es $U = Z \times T$, donde cada $u \in U$ es de la forma $u = (z, t)$. Ahora, definiremos $n_z, n_t \in \mathbb{N}$, tales que $n_z, n_t > 0$. n_z nos dira el número de pedazos en los que será dividido Z y de igual forma con n_t para T.

Si $n_z = 2$, entonces existe z^* tal que $Z = [0, z^*] \cup [z^*, 15]$, llamando $z_0 = 0$ y $z_f = 15$. Decimos que la colección $P_Z = \{z_0, z^*, z_f\}$ forman una partición de Z. Todo esto aplica de igual forma para T. Entonces, llamaremos malla (o 'grid' traducido al ingles) al conjunto G, donde

$$G = P_Z \times P_T,$$

donde P_Z y P_T son particiones de Z y T respectivamente.

Dentro de una partición $P = \{x_0, \dots, x_n\}$ llamaremos $\Delta x_k = x_{k+1} - x_k, n - 1 \geq k \geq 0$. Entonces para P_Z y P_T existe $\Delta z_k, \Delta t_k$ respectivamente. Entonces, es claro que

$$t_k = t_0 + \sum_{i=0}^{k-1} \Delta t_k$$

$$z_k = z_0 + \sum_{i=0}^{k-1} \Delta z_k$$

Entonces, entonces para cada $u \in G$ existen indices i, j tales que

$$u \in G \Rightarrow u(z, t) = u(z_i, t_j) = u_j^i$$

3.2.2 Método Upwind

Una vez definida la malla, y conociendo el sistema a resolver, debemos revisar como vamos a resolver el sistema. Para ello usaremos una versión del método upwind la cual consta de 2 elementos.

- Diferencias Finitas
- Criterio CFL

3.2.2.1 Diferencias Finitas

Recordando la definición de derivada.

$$f'_+(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Es claro que,

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

También, según la situación pueden usarse variaciones para aproximarla.

$$f'_-(x) = \frac{f(x) - f(x-h)}{h}, f'_{\cdot}(x) = \frac{f'_+(x) + f'_-(x)}{2}$$

Sin embargo, esto es para funciones reales. ($f : \mathbb{R} \to \mathbb{R}$). Para nuestro caso $f : G \to \mathbb{R}$, requeriremos las derivadas parciales, tales que sin detalles pueden aproximarse mediante la siguiente fórmula.

$$\partial_z f(z,t) \approx \frac{f(z+h,t) - f(z,t)}{h}, \partial_t f(z,t) \approx \frac{f(z,t+h) - f(z,t)}{h}$$

3.2.2.2

4 Modelo 1: Etapa 1

Al estar recorriendo el modelo por submodelos, supondremos la velocidad constante.

$$\omega\left(z,t\right)=1,\forall\left(z,t\right)\in\mathbb{R}^{2}$$

Entonces, el sistema queda como sigue

$$\begin{aligned}\omega\left(z,t\right)&=1\\ \partial_t\left(\theta\right)+\partial_z\left(\omega\theta\right)&=0\\ \partial_t\left(q_v\right)+\partial_z\left(\omega q_v\right)&=0\\ \partial_t\left(q_r\right)+\partial_z\left(\omega q_r\right)&=0\\ \partial_t\left(q_N\right)+\partial_z\left(\omega q_N\right)&=0\end{aligned}$$

Entonces, pretendemos resolver un sistema desacoplado donde cada variable representa un problema de transporte.

Para ello. partimos importando las librerías básicas y los parámetros definidos para el modelo.

```
import numpy as np
import parameters as p
```

Entonces, proseguimos con la "entrada" del modelo.

```
z_0 = 0  # Km, nunca olvide las unidades...
z_0 = z_0 / p.length_scale

z_f = 15
z_f = z_f / p.length_scale

print(z_0, z_f)
```

```
0.0 1.5
```

Una vez definida $Z=[z_0,z_f]$, seguimos con n_z, en nuestro caso, comenzaremos con $n_z=150$.

```python
n_z = 150

height = np.linspace(z_0,z_f,n_z)
delta_z = height[1] - height[0]

print(n_z, delta_z)
```

```
150 0.010067114093959731
```

En nuestro caso, al usar la función "linspace", dados por sentado que Δz es constante, es decir, $\Delta z_k = c, c \in \mathbb{R}$.

Ahora, seguimos con el area de trabajo.

```python
variables = ['omega','theta','qv','qr','qn']

workspace = np.zeros((n_z,len(variables)))
```

En nuestro caso, nuestra areá de trabajo es una matriz de $n_z \times n_v$ donde n_v es el número de variables. Seguido con las condiciones iniciales, para ello, de nuevo llamamos a las librerías adecuadas.

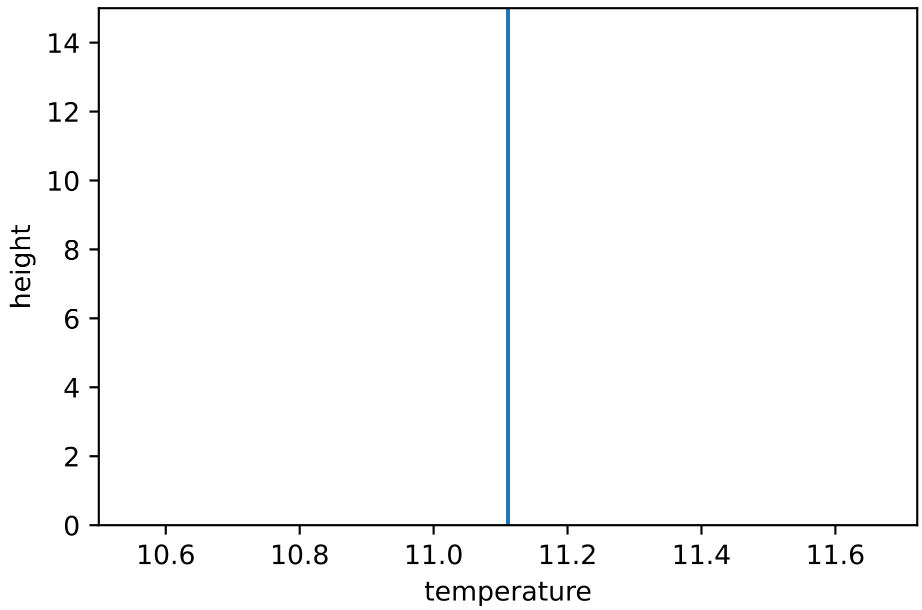
```python
import matplotlib.pyplot as plt
import init_functions as init
```

Para el primer modelo. Usaremos funciones escalonadas o de Heaviside.

$$U(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

```python
y = np.array([init.heaviside(i) for i in height])
plt.plot(y * p.velocity_scale, p.length_scale * height)
plt.ylim([0.0,15.0])
plt.xlabel('temperature')
plt.ylabel('height')
```

```
Text(0, 0.5, 'height')
```



En este caso era claro que saldría constante, por esto vamos a trasladar la función una distancia de a, además sirve para comentar que al estar tratando con alturas, es más conveniente gráficar con el formato anterior. $(f(z), z)$.

```
# Vamos a desplazar la grafica 3 unidades.
a_omega = 3
a_omega = a_omega / p.velocity_scale
a_theta = 3
a_theta = a_theta / p.temperature_scale
a_qv = 300
a_qv = a_qv / p.ratio_scale
a_qr = 300
a_qr = a_qr / p.ratio_scale
a_qn = 300
a_qn = a_qn / p.ratio_scale

print(a_omega, a_theta, a_qv, a_qr, a_qn)
```

```
0.27 1.0 0.3 0.3 0.3
```

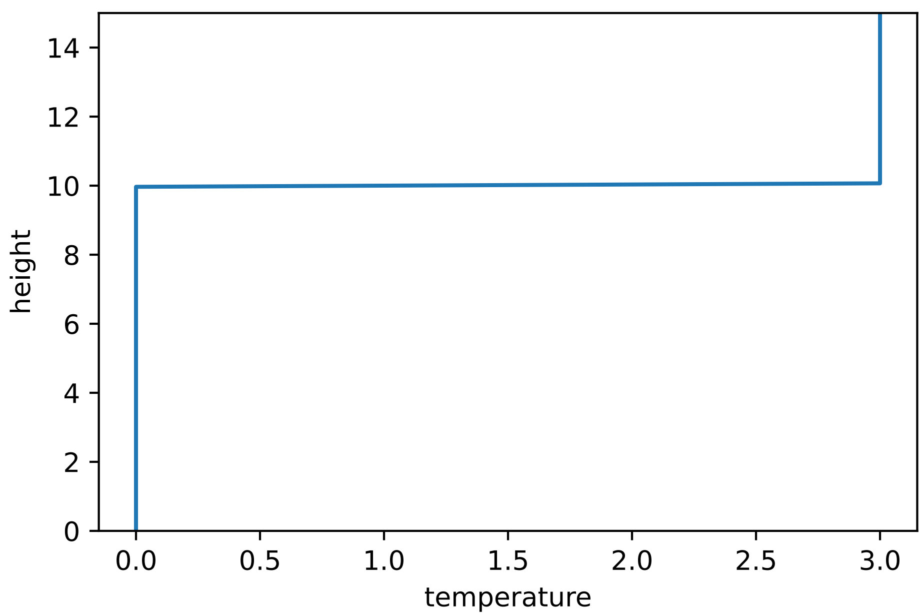
Recordando que

$$U_a(x) = \begin{cases} 0 & x < a \\ 1 & x \geq a \end{cases} = U(x-a), \forall a \in \mathbb{R}$$

podemos graficar ω.

```
y_omega = np.array([init.heaviside(i - a_theta) for i in height])
plt.plot(y_omega * p.temperature_scale, p.length_scale * height)
plt.ylim([0.0,15.0])
plt.xlabel('temperature')
plt.ylabel('height')
```

```
Text(0, 0.5, 'height')
```



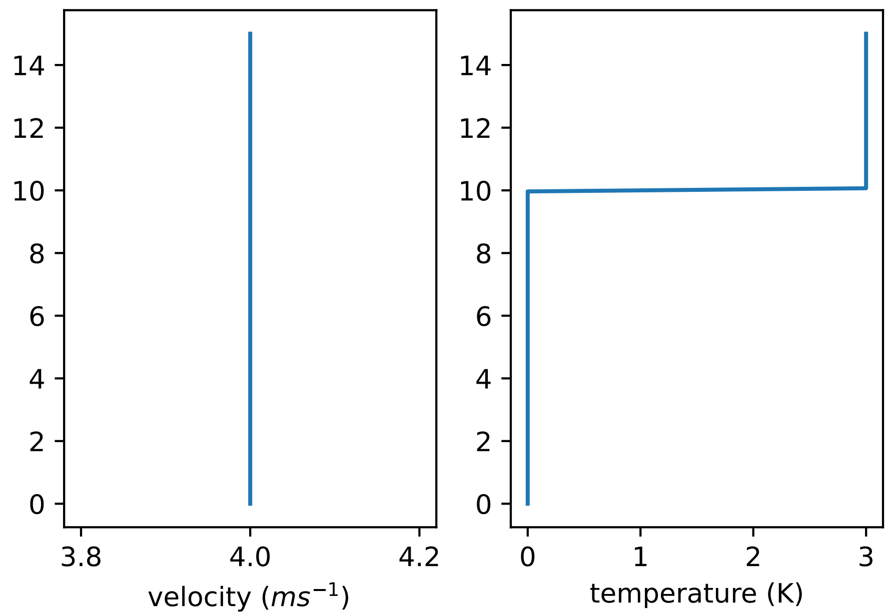
Entonces, ahora podemos graficar todas en conjunto. Para eso usaremos una función auxiliar

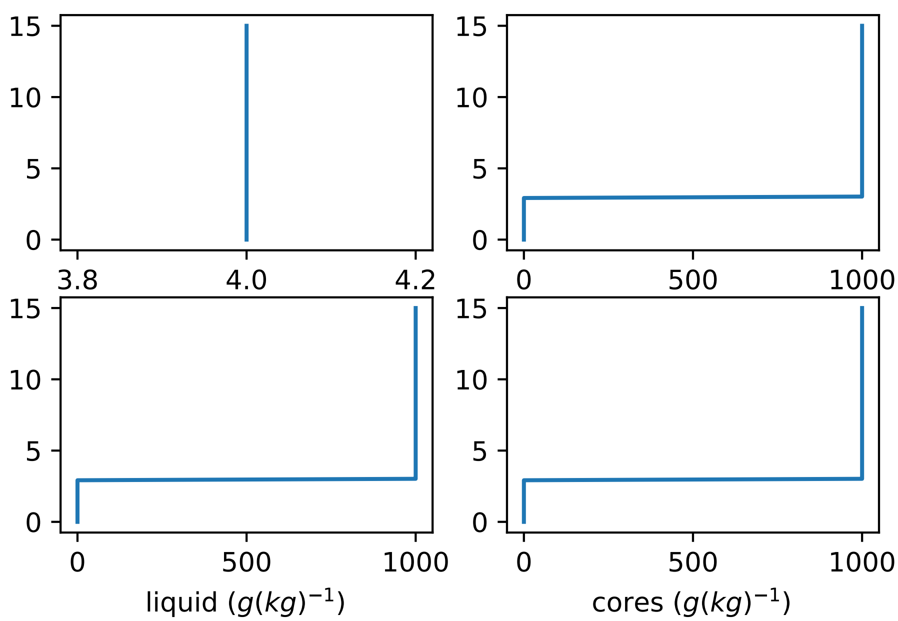
```
def system_plots(workspace, y):
    velocity = workspace[:, 0]
    temperature = workspace[:, 1]
    vapor = workspace[:, 2]
    water = workspace[:, 3]
    core = workspace[:, 4]
    wt_plots, ax = plt.subplots(1, 2)
    qvrn_plots, axq = plt.subplots(2, 2)
    ax[0].plot(velocity * p.velocity_scale, y * p.length_scale)
    ax[0].set_xlabel('velocity ('r'$ms^{-1}$'')')
```

```
    ax[1].plot(temperature * p.temperature_scale, y * p.length_scale)
    ax[1].set_xlabel('temperature (K)')
    axq[0, 0].plot(velocity * p.velocity_scale, y * p.length_scale)
    axq[0, 0].set_xlabel('velocity ('r'$ms^{-1}$'')')
    axq[0, 1].plot(vapor * p.ratio_scale, y * p.length_scale)
    axq[0, 1].set_xlabel('vapor ('r'$g(kg)^{-1}$'')')
    axq[1, 0].plot(water * p.ratio_scale, y * p.length_scale)
    axq[1, 0].set_xlabel('liquid ('r'$g(kg)^{-1}$'')')
    axq[1, 1].plot(core * p.ratio_scale, y * p.length_scale)
    axq[1, 1].set_xlabel('cores ('r'$g(kg)^{-1}$'')')
    plt.show()
```

Entonces, graficamos las condiciones iniciales.

```
v0 = 4  # Etapa 1: Velocidad Constante
v0 = v0 / p.velocity_scale

workspace[:, 0] = v0
workspace[:, 1] = [init.heaviside(i - a_theta) for i in height]
workspace[:, 2] = [init.heaviside(i - a_qv) for i in height]
workspace[:, 3] = [init.heaviside(i - a_qr) for i in height]
workspace[:, 4] = [init.heaviside(i - a_qn) for i in height]

system_plots(workspace, height)
```





Una vez podemos ver las condiciones iniciales, procedemos a la implementación del método. Primero definimos $T = [0, t]$ y fijamos el criterio CFL.

```
t = 5

cfl = 0.9

delta_t = cfl * delta_z
print(delta_t)
```

```
0.009060402684563758
```

Una vez definido CFL, podemos calcular Δ_t mediante la formula

$$\Delta t = \frac{\text{CFL}}{\Delta z}$$

Para Δ_t inicial, pero como se mencionó anteriormente, Δt ira variando mediante la siguiente fórmula.

$$\Delta t(c_w) = \frac{\text{CFL}}{\Delta z} c_w$$

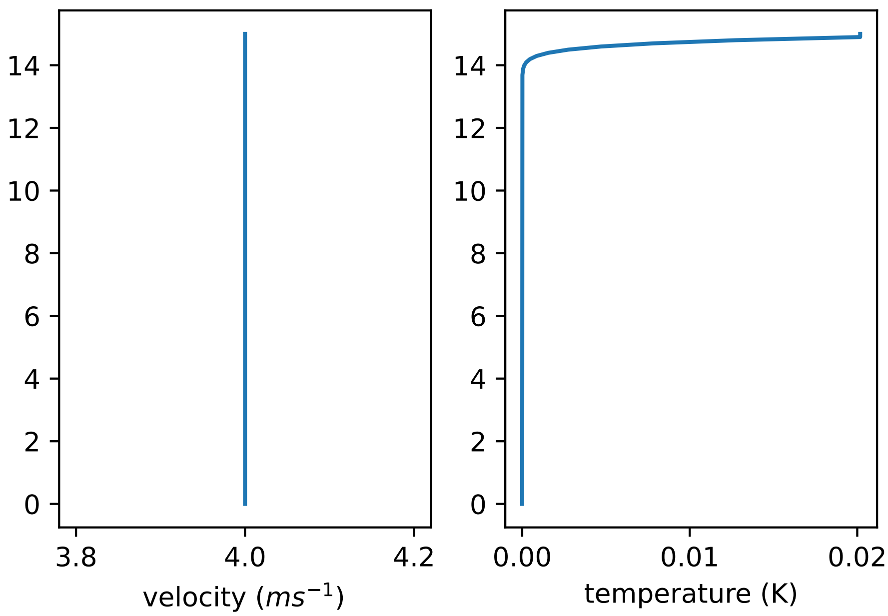
Donde c_w variará según el modelo. Aplicando el método de upwind, resolvemos el primer modelo.

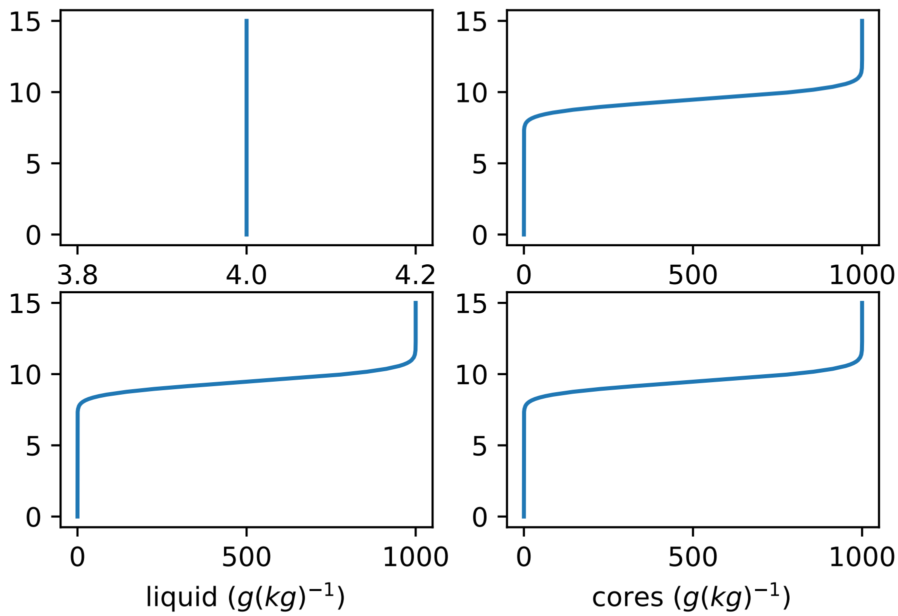
```
def upwind_model(dt, dz, u, ve):
    m, n = np.shape(u)
    aux = np.zeros((m, n))
    v = u[:, 0]
    t = u[:, 1]
    qv = u[:, 2]
    qr = u[:, 3]
    qn = u[:, 4]
    dzt = dt / dz
    for i in range(1, m - 1):
        if v0 < 0:
            aux[i, 0] = ve
            aux[i, 1] = t[i] - ve * dzt * (t[i + 1] - t[i])
            aux[i, 2] = qv[i] - ve * dzt * (qv[i + 1] - qv[i])
            aux[i, 3] = qr[i] - ve * dzt * (qr[i + 1] - qr[i])
            aux[i, 4] = qn[i] - ve * dzt * (qn[i + 1] - qn[i])
        else:
            aux[i, 0] = ve
```

```
            aux[i, 1] = t[i] - ve * dzt * (t[i] - t[i - 1])
            aux[i, 2] = qv[i] - ve * dzt * (qv[i] - qv[i - 1])
            aux[i, 3] = qr[i] - ve * dzt * (qr[i] - qr[i - 1])
            aux[i, 4] = qn[i] - ve * dzt * (qn[i] - qn[i - 1])
        aux[0, :] = aux[1, :]
        aux[-1,:] = aux[-2, :]
    return aux
```

Una vez construido el modelo. Preparamos la implementación

```
t_star = 0
while t_star < t:
    workspace = upwind_model(delta_t, delta_z, workspace, v0)
    dt = cfl * delta_z / np.abs(v0)
    t_star += dt

system_plots(workspace, height)
```



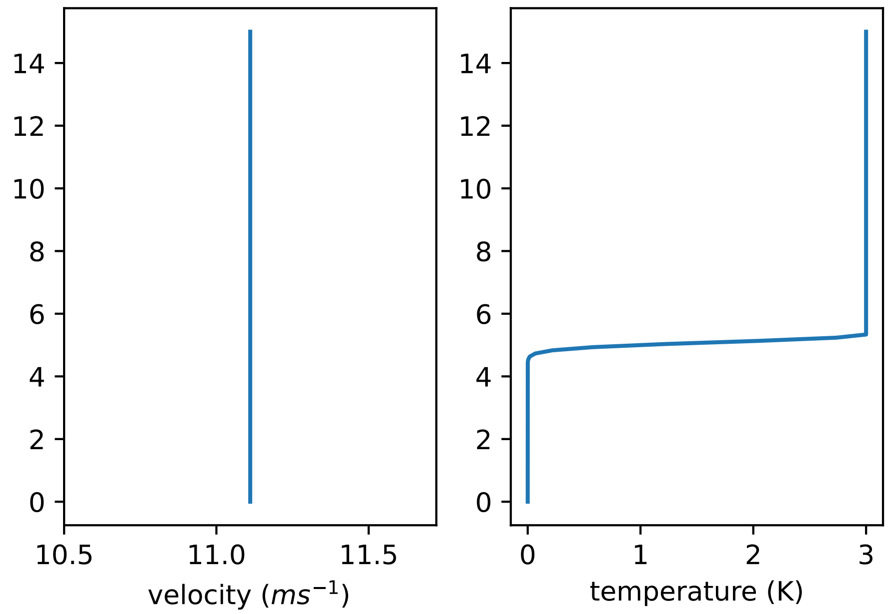


Finalmente, una función que implemente todo lo visto.

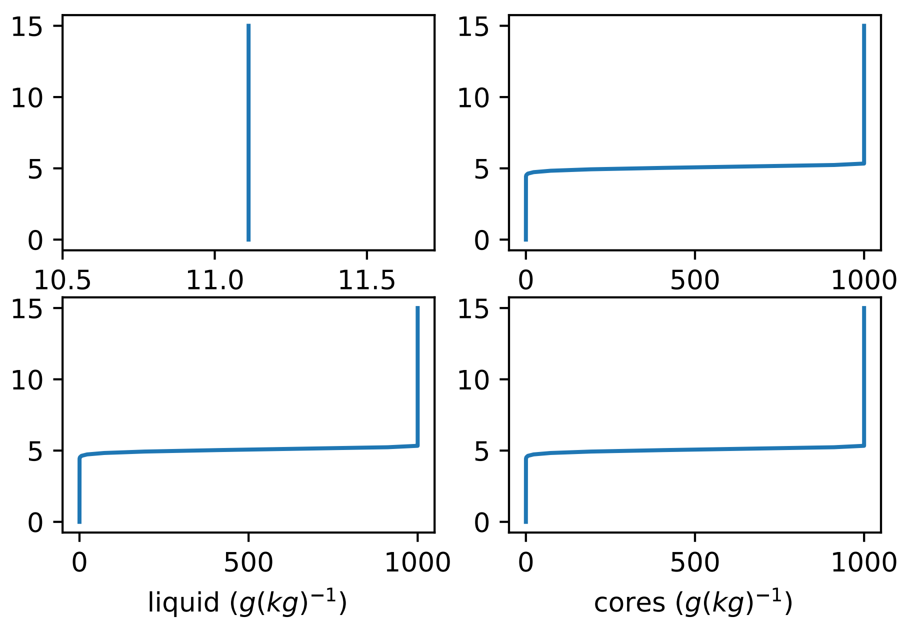
```
def model1(variables,  a, v0, Z, n_z, t, cfl):
    def heaviside(x): # Tarea, modificar model1 para que acepte diferentes
        if x < 0:        # funciones
            y = 0
        else:
            y = 1
        return y
    height = np.linspace(Z[0],Z[1],n_z)
    delta_z = height[1] - height[0]
    workspace = np.zeros((n_z,len(variables)))

    a_omega = a[0]
    a_theta = a[1]
    a_qv = a[2]
    a_qr = a[3]
    a_qn = a[4]

    workspace[:, 0] = v0
    workspace[:, 1] = [init.heaviside(i - a_theta) for i in height]
    workspace[:, 2] = [init.heaviside(i - a_qv) for i in height]
```

```python
    workspace[:, 3] = [init.heaviside(i - a_qr) for i in height]
    workspace[:, 4] = [init.heaviside(i - a_qn) for i in height]

    de_t = cfl * delta_z

    t_star = 0
    m, n = np.shape(workspace)
    while t_star < t:
        v = workspace[:, 0]
        temp = workspace[:, 1]
        qv = workspace[:, 2]
        qr = workspace[:, 3]
        qn = workspace[:, 4]
        aux = np.zeros((m, n))
        dzt = de_t / delta_z
        for i in range(1, m - 1):
            if v0 < 0:
                aux[i, 0] = v0
                aux[i, 1] = temp[i] - v0 * dzt * (temp[i + 1] - temp[i])
                aux[i, 2] = qv[i] - v0 * dzt * (qv[i + 1] - qv[i])
                aux[i, 3] = qr[i] - v0 * dzt * (qr[i + 1] - qr[i])
                aux[i, 4] = qn[i] - v0 * dzt * (qn[i + 1] - qn[i])
            else:
                aux[i, 0] = v0
                aux[i, 1] = temp[i] - v0 * dzt * (temp[i] - temp[i - 1])
                aux[i, 2] = qv[i] - v0 * dzt * (qv[i] - qv[i - 1])
                aux[i, 3] = qr[i] - v0 * dzt * (qr[i] - qr[i - 1])
                aux[i, 4] = qn[i] - v0 * dzt * (qn[i] - qn[i - 1])
            aux[0, :] = aux[1, :]
            aux[-1,:] = aux[-2, :]
            workspace = aux
        dt = cfl * delta_z / np.abs(v0)
        t_star += dt
    return workspace
```

```python
s = model1(['omega','temp','qv','qr','qn'],
[3,3,3,3,3], 1, [0, 15], 150, 2, 0.9)

system_plots(s, height)
```



velocity (ms^{-1})
temperature (K)



liquid ($g(kg)^{-1}$)
cores ($g(kg)^{-1}$)

5 Modelo 1: Etapa 2

Una vez concluido la etapa 1, seguiremos agregando las funcionces $V_{\mathrm{T}}, V_{\mathrm{TN}}$. Las cuales tienen la siguiente forma.

$$\begin{aligned}
V_{\mathrm{T}}\left(q_r\right) =& V_0 \frac{q_r}{q_*} \\
V_{\mathrm{TN}}\left(q_r\right) =& V_{\mathrm{TN}d} + \max\left(\frac{q_r}{q_*}, 1\right) \max\left(V_T\left(q_r\right) - V_{\mathrm{TN}d}, 0\right)
\end{aligned}$$

6

References