Oblig_termodynamikk

January 29, 2019

1 Obligatorisk oppgave GEO2310: Termodynamikk

I atmosfæren har vi ulik oppvarming/avkjøling i forskjellige høyder (mer om det i senre kapittler). Dette kan føre til at atmosfæren blir statisk instabil og at dermed blir vertikal blanding av luftmassene.

I denne oppgaven skal vi programmere en svært enkel kolonnemodell. Vi begynner med å lage en modell som tar en temperaturprofil inn og så løfter/senker luft til vi har en stabil profil.

Antagelser for modellen:

- Vi bruker trykkoordinater.

Vi gjør noen forenklinger

- Vi antar null utveksling av luft med omgivelsene (hver boks er en isolert luftpakke) når luften blandes vertikalt.
- Vi antar at luftsøylen består kun av tørr luft (potensiell temperatur er bevart)

1.1 Oppgave 1

Vi begynner med å lage en klasse i python for griddet for modellen.

Vi bruker trykkoordinater. Boksene i kolonnemodellen skal være fordelt med lik trykkdifferanse mellom hver boks (slik blir massen i hver boks lik og vi slipper å ta hensyn til dette når vi løfter/senker boksene) I tillegg skal du legge til legge til en array for høyde i meter. Vi antar at vi kan bruke en konstant skalahøyde og dermed likning 3.25 i boka.

Sjekk at det ser riktig ut ved å plotte trykk mot høyde med 100 bokser, høyeste trykk være 10 hPa og laveste trykk 1000 hPa. La skalahøyden være 8000 m (basert på T=273 K). (Plot med punkter, ikke linje (bruk f.eks '.'), slik at du kan se fordelingen).

Hvorfor er det egentlig ikke riktig å bruke konstant skalahøyde her? Hva skulle vi gjort for å få en riktig høyde?

```
In [2]: import numpy as np
        import sys
        import pandas as pd
        Lv = 3.34*10**5 # J/kg Latent varmekapasitet
        cp=1004. # Spesifikk varmekapasitet tørr luft, konstant trykk
        r2cp=287/1004.# 0.286 #R/cp
```

```
class Grid:
    #scale_H = 8000 # assume constant, T = 273 K, H = 8000 m
    def __init__(self, nr_boxes, top_pressure, p0=1000.,scale_H=8000):
        self.p0=p0 # hPa
        self.nr_boxes = nr_boxes # number of boxes
        self.scale_H = scale_H # [m] scale height atmosphere

        #p_mid = # mid box pressure [hPa]
        #self.z_mid= # [m] mid box height
        return
```

1.2 Oppgave 2

Vi skal nå lage en klasse som holder en temperaturprofil. Den skal være basert på grid-klassen du lagde i forrige oppgave men i tillegg tar den et temperaturprofil (dvs. at den inneholder de samme variablene).

1. Legg til en initialiseringsfunksjon (def __init__(self,input1, input2 osv)) som tar variable for å initialisere griddet (antall bokser, topp trykket, bunntrykk og skaleringshøyde). Legg til i initialiseringsfunksjonen at at den tar inn en temperaturprofil og legger den til som self.temperatures i klassen.

2. Legg til en funksjon i klassen som regner potensiell temperatur θ fra temperatur og en som regner fra potensiell temperatur til temperatur (temp2theta(..) og theta2temp(..)). Initialiser modellen med en temperaturprofil (du kan velge hvordan den ser ut) og plot både temperatur og potensiell temperatur.

3. Endre initialiseringsfunksjonen du lagde i punkt 1 slik at den også kan initialiseres med en potensiell temperaturprofil. Plot temperatur og potensielltemperatur for enn stabil profil og en instabil profil.

```
In [7]: class Temperature_profile(Grid):
            r2cp=287/1004.#0.286 #R/cp

            def __init__(self, nr_boxes, top_pressure, temperatures=np.array([]), theta=np.arra
                # initialiser gridded ved å kalle init funksjonen til super klassen.
                super().__init__(nr_boxes, top_pressure, p0, scale_H)
                #self.temperatures
                #self.theta
                return



            def temp2theta(self, temperatures):
                #compute potential temperature from temperature profile
                return# theta
```

```
    def theta2temp(self,theta):
        # compute temperature from potential temperature profile
        return #temperatures

    def lift_box(self,i):
        # Løfter en boks til den er stabil.
        # Oppgave 3
        return #box_shifted


    def equilibrium(self):
        # Lar luft løftes/senkes til likevekt
        # Oppgave 3
        return

    def heat_profile(self,heating_rate, dt):
        return
```

1.3 Oppgave 3

1. Legg til en funksjon som løfter en boks opp til den er stabil og forskyver de andre boksene nedover (`lift_box(self,i)`).
2. Lag et script som lar luftpakkene løftes/senkes til luftsøylen er stabil (ved å kalle funksjonen fra punkt 1). Test at scriptet ditt gjør det du forventer ved å lage en istabil temperaturprofil (θ synker med høyden) og 5 bokser totalt og plot hver gang en boks er flyttet på. Legg så til en funksjon i klassen som gjør dette (`equilibrium(self)`). Du kan bruke scriptet under som inspirasjon.

In [ ]:
```
# 2: Under følger en eksempelkode som du kan ta utgangspunkt i
import numpy as np
nr_of_boxes=5
top_pres=10. # hPa
theta= 300+np.arange(nr_of_boxes)/nr_of_boxes

temp_prof = Temperature_profile(nr_of_boxes, top_pres, theta=theta)

# Lar luft løftes/senkes til likevekt
i=0 #len(temp_prof.p_mid)-1
while i<nr_of_boxes: # for i in np.arange(0,len(self.grid.p_mid))[::-1]:
    plt.plot(temp_prof.theta, temp_prof.p_mid, label='%.0f'%i)
    box_shifted = temp_prof.lift_box(i) # Løft en boks
    if not box_shifted: i=i+1 # Hvis boksen ikke er løftet kan du gå et steg ned/opp:
```

```
            temp_prof.temperatures=temp_prof.theta2temp(temp_prof.theta)

        plt.gca().invert_yaxis()
        plt.legend()
        plt.yscale('log')
        plt.show()
```

1.4 Oppgave 4

La øverste trykk være 10 hPa og antall bokser være 1000.

Initialiser 3 temperaturprofiler: En stabil, en instabil og profil som du kan lese ut ved vedlagt funksjon:

```
import read_in_temp
temp3=read_in_temp.read_temperatures_and_interpolate_to_grid('temperatures.csv', Grid(nr_of_bo
```

Denne gir deg en semirealistisk temperaturprofil.

Plot profilen før og etter likevekt.

Prøv å variere topptrykket i modellen slik at du kan se hva som skjer med den siste profilen. Beskriv endringene og forklar hva slags naturlig fenomen de kan representere.

1.5 Oppgave 5: Diabatisk oppvarming

Vi skal nå legge til en oppvarmingsrate til modellen. Vi skal legge inn diabatisk oppvarming for å imitere oppvarming ved solinnstråling.

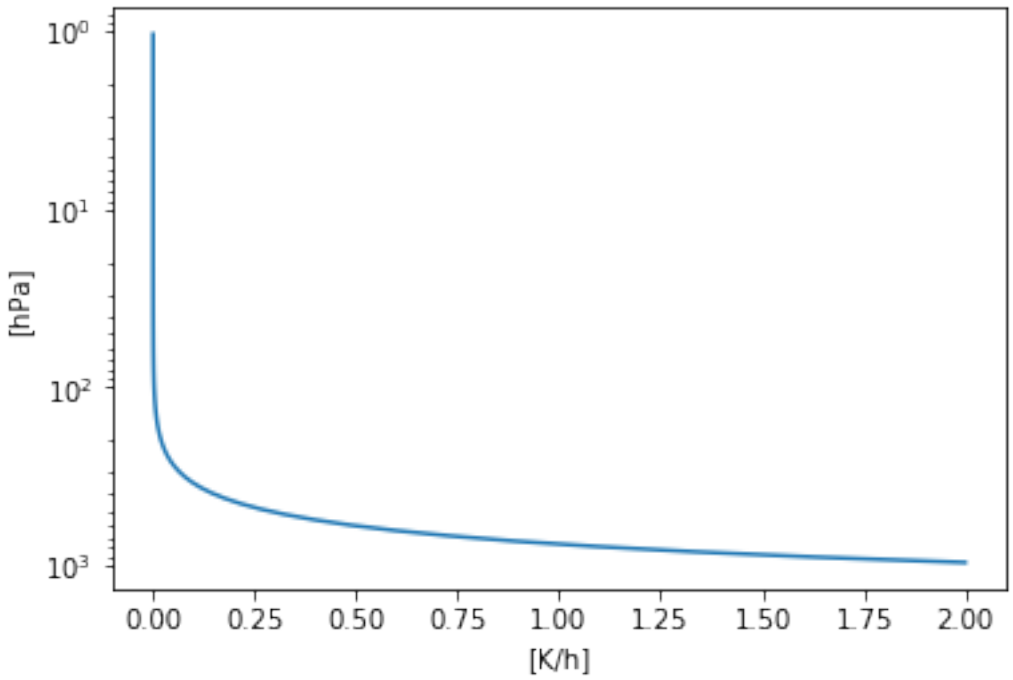
Legg til en funksjon som tar inn en oppvarmingsrate (Kelvin per time) for hver boks i modellen og et tidssteg (f.eks 1 time) og hever temperaturen i henhold til oppvarmingsraten og tidssteget. Altså, hvis jeg gir funksjonen 1 Kelvin per time for alle boksene, og et tidssteg på 1/2 time, skal temperaturen gå opp 0.5K overalt.

Start med en temperaturprofil som den siste i forrige oppgave:

```
import read_in_temp
temp3=read_in_temp.read_temperatures_and_interpolate_to_grid('temperatures.csv', Grid(nr_of_bo
```

Gi modellen følgende profil for oppvarming:

```
In [18]: #
         heating_rate =2*np.exp(- 20*np.linspace(0,1,nr_of_boxes)[::-1]) # K/h
         grid= Grid(nr_of_boxes, top_pres)
         plt.plot( heating_rate, grid.p_mid)#
         plt.xlabel('[K/h]')
         plt.ylabel('[hPa]')
         plt.yscale('log')
         plt.gca().invert_yaxis()
```



Bruk tidssteg 1 time. Plot resultatet før oppvarming i et tidssteg, etter oppvarming og når du har kjørt modellen til en ny likevekt.

1.6 Oppgave 6:

Tilslutt skal vi lage en liten film av modellen vår når vi legger til en oppvarmingsrate! Filmen skal vise hvordan temperatur og potensiell temperatur endrer seg med tid.

For å simulere døgnvariasjon, multipliserer vi oppvarmingsraten fra forrige oppgave med en sinusfunksjon. F.eks slik:

```
heating_rate*np.sin(i*timestep/24*2*np.pi)
```

hvor i er en indeks. Bruk tidssteg 1 time, men prøv også å variere tidssteget. Kjør modellen i 2 dager. Beskriv hva som skjer. Hva slags naturlig fenomen kan dette sammenliknes med? Hva blir forskjellig hvis du endrer tidssteget?

Bruk oppsettet under (eller finne noe eget) :)

```
In [ ]: %matplotlib inline
        from matplotlib import animation, rc
        from IPython.display import HTML
        # First set up the figure, the axis, and the plot element we want to animate
        top_pres=400
        nr_of_boxes=1000
        timestep=1. #h
        # Les inn temperaturprofil:
        temperatures = read_in_temp.read_temperatures_and_interpolate_to_grid('temperatures.csv
```

```
# Initialiser:

temp_profil = Temperature_profile(nr_of_boxes, top_pres,temperatures=temperatures)


# La stå:
fig, ax = plt.subplots()
ax.set_xlim(( 225, 400))
ax.set_ylim((min(temp_profil.p_mid),max(temp_profil.p_mid))
lines = []#
lines.append(plt.semilogy([],[], label=r'$\theta$')[0])
lines.append(plt.semilogy([],[], label='Temp')[0])
plt.legend()
plt.ylabel('Pressure [hPa]')
plt.xlabel('[K]')
plt.gca().invert_yaxis()

# initialization function: plot the background of each frame
def init():
    for line in lines:
        line.set_data([], [])
    return lines
# animation function. This is called sequentially
timestep=1
def animate(i):
    # Regn ut theta og temperaturen for hvert tidssteg
    # her og la x være theta, og x2 være temperatur.
    # La y være trykk eller høyde
    lines[0].set_data(x, y)
    lines[1].set_data(x2,y)
    return lines

#plt.legend()
# call the animator. blit=True means only re-draw the parts that have changed.
nr_of_days = 2
frames = round(24*nr_of_days/timestep)
interval = 200*timestep # Delay between frames in milliseconds

anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=frames, interval=interval, blit=True)
HTML(anim.to_html5_video())
```

2 SLUTT