UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

DEPARTMENT OF SPACE AND APPLICATIONS

INTRODUCTION TO EARTH SYSTEM

LECTURER: ASSOC. PROF. NGO DUC THANH

Midterm Report

Duong Thu Phuong
22BI13362

3 June, 2024

Contents

1 Overview

This report is for solving and explaining all the practices in the ”Introduction to Earth System” course given at University of Science and Technology of Hanoi. This course aims to provide a comprehensive understanding of the interconnections and dynamics that define the Earth’s complex system, including the lithosphere, atmosphere, hydrosphere, biosphere, and even the human element. By studying these relationships, students gain insights into the processes of shaping our planet.

This report covers practices related to understanding the Earth’s system, including:

- Analyzing the vertical pressure profile in the atmosphere, including plotting the pressure and density profiles and comparing analytical and numerical results.
- Plotting Planck curves for different blackbody temperatures and calculating the wavelength of peak emission and total energy emitted.
- Implementing a 0-D Energy Balance Model (EBM) to estimate the incoming solar flux and blackbody equivalent temperatures for different planets.
- Further exploring the 0-D EBM, investigating the dependence of surface temperature on parameters like albedo and emissivity.
- Modeling the Daisy World system with black and white daisies, analyzing the equilibrium populations and temperatures.
- Extending the 0-D EBM to include feedback effects, studying the dependence of equilibrium temperature on initial conditions and solar flux.
- Developing a 1-D EBM and using it to estimate the solar flux required for the Earth to be entirely ice-covered, as well as investigating the sensitivity to model parameters.
- Analyzing sea level rise data from satellite altimetry and describing the understanding of altimetry missions.
- Examining sea surface temperature data and its relationship to the El Niño-Southern Oscillation (ENSO) phenomenon.

Overall, this report demonstrates a thorough investigation of fundamental Earth system processes through a combination of theoretical analysis and practical computer simulations.

2 Practices

2.1 Vertical pressure profile in the atmosphere

Given:

$$p(z) = p_0 \cdot \exp\left(-\frac{z}{H}\right)$$

where H=8000(m); p0=1000.; Z=0., 1000., ...9000.; dz=1000.

2.1.1 Plot the vertical profile of pressure.

Listing 1: Practice 1.1

```
import matplotlib.pyplot as plt
import numpy as np

H=8000
p0=1000
dz=1000
g=9.8

def p(x):
    p=p0*np.exp(-x/H)
    return p

z=np.arange(0,10000,dz)

plt.plot(p(z),z, color = 'deeppink')
plt.ylabel('Height')
plt.xlabel('Pressure')
plt.grid(True)
```

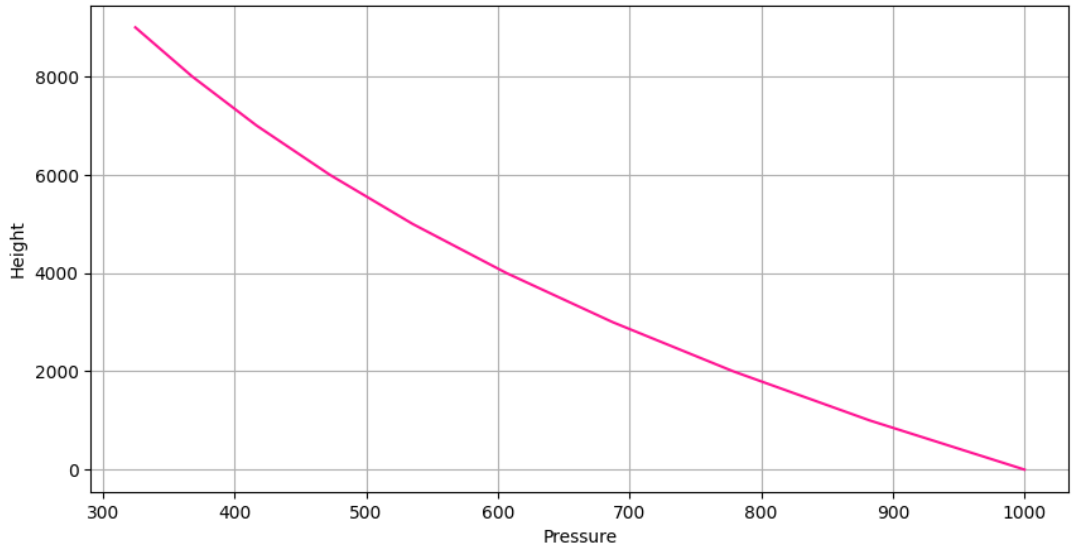

Figure 1: Vertical profile of pressure.

At lower heights, which is closer to the surface of the Earth, the pressure values are high. As the height increases, there is less air, the pressure decreases, resulting in a curve that gradually flattens out.

2.1.2 Plot the analytical density at different atmospheric levels.

First I import the constants and define the array of altitude values as below, in which R is ideal gas constant ($\frac{J}{mol \cdot K}$), M is molar mass of dry air (kg/mol), T_0 is the reference temperature at sea level (K).

I use the hydrostatic equation to calculate the density at each level using a temperature lapse rate of $-0.0065 K/m$, and plot the density profile in which the vertical axis is altitude and the other one is density.

Listing 2: Practice 1.2

```
R = 8.314
M = 0.02897
T0 = 288.15

altitude = np.arange(0, 10000, dz)

density = []
for h in altitude:
    T = T0 - 0.0065 * h
    P = p0 * (1 - 0.0065 * h / T0) ** (g * M / (R * 0.0065))
    rho = P * M / (R * T)
    density.append(rho)

plt.plot(density, altitude, color = 'deeppink')
plt.xlabel('Density')
plt.ylabel('Altitude')
plt.grid(True)
plt.show()
```

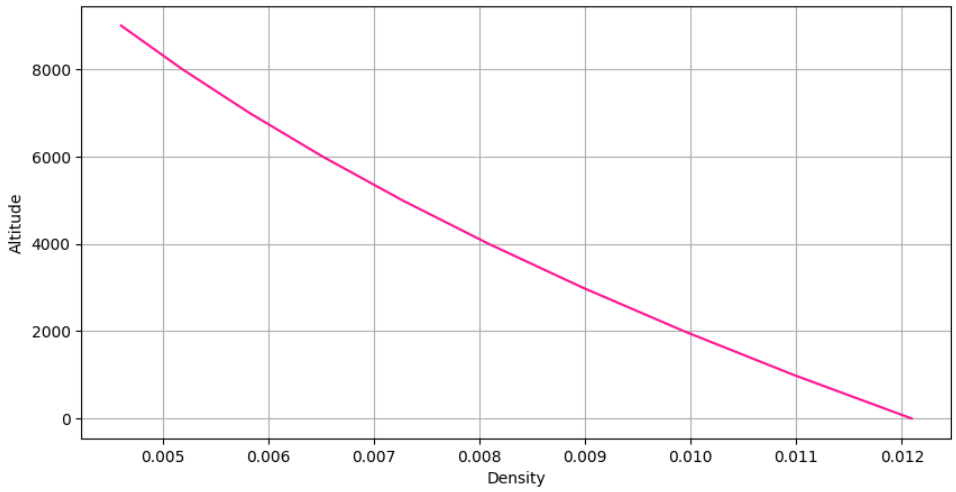

Figure 2: Analytical density at different atmospheric levels.

From the graph plotted, it can easily be seen that as the altitude increases, the density of the atmosphere decreases. This is due to the fact that as it go higher in the atmosphere, the air becomes less dense because there is less mass of air above it.

2.1.3 Calculate (approximately) the density at different atmospheric levels by using forward, backward, central difference method, etc.

I start by defining the range of height and the three functions forward, backward, central and then creating empty arrays of rhof, rhob, and rhoc to store the values calculated using each method, respectively.

Listing 3: Practice 1.3

```
z = np.arange(0, 10000, dz)
n = len(z)

def forward(b):
    densf = (-1 / g) * (p(b + dz) - p(b)) / dz
    return densf

def backward(b):
    densb = (-1 / g) * (p(b) - p(b - dz)) / dz
    return densb

def central(b):
    densc = (-1 / g) * (p(b + dz) - p(b - dz)) / (2 * dz)
    return densc

rhof = np.zeros(n)
rhob = np.zeros(n)
rhoc = np.zeros(n)

for i in range(0, n):
    rhof[i] = forward(z[i])

for i in range(0, n):
    rhob[i] = backward(z[i])

for i in range(0, n):
    rhoc[i] = central(z[i])

plt.plot(rhof, z, color='deeppink', linestyle='', marker='.',
   label='Forward Difference')
plt.plot(rhob, z, color='deeppink', linestyle='', marker='+',
   label='Backward Difference')
plt.plot(rhoc, z, color='deeppink', linestyle='', marker='o',
   label='Central Difference')
plt.legend()
plt.ylabel('Height (m)')
plt.xlabel('Density (kg/m^3)')
plt.grid(True)
plt.show()
```

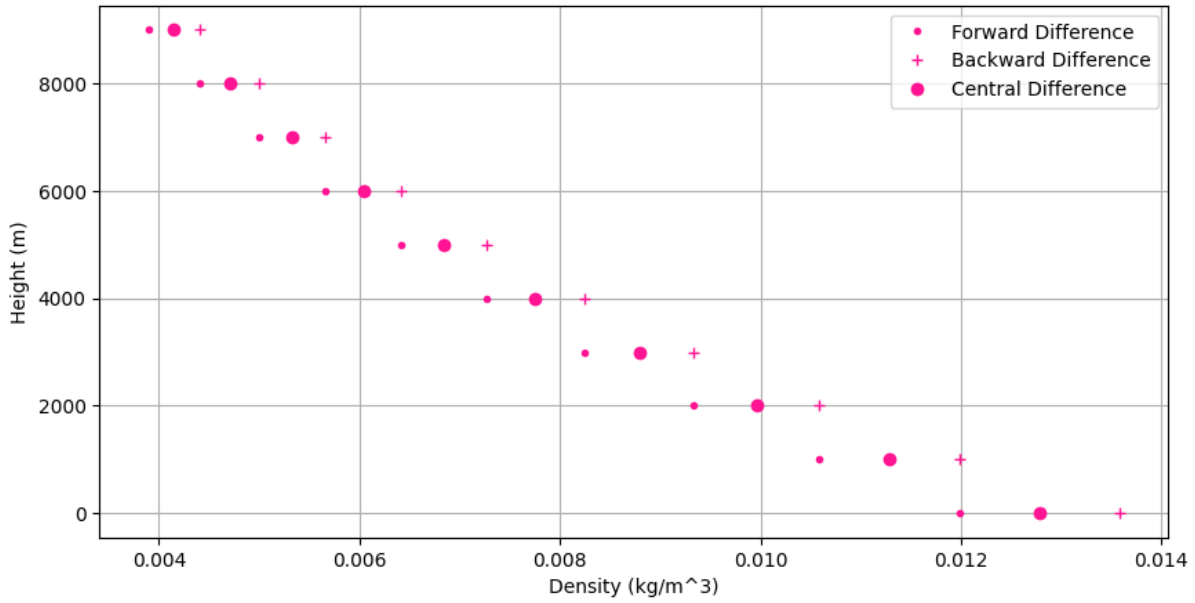

Figure 3: Density variations with height (dz=1000).

The points calculated by forward difference method is on the left, the backward difference points is in the middle, and the central difference points is on the right. This arrangement is because forward difference method calculates the gradient by considering the next higher point, shifting the resulting data points to the left; the backward difference method considers the next lower point, placing the data points towards the middle; and the central difference method involves neighboring points on both sides, aligning the data points more towards the right.

2.1.4 Compare the obtained simulation results with the analytical results.

I take the analytical results from question 2 minus the simulation results, which are rhof, rhob and rhoc corresponding to three methods.

Listing 4: Practice 1.4

```
analytical_results = np.array(density)

dif_forward = np.abs(analytical_results - rhof)
dif_backward = np.abs(analytical_results - rhob)
dif_central = np.abs(analytical_results - rhoc)

print("Difference using Forward method:")
print(dif_forward)

print("Difference using Backward method:")
print(dif_backward)

print("Difference using Central method:")
print(dif_central)
```

```
Difference using Forward method:
[0.00010249 0.00039295 0.00059875 0.00073521 0.00081547 0.00085077
```

```
 0.0008507  0.00082339 0.0007757  0.00071339]
Difference using Backward method:
[1.49397229e-03 1.01592931e-03 6.44577840e-04 3.62021643e-04
 1.52833544e-04 3.75182991e-06 9.65875880e-05 1.57887433e-04
 1.88397458e-04 1.95098251e-04]
Difference using Central method:
[6.95739856e-04 3.11491661e-04 2.29137974e-05 1.86594949e-04
 3.31318899e-04 4.23511202e-04 4.73645890e-04 4.90640216e-04
 4.82050759e-04 4.54246379e-04]
```

As being seen, the differences are really small and insignificant, which means that the numerical methods are performing well and providing accurate approximations of the analytical solution.

2.1.5 If dz=500 (i.e. the levels are 0., 500., ..., 9000.). Compare the new simulation results with those obtained with dz=1000.

This is the graph plotted when I changed dz to 500 in practice 1.3's code.

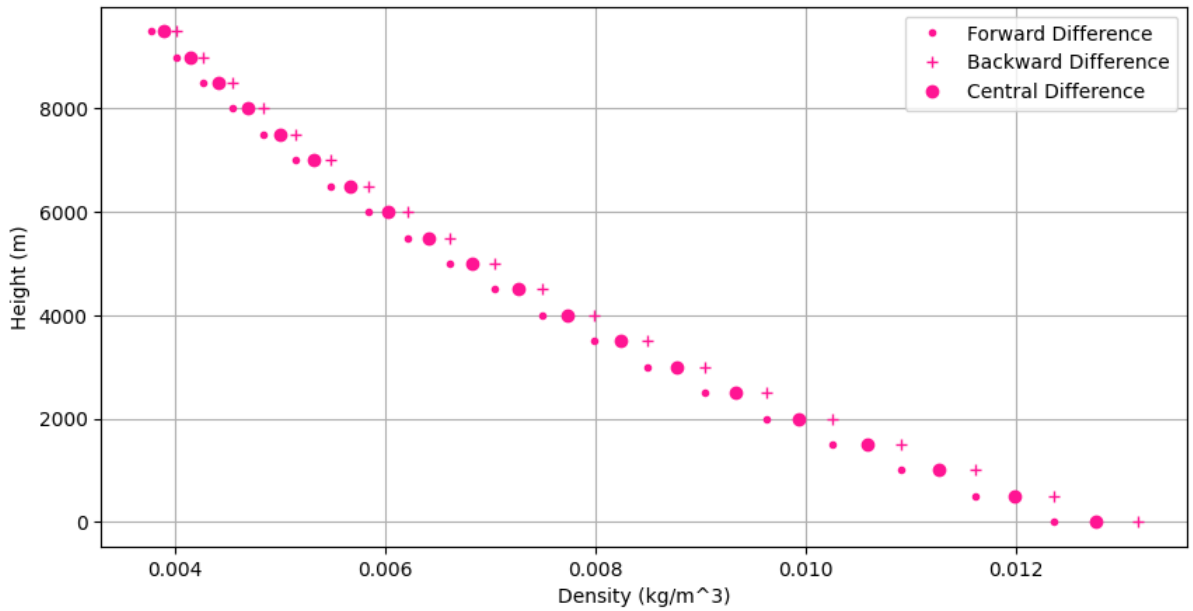

Figure 4: Density variations with height (dz=500).

To compare the two graphs, I overlap them. In the following code, the three functions forward, backward, central for dz=500 are restored in arrays of rhof_new, rhob_new, and rhoc_new, respectively. The color blue represents the results when dz = 1000, and the color pink represents the results when dz = 500.

Listing 5: Practice 1.5

```
1 dz = 500
2 z_new = np.arange(0, 10000, dz)
3 n_new = len(z_new)
4
```

```
rhof_new = np.zeros(n_new)
rhob_new = np.zeros(n_new)
rhoc_new = np.zeros(n_new)

for i in range(0, n_new):
    rhof_new[i] = forward(z_new[i])
for i in range(0, n_new):
    rhob_new[i] = backward(z_new[i])
for i in range(0, n_new):
    rhoc_new[i] = central(z_new[i])

plt.plot(rhof, z, 'blue', label= 'dz=1000')
plt.scatter(rhof, z, color='blue', marker='o')
plt.plot(rhob, z, 'blue')
plt.scatter(rhob, z, color='blue', marker='o')
plt.plot(rhoc, z, 'blue')
plt.scatter(rhoc, z, color='blue', marker='o')

plt.plot(rhof_new, z_new, 'pink', label='dz=500')
plt.scatter(rhof_new, z_new, color='pink', marker='o')
plt.plot(rhob_new, z_new, 'pink')
plt.scatter(rhob_new, z_new, color='pink', marker='o')
plt.plot(rhoc_new, z_new, 'pink')
plt.scatter(rhoc_new, z_new, color='pink', marker='o')

plt.legend()
plt.ylabel('Height (m)')
plt.xlabel('Density (kg/m^3)')
plt.grid(True)
plt.show()
```

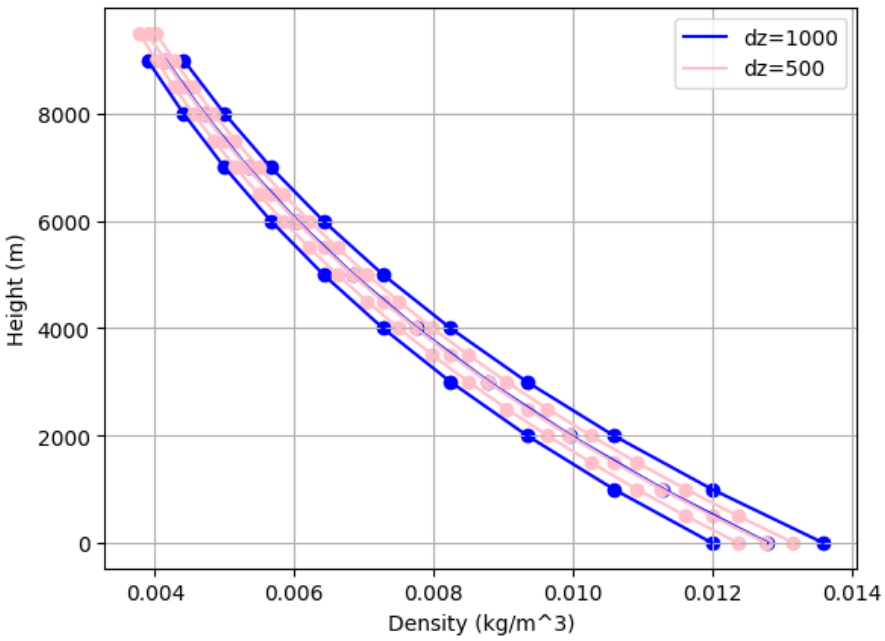

Figure 5: Comparison of dz=500 and dz=1000.

As being seen, compared to the when dz=1000, the overall shape of the plot when dz=500 remains the same, but it is more detailed. This plot contains more data points and less gaps between the points than the previous one. The additional data points provide a more detailed representation. So it can be concluded that changing the value of dz affects the density of data points on the plot, with a larger dz resulting in sparser data and a smaller dz resulting in denser data.

2.2 Planck curves

Given:

$$B_\lambda(T) = \frac{c_1 \lambda^{-5}}{\pi(\exp\left(\frac{c_2}{\lambda T}\right) - 1)}$$

with c1 = 3.74E-16 Wm^2 and c2 = 1.45E-2 mK

2.2.1 Plot the Planck curves for different black body temperatures T=6000K, 5000K, 4000K, 3000K.

I first create a list T that contains the temperatures. I choose the range of wavelength from 0.01 to 3 with a step size of 0.01. Then I apply the Planck function to calculate the wavelength $B_\lambda(T)$ for each temperature in the list.

Listing 6: Practice 2.1

```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [10, 5]

c1 = 3.74E-16
c2 = 1.45E-2

T = [6000, 5000, 4000, 3000]

lamda = np.arange(0.01, 3, 0.01)

for t in T:
    b = (c1 * (lamda * 10 ** -6) ** -5) / (np.pi * (np.e ** (c2 /
        (lamda * 10 ** -6 * t)) - 1))
    la = str(t) + "K"
    plt.plot(lamda, b, label=la)

plt.legend()
plt.show()
```

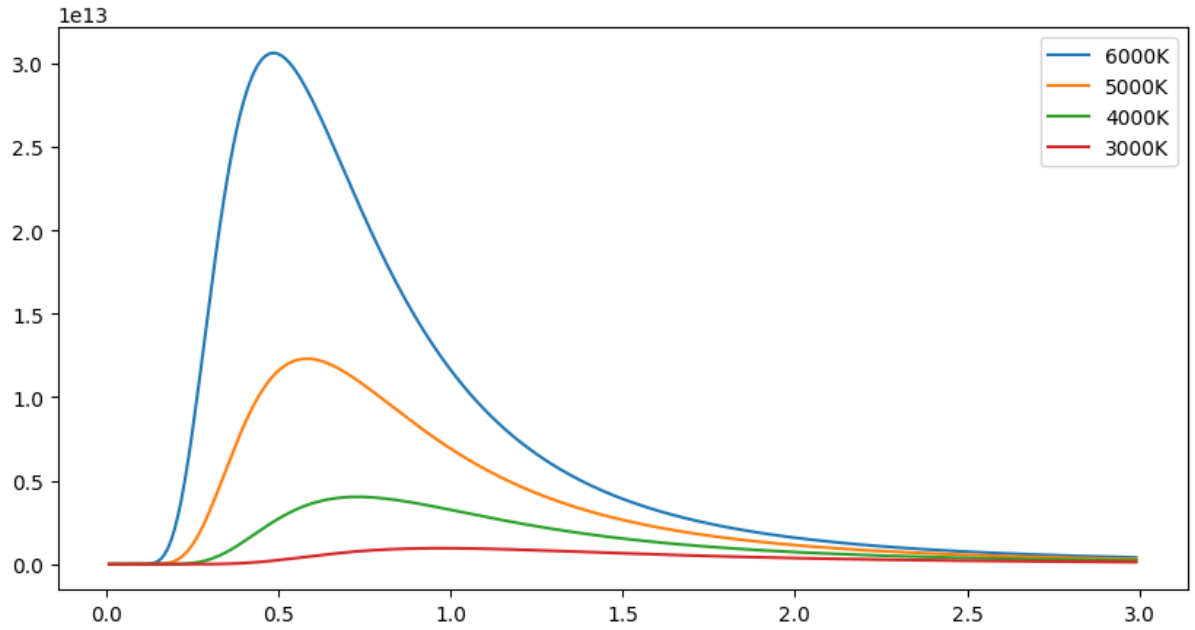

Figure 6: Plank curves for different black body temperatures.

As being seen, the higher the temperature, the higher the peaks are. This is because as the temperature increases, the intensity of radiation emitted by the black body also increases.

2.2.2 Print out the wavelength of peak emission for each body.

For this question, I use the Wien law's formula:

$$\lambda_{\max} = \frac{b}{T}$$

where λmax is the wavelength at which the spectral radiance of a black body is maximum (m), b is Wien's constant ($m \cdot K$) and T is the temperature of the black body (K).

First I the Wien's constant. I use for loop function to iterates through each value in the temperature list T from the previous code and let it calculates the wavelength of peak emission for that temperature then print.

Listing 7: Practice 2.2

```
b = 2.897771955E-3

for t in T:
    lambda_peak = b / t
    print(f"Wavelength of peak emission for a black body at {t} K
        is {lambda_peak} m")
```

```
Wavelength of peak emission for a black body at 6000K is 4.829619925e-07 m
Wavelength of peak emission for a black body at 5000K is 5.79554391e-07 m
Wavelength of peak emission for a black body at 4000K is 7.2444298875e-07 m
Wavelength of peak emission for a black body at 3000K is 9.65923985e-07 m
```

According to Wien's displacement law, the hotter an object is, the shorter the wavelength at which it emits the most energy. As being seen, the black body at 6000K emits the most energy at a wavelength of about 4.83E-07 m, followed by the one at 5000K and 4000K at longer wavelengths of 5.79E-07 m and 7.24E-07m, respectively. A black body at 3000K has a peak emission wavelength of 9.66E-07 m, therefore it is the body that emits the least energy among all.

2.2.3 Print out the total energy emitted by each body by using the Stefan-Boltzmann Law, and the Planck function.

Here is the Stefan-Boltzmann Law:

$$E = \sigma \cdot T^4$$

where E is the total energy emitted, σ is the Stefan-Boltzmann constant ($W/m^2 \cdot K^4$) and T is the temperature (K).

And the Planck function:

$$B(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{\exp\left(\frac{hc}{\lambda k T}\right) - 1}$$

where $B(\lambda, T)$ is the spectral radiance at wavelength λ at temperature T, h is the Planck constant ($J \cdot s$), c is the speed of light in a vacuum (m/s), λ is the wavelength, k is the Boltzmann constant (J/K) and T is the temperature (K).

As being seen, the Stefan-Boltzmann Law directly calculates the total energy emitted, but the Plank function doesn't, it calculates the amount of energy emitted per unit wavelength at a given temperature instead. My algorithm is to first choose a range that covers the peak wavelengths for the black body temperatures above, which is from 4E-7m to 1E-6m to consider, and divide the chosen range into an array of 1000 wavelengths. For each interval, I use the Planck function to calculate the spectral radiance and then integrate them over the wavelength range to obtain the total energy emitted.

In the following code, for Planck function, I use trapezoidal rule to integrate the radiance over the wavelength range. It is a method that approximates the area under a curve by dividing the interval of integration into a series of trapezoids and summing their areas. The formula for this rule is $a^b f(x)dx(b-a) * [f(a) + f(b)]/2$ but I just use the np.trapz() function in Python instead. First I import the needed constant, in which sigma is the Stefan-Boltzmann constant ($W/m^2 \cdot K^4$), h is the Planck constant ($J \cdot s$), c is the speed of light (m/s) and k is the Boltzmann constant (J/K). Then I use above formulas to calculate E and $B(\lambda, T)$ for each black body in the array T, which is the array I defined in previous practices that includes the temperatures, and print them out.

Listing 8: Practice 2.3

```
1 sigma = 5.670E-8
2 h = 6.626E-34
3 c = 3E8
4 k = 1.380649E-23
```

```
5
6  boltzmann = [sigma*t**4 for t in T]
7
8  for t, e in zip(T, boltzmann):
9      print(f"Total energy emitted by a black body at {t} K using
           Stefan-Boltzmann law is {e:4.0f} W/m^2")
10
11 lamda = np.linspace(4E-7, 1E-6, 1000)
12
13 planck = []
14 for t in T:
15     radiance = 2*h*c**2/lamda**5 * 1/(np.exp(h*c/(lamda*k*t))-1)
16     planck.append(np.trapz(radiance, lamda))
17
18 for t, e in zip(T, planck):
19     print(f"Total energy emitted by a black body at {t}K using
           Planck function is {e:4.0f} W/m^2")
```

```
Total energy emitted by a black body at 6000K using Stefan-Boltzmann law
is 73483200 W/m^2
Total energy emitted by a black body at 5000K using Stefan-Boltzmann law
is 35437500 W/m^2
Total energy emitted by a black body at 4000K using Stefan-Boltzmann law
is 14515200 W/m^2
Total energy emitted by a black body at 3000K using Stefan-Boltzmann law
is 4592700 W/m^2
Total energy emitted by a black body at 6000K using Planck function is
13958355 W/m^2
Total energy emitted by a black body at 5000K using Planck function is
6385094 W/m^2
Total energy emitted by a black body at 4000K using Planck function is
2126014 W/m^2
Total energy emitted by a black body at 3000K using Planck function is
395149 W/m^2
```

As being seen, the total energy emitted calculated using Stefan-Boltzmann law and Plank function have significant differences. This is due to the different assumptions and mathematical models of two methods. For Planck function, to get the total energy, it is a must to integrate it over the entire wavelength range. Therefore, it is able to describe the detailed distribution of the emitted radiation and provide a more accurate picture of the actual energy emitted by the black body. On the other hand, the Stefan-Boltzmann law is a simpler formula that doesn't account for the detailed wavelength dependence of the black body radiation. As a result, it tends to overestimate the total energy, especially at higher temperatures. In summary, the Stefan-Boltzmann law gives a higher but less accurate estimate of the total energy. The Planck function integration provides a more detailed and precise calculation, especially at higher temperatures .

2.3 0-D EBM

Given

$$\frac{(1-\alpha)S}{4} = \sigma T_e^4$$

with S is Solar constant, for the Earth, S=1370$W/m2$, α is planetary albedo, σ is Stefan-Boltzman constant ($W/m^2 \cdot K^4$) and T_e is the temperature.

Table 1: Data for practice 3

Planet	Distance (AU)	Albedo	Average Surface Temperature	Atmosphere
Venus	0.723	0.76	425°C	95 Atm, 96% CO2
Earth	1.000	0.32	15C	1 Atm, N2, O2, traceH2, CO2
Mars	1.524	0.16	-50°C	0.02 Atm, 95% CO2
Europa	5.203	0.64	-145°C	no atmosphere

2.3.1 Estimate the incoming solar flux for those planets.

The relationship between solar flux of a planet and its distance to the Sun is represented by the formula:

$$F = \frac{S}{D^2}$$

in which F is the solar flux (W/m^2), S is the amount of solar radiation received at the top of Earth's atmosphere, also known as the solar constant (W/m^2) and D is the distance from that planet to the Sun (AU).

First I import the solar constant S and the data about planet's names and their distances to the Sun. Then I calculate the solar fluxes and print them out with two decimal places.

Listing 9: Practice 3.1

```
def solar_flux(s, d):
    return s / (d**2)

s = 1370

data = {
    "Venus": 0.723,
    "Earth": 1.000,
    "Mars": 1.524,
    "Europa": 5.203
}

```

```
solar_fluxes = {
    planet: solar_flux(s, d)
    for planet, d in data.items()
}

for planet, flux in solar_fluxes.items():
    print(f"The solar flux for {planet} is {flux:.2f} W/m^2")
```

```
The solar flux for Venus is 2620.86 W/m^2
The solar flux for Earth is 1370.00 W/m^2
The solar flux for Mars is 589.86 W/m^2
The solar flux for Europa is 50.61 W/m^2
```

As being seen, the solar flux decreases as the distance increases. That's why the solar flux for Venus, the closet planet to the Sun, is highest among all at 2620.86 W/m^2. The second highest amount of solar radiation received is on Earth's surface, 1370 W/m^2, which is also referred to as the solar constant. The solar flux for Mars is 589.86 W/m^2. Compared to Earth, it receives significantly less solar radiation due to its greater distance from the Sun. Europa, the farthest object from the Sun receives the lowest solar flux at 50.61 W/m^2.

2.3.2 Estimate their black body equivalent temperature.

I rearrange the given formula with black body equivalent temperature on one side:

$$T_e = \sqrt[4]{\frac{(1-\alpha)S}{4\sigma}}$$

First I import the Stefan-Boltzmann constant ($W/m^2 \cdot K^4$) as sigma, as well as the data about the solar flux and albedo for each planet. Then I use the above formula to calculate black body temperature t and print them out.

Listing 10: Practice 3.2

```
sigma = 5.670E-8

def blackbody_t(flux, albedo):
    t = ((1 - albedo) * flux / (4 * sigma))**(1/4)
    return t

flux = {
    "Venus": 2620.86,
    "Earth": 1370.00,
    "Mars": 589.86,
    "Europa": 50.61
}

albedos = {
    "Venus": 0.76,
    "Earth": 0.32,
```

```
    "Mars": 0.16,
    "Europa": 0.64
}

for planet, flux in flux.items():
    albedo = albedos[planet]
    t = blackbody_t(flux, albedo)
    print(f"The black body equivalent temperature for {planet} is
        {t} K")
```

```
The black body equivalent temperature for Venus is 229.48431298446596 K
The black body equivalent temperature for Earth is 253.16088628145764 K
The black body equivalent temperature for Mars is 216.19532352212696 K
The black body equivalent temperature for Europa is 94.67252209502239 K
```

The highest to lowest temperature rank is Earth, Venus, Mars and Europa, respectively. Venus has the highest solar flux and the highest albedo among all, which mean it receives the most incoming solar radiation but also reflects a significant amount of the radiation. The combination of high flux and high albedo results in a relatively high black body equivalent temperature. The solar flux of the Earth is lower than that of Venus due to its greater distance from the Sun. Its albedo is lower than Venus too, which means it absorbs a larger amount of the incoming solar radiation, results in a black body equivalent temperature that is close to the actual average surface temperature of Earth, which is around 288K. Mars has low solar flux and low albedo, also. This means that Mars receives and reflects low amount of solar radiation, resulting in a black body equivalent temperature that is close to the actual average surface temperature of Mars, which is around 210K. Europa has the lowest solar flux and lowest albedo among all due to it great distance to the Sun. Its black body equivalent temperature is much lower than the actual surface temperature, which is around 100-130K due to tidal heating from Jupiter.

2.3.3 Plot: T_e versus Solar fluxes and comment on the obtained results.

I import the data calculated earlier about the four planets' solar flux and black body equivalent temperature as flux and t, respectively. Then I plot the graph with solar flux as the x-axis and black body temperature as y-axis.

Listing 11: Practice 3.3

```
import matplotlib.pyplot as plt

flux = {"Venus": 2620.86, "Earth": 1370.00, "Mars": 589.86, "
    Europa": 50.61}
t = {"Venus": 229.48, "Earth": 253.16, "Mars": 216.19, "Europa":
    94.67}

plt.figure()
plt.scatter(list(flux.values()), list(t.values()), color = '
    deeppink')
plt.xlabel("Solar Flux")
plt.ylabel("Black Body Equivalent Temperature")
```

```
for planet in flux:
    plt.annotate(planet, (flux[planet], t[planet]))
plt.grid()
plt.show()
```

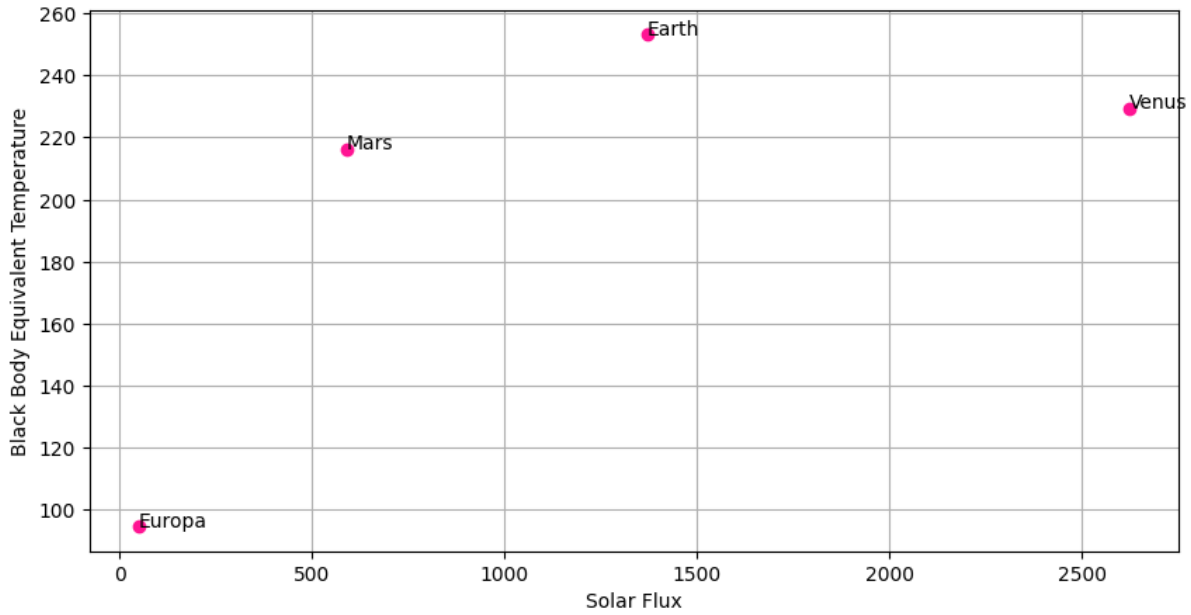

Figure 7: Black body equivalent temperature versus Solar flux.

As being seen, when the solar flux increases, the black body equivalent temperature also tend to increases, however the relationship between these two features is not perfectly linear. This is because the black body temperature depends not only on the solar flux, but also on the albedo. For instance, Venus has the highest solar flux among all, but also high albedo and it reflects a large amount of solar radiantion received, therefore it doesn't have the highest black body equivalent temperature. Planets with higher albedo, like Europa, will have lower black body temperatures compared to planets with lower albedo, like Mars, even if they receive the same solar flux. Overall, higher solar flux leads to higher black body temperatures, which in turn affects the overall climate and habitability of the planets.

Moreover, the surface temperature of Venus is 698.15K, while its black body temperature is 229.48K. For Earth, the surface temperature is 288.15K and the black body temperature is 253.16K. On Mars, the surface temperature is 223.15K and the black body temperature is 216.19K. Finally, the surface temperature of Europa is 128.15K, while its black body temperature is 94.67K. As being seen, when compared to the surface temperature, the black body equivalent temperature don't have a too much of the different except for Venus. This is because of its high albedo.

2.4 0-D EBM (cont.)

Given the solar constant of 1368 W/m^2, σ=5,67E-8 W/m^2K^4 is the Stefan-Boltzmann constant. α is the planetary albedo. Given the outgoing longwave radiation of the planet by $\epsilon\sigma T_s^4$, where ϵ is the emissivity of the atmosphere , T_s is the surface temperature ($°C$.).

Knowing that the relationship between the the outgoing longwave radiation and the albedo is expressed as:

$$O = (1-\alpha)S$$

where O is the outgoing longwave radiation.

Substituting the given formula of O, we have:

$$(1-\alpha)S = \epsilon\sigma T_s^4$$

And I rearrange to get a formula with surface temperature on one side:

$$T_s = \sqrt[4]{\frac{(1-\alpha)S}{\epsilon\sigma}}$$

2.4.1 α=0,32 , write a Python program to plot the dependence of T_s on ϵ.

I import the solar constant (W/m^2) as S, Stefan-Boltzmann constant (W/m^2K^4) as sigma and albedo as albedo. The full range of emissivity values is from 0, where the atmosphere is the leass emissive, to 1, where the atmosphere is the most emissive. However, in the above formula, ϵ is at the denominator, and to avoid a division by zero, I choose the range of ϵ from 0.01 to 1. Then I plot the relationship between surface temperature and emissivity with T_s along the y-axis and ϵ along the x-axis.

Listing 12: Practice 4.1

```
import numpy as np
import matplotlib.pyplot as plt

S = 1368
sigma = 5.67E-8
albedo = 0.32

epsilon = np.linspace(0.01, 1, 100)

Ts = []
for e in epsilon:
    Ts.append(((1 - albedo) * S / (e * sigma))**(1/4))

plt.figure()
plt.plot(epsilon, Ts, color = 'deeppink')
plt.xlabel("Emissivity of the atmosphere")
plt.ylabel("Surface temperature")
plt.grid()
plt.show()
```

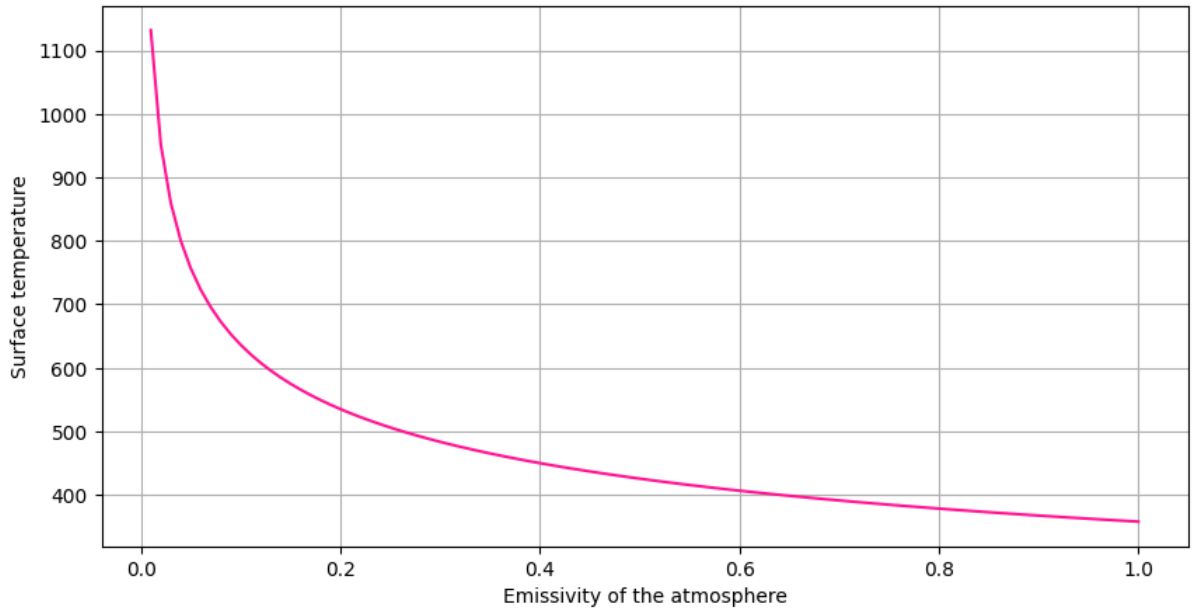

Figure 8: Dependence of Surface temperature on Emissivity.

Emissivity is a measurement of how efficiently the surface radiates energy, therefore when the emissivity goes up, the surface is better able to radiate heat away, leading to a lower temperature. As being seen in the graph, when emissivity increases from 0.01 to 1, the surface temperature decreases.

2.4.2 ϵ=0.66, write a Python program to plot the dependence of T_s on α.

After define emissitivy as epsilon, I choose the range of albedo, as alpha, from 0, where there is no reflection, to 1, where the surface reflects all the incoming solar radiation. Then I use the same formula to calculate and plot the relationship between surface temperature and albedo.

Listing 13: Practice 4.2

```
epsilon = 0.66

alpha = np.linspace(0, 1, 100)
Ts = (S * (1 - alpha) / (epsilon * sigma))**(1/4)

plt.figure()
plt.plot(alpha, Ts, color = 'deeppink')
plt.xlabel('Albedo')
plt.ylabel('Surface Temperature')
plt.grid()
plt.show()
```

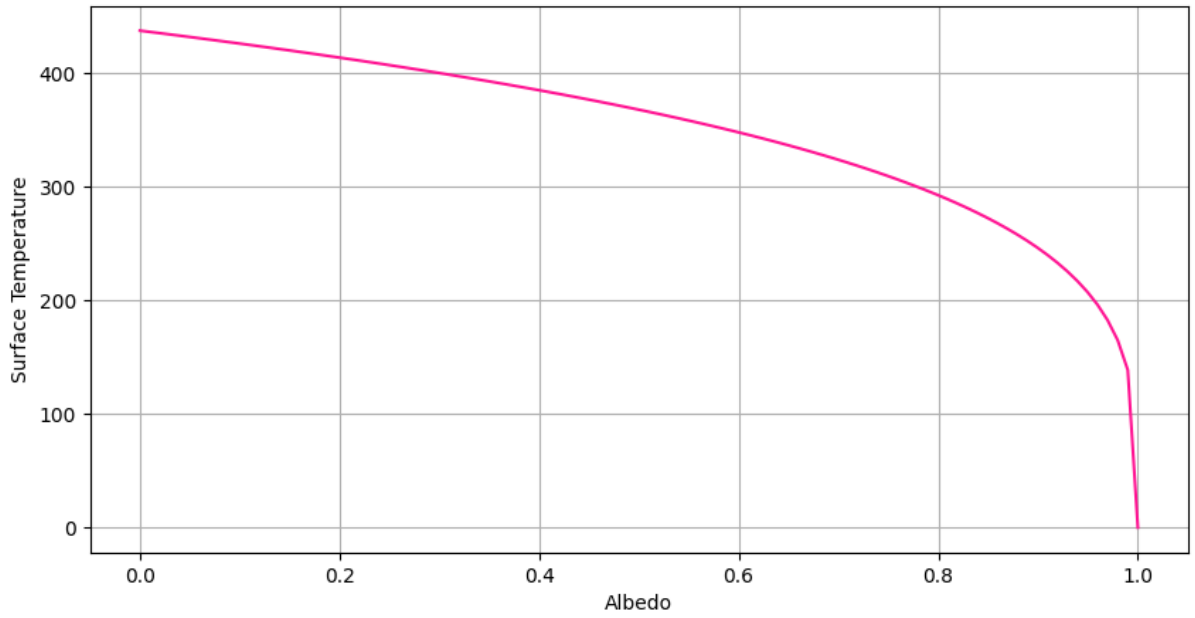

Figure 9: Dependence of Surface temperature on Albedo.

As being seen, the curve is decreasing, which means that when the albedo increases from 0 to 1, the surface temperature steadily drops. At an albedo of 0, all the incoming solar radiation is absorbed, resulting in the maximum surface temperature. On the other hand, an albedo of 1 means that all the solar radiation is reflected, and there is nothing absorbed by the surface, so that the surface temperature would be the lowest. In summary, higher albedo results in less solar radiation being absorbed, leading to reduced temperature of the surface.

2.4.3 Write a Python program to plot the dependence of T_s on both ϵ and α

I choose the range of emissivity from 0.1 to 1, and the range of albedo from 0 to 1. I set the x axis to epsilon and the y axis to alpha. Then I use the same formula to calculate surface temperature, just that ϵ and α are replaced with x and y, respectively. The surface temperature T_s is set to the z axis.

Listing 14: Practice 4.3

```
from mpl_toolkits.mplot3d import Axes3D

epsilon = np.linspace(0.1, 1, 100)
alpha = np.linspace(0, 1, 100)

x, y = np.meshgrid(epsilon, alpha)
Ts = ( (1 - y)*S / (x*sigma) ) ** (1/4)

fig = plt.figure(figsize=(15, 5))
ax = plt.axes(projection='3d')
surf = ax.plot_surface(x, y, Ts, cmap='Blues')

ax.set_xlabel('Emissivity')
ax.set_ylabel('Albedo')
ax.set_zlabel('Surface Temperature')
```

```
16
17 cbar = fig.colorbar(surf)
18
19 plt.show()
```

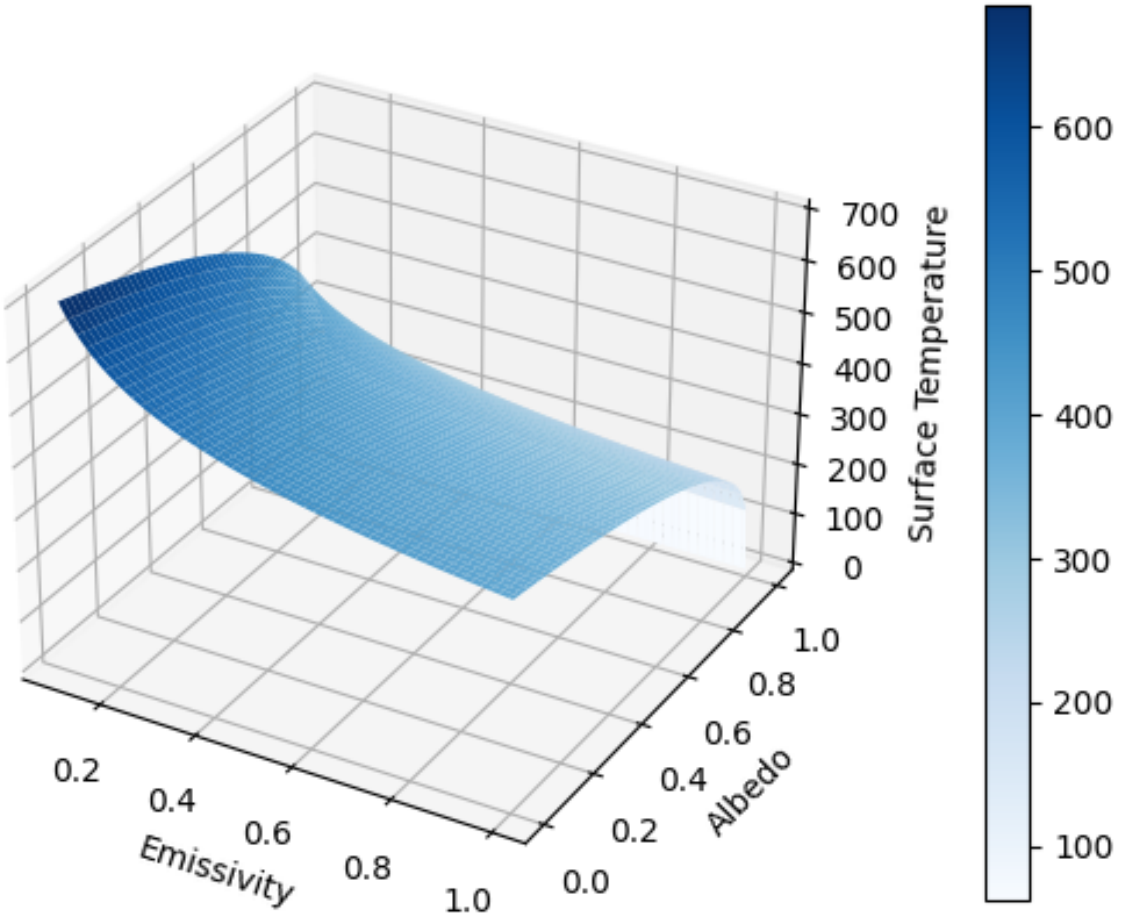

Figure 10: Dependence of Surface temperature on both Emissitivy and Albedo.

As being seen, the variation of function if only one variable change (when look across one axis) are the same as previous problems. The hottest surface temperature occurs at the lowest emissivity and albedo, when they are close to 0, because this is when the absorption of solar radiation is maximum while the reflection is minimum. On the other hand, at the highest albedo and emissivity, when they nearly reach 1, when the absorption of solar radiation is minimum and the emission is maximum, the surface temperature is lowest.

2.5 Daisy world

2.5.1 Write a program to represent the Daisy World with two species: black and white.

In Daisy World, the white daisy reflects solar radiation while the black daisy absorbs it. When the sun's luminosity is low, the black daisy takes advantage and grow rapidly. As the black daisy cover more area, they absorb more sunlight, causing the temperature to rise. However, as the temperature rises, the white daisy begins to grow to cool down the planet. This interplay between the two species creates a loop, where the temperature remains stable despite the changes in the sun's luminosity.
The temperature is calculated by:

$$T_e = \sqrt[4]{\frac{(1-\alpha)S_0}{4\epsilon\sigma}} - 273.15$$

in which S is the solar constant, ϵ is the emissivity, and σ is the Stefan-Boltzmann constant.

The growth rate of black daisy and white daisy is calculated by:

$$f_b = max(0, 1 - k(T_0 - T_b)^2)$$

$$f_w = max(0, 1 - k(T_0 - T_w)^2)$$

in which k is the growth rate constant, T_0 is the peak growth temperature, and T_b and T_w are the temperature for the black and white Daisy World, respectively.

I use the above formulas to shows how the coverage of the two daisy species can help stabilize the planetary temperature. In the following code, L is the luminosity of the sun, T_opt is the optimal temperature for daisies, T_max and T_min are the maximum and minimum temperature for species, k is the growth rate constant.

Listing 15: Practice 5.1

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

L = 1.0
S = 1368
sigma = 5.67E-8
albedo_black = 0.25
albedo_white = 0.75
albedo_surface = 0.5
T_opt = 295
temp_range = 10

def planetary_temperature(w, b, L):
    albedo = w * albedo_white + b * albedo_black + (1 - w - b) *
       albedo_surface
    return ((L * S * (1 - albedo)) / (4 * sigma)) ** 0.25
```

```
17
18 def growth_rate(T):
19     return max(0, 1 - ((T - T_opt) / temp_range) ** 2)
20
21 def simulate_daisy_world(w, b):
22     T = planetary_temperature(w, b, L)
23     f_w = growth_rate(T)
24     f_b = growth_rate(T)
25     dw_dt = f_w * w * (1 - (w + b))
26     db_dt = f_b * b * (1 - (w + b))
27     return T, dw_dt, db_dt
28
29 w_range = np.linspace(0, 1, 50)
30 b_range = np.linspace(0, 1, 50)
31 w, b = np.meshgrid(w_range, b_range)
32
33 T = np.zeros_like(w)
34 for i in range(len(w_range)):
35     for j in range(len(b_range)):
36         T[i, j], _, _ = simulate_daisy_world(w[i, j], b[i, j])
37
38 fig = plt.figure(figsize=(15, 5))
39 ax = fig.add_subplot(111, projection='3d')
40 surf = ax.plot_surface(w, b, T, cmap='Blues')
41
42 ax.set_xlabel('White Daisy Coverage')
43 ax.set_ylabel('Black Daisy Coverage')
44 ax.set_zlabel('Temperature')
45
46 cbar = fig.colorbar(surf)
47
48 plt.show()
```

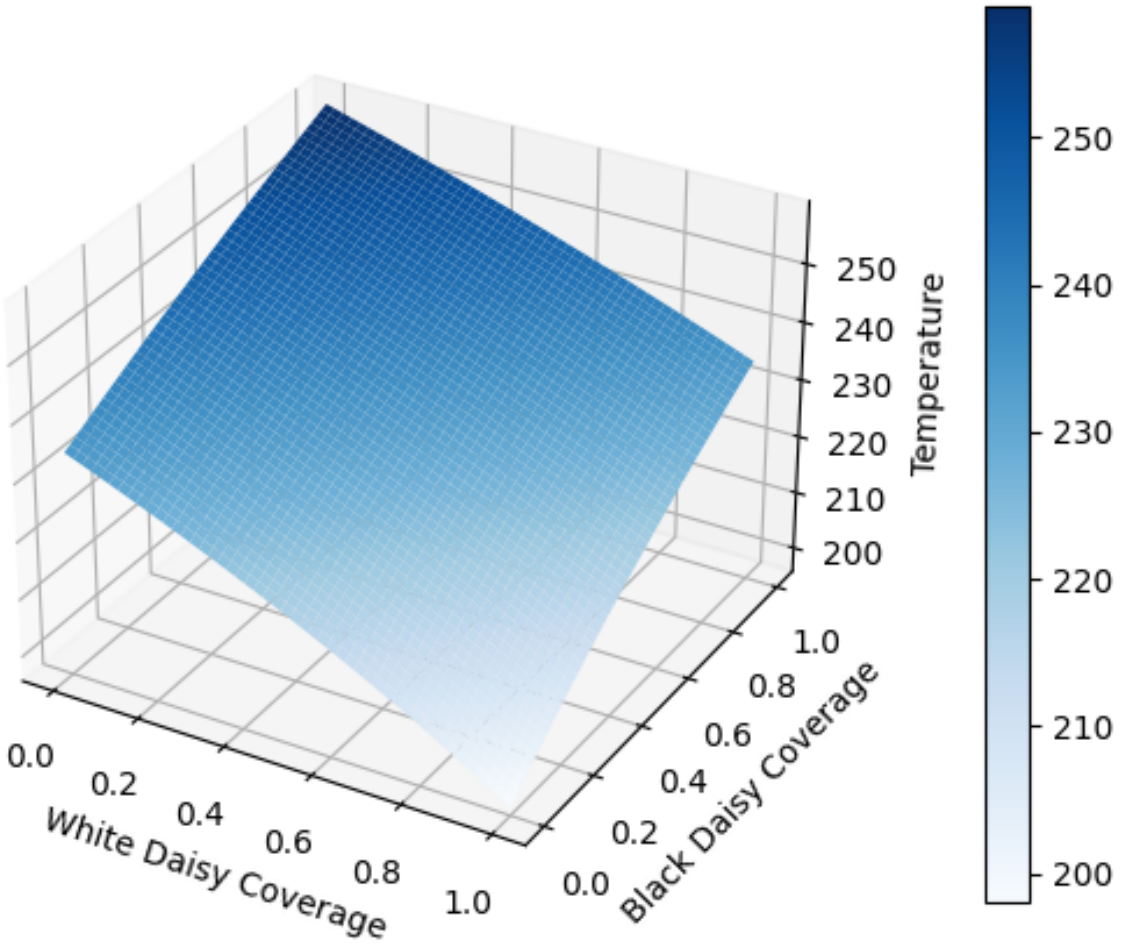

Figure 11: Relationship between White and Black Daisy Coverage versus Temperature.

As being seen, in the central region of the plot, where the white and black daisy both exist, the temperature remains stable and close to 25 °C, the optimal value. When the temperature is too high or too low, the growth rate functions approach zero, and the delicate balance is disrupted, leading to the collapse of the daisy populations. As the temperature increases, the black daisy grows fast to absorbs the solar radiation, so that its area coverage goes up too. On the other hand, the white daisy coverage increases when the temperature decreases, because white daisy grows to reflect solar radiation to help balance the planetary temperature.

2.5.2 Plot the growth rate function f_b and f_w versus temperature.

First I import the needed constant, in which k is the growth rate constant and T_0 is the peak growth temperature. Then I choose the same range of temperature T from 0 degree to 50 degrees for both black and white Daisy World. Then I use the given formula to calculate the growth rate for them and plot. The y-axis is growth rate and the x-axis is temperature. I set the colour of black Daisy World to blue and that of white Daisy World to pink.

Listing 16: Practice 5.2

```
1 iimport numpy as np
2 import matplotlib.pyplot as plt
```

```
3
4  k = 0.003265
5  T0 = 22.5
6  T = np.arange(0, 50, 1)
7
8  fb_array = []
9  fw_array = []
10
11 for i in T:
12     fb = max(0,1-k*(T0-i)**2)
13     fb_array.append(fb)
14     fw = max(0,1-k*(T0-i)**2)
15     fw_array.append(fw)
16
17 fig,(ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
18
19 ax1.plot(T, fb_array, color='blue')
20 ax1.set_xlabel("Temperature")
21 ax1.set_ylabel("Growth Rate")
22
23 ax2.plot(T, fw_array, color='pink')
24 ax2.set_xlabel("Temperature")
25 plt.tight_layout(w_pad=0)
26
27 plt.show()
```

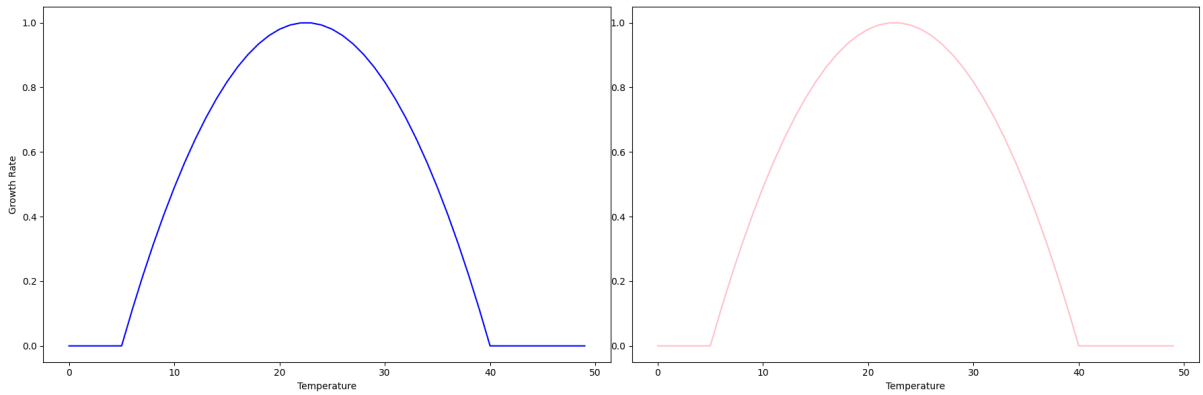

Figure 12: Growth rate for black and white Daisy World, respectively.

As being seen, the two graph plotted for black and white Daisy World are identical. The parabolic curve extends from approximately 5 °C to 40 °C, this is the range in which daisies can thrive. The peak at the middle of the graph is at 22.5 °C, where the daisies grow the fastest.

2.5.3 S=S0=1000; at the initial stage b=w=0.2. Find the equilibrium b and w values.

Given that the growth rate function is:

$$f_b = max(0, 1 - k(T_0 - T_b)^2)$$

$$f_w = max(0, 1 - k(T_0 - T_w)^2) \tag{1}$$

In that, T_b and T_w are the temperatures of the black and white daisy and is calculated by:

$$T_b = T_e + (\alpha - \alpha_b)$$
$$T_w = T_e + (\alpha - \alpha_w) \tag{2}$$

in which α is the effective albedo and T_e is the temperate.

The albedo is calculated by:

$$\alpha = b\alpha_b + w\alpha_w + (1 - b - w)\alpha_g$$

The temperature is calculated by:

$$T_e = \sqrt[4]{\frac{(1-\alpha)S_0}{4\epsilon\sigma}} - 273.15$$

Substituing 1 into 2:

$$f_b = max(0, 1 - k(T_0 - (T_e + (\alpha - \alpha_b)))^2)$$

$$f_w = max(0, 1 - k(T_0 - (T_e + (\alpha - \alpha_w)))^2)$$

At equilibrium, the changes in b and w is zero

$$db = b(xf_b - d) = 0$$

$$dw = w(xf_w - d) = 0$$

in which x = 1 - b - w. Solving this system of equations would give the equilibrium values of b and w.

In the following code, first I import the given solar flux S0, given b and w, and other needed constants, in which epsilon is the emissivity of the surface, q is the number of iterations, k is the growth rate constant, T0 is the initial temperature, d is the decay rate of the variables b and w, sigma is the Stefan-Boltzmann constant, alpha_black, alpha_white, and alpha_ground are the albedo values for black, white, and ground surfaces, respectively, and eps_ref is the reference value for the convergence criterion. I define a low albedo of 0.25 for black daisy, a high albedo of 0.75 for white daisy, and a medium albedo of 0.5 for bare ground. I use the algorithm explained above to calculate the temperature and the area. It repeats until the changing in area is very small, which mean it is stable, and then print it out.

Listing 17: Practice 5.3

```
1 S0 = 1000
2 b = 0.2
3 w = 0.2
4
```

```
epsilon = 0.3
q = 20
k = 0.003265
T0 = 22.5
d = 0.3
sigma = 5.67e-8
alpha_black = 0.25
alpha_white = 0.75
alpha_ground = 0.5

eps = 1
eps_ref = 1E-8

while eps > eps_ref:
    x = 1-b- w
    alpha = b*alpha_black + w*alpha_white + x*alpha_ground
    Te = ((1-albedo)*S0 / (4*epsilon*sigma)) ** (1/4) - 273.15
    T_black = Te+(albedo-alpha_black)
    T_white = Te+(albedo-alpha_white)
    f_black = max(0,1-k*(T0-T_black) ** 2)
    f_white = max(0,1-k*(T0-T_white) ** 2)
    db = b*(x*f_black-d)
    dw = w*(x*f_white-d)
    b = b+db
    w = w+dw

    eps = abs(db+dw)

print("Equilibrium value of b is:", b)
print("Equilibrium value of w is:", w)
```

```
Equilibrium value of b is: 0.3500872084464622
Equilibrium value of w is: 0.3498501488641292
```

As being seen, the equilibrium values of b and w are both around 0.35, which means that at equilibrium, the surface area is roughly equally divided between the black, white, and ground surface (since x = 1 - b - w). This means that the system has reached a balance between the competing effects of the different surface on the overall albedo and temperature.

2.5.4 $\frac{S}{S_0}$ varies from 0.4 to 1.6. Plot the equilibrium daisy black/white area versus $\frac{S}{S_0}$. Plot the planetary temperature versus $\frac{S}{S_0}$.

First I import the constants, in which sigma is the Stefan-Boltzmann constant (W/m^2K^4), epsilon is the emissivity of the surface, d is the decay rate for the daisies, s0 is the incoming solar flux, alb_white, alb_black and alb_bare are the albedo of the white daisies, black daisies and bare surface respectively, eps_ref is the reference value for the convergence criterion, T0 is the initial temperature, k is the growth rate constant, a_ini_white and a_ini_black are the initial fraction of the white daisies and black daisies, mf_it is the maximum fraction of the planet's surface that can be covered by daisies in each

iteration, ns is the number of steps or iterations in the loop.

The interactions between the solar flux ratio, temperature, and equilibrium area are nonlinear and can exhibit multiple stable states or equilibria. To explore the full range of possible stable states, I let the loop run in two different directions. It starts in the middle of the data and goes towards the end, and then it starts in the middle and goes towards the beginning.

For each step of the loop, I let the algorithm uses the following formula to calculate.

The planetary albedo:

$$\alpha_p = w\alpha_w + b\alpha_b + x\alpha_g$$

where α_w, α_b, and α_g are the albedos of white daisies, black daisies, and bare ground, respectively.

Stefan-Boltzmann law to calculate the planet temperature:

$$T_e = \sqrt[4]{\frac{S(1-\alpha_p)}{4\epsilon\sigma}} - 273.15$$

where S is the solar constant, α is the albedo, ϵ is the emissivity, and σ is the Stefan-Boltzmann constant.

The local temperatures of the black and white daisies:

$$T_b = T_e + q(\alpha_p\alpha_b)$$

$$T_w = T_e + q(\alpha_p\alpha_w)$$

where T_b and T_w are the local temperatures for black and white daisies, and q is a constant used to calculate local temperature as a function of albedo.

The growth rate of the white and black daisies:

$$f_b = max(0, 1 - k(T_0 - T_b)^2)$$

$$f_w = max(0, 1 - k(T_0 - T_w)^2)$$

where f_b and f_w are the growth rates for black and white daisies, k is a constant, and T_0 is the temperature at which growth occurs within a range and peaks.

The change in area of black and white daisies over time:

$$\frac{db}{dt} = b(xf_b - d)$$

$$\frac{dw}{dt} = w(xf_w - d)$$

where $\frac{db}{dt}$ and $\frac{dw}{dt}$ is the change in area of black and white daisies, d is the death rate, and t is time.

The new area of black and white daisies:

$$b = b + \frac{db}{dt}$$

$$w = w + \frac{dw}{dt}$$

I store the results in the corresponding arrays and plot two graphs. The first graph is the relationship between the solar flux ratio and the area fractions of the black Daisy World, white Daisy World, and the bare surface. The second one is the relationship between the solar flux ratio and the temperatures of the system with and without the daisies.

Listing 18: Practice 5.4

```
sigma = 5.6696e-8
epsilon = 0.3
d = 0.3
s0 = 1000
alb_white = 0.75
alb_black = 0.25
alb_bare = 0.5
q = 20
eps_ref = 1e-8
T0 = 22.5
k = 0.003265
a_ini_white = 0.2
a_ini_black = 0.2
mf_it = 0.01
ns = 121

sol_flx_fnl = np.zeros(ns)
a_bare_fnl = np.zeros(ns)
a_black_fnl = np.zeros(ns)
a_white_fnl = np.zeros(ns)
tem_ini_fnl = np.zeros(ns)
tem_pln_fnl = np.zeros(ns)
fw_fnl = []
fb_fnl = []
tmp_b = []
tmp_w = []

ilogic = 1
while ilogic != 0:
    a_black = a_ini_black
    a_white = a_ini_white
    if ilogic == 1:
        irange = range(int(ns/2), ns, 1)
    elif ilogic == -1:
        irange = range(int(ns/2), -1, -1)
    for i in irange:
        mf = 1 + (i - int(ns/2)) * mf_it
```

```
        sol_flx = s0 * mf
        eps = 1.0

        while eps > eps_ref:
            a_bare = 1 - a_black - a_white
            alb_pln = a_bare * alb_bare + a_black * alb_black +
               a_white * alb_white
            tem_pln = (sol_flx * (1 - alb_pln) / sigma) ** (1/4)
               - 273.15
            tem_black = tem_pln + q * (alb_pln - alb_black)
            tem_white = tem_pln + q * (alb_pln - alb_white)
            fb = max(0., 1 - k * (T0 - tem_black) ** 2)
            fw = max(0., 1 - k * (T0 - tem_white) ** 2)
            if -10. <= tem_black <= 70. and fb >= 0:
                tmp_b.append(tem_black)
                fb_fnl.append(fb)
            if -10. <= tem_white <= 70. and fw >= 0:
                tmp_w.append(tem_white)
                fw_fnl.append(fw)
            db_dt = a_black * (a_bare * fb - d)
            dw_dt = a_white * (a_bare * fw - d)
            a_black = a_black + db_dt
            a_white = a_white + dw_dt
            eps = abs(db_dt) + abs(dw_dt)
            sol_flx_fnl[i] = mf
            a_bare_fnl[i] = a_bare
            a_black_fnl[i] = a_black
            a_white_fnl[i] = a_white
            tem_ini_fnl[i] = (sol_flx * (1 - alb_bare) / sigma)
               ** (1/4) - 273.15
            tem_pln_fnl[i] = tem_pln
    if ilogic == 1:
        ilogic = -1
    else:
        ilogic = 0

fig = plt.figure()
plt.plot(sol_flx_fnl, a_black_fnl, 'blue', label="Black Daisy")
plt.plot(sol_flx_fnl, a_white_fnl, 'green', label='White Daisy')
plt.plot(sol_flx_fnl, a_bare_fnl, 'pink', label="Bare")
plt.legend(loc='upper center')
plt.xlabel("S/S0")
plt.ylabel("Equilibrium Daisy area")
plt.grid(True)

fig = plt.figure()
plt.plot(sol_flx_fnl, tem_pln_fnl, 'red', label="With Daisy")
plt.plot(sol_flx_fnl, tem_ini_fnl, 'yellow', label="Bare")
plt.legend()
plt.xlabel("S/S0")
plt.ylabel("Temperature")
```

```
86 plt.grid(True)
```

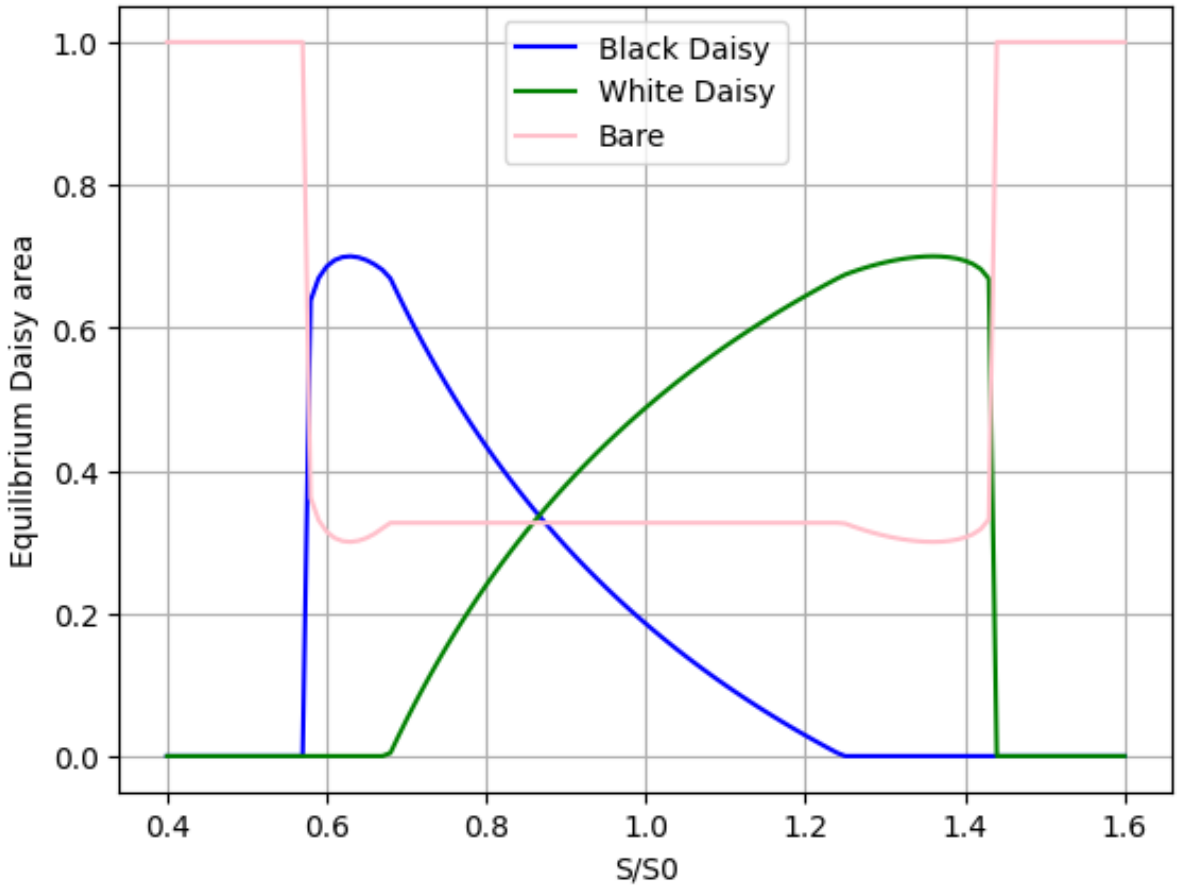

Figure 13: S/S0 versus black and white daisy area.

As being seen, there is clearly an opposite in behavior of the black and white daisy in response to the change of solar flux. As the solar flux goes higher and the planet temperature be hotter, black daisy growth decrease and so is their area. The white daisy enhance their growth as well as area. As the solar flux goes lower and the planet temperature be cooler, the opposite happens, the white daisy growth rate goes down and the black daisy dominates. When the solar flux is too high or too low, both white and black daisy will die.

Moreover, when there are no daisies, in that range of solar flux, the equilibrium area stays stable.

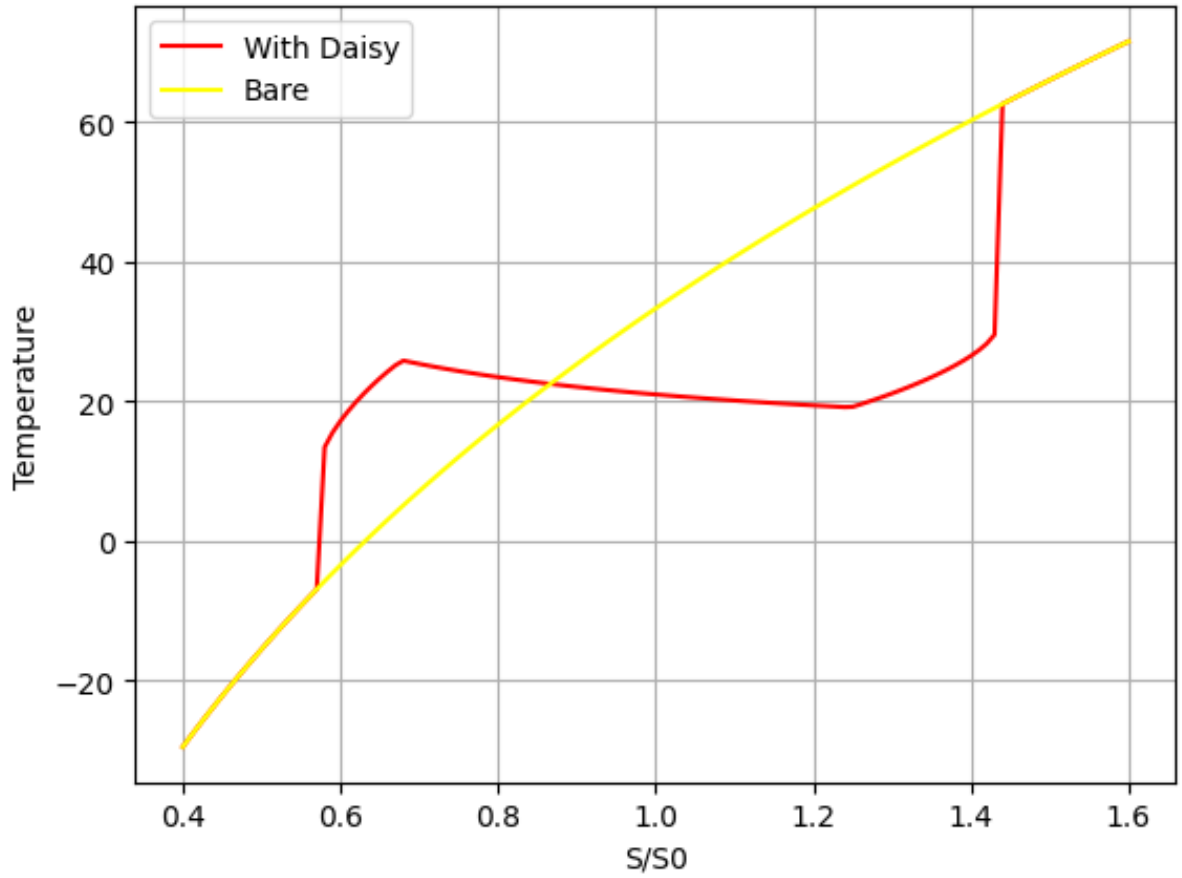

Figure 14: S/S0 versus planetary temperature.

As being seen, when there are daisies, within the range of solar flux where they can live, when it is low, the temperature is cold. In this scenario, the black daisies will thrive, as they can absorb more of the incoming solar flux and raise the planetary temperature. On the other hand, when the solar flux is high and it becomes hotter, the white daisies will thrive, as they can reflect the incoming solar flux and prevent the planet from becoming too hot. At the range of $\frac{S}{S_0}$ from $\tilde{0}.6$ to $\tilde{1}.4$, the temperature is quite stable, the white daisy reflect solar radiation as well as the black daisy help absorb it to balance the planetary temperature.

When there are no daisies, the planetary temperature is completely dependent on the solar flux and seems to be linear. The higher the solar flux, the hotter the planet is and vice versa.

2.5.5 Suppose that the Daisy world has only one specie, black daisy. How does the behavior of the system change?

For this problem, I use the same algorithm as the previous ones, just that I remove all the constants and functions related to white daisy.

Listing 19: Practice 5.5

```
import numpy as np
import matplotlib.pyplot as plt

sigma = 5.6696E-8
epsilon = 0.3
d = 0.3
s0 = 1000
alb_black = 0.25
alb_bare = 0.5
q = 20
eps_ref = 1e-8
T0 = 22.5
k = 0.003265
a_ini_black = 0.4
mf_it = 0.01
ns = 121

sol_flx_fnl = np.zeros(ns)
a_bare_fnl = np.zeros(ns)
a_black_fnl = np.zeros(ns)
tem_ini_fnl = np.zeros(ns)
tem_pln_fnl = np.zeros(ns)
fb_fnl = []
tmp_b = []

ilogic = 1

while ilogic != 0:
    a_black = a_ini_black

    if ilogic == 1:
        irange = range(int(ns/2), ns, 1)

    elif ilogic == -1:
        irange = range(int(ns/2), -1, -1)

    for i in irange:
        mf = 1 + (i - int(ns/2)) * mf_it
        sol_flx = s0 * mf
        eps = 1.0

        while eps > eps_ref:
            a_bare = 1 - a_black
```

```
            alb_pln = a_bare * alb_bare + a_black * alb_black
            tem_pln = (sol_flx * (1 - alb_pln) / sigma) ** (1/4)
               - 273.15
            tem_black = tem_pln + q * (alb_pln - alb_black)
            fb = max(0., 1 - k * (T0 - tem_black) ** 2)
            if -10. <= tem_black <= 70. and fb >= 0:
                tmp_b.append(tem_black)
                fb_fnl.append(fb)
            db_dt = a_black * (a_bare * fb - d)
            a_black = a_black + db_dt
            eps = abs(db_dt)
            sol_flx_fnl[i] = mf
            a_bare_fnl[i] = a_bare
            a_black_fnl[i] = a_black
            tem_ini_fnl[i] = (sol_flx * (1 - alb_bare) / sigma)
               ** (1/4) - 273.15
            tem_pln_fnl[i] = tem_pln

    if ilogic == 1:
        ilogic = -1

    else:
        ilogic = 0

fig = plt.figure()

plt.plot(sol_flx_fnl, a_black_fnl, 'blue', label="Black Daisy")
plt.plot(sol_flx_fnl, a_bare_fnl, 'pink', label="Bare")
plt.legend(loc='upper right')
plt.xlabel("S/S0")
plt.ylabel("Equilibrium Daisy area")

plt.grid(True)

fig = plt.figure()

plt.plot(sol_flx_fnl, tem_pln_fnl, 'blue', label="With Daisy")
plt.plot(sol_flx_fnl, tem_ini_fnl, 'pink', label="Bare")
plt.legend()
plt.xlabel("S/S0")
plt.ylabel("Temperature")

plt.grid(True)
```

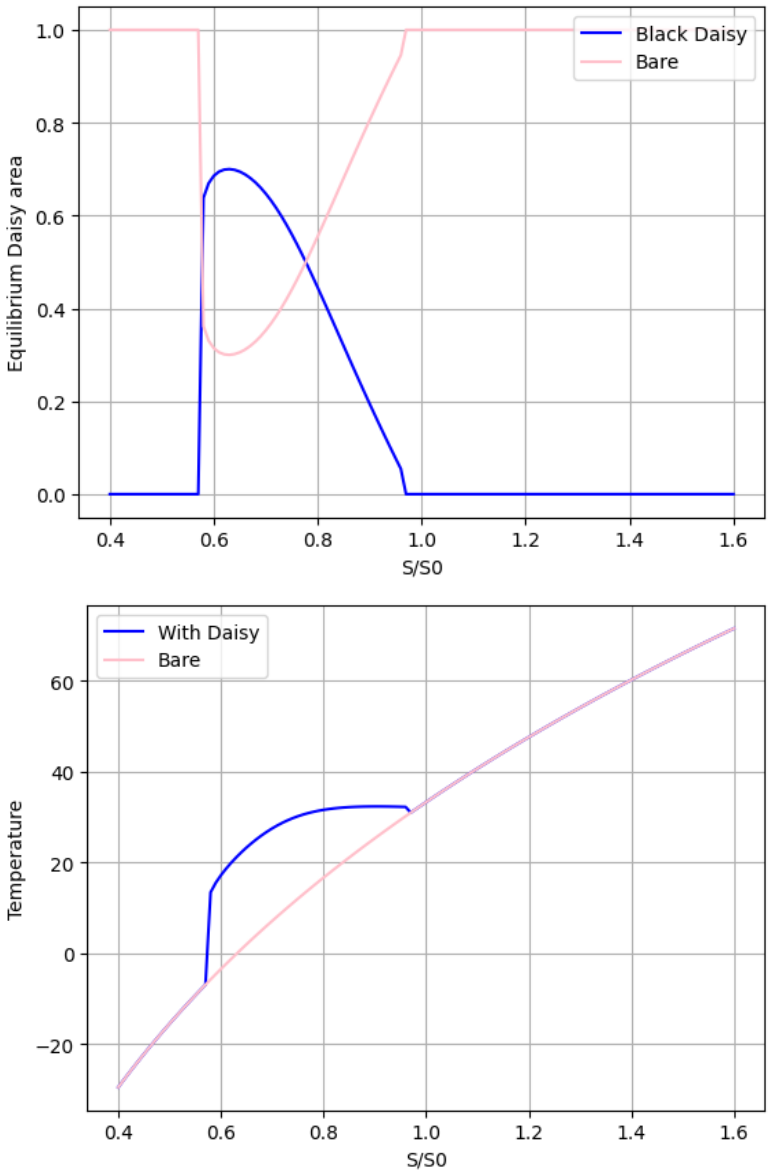

Figure 15: S/S0 versus area and temperature when there is only black daisy.

As being seen, where there is only black daisy species and no competing white daisy species, the black daisy area will increase more rapidly with increasing solar flux. The temperature also rise more sharply. The point where the black daisy begin to grow remains the same, but it die out sooner. The black daisy are completely wiped out before the ratio reaches 1, while when there are both black and white daisy, it is more than 1.2.

2.6 EBM-0D & Feedback (cont.)

The planetary albedo depends on the surface temperature. Let's assume the following relations:
- If $T_{sf} < T_{ice}, \alpha = \alpha_{ice}$
- If $T_{sf} > T_{land}, \alpha = \alpha_{land}$
- If $T_{land} \geq T_{sf} \geq T_{\mathbf{ice}}, \alpha = \alpha_{ice} + (\alpha_{land} - \alpha_{ice}) \cdot \frac{T_{sf} - T_{ice}}{T_{land} - T_{ice}}$

where $\alpha_{ice} = 0{,}6$; $\alpha_{land} = 0{,}32$; $\epsilon = 0{,}62$; T_{ice} = -10°C is the temperature where the Earth becomes a snowball; T_{land} = 10°C is the temperature where the Earth remains in the nowadays state.

2.6.1 Write a program to estimate the equilibrium T_{sf} with the initial temperature varying from 13°C to 3°C with a step of 1°C. Plot a figure showing the dependency of the equilibrium temperature on the initial one. The solar constant remains unchanged at S_0=1368W/m^2.

After defining the constants, I use the given formulas to calculate the albedo and the temperature by:

$$T = \sqrt[4]{\frac{S(1-\alpha)}{4\epsilon\sigma}}$$

in which S is the solar constant, α is the albedo, ϵ is the emissivity, and σ is the Stefan-Boltzmann constant.

The algorithm keeps calculating with fixed albedo until the difference between the current temperature and the next temperature in the loop is less than 0.01, which means it is equilibrium.

Listing 20: Practice 6.1

```
import numpy as np
import matplotlib.pyplot as plt

alpha_ice = 0.6
alpha_land = 0.32
epsilon = 0.62
T_ice = -10
T_land = 10
S0 = 1368
sigma = 5.67E-8

def albedo(T):
    if T < T_ice:
        return alpha_ice
    elif T > T_land:
        return alpha_land
    else:
        return alpha_ice + (alpha_land - alpha_ice) * (T - T_ice)
            / (T_land - T_ice)

def equilibrium_t(T_initial, S=S0):
```

```
    T_current = T_initial + 273.15
    T_next = T_current
    while True:
        alpha = albedo(T_current - 273.15)
        T_next = ((S * (1 - alpha)) / (4 * sigma * epsilon)) **
           0.25
        if abs(T_next - T_current) < 0.01:
            break
        T_current = T_next
    return T_next - 273.15

initial_ts = np.arange(3, 14, 1)
equilibrium_ts = [equilibrium_t(T) for T in initial_ts]

plt.figure()
plt.plot(initial_ts, equilibrium_ts, color = 'deeppink')
plt.xlabel('Initial Temperature')
plt.ylabel('Equilibrium Temperature')
plt.grid(True)
plt.show()
```

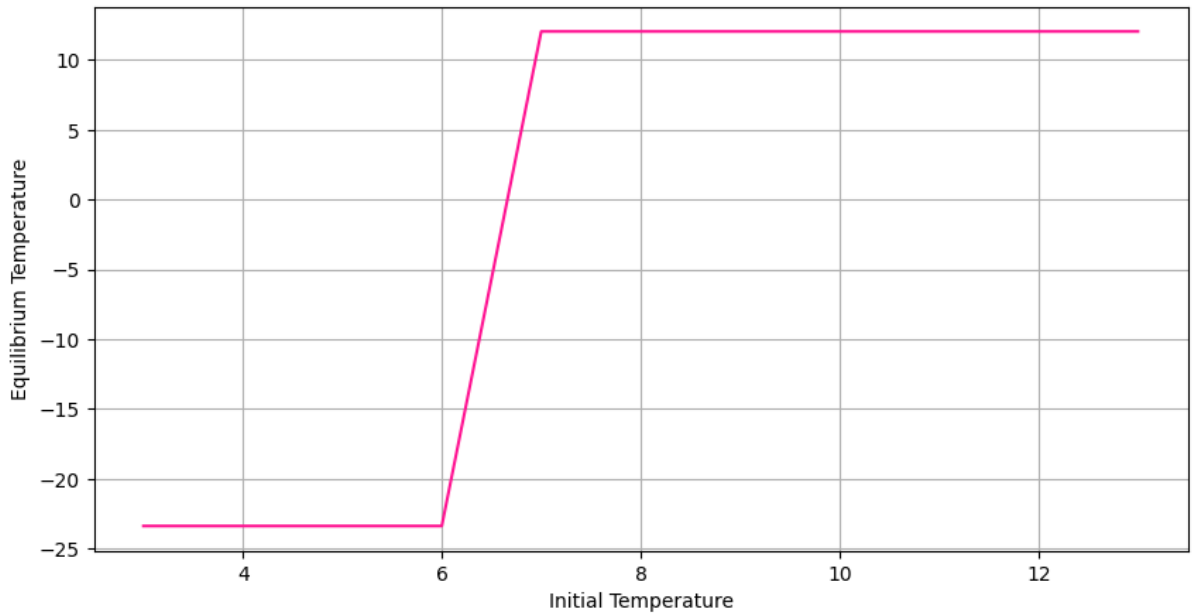

Figure 16: Dependency of the equilibrium temperature on the initial one.

As being seen, when the initial temperature is less than 6, the system reaches a quite low equilibrium temperature at about between $-20\,°C$ and $-25\,°C$, this is because the planet has high albedo ($\alpha\_ice=0.6$) leading to low temperatures. While the higher initial temperature corresponding to lower albedo ($\alpha\_land=0.32$) gives the equilibrium temperature at more than $10\,°C$. In the middle range between two equilibrium points, the data increases linearly.

2.6.2 Write a program to estimate the equilibrium surface temperature with varying solar flux S (S/S_0 ranges from 0.2 to 2 with a step of 0.05). The initial temperature varies from 15°C down to -15°C with a step of 1°C

For this problem, I use the same algorithm and the same formula:

$$T = \sqrt[4]{\frac{S(1-\alpha)}{4\epsilon\sigma}}$$

in which S is the solar constant, α is the albedo, ϵ is the emissivity, and σ is the Stefan-Boltzmann constant to calculate the temperature.

For each solar flux ratio, the algorithm calculates the corresponding equilibrium temperature using the above formula and finally plot them out as separated lines.

Listing 21: Practice 6.2

```
import numpy as np
import matplotlib.pyplot as plt

alpha_ice = 0.6
alpha_land = 0.32
epsilon = 0.62
T_ice = -10
T_land = 10
S0 = 1368
sigma = 5.67E-8

def albedo(T):
    if T < T_ice:
        return alpha_ice
    elif T > T_land:
        return alpha_land
    else:
        return alpha_ice + (alpha_land - alpha_ice) * (T - T_ice)
            / (T_land - T_ice)

def equilibrium_t(T_initial, S=S0):
    T_current = T_initial + 273.15
    T_next = T_current
    while True:
        alpha = albedo(T_current - 273.15)
        T_next = ((S * (1 - alpha)) / (4 * sigma * epsilon)) **
           0.25
        if abs(T_next - T_current) < 0.01:
            break
        T_current = T_next
    return T_next - 273.15

initial_ts = np.arange(-15, 16, 1)
solar_flux_ratios = np.arange(0.2, 2.05, 0.05)

```

```
plt.figure(figsize=(10, 11))
for S_ratio in solar_flux_ratios:
    S = S_ratio * S0
    equilibrium_ts = [equilibrium_t(T, S) for T in initial_ts]
    plt.plot(initial_ts, equilibrium_ts, label=f"S/S0 = {S_ratio
        :.2f}")

plt.xlabel('Initial Temperature')
plt.ylabel('Equilibrium Temperature')
plt.legend()
plt.grid(True)
plt.show()
```

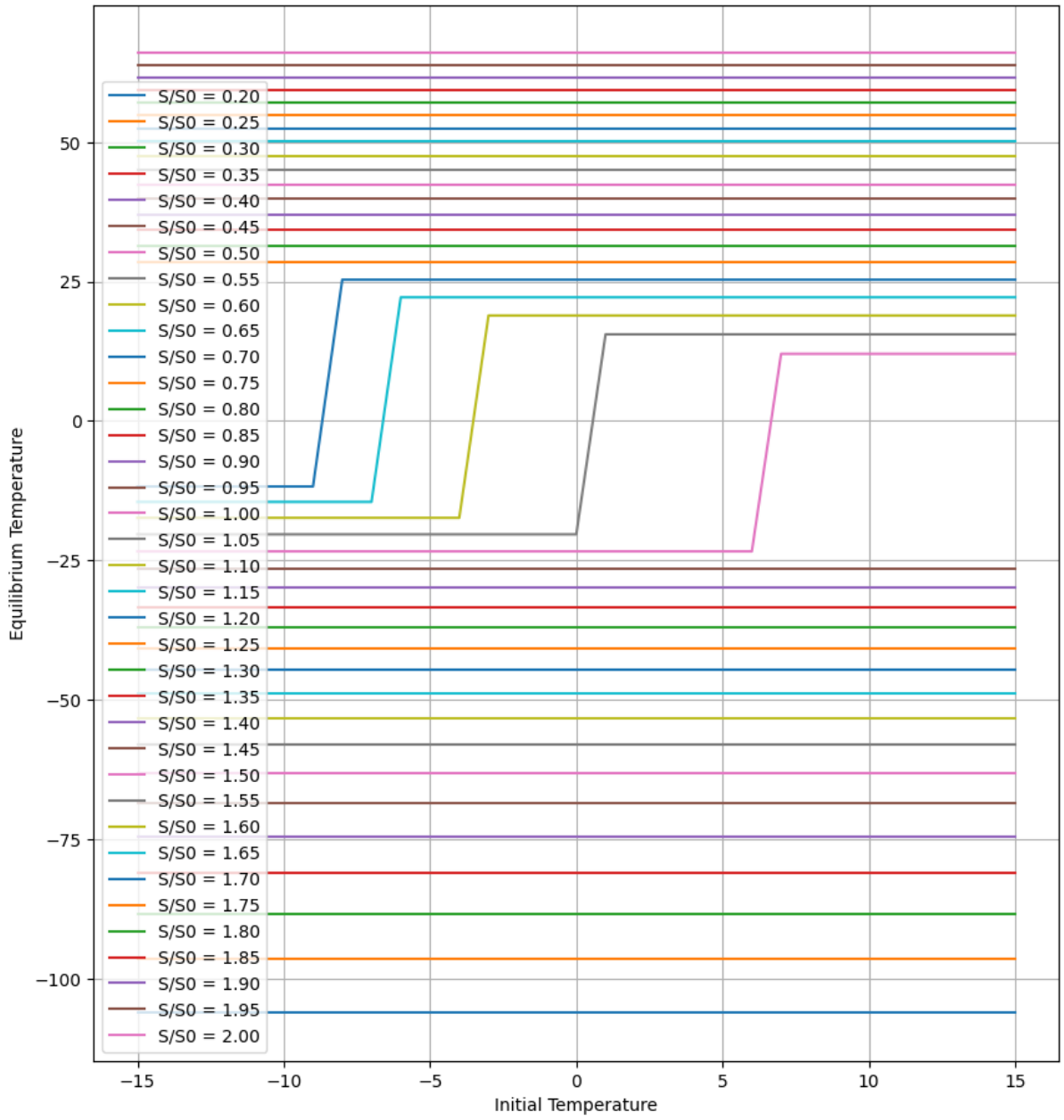

Figure 17: Equilibrium surface temperature with varying solar flux ratio.

As being seen, the equilibrium temperatures shift with fixed solar flux ratios. Higher flux ratios generally result in higher equilibrium temperatures. For many initial temperatures, the equilibrium temperature remains constant and forms horizontal lines. This means that when $\frac{S}{S_0}$ is more than 1.25 or less than 0.95, the temperature will be more than 25 or less than -25 respectively, and the system quickly stabilizes to the equilibrium temperature.

2.7 EBM-1D

2.7.1 Make a complete code of the EBM-1D.

A 1D Energy Balance Model is a simplified climate model used to study and understand the basic mechanisms of the Earth's energy balance. It represents the Earth as a single column or latitude band, averaging the processes and variables across that band.

To solve this problem, I pretend each band of the 1D is a 0D and use the principle of energy balance, which states that the temperature of a given location is determined by the balance between the incoming solar radiation, the albedo, and the heat exchange with the surrounding environment to calculate all the variable needed for that band and repeat until the system is stable.

I define lists of constants, in which so0 is the solar constant (W/m), albi is the albedo of ice, tc1 is the lower threshold temperature for ice (°C), tc2 is the upper threshold temperature for ice (°C), ny is the number of latitudinal bands, it is the number of iterations, and alb0 and so_frac are arrays for the initial albedo and fractional solar radiation for each latitudinal band, respectively.

For each latitude band j, I use the following formulas to calculate its features.

The area of each band is calculated by:

$$\text{area}[j] = \sin\left(\phi[j] + \frac{\Delta\phi}{2}\right) - \sin\left(\phi[j] - \frac{\Delta\phi}{2}\right)$$

where area[j] is the area of the j-th latitude band and $\Delta\phi$ is the angular width of each band.

Albedo is calculated by:

$$\alpha_j = ice \cdot \alpha_i + (1 - ice)\alpha_0[j]$$

where α_i is the albedo of ice, α_0[j] is the initial albedo of the j-th band, and ice is a factor that ranges from 0 to 1, representing the fraction of the band covered by ice, based on temperature.

Incoming solar radiation is calculated by:

$$S = S_0 \cdot so\_frac[j]$$

where S_0 is the solar constant, $so\_frac[j]$ is the fraction of the solar constant received by the j-th latitude band.

Temperature is calculated by:

$$T_j = \frac{S_j(1 - \alpha_j) + kT_m - a}{b + k}$$

where S_j is the solar radiation received by the j-th band, k is a constant (W/m²/°C), and T_m is the mean temperature across all bands.

The fraction of ice cover is calculated by:

$$\text{ice} = \begin{cases} 1 & \text{if } T_j < tc1 \\ 0 & \text{if } T_j > tc2 \\ \frac{tc2 - tj[j]}{tc2 - tc1} & \text{otherwise} \end{cases}$$

where T_j is the temperature of the j-th latitude band tc1 is the lower threshold temperature for ice cover and tc2 is the upper threshold temperature for ice cover.

The loop continues until the different between the temperature between each iterations is smaller than ϵ, which means the system is quilibrium.

Listing 22: Practice 7.1

```
import numpy as np
import matplotlib.pyplot as plt

a = 204
b = 2.17
k = 3.81
so0 = 1369
albi = 0.6
tc1 = -10
tc2 = 0
ny = 9
it = 25

alb0 = [0.23, 0.24, 0.25, 0.29, 0.35, 0.4, 0.46, 0.5, 0.5]
so = np.zeros(ny)
so_frac = [0.30475, 0.29725, 0.28, 0.25525, 0.223, 0.1925, 0.156,
    0.13275, 0.125]

phi = np.zeros(ny)
phid = np.zeros(ny)
area = np.zeros(ny)
alb = np.zeros(ny)
tj = np.zeros(ny)
tj_ini = np.zeros(ny)
tj_pre = np.zeros(ny)

dp = np.pi / (2 * ny)
for j in range(ny):
    phi[j] = (0.5 + j - 1) * dp
    phid[j] = phi[j] * 180 / np.pi
    area[j] = np.sin(phi[j] + dp / 2.) - np.sin(phi[j] - dp / 2.)
    so[j] = so0 * so_frac[j]

albm = 0.0
for j in range(ny):
    albm += alb0[j] * area[j]

tm = ((so0 / 4) * (1 - albm) - a) / b
```

```
38
39 for j in range(ny):
40     tj_ini[j] = tm
41     tj_pre[j] = tm
42     tj[j] = tm
43
44 epsilon = 1E-6
45 diff = 1
46
47 i = 0
48 while abs(diff) > epsilon:
49     i += 1
50     albm = 0
51     tm = 0
52     diff = 0
53     for j in range(ny):
54         if tj[j] < tc1:
55             ice = 1
56         elif tj[j] > tc2:
57             ice = 0
58         else:
59             ice = (tc2 - tj[j]) / (tc2 - tc1)
60         alb[j] = ice * albi + (1 - ice) * alb0[j]
61         tj[j] = (so[j] * (1 - alb[j]) + k * tm - a) / (b + k)
62         albm += alb[j] * area[j]
63         tm += tj[j] * area[j]
64         diff += tj[j] - tj_pre[j]
65         tj_pre[j] = tj[j]
66
67 print("Number of iterations to reach equilibrium:", i)
68 for j in range(ny):
69     print('Band', j, np.round(tj_ini[j], 4), np.round(tj[j], 4))
70
71 plt.figure()
72 plt.plot(range(ny), tj, color = 'deeppink')
73 plt.xlabel('Latitudinal Bands')
74 plt.ylabel('Temperature')
75 plt.legend()
76 plt.grid(True)
77 plt.show()
```

```
Number of iterations to reach equilibrium: 5
Band 0 4.694 19.6064
Band 1 4.694 19.773
Band 2 4.694 18.3183
Band 3 4.694 13.6965
Band 4 4.694 6.7701
Band 5 4.694 0.644
Band 6 4.694 -11.4616
Band 7 4.694 -14.3208
Band 8 4.694 -15.7026
```

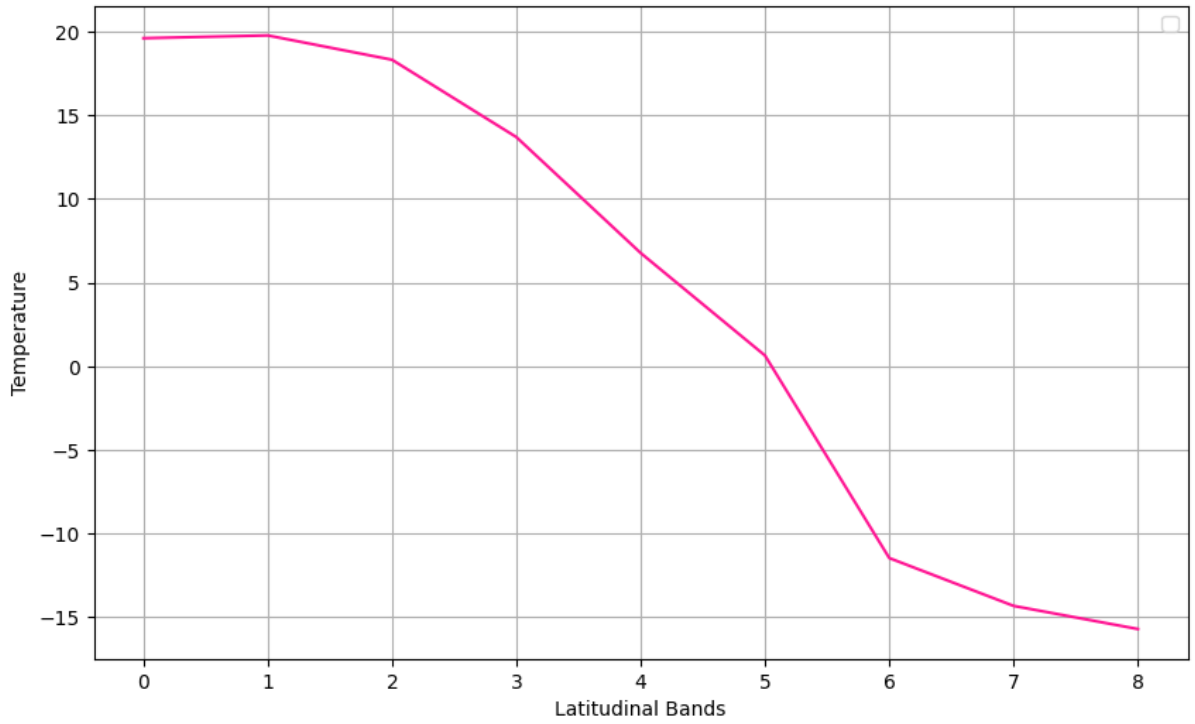

Figure 18: Temperature profile across latitude bands.

As being seen, it requires 5 iterations to reach the equilibrium state. The initial temperature for all bands is 4.694°C. The final temperatures vary from -15.7026°C in the northernmost band to 19.6064°C in the southernmost band. From band 0 to 4, the temperature is quite warm, while the colder temperatures are in the higher latitudes (bands 5-8). The temperature in bands 6 to 8 is below the freezing point of water, suggesting the presence of ice in these regions.

2.7.2 Estimate the value of solar flux so that the Earth will be entirely covered by ice.

The Earth entirely covered by ice means that all the zones have the equilibrium temperature lower than 0 °C. To meet this condition, I use the same algorithm as the previous problem but lets the loop calculates for each latitude bands until all their temperatures are less than 0 °C.

Listing 23: Practice 7.2

```
import numpy as np
import matplotlib.pyplot as plt

a = 204
b = 2.17
k = 3.81
albi = 0.6
tc1 = -10
tc2 = 0
ny = 9
```

```
11
12 alb0 = [0.23, 0.24, 0.25, 0.29, 0.35, 0.40, 0.46, 0.5, 0.5]
13 so_frac = [0.30475, 0.29725, 0.28, 0.25525, 0.223, 0.1925, 0.156,
       0.13275, 0.125]
14
15 phi = np.zeros(ny)
16 area = np.zeros(ny)
17 dp = np.pi / (2 * ny)
18 for j in range(ny):
19     phi[j] = (0.5 + j - 1) * dp
20     area[j] = np.sin(phi[j] + dp / 2.) - np.sin(phi[j] - dp / 2.)
21
22 def calculate_temperatures(so0):
23     so = so0 * np.array(so_frac)
24
25     albm = sum(alb0[j] * area[j] for j in range(ny))
26     tm = ((so0 / 4.0) * (1 - albm) - a) / b
27
28     tj = np.full(ny, tm)
29     tj_pre = np.full(ny, tm)
30
31     epsilon = 1E-6
32     diff = 1
33
34     while abs(diff) > epsilon:
35         albm = 0
36         tm = 0
37         diff = 0
38         for j in range(ny):
39             if tj[j] < tc1:
40                 ice = 1
41             elif tj[j] > tc2:
42                 ice = 0
43             else:
44                 ice = (tc2 - tj[j]) / (tc2 - tc1)
45             alb = ice * albi + (1 - ice) * alb0[j]
46             tj[j] = (so[j] * (1 - alb) + k * tm - a) / (b + k)
47             albm += alb * area[j]
48             tm += tj[j] * area[j]
49             diff += tj[j] - tj_pre[j]
50             tj_pre[j] = tj[j]
51
52     return tj
53
54 so_range = np.linspace(1000, 1500, 10)
55
56 temperature_matrix = np.zeros((len(so_range), ny))
57
58 for i, so0 in enumerate(so_range):
59     temperature_matrix[i, :] = calculate_temperatures(so0)
60
```

```
plt.figure(figsize=(10, 8))
plt.contourf(range(ny), so_range, temperature_matrix, cmap = 'jet
    ', levels=100)
plt.colorbar(label='Temperature')
plt.xlabel('Latitude Bands')
plt.ylabel('Solar Flux')
y_ticks = np.arange(1000, 1500, 20)
plt.yticks(y_ticks)

plt.show()
```

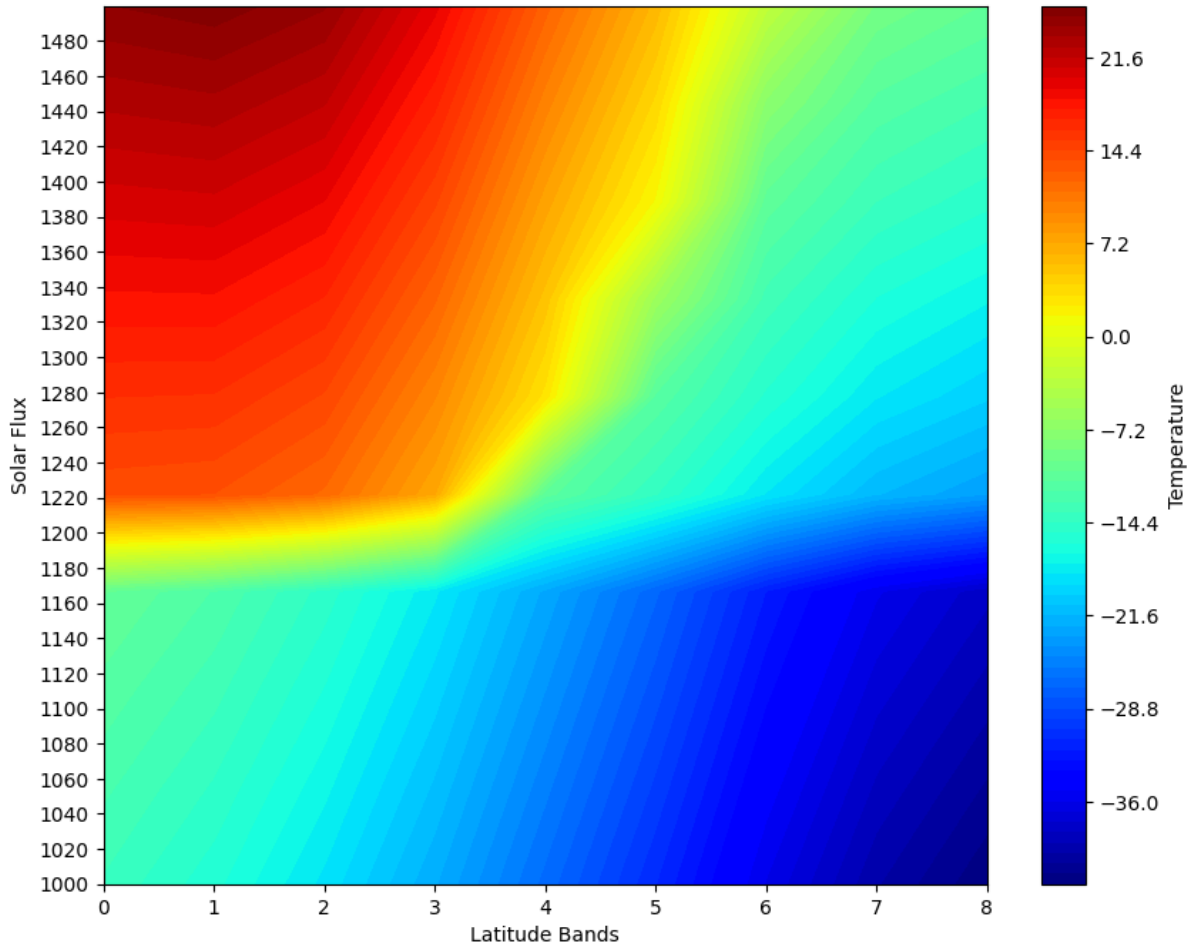

Figure 19: Temperature profile across latitude bands with varying solar flux.

As being seen in the graph, the red regions indicate higher temperatures, with the highest temperature shown being around 21.6°C. The blue regions indicate lower temperatures, with the lowest temperature shown being around -36°C. There is a gradient transition from red to blue when it moves from the top left, where the solar flux is high and the latitude band is low, to the bottom right, where the solar flux is low and the latitude band is high. The Earth is entirely covered in ice when the temperature around all latitude bands are under 0 °C when the solar flux is around 1180 W/m^2 or lower.

2.7.3 K could vary. For example Budyko (1969) let k_t =3.81. Warren and Schneider (1979) let k_t=3.74. Investigate the sensitivity of the model with K.

Knowing that k is the transport coefficient which implies the linear relationship of the energy transport from each band with the difference in the zonal temperature and global temperature. For this problem, I used the same mathematical formulas as previous, just change the k values, and plot the graph in which blue represents 3.81 and pink represents 3.74 to see what happen.

Listing 24: Practice 7.3

```
def EBM_1D(a,b,k):
    so0=1368
    albi=0.6
    tc1=-10
    tc2=0
    ny=9
    it=25

    alb0=[0.23, 0.24, 0.25, 0.29, 0.35, 0.40, 0.46, 0.5, 0.5]
    so=np.zeros(ny)
    so_frac
       =[0.30475,0.29725,0.28,0.25525,0.223,0.1925,0.156,0.13275,0.125]


    phi=np.zeros(ny)
    phid=np.zeros(ny)
    area=np.zeros(ny)
    alb=np.zeros(ny)
    tj=np.zeros(ny)
    tj_ini=np.zeros(ny)
    tj_pre=np.zeros(ny)

    dp=np.pi/(2*ny)
    for j in range(0, ny):
        phi[j]=(0.5+j-1)*dp
        phid[j]=phi[j]*180/np.pi
        area[j]=np.sin(phi[j]+dp/2.)-np.sin(phi[j]-dp/2.)
        so[j]=so0*so_frac[j]

    albm=0.0
    for j in range(0,ny):
        albm=albm+alb0[j]*area[j]

    tm=((so0/4)*(1-albm)-a)/b

    for j in range(0,ny):
        tj_ini[j]=tm
        tj_pre[j]=tm
        tj[j]=tm

```

```
    epsilon=1.0*(10**(-6))
    diff=1.

    i=0
    while abs(diff)>epsilon:
        i=i+1
        albm=0.
        tm=0.
        diff=0.
        for j in range(0,ny):
            if tj[j]<tc1:
                ice=1
            elif tj[j]>tc2:
                ice=0
            else:
                ice=(tc2-tj[j])/(tc2-tc1)
            alb[j]=ice*albi+(1-ice)*alb0[j]
            tj[j]=(so[j]*(1-alb[j])+k*tm-a)/(b+k)
            albm=albm+alb[j]*area[j]
            tm=tm+tj[j]*area[j]
            diff=diff+tj[j]-tj_pre[j]
            tj_pre[j]=tj[j]
    return(tj,so)

a=204
b=2.17
k_Budyko=3.81
k_Warren=3.74

Budyko=EBM_1D(a,b,k_Budyko)
Warren_Schneider=EBM_1D(a,b,k_Warren)

plt.plot(Budyko[1],Budyko[0],'k', color = 'blue', label='3.81')
plt.plot(Warren_Schneider[1],Warren_Schneider[0],'y', color = '
   pink', label='3.74')

plt.legend()
plt.xlabel("Solar flux")
plt.ylabel("Temperature")
plt.grid("True")
```

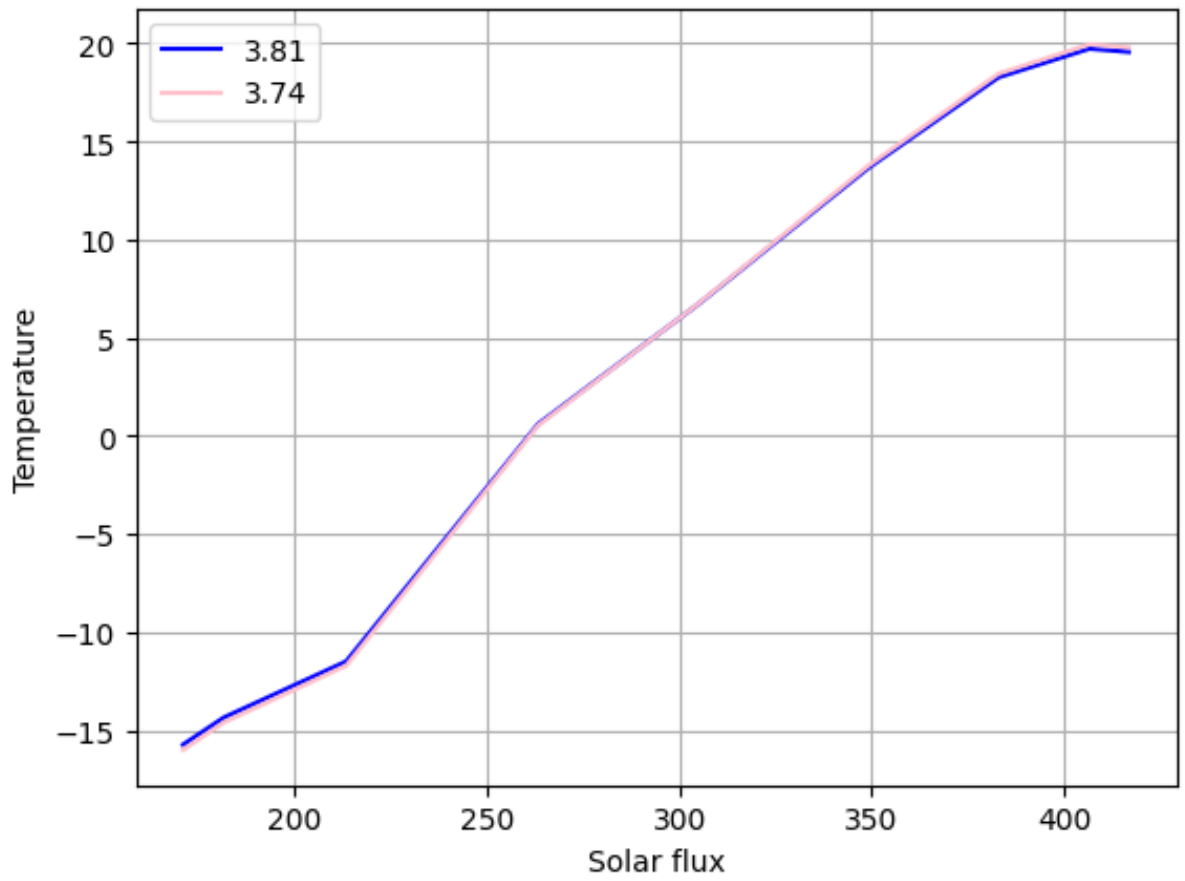

Figure 20: Sensitivity of model with k = 3.81 and 3.74.

Knowing that temperature is calculated by:

$$T_j = \frac{S_j(1-\alpha_j) + kT_m - a}{b+k}$$

where S_j is the solar radiation received by the j-th band, k is a constant (W/m²/°C), and T_m is the mean temperature across all bands. It can be seen that k appears in both the numerator and denominator in the term kT_m and $b+k$, respectively. kT_m is covariant with T, while $b+k$ not. So mathematically, T should be really sensitive while k values change.

However, as being seen in the graph, there is not much difference. The differences can still be seen at the bottom left and top right of the graph, but in the middle, the two lines completely overlap. This is because the two k values are too close to each other.

To explore more and visualize better, I also plot a few other values of k, range from 1 to 5.

Listing 25: Also practice 7.3

```
k=np.arange(1,6,1)

for i in range(len(k)):
```

```
    temp_fin,so_flux=EBM_1D(a,b,k[i])
    plt.plot(so_flux,temp_fin,label= 'k='+str(k[i]))

plt.legend()
plt.xlabel("Solar flux")
plt.ylabel("Temperature")
plt.grid("True")
```

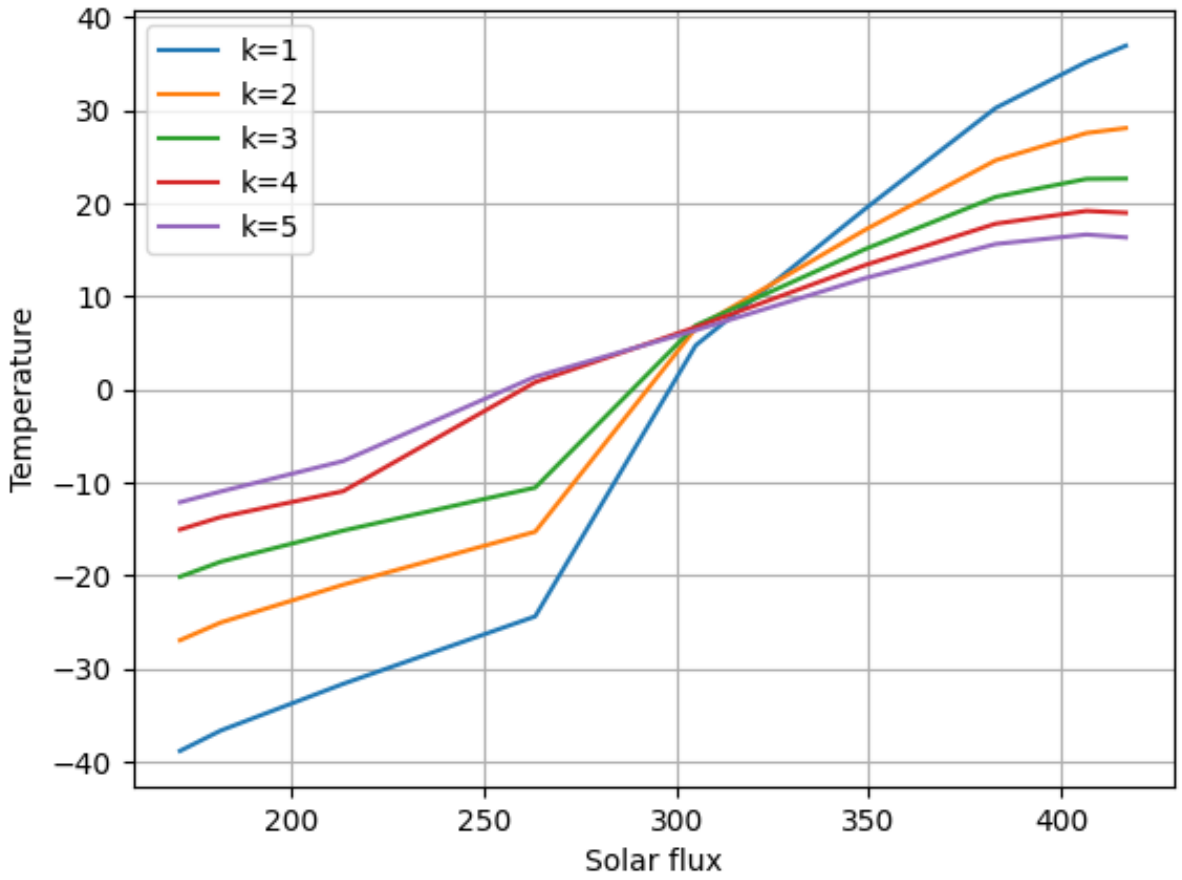

Figure 21: Sensitivity of model with variation of k values.

As being seen, the slope of the lines change significantly. As the value of the k increases, the slope of the temperature variation with solar flux decreases. This is because a higher k value indicates a stronger feedback mechanism in the model that maintain the temperature balance, which results in smaller variations in temperature between different latitude bands. Before the intersection point, the lower the k value is, the lower the temperature is.

2.7.4 A and B can vary. For example, Bydyko (1969) let A = 202 Wm and B = 1.45 $Wm^{-2}°C$. Cess (1976) let A = 212 Wm and B = 1.6 $Wm^{-2}°C$. Investigate the sensitivity of the model with B. What are the physical meanings behind?

To solve this problem, I still use the same algorithm as the above one, just with different A and B values. To better visualize the sensitivity of the model with the two given B

values, I define A value as an average of the two given A values, which is 207 Wm. Then I plot the graph, in which the color blue is the Budyko B value, and the color pink is the Cess B value.

Listing 26: Practice 7.4

```
def EBM_1D(a,b,k):
    so0=1368
    albi=0.6
    tc1=-10
    tc2=0
    ny=9
    it=25

    alb0=[0.23, 0.24, 0.25, 0.29, 0.35, 0.40, 0.46, 0.5, 0.5]
    so=np.zeros(ny)
    so_frac
       =[0.30475,0.29725,0.28,0.25525,0.223,0.1925,0.156,0.13275,0.125]


    phi=np.zeros(ny)
    phid=np.zeros(ny)
    area=np.zeros(ny)
    alb=np.zeros(ny)
    tj=np.zeros(ny)
    tj_ini=np.zeros(ny)
    tj_pre=np.zeros(ny)

    dp=np.pi/(2*ny)
    for j in range(0, ny):
        phi[j]=(0.5+j-1)*dp
        phid[j]=phi[j]*180/np.pi
        area[j]=np.sin(phi[j]+dp/2.)-np.sin(phi[j]-dp/2.)
        so[j]=so0*so_frac[j]

    albm=0.0
    for j in range(0,ny):
        albm=albm+alb0[j]*area[j]

    tm=((so0/4)*(1-albm)-a)/b

    for j in range(0,ny):
        tj_ini[j]=tm
        tj_pre[j]=tm
        tj[j]=tm

    epsilon=1.0*(10**(-6))
    diff=1.

    i=0
    while abs(diff)>epsilon:
```

```
        i=i+1
        albm=0.
        tm=0.
        diff=0.
        for j in range(0,ny):
            if tj[j]<tc1:
                ice=1
            elif tj[j]>tc2:
                ice=0
            else:
                ice=(tc2-tj[j])/(tc2-tc1)
            alb[j]=ice*albi+(1-ice)*alb0[j]
            tj[j]=(so[j]*(1-alb[j])+k*tm-a)/(b+k)
            albm=albm+alb[j]*area[j]
            tm=tm+tj[j]*area[j]
            diff=diff+tj[j]-tj_pre[j]
            tj_pre[j]=tj[j]
    return(tj,so)

k = 3.81
A = 207
B_Budyko = 1.45
B_Cess = 1.6

Budyko2 = EBM_1D(A, B_Budyko, k)
Cess = EBM_1D(A, B_Cess, k)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(Budyko2[1], Budyko2[0], 'k', color='blue', label='Budyko'
    )
ax.plot(Cess[1], Cess[0], 'y', color='pink', label='Cess')

ax.legend()
ax.set_xlabel("Solar flux")
ax.set_ylabel("Temperature")
ax.grid("True")
```

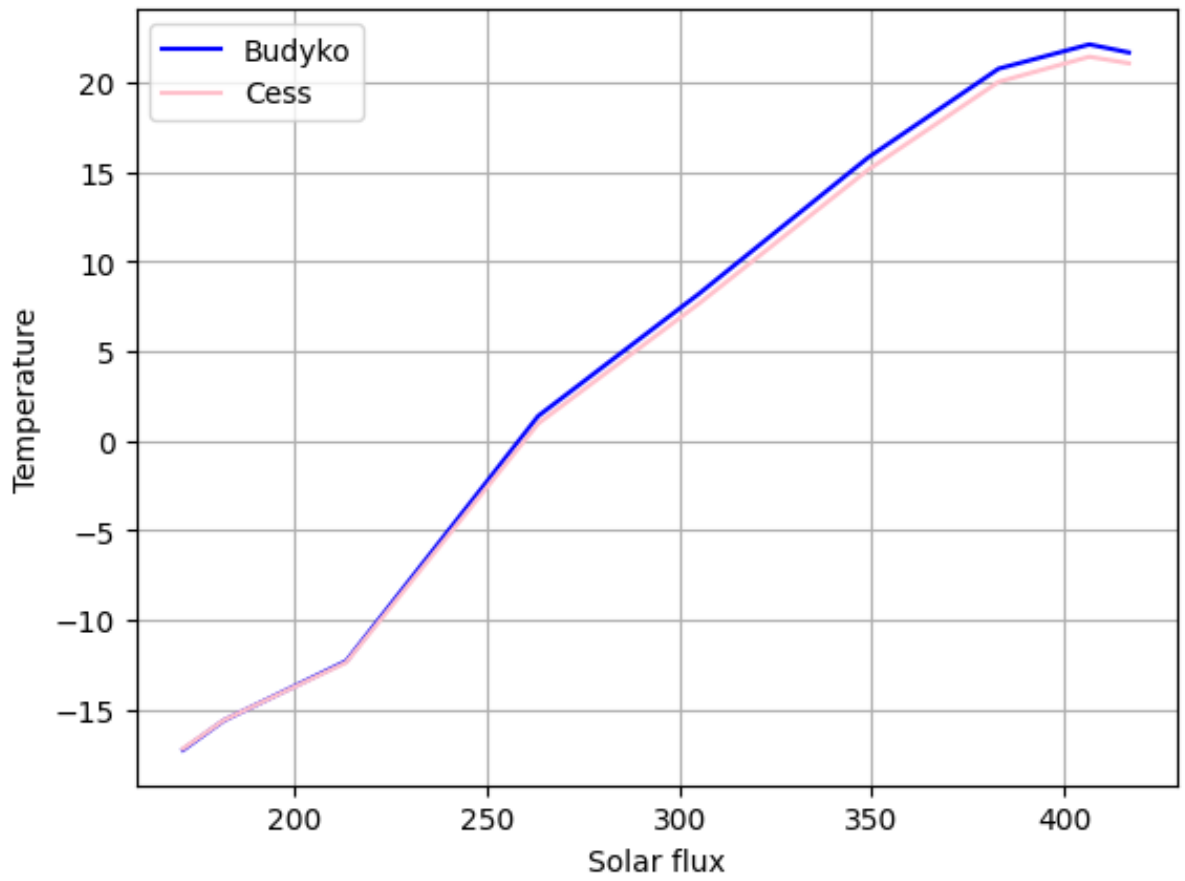

Figure 22: Sensitivity of model when B = 1.45 and 1.6.

Just like the previous problem, the two B values are too close so it is not much a significant difference to be seen. The two lines mostly overlap when the solar flux is low. To better explore the sensitivity of the model with B values, I also plot a few other values of B, ranges from 1 to 5.

Listing 27: Also practice 7.4

```
k = 3.81
A = 207
B = np.arange(1, 6, 1)

fig, ax = plt.subplots()

for i in range(len(B)):
    temp_fin,so_flux=EBM_1D(A,B[i],k)
    plt.plot(so_flux,temp_fin,label= 'B=' + str(np.round(B[i],1))
       )

ax.set_xlabel('Solar Flux')
ax.set_ylabel('Temperature')
ax.legend()
ax.grid("True")

plt.show()
```

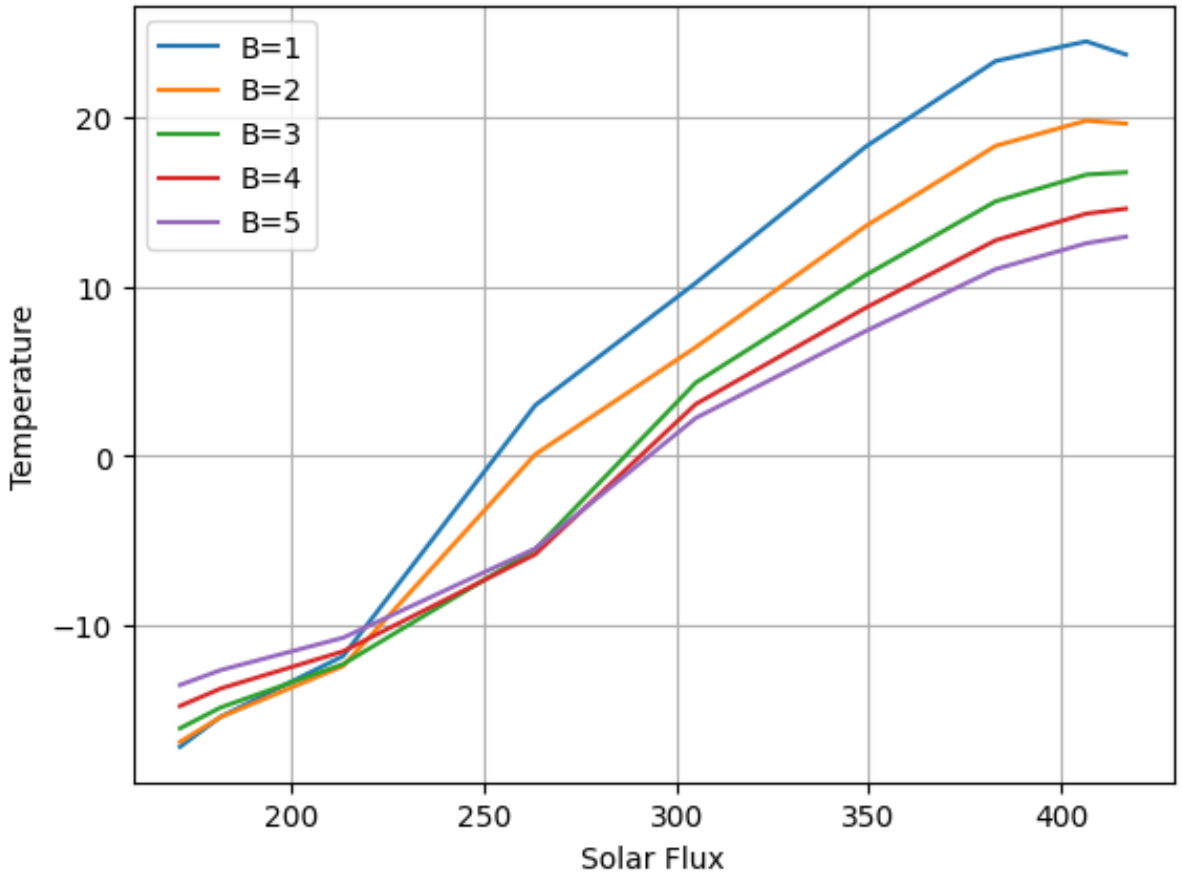

Figure 23: Sensitivity of model with variation of B values.

As being seen, the higher the B is, the steeper the slope becomes. For lower values of B, when solar flux increases, the temperature goes up more rapidly. For higher values of B, the temperature increases more gradually. In conclusion, low B value indicates a climate system where temperature responds strongly to changes in solar forcing, while high B value indicates a more stable climate system with less temperature variation for given changes in solar forcing.

2.8 Plot sea level rise figure

2.8.1 Download the altimetry data from here.

The file is a dataset of sea level rise collected by a series of satellite missions, which are TOPEX/Poseidon, Jason-1, Jason-2, Jason-3 and Sentinel-6MF. Due to the combination of data from multiple satellites, there are some missing values in the dataset.

This is what the file looks like when opened on my laptop.

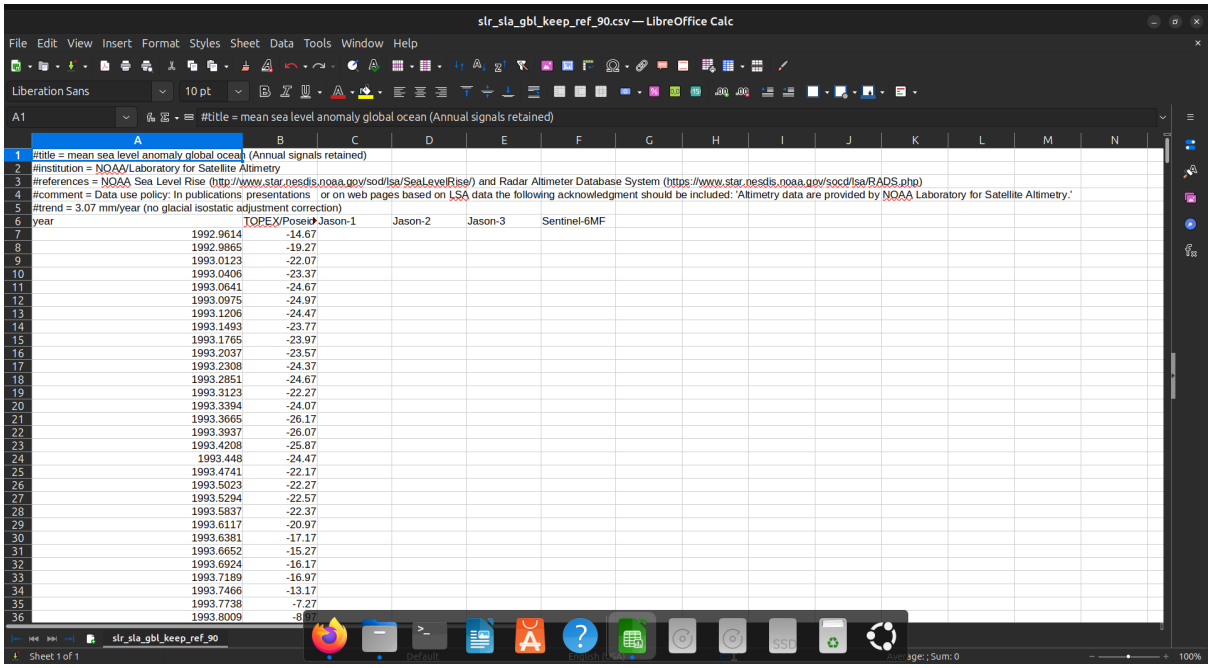

Figure 24: Dataset for practice 8.

First of all, this is my code to read the dataset and the outcome.

Listing 28: Code to read the dataset for practice 8

```
import pandas as pd
df = pd.read_csv ("slr_sla_gbl_keep_ref_90.csv", comment = "#",
   delimiter = ",", index_col =0)
df
```

Table 2: Dataset read for practice 8

Year	TOPEX/ Poseidon	Jason-1	Jason-2	Jason-3	Sentinel-6MF
1992.9614	-14.67	NaN	NaN	NaN	NaN
1992.9865	-19.27	NaN	NaN	NaN	NaN
1993.0123	-22.07	NaN	NaN	NaN	NaN
1993.0406	-23.37	NaN	NaN	NaN	NaN
1993.0641	-24.67	NaN	NaN	NaN	NaN

⋮	⋮	⋮	⋮	⋮	⋮
2024.2394	NaN	NaN	NaN	77.87	NaN
2024.2611	NaN	NaN	NaN	NaN	71.84
2024.2665	NaN	NaN	NaN	73.77	NaN
2024.2831	NaN	NaN	NaN	NaN	71.64
2024.2857	NaN	NaN	NaN	76.97	NaN

2.8.2 Plot a figure from the data above, to represent the change of sea level since 1992.

As being seen, there are NaN value in the data set, they are the missing values. To remove them, I use built-in function dropna(). After that I plot the data, and set x-axis label is set to Year and y-axis label is set to Sea level rise (mm).

Listing 29: Practice 8.2

```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5), dpi=150)

for col in df.columns:
    plot_data = df[col].dropna()
    plt.plot(plot_data.index.values, plot_data.values, label=col)

plt.xlabel('Year')
plt.ylabel('Sea level rise (mm)')
plt.grid(True)
plt.legend()
plt.show()
```

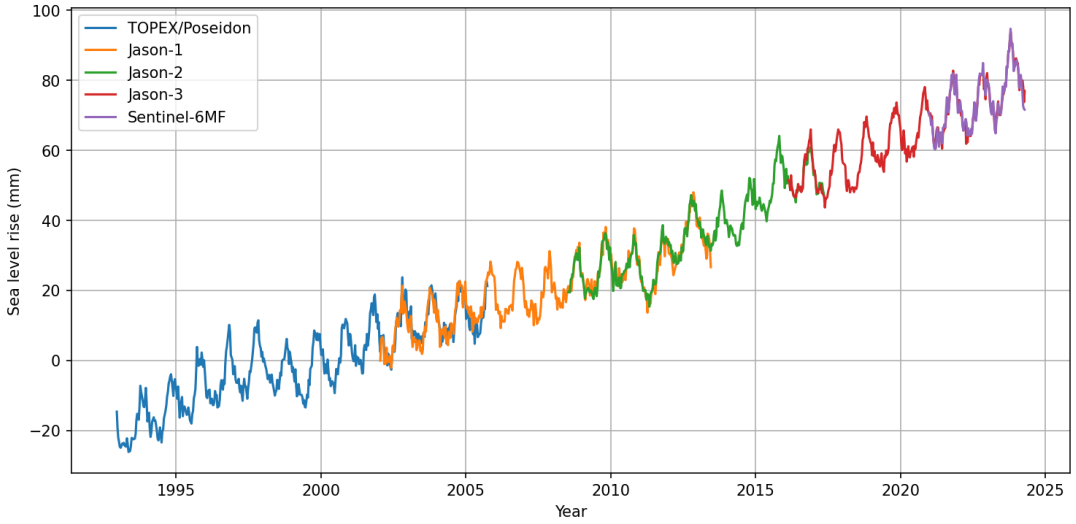

Figure 25: Global mean sea level.

As being seen, the global mean sea level has increased about 100mm - 120mm from 1992 to 2024 and is still going up. There are changes in the weather throughout the year which create an annual pattern of sea level. In winter, the northern hemisphere experiences more rain and snow, causing water to build up on land. It takes time for this water to flow back into the oceans. As a result, the global mean sea level follows a pattern on the plot each year and usually changes by about 10mm. The size of this pattern can change a bit during El Niño and La Niña year.

2.8.3 Search and describe your understanding of altimetry missions in your report.

TOPEX/Poseidon: TOPEX/Poseidon was a project of NASA and CNES. It was launched on August 10, 1992. Its objective is to revolutionize the field of satellite altimetry, which is the measurement of the height of the ocean's surface from space. TOPEX/Poseidon provided highly accurate data on sea surface height (SSH) and greatly improved the understanding of the ocean currents and their impact on climate. The mission's goals were to measure changes in global sea levels, track ocean currents and eddies, study the El Niño and La Niña weather patterns, and improve weather forecasting and climate models. TOPEX/Poseidon's data confirmed the ongoing rise in global sea levels and helped improve the knowledge of ocean circulation patterns. It also contributed to the development of operational oceanography.

Jason-1: Jason-1 was a follow-up mission to TOPEX/Poseidon, launched on December 7, 2001. It continued the important work of its predecessor by providing ongoing measurements of sea surface height. The goals of Jason-1 were to maintain the long-term climate data record started by TOPEX/Poseidon, increase the accuracy and coverage of SSH measurements. Jason-1 extended the time series of global sea level measurements, improved global climate models and weather predictions, and provided valuable data for studying large scale ocean features, such as eddies and boundary currents.

Jason-2: Jason-2 mission, also known as the Ocean Surface Topography Mission, was launched on June 20, 2008. It was a mission of NASA, CNES, NOAA, and EUMETSAT. The mission aimed to continue and enhance the ocean altimetry data record. Its objectives were to maintain the continuity of sea surface height measurements, improve data accuracy and spatial resolution, and support operational oceanography and climate monitoring. Jason-2 contributed to human's understanding of sea level rise and provided data for operational applications such as marine forecasting and climate research.

Jason-3: Jason-3 was a follow-up mission to Jason-2 and was launched on January 17, 2016. It continues the ocean altimetry data record and the climate data. The mission's goals are to extend the time series of global sea level measurements, monitor ocean circulation and its impact on climate, and support operational oceanography and

meteorology. Jason-3 has improved the accuracy of sea level rise measurements, enhanced human's understanding of ocean circulation patterns and their variability, and provided critical data for weather prediction and climate models.

Sentinel-6MF: Sentinel-6 Michael Freilich was launched on November 21, 2020. It is a project of NASA, ESA, EUMETSAT, and NOAA, and is named after Dr. Michael Freilich, a former director of NASA's Earth Science Division. The mission's goals are to continue the long-term record of sea surface height measurements, provide high-precision data for climate monitoring and research. Among all of the missions above, Sentinel-6 Michael Freilich provides the most accurate data on global sea level rise and ocean circulation. It has helped improved weather forecasting and climate modeling, and contribute to the study of climate change and its impacts on global sea levels.

2.9 Sea surface temperature & ENSO

2.9.1 Download sst.mnmean.nc from here.

This is what the file looks like when opened on my laptop.

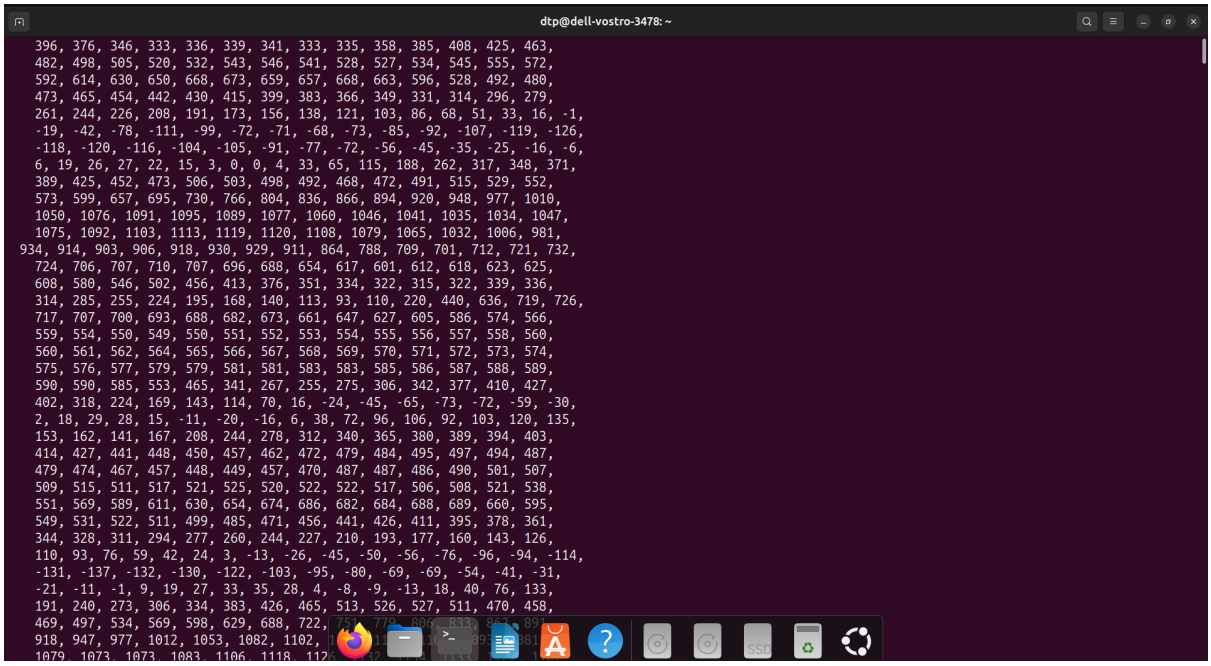

Figure 26: Dataset for practice 9.

And this is when opened with Python.

Listing 30: Code to read the dataset for practice 9.

```
import xarray as xr

data = xr.open_dataset('sst.mnmean.nc')
print(data)
```

```
<xarray.Dataset>
Dimensions:    (lat: 180, lon: 360, time: 494, nbnds: 2)
Coordinates:
  * lat        (lat) float32 89.5 88.5 87.5 86.5 ... -86.5 -87.5 -88.5 -89.5
  * lon        (lon) float32 0.5 1.5 2.5 3.5 4.5 ... 356.5 357.5 358.5 359.5
  * time       (time) datetime64[ns] 1981-12-01 1982-01-01 ... 2023-01-01
Dimensions without coordinates: nbnds
Data variables:
    sst        (time, lat, lon) float32 ...
    time_bnds  (time, nbnds) datetime64[ns] ...
Attributes:
    title:          NOAA Optimum Interpolation (OI) SST V2
    Conventions:    CF-1.0
    history:        Wed Apr  6 13:47:45 2005: ncks -d time,0,278 SAVEs/sst.mn...
    comments:       Data described in  Reynolds, R.W., N.A. Rayner, T.M.\nSmi...
    platform:       Model
```

```
    source:          NCEP Climate Modeling Branch
    institution:     National Centers for Environmental Prediction
    References:      https://www.psl.noaa.gov/data/gridded/data.noaa.oisst.v2....
    dataset_title:   NOAA Optimum Interpolation (OI) SST V2
    source_url:      http://www.emc.ncep.noaa.gov/research/cmb/sst_analysis/
```

As being seen, the dataset has four dimensions: latitude (lat), longitude (lon), time (time) and nbnds (number of bounds). It has coordinate variables for lat, lon, and time. It has two data variables sst (sea surface temperature) and time_bnds (time bounds). It has several attributes, including the title, conventions, history, comments, platform, source, institution, references, dataset title, and source URL.

2.9.2 Plot the average SST over the oceans.

First I extract the needed informations from the file, which are sst and time. I calculate the average SST over the oceans by taking the mean of the SST data along the latitude and longitude. I use the PlateCarree function and ax.add_feature(cfeature.LAND) function to plot the global map as shown below.

Listing 31: Practice 9.2

```
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

data = xr.open_dataset('sst.mnmean.nc')

sst = data['sst']
time = data['time']

sst_mean = sst.mean(dim=['lat', 'lon'])

sst_mean_global = sst.mean(dim='time')
fig, ax = plt.subplots(figsize = (13, 6),subplot_kw={'projection'
    : ccrs.PlateCarree()})

im = ax.contourf(sst['lon'], sst['lat'], sst_mean_global, cmap='
    jet', levels=100, transform=ccrs.PlateCarree())

ax.add_feature(cfeature.LAND, zorder=1)
plt.colorbar(im, ax=ax, label='Sea surface temperature')
plt.show()
```

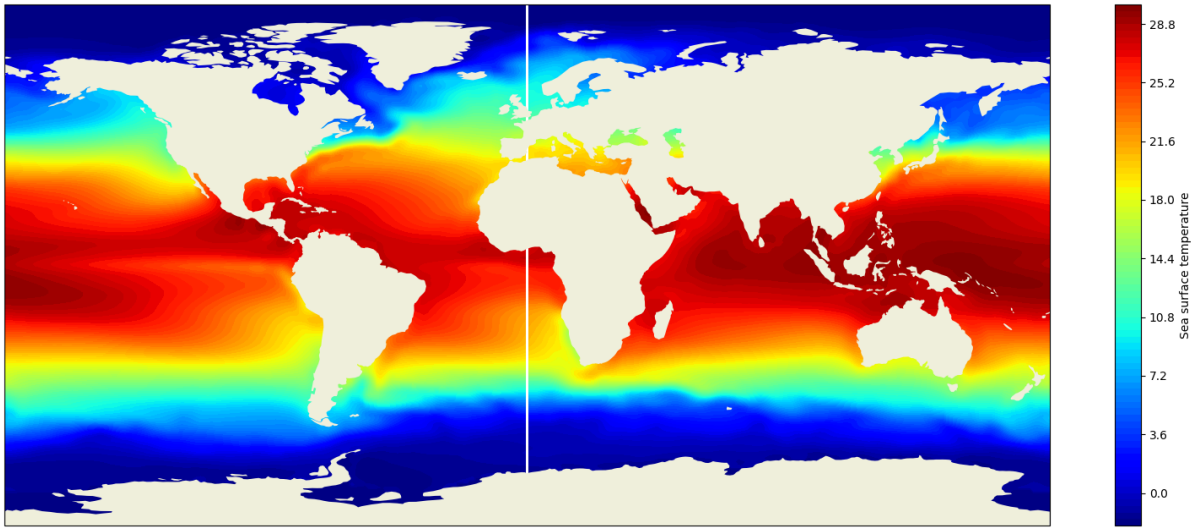

Figure 27: Average SST over the oceans.

As being seen, the red regions indicate higher SST, with the highest temperature shown being around 28°C. The blue regions indicate lower SST, with the lowest temperature shown being around 0°C. The color red is seen in the middle region, which means that equatorial regions have the highest SST, while the color blue is mostly in the top and bottom edge of the graph, which means the polar regions have the lowest SST. The temperature is very high near the equator and get cooler moving toward the poles.

2.9.3 Plot the SST departures.

SST departure is the difference between the current temperature and the average temperature in each location. It is calculated by:

$$T_{de} = T_o - T_{long}$$

in which T_{de} is the SST departure, ST_o is the observed SST and T_{long} is the long-term SST.

The resulting SST departure values can be positive or negative, which shows that whether the ocean temperatures are higher or lower than normal, respectively.

Listing 32: Practice 9.3

```
import numpy as np
import netCDF4 as nc
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fh = nc.Dataset('sst.mnmean.nc', 'r')

lat = fh.variables['lat'][:].data
long = fh.variables['lon'][:].data

sst_average = np.mean(sst, axis=(2))
```

```
13 sst_departure = sst - np.expand_dims(sst_average, axis=(2))
14
15 max_abs_value = np.max(np.abs(sst_departure))
16
17 fig = plt.figure(figsize = (15, 6))
18 ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
19
20 im = ax.contourf(long, lat, np.mean(sst_departure, axis=0), cmap=
       'jet', levels=100, transform=ccrs.PlateCarree())
21
22 ax.add_feature(cfeature.LAND, zorder=1)
23
24 cbar = fig.colorbar(im, ax=ax, label='SST departures')
25
26 plt.tight_layout()
27 plt.show()
```

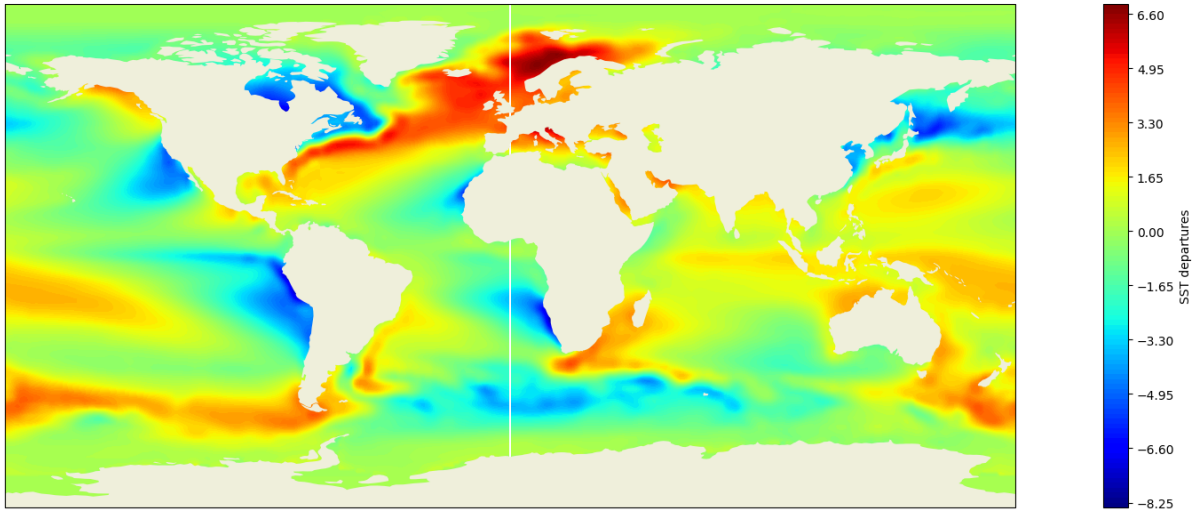

Figure 28: SST departures.

As being seen, the red color represents positive departure SST values, which means that the observed SST is warmer than the reference value, while the color blue represents negative departure SST values, which means that the observed SST is cooler than the reference value.

The highest SST departure values are mostly seen in the middle top edge of the graph, which is where Atlantic Ocean and Arctic Ocean meet. This due to the global warming trend due to climate change. Warmer temperature in this region can lead to reduced sea ice extent, creating a feedback loop that increases SST over the world. High SST departure values can also be found at the central Pacific Ocean. This is due to the El Niño events. During El Niño, the trade winds weaken, and warm water from the western Pacific moves eastward.

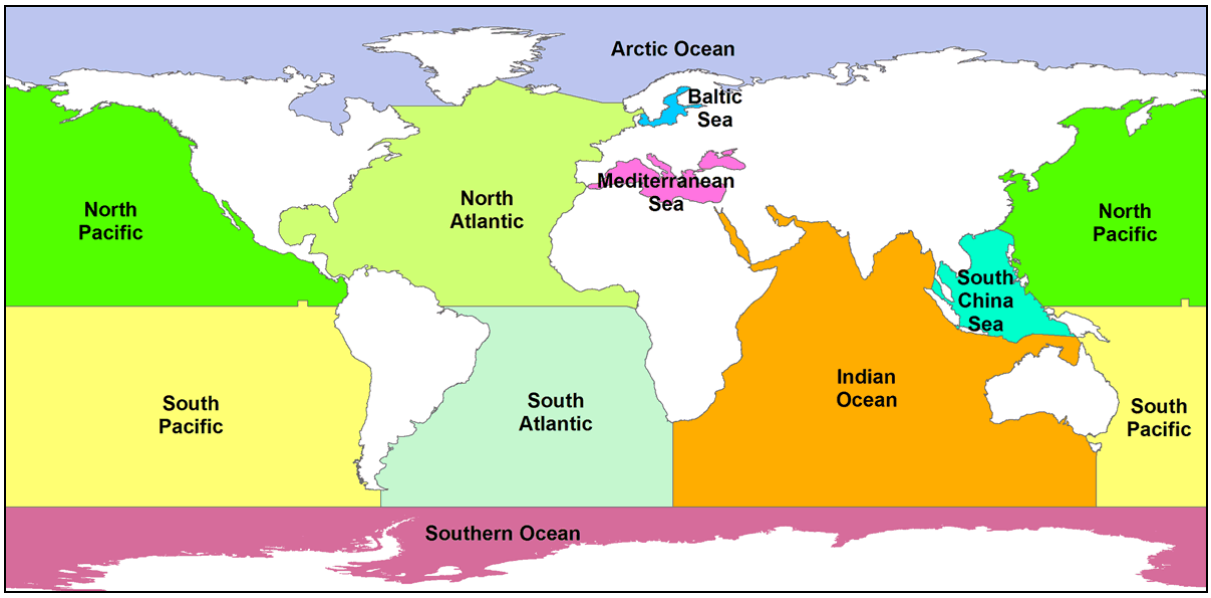

Figure 29: Just a global ocean map for reference.

The lowest SST departure values can be seen at the Eastern Pacific Ocean, which is also the West coast of South America and North America. This is because the winds typically blow from the north or northwest parallel to the coastline. It pushes the warm surface water away from the coast, allowing the cold water from the deeper ocean to rise to the surface. Western North Atlantic Ocean also has low SST departure values, this is the influence of the ice melted from the region where North Atlantic Ocean and Artic Ocean meet.

2.9.4 Calculate the Niño3 index and identify the El Niño and La Niña years for the period 1981-present.

El Niño is a climate pattern that occurs when the ocean waters in the central and eastern Pacific Ocean get unusually warm. El Niño events usually causes the world be warmer. It also changes the normal weather patterns, leading to droughts, heavy rains, and strong storms in some areas. On the other hand, La Niña is the event when the ocean water in the central and eastern Pacific Ocean are much colder than normal. It results in cooler global temperatures, and can also influence the frequency and intensity of tropical cyclones and other weather phenomena.

The Niño3 index is a measure focuses on the region bounded by 5°N to 5°S latitude and 150°W to 90°W longitude. In the following code, first I define the region latitudes are between -5° and 5°, and longitudes are between 210° and 270°. After extracting the SST data for the Niño 3 region using indexing, I let the algorithm calculates the mean SST for the Niño 3 region for each year.

Finally, the algorithm calculates the Niño3 index by:

$$Nino = SST_{nino} - SST_{entire}$$

in which $Nino$ is the Niño 3 index, SST_{entire} is the mean SST for the Niño 3 region for each year, and SST_{entire} is the overall mean SST for the entire region.

Listing 33: Practice 9.4

```
import numpy as np
import matplotlib.pyplot as plt
import netCDF4 as nc
from netCDF4 import Dataset
import datetime as dt

data = Dataset('sst.mnmean.nc', 'r')

lats = data.variables['lat'][:]
lons = data.variables['lon'][:]
times = data.variables['time'][:]
sst = data.variables['sst'][:]
units = data.variables['time'].units

d_times = nc.num2date(times, units, only_use_cftime_datetimes=
   False, only_use_python_datetimes=True)

latidx_nino3 = (lats >= -5.) & (lats <= 5.)
lonidx_nino3 = (lons >= 210.) & (lons <= 270.)

sst_nino3 = sst[:, latidx_nino3][:, :, lonidx_nino3]

nino3 = np.mean(sst_nino3, axis=(1, 2)) - np.mean(sst_nino3)

plt.figure(figsize=(10, 5))

plt.plot(d_times, nino3, color = 'deeppink')
plt.xlabel('Year')
plt.ylabel('SST Anomaly')
plt.grid()
plt.show()
```

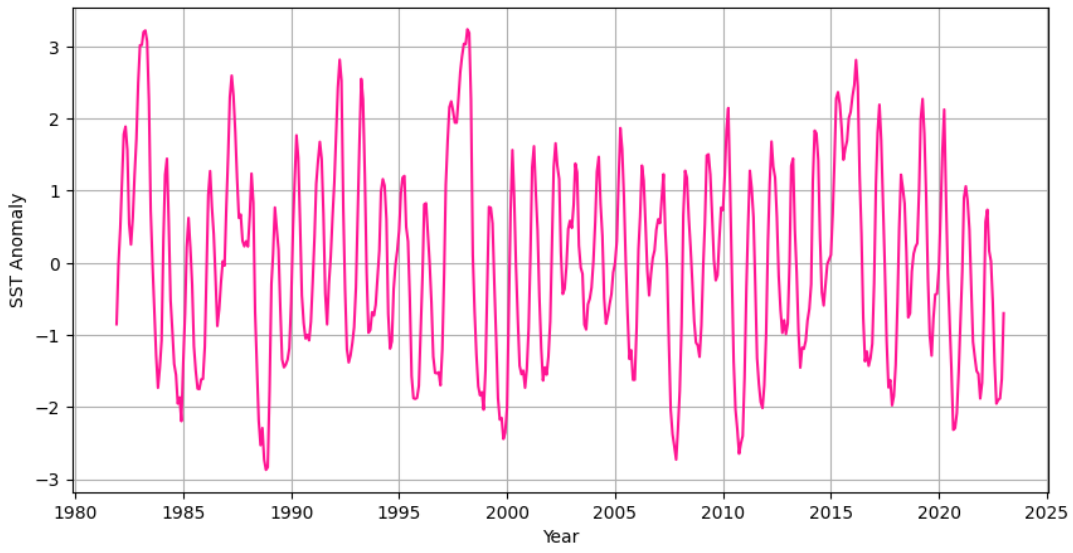

Figure 30: Anomaly over Niño3.

As being seen, the peaks are the El Niño events where the SST is warmer than average. Significant peaks are being seen around 1983, 1998, 2016, and early 2020s. The troughs are La Niña events where the SST is cooler than average. Significant troughs are being seen around 1989, 1999, 2008, and 2011. The anomalies can be quite high, with some El Niño events showing anomalies greater than 2°C and some La Niña events showing anomalies below -2°C.

The El Niño-Southern Oscillation (ENSO) criteria can be used to identify the El Niño and La Niña years. By looking at the SST anomalies and finding times when the temperature is a lot higher than normal, which means it is the El Niño event, or lower than normal, which means it is the La Niña event.

In the following code, I define el_nino_threshold and la_nina_threshold to 0.5 and -0.5, respectively. When the Niño 3 index is greater than or equal to 0.5, it's considered an El Niño event. When the Niño 3 index is less than or equal to -0.5, it's considered a La Niña event. Then I create true/false masks to check if the sum of 5 consecutive days meets the condition for identifying El Niño and La Niña events. If it is the El Niño event, it is displayed by the color pink. If it is the La Niña event, it is displayed by the color blue.

Listing 34: Also practice 9.4

```
el_nino_threshold = 0.5
la_nina_threshold = -0.5

el_nino_mask = np.convolve(nino3 >= el_nino_threshold, np.ones(5)
    , mode='valid') == 5
el_nino_mask = np.concatenate((np.zeros(4, dtype=bool),
    el_nino_mask))

la_nina_mask = np.convolve(nino3 <= la_nina_threshold, np.ones(5)
    , mode='valid') == 5
la_nina_mask = np.concatenate((np.zeros(4, dtype=bool),
    la_nina_mask))

plt.figure(figsize=(10, 5))

plt.plot(d_times, nino3, color="purple", linewidth=2, label="Nino
    3")
plt.fill_between(d_times, nino3, where=el_nino_mask, color="
    deeppink", alpha=0.5, label="El Nino")
plt.fill_between(d_times, nino3, where=la_nina_mask, color="blue"
    , alpha=0.5, label="La Nina")

plt.xlabel('Year')
plt.ylabel('SST Anomaly')
plt.legend()
plt.grid()
plt.show()
```

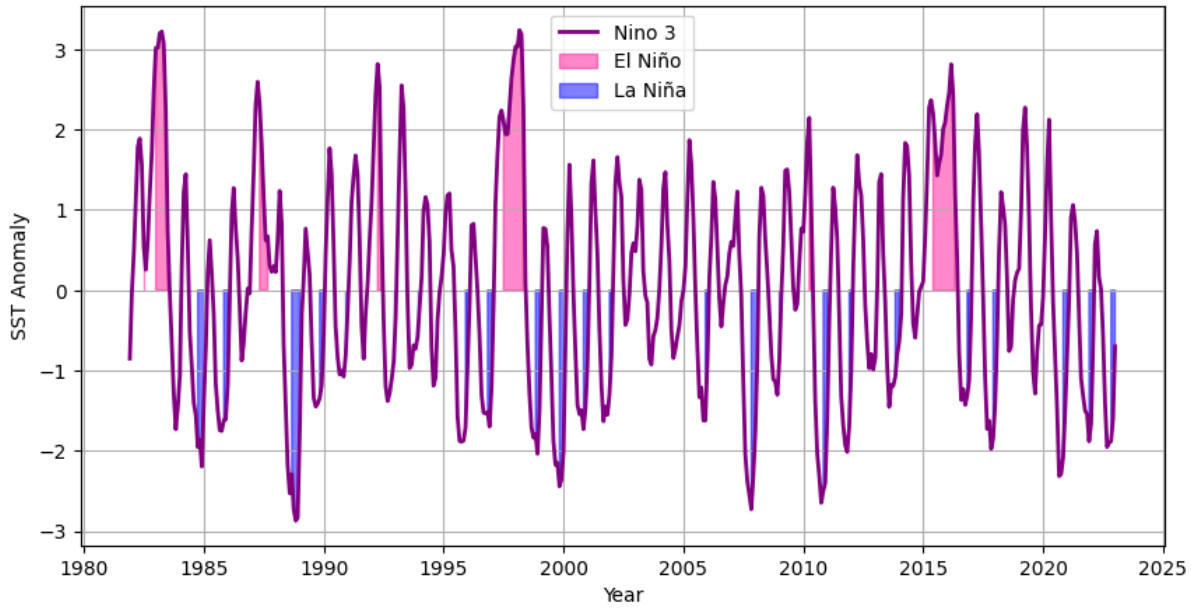

Figure 31: El Niño and La Niña over Niño3 using ENSO criteria.

As being seen, significant peaks, which are El Niño events, are being seen in early 1983, 1987, 1997-1998, 2002-2003, 2009-2010, and 2015-2016. Significant troughs, which are La Niña events, are being seen in 1988-1989, 1998-2000, 2007-2008, 2010-2012, and 2016-2018. It can easily be seen that both El Niño and La Niña events occur irregularly over time. The 1990s and 2000s is the time when it shows multiple strong events of both types.

2.10 QBO

The Quasi-Biennial Oscillation (QBO) is a repeating pattern in the wind direction in the upper atmosphere near the equator. It alternates between easterly and westerly directions approximately every 2 years.

Download QBO data from here.

This is what the file looks like when opened on my laptop.

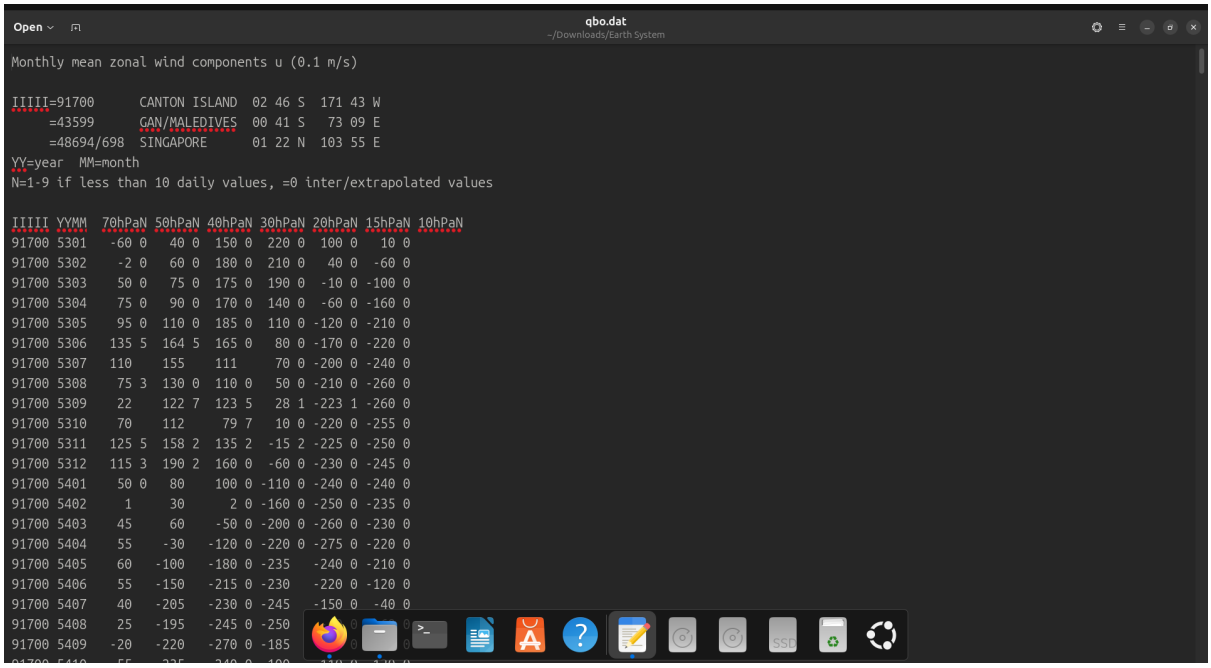

Figure 32: Dataset for practice 10.

And when opened with Python.

Listing 35: Code to read the dataset for practice 10

```
with open('qbo.dat', 'r') as file:
    lines = file.readlines()

    for _ in range(7):
        lines.pop(0)

    for line in lines:
        print(line.strip())
```

IIIII	YYMM	70hPaN	50hPaN	40hPaN	30hPaN	20hPaN	15hPaN	10hPaN
91700	5301	-60 0	40 0	150 0	220 0	100 0	10 0	
91700	5302	-2 0	60 0	180 0	210 0	40 0	-60 0	
91700	5303	50 0	75 0	175 0	190 0	-10 0	-100 0	
91700	5304	75 0	90 0	170 0	140 0	-60 0	-160 0	
91700	5305	95 0	110 0	185 0	110 0	-120 0	-210 0	
91700	5306	135 5	164 5	165 0	80 0	-170 0	-220 0	
91700	5307	110	155	111	70 0	-200 0	-240 0	
91700	5308	75 3	130 0	110 0	50 0	-210 0	-260 0	
91700	5309	22	122 7	123 5	28 1	-223 1	-260 0	
91700	5310	70	112	79 7	10 0	-220 0	-255 0	
91700	5311	125 5	158 2	135 2	-15 2	-225 0	-250 0	
91700	5312	115 3	190 2	160 0	-60 0	-230 0	-245 0	
91700	5401	50 0	80	100 0	-110 0	-240 0	-240 0	
91700	5402	1	30	2 0	-160 0	-250 0	-235 0	
91700	5403	45	60	-50 0	-200 0	-260 0	-230 0	
91700	5404	55	-30	-120 0	-220 0	-275 0	-220 0	
91700	5405	60	-100	-180 0	-235	-240 0	-210 0	
91700	5406	55	-150	-215 0	-230	-220 0	-120 0	
91700	5407	40	-205	-230 0	-245	-150 0	-40 0	
91700	5408	25	-195	-245 0	-250	-40 0	60 0	
91700	5409	-20	-220	-270 0	-185	40 0	110 0	
91700	5410	-55	-235	-240 0	-100	110 0	130 0	
91700	5411	-80	-210	-200 0	60	120 0	155 0	
91700	5412	-90	-195	-80 0	75	150 0	160 0	
91700	5501	-40	-105	10 0	105	155 0	155 0	
91700	5502	20	-25	70 0	108	150 0	160 0	
91700	5503	55	5	105 0	105	155 0	165 0	
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Table 3: Dataset read for practice 10.

This file contains monthly mean zonal wind component (*m/s*) for Canton Island (02°46'S, 171°43'W), Gan/Maldives (00°41'S, 73°09'E), and Singapore (01°22'N, 103°55'E). YYMM is year and month. For each location and month, it shows the zonal wind component at seven pressure levels: 70 hPa, 50 hPa, 40 hPa, 30 hPa, 20 hPa, 15 hPa, and 10 hPa. If the value is less than 10, it represents the number of daily values used. If the value is 0, it means the data was inter/extrapolated.

2.10.1 From the data downloaded, prove that QBO exists.

In the following code, I skip the first 9 rows and drop the first 2 columns, which are unnecessary information. I let the algorithm separates the year and month components from the time values, where the year is obtained by integer division by 100, and the month is obtained by taking the modulo of 100 and creates a datetime series using these. After assigning data to the corresponding locations and plot 7 subplots as below.

Listing 36: Practice 10.1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

qbo = pd.read_fwf('qbo.dat', skiprows=9, header=None)

time = np.asarray(qbo[1])
year = time // 100
year += np.where(year >= 53, 1900, 2000)
month = time % 100

datetime_series = pd.to_datetime({'year': year, 'month': month, 'day': 1})

df_qbo = qbo.drop([0, 1], axis=1) * 0.1
df_qbo.columns = ['70 hPa', 'N', '50 hPa', 'N', '40 hPa', 'N', '30 hPa', 'N', '20 hPa', 'N', '15 hPa', 'N', '10 hPa', 'N']

location = np.empty_like(year, dtype='U15')
location[qbo[0] == 91700] = 'CANTON ISLAND'
location[qbo[0] == 43599] = 'GAN / MALEDIVES'
location[(qbo[0] == 48694) | (qbo[0] == 48698)] = 'SINGAPORE'

fig, ax = plt.subplots(7, 1, figsize=(25, 20), sharex=True, sharey=True)
pressure_levels = ['70 hPa', '50 hPa', '40 hPa', '30 hPa', '20 hPa', '15 hPa', '10 hPa']

for i, level in enumerate(pressure_levels):
    ax[i].plot(datetime_series, df_qbo[level], color='deeppink')
    ax[i].set_ylabel("Velocity")
    ax[i].set_title("QBO at {}".format(level))
    ax[i].grid(True)

plt.xlabel("Year")
plt.grid(True)
plt.show()
```

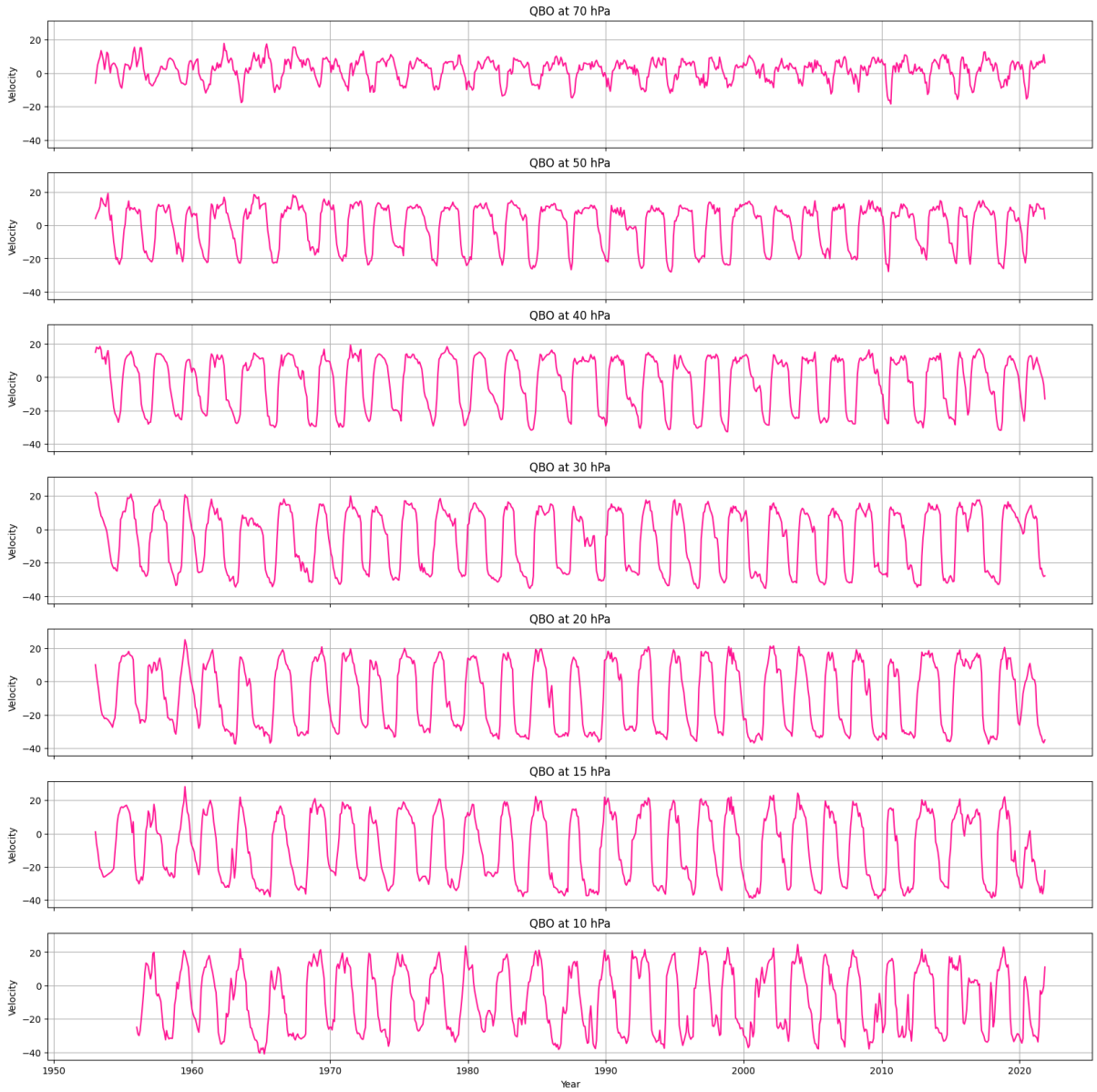

Figure 33: Wind speed at various pressure levels.

As being seen, the oscillation is weaker at higher altitudes, especially at 70 hPa, and it fluctuates more, which means it gets stronger as going down to lower altitudes. It can also be seen in every pressure bands that the space between two peak each two valley is about two years in a cycle. And these valley represent easterlies zonal wind. Therefore it can be concluded that QBO exist.

2.10.2 Identify QBO's characteristics from the data.

This is what the file looks like when opened on my laptop.

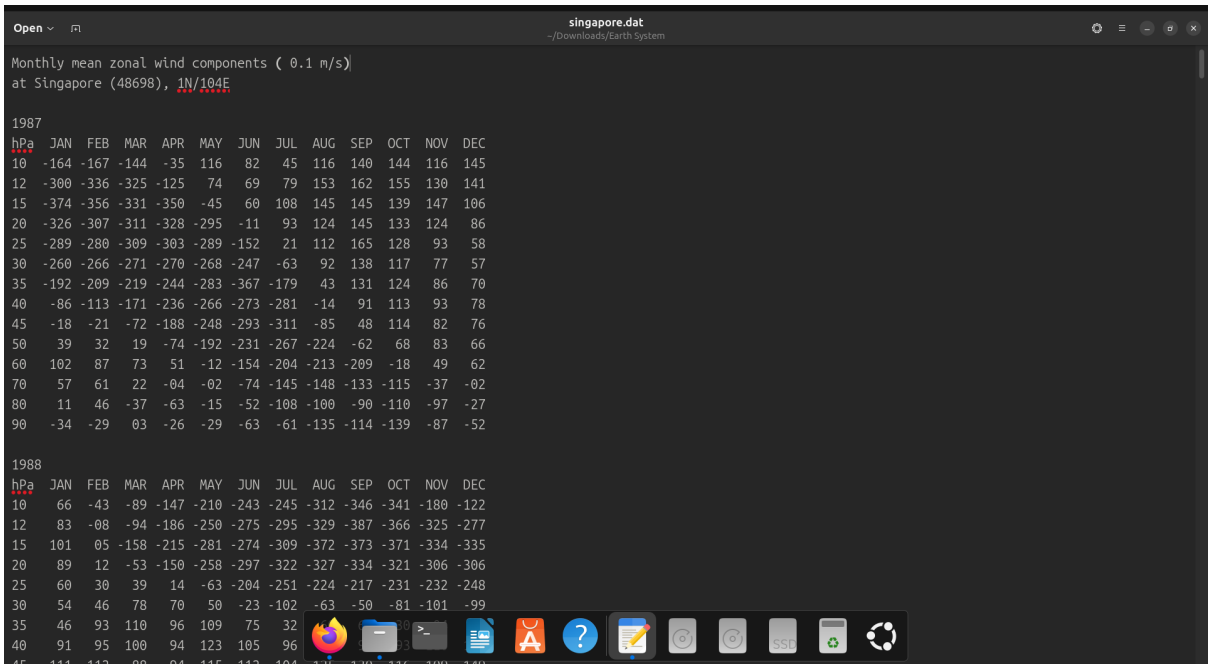

Figure 34: Also dataset for practice 10.

And when opened with Python.

Listing 37: Another code to read another dataset for practice 10

```
with open('singapore.dat', 'r') as file:
    lines = file.readlines()

    for _ in range(3):
        lines.pop(0)

    for line in lines:
        print(line.strip())
```

						1987						
hPa	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
10	-164	-167	-144	-35	116	82	45	116	140	144	116	145
12	-300	-336	-325	-125	74	69	79	153	162	155	130	141
15	-374	-356	-331	-350	-45	60	108	145	145	139	147	106
20	-326	-307	-311	-328	-295	-11	93	124	145	133	124	86
25	-289	-280	-309	-303	-289	-152	21	112	165	128	93	58
30	-260	-266	-271	-270	-268	-247	-63	92	138	117	77	57
35	-192	-209	-219	-244	-283	-367	-179	43	131	124	86	70
40	-86	-113	-171	-236	-266	-273	-281	-14	91	113	93	78
45	-18	-21	-72	-188	-248	-293	-311	-85	48	114	82	76
50	39	32	19	-74	-192	-231	-267	-224	-62	68	83	66
60	102	87	73	51	-12	-154	-204	-213	-209	-18	49	62

70	57	61	22	-04	-02	-74	-145	-148	-133	-115	-37	-02
80	11	46	-37	-63	-15	-52	-108	-100	-90	-110	-97	-27
90	-34	-29	03	-26	-29	-63	-61	-135	-114	-139	-87	-52
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.

Table 4: Another dataset read for practice 10.

The file contains information about monthly mean zonal wind components (0.1 m/s) at Singapore from January 1987 to November 2021. As being seen, the hPa levels are from 10 to 90. There are 16 rows per year.

In the following code, I skip the first 4 rows where the information is unnecessary, and rows where all data columns are NaN and finally plot the data as below.

Listing 38: Practice 10.2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv('singapore.dat', skiprows=4, header=None,
    delim_whitespace=True, comment='h')

data = data[data[0] < 1900]

data = data.dropna(how='all')

data_values = data.values[:, 1:] * 0.1

total_months = data.shape[0]

start_year = 1981
years_per_record = 1 / 16

years = np.arange(start_year, start_year + total_months *
    years_per_record, years_per_record)

pa = np.arange(10, 100, 7.5)

fig = plt.figure(figsize=(15, 5))
plt.contourf(years, pa, data_values.T, cmap='RdBu')

plt.gca().invert_yaxis()
plt.colorbar()
plt.xlabel("Year")
plt.ylabel("Pressure")

```

```
30 plt.show()
```

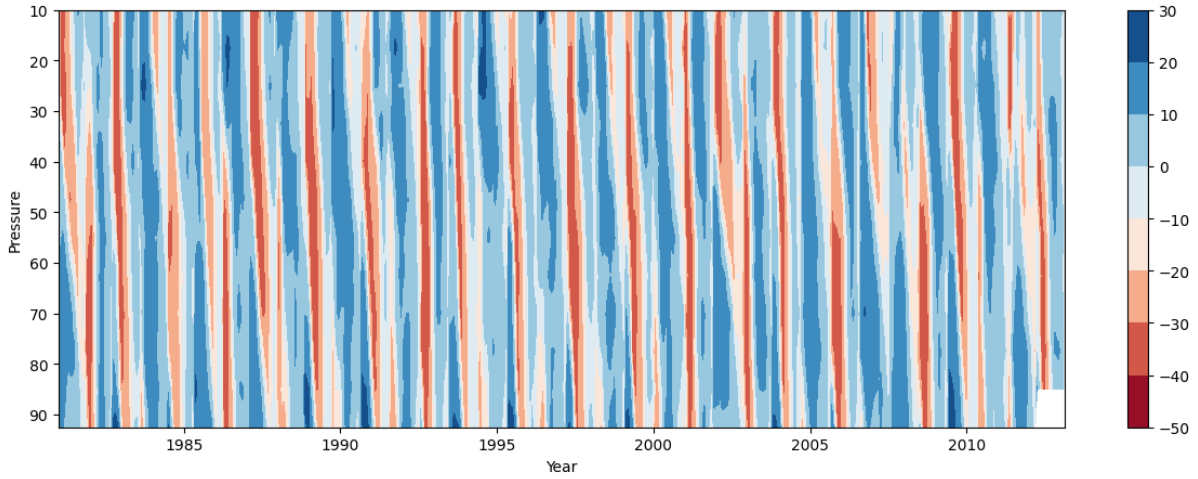

Figure 35: QBO in Singapore from 1987 to 2021.

As being seen, the color blue represents positive values, which are the westerly winds, while the color red represents negative values, which are the easterly winds. The most pronounced variations are typically observed at mid-stratospheric levels. There is a blank area at the bottom right of the graph. It is the space for December 2021, and since the file only contains data till November 2021, it is blank.

And by looking at the graph, some characteristic of QBO can be concluded:

- The QBO periods range from 20 to 36 months, with an average of 28 months.
- The amplitude of the easterlies is usually stronger (about 2 times) than the amplitude of the westerlies.
- The QBO signal moves downward over time, from around 30 km (10 hPa) down to 16 km (100 hPa) or lower.
- The downward movement speed is about 1 km/month.
- At the top of the QBO region, the easterly winds are dominant, while at the bottom, the westerly winds are more common.
- The westerlies propagate downward faster than the easterlies.
- QBO amplitude decreases as the altitude decreases. The maximum amplitude of 40-50 m/s is usually be seen around 20 mb (25 km).
- The transition from the westerlies to the easterlies usually slows down between 30-50 mb (20-23 km).
- There are significant fluctuations in QBO periods and amplitudes.

2.11 MJO

The Madden-Julian Oscillation (MJO) was discovered in 1971 by scientists named Roland Madden and Paul Julian. It is a weather pattern that moves across the Indian and Pacific Oceans. It shows up as large groups of clouds, rain, winds, and air pressure that move eastward. The MJO happens on a timescale of 30 to 60 days. This means it repeats itself about once a month to two months.

Using GPCP global rainfall data to detect MJO for a period of any 2 years of your choice; and to show MJO's characteristics.

For this practice, I found the GPCP Monthly dataset which contains monthly values from 1979 to 2024, and long term monthly means derived from years 1981 to 2010 also. The dataset is on a 2.5 degree latitude by 2.5 degree longitude global grid, covering 88.75°N to 88.75°S and 1.25°E to 358.75°E, with surface level measurements. Missing data in the file is flagged with a value of -9.96921e+36f. I choose to download a subset that contains long term mean surface precipitation from 2003 to 2004.

This is what the file looks like when opened on my laptop.

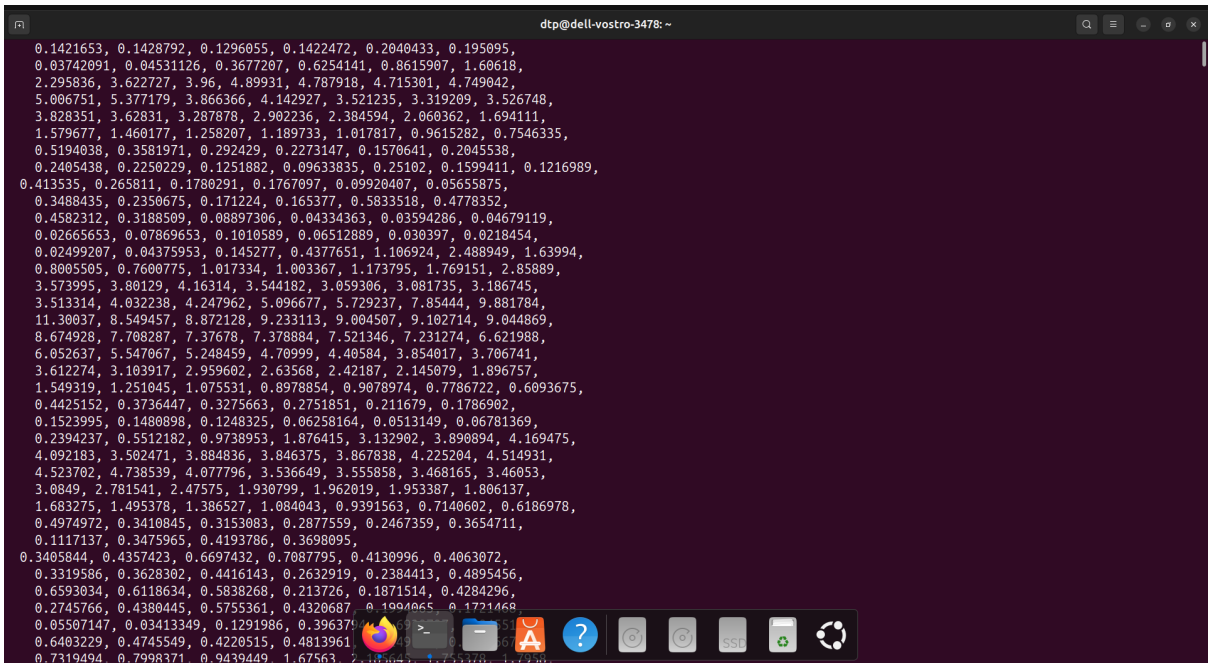

Figure 36: Dataset for practice 11.

And when opened with Python.

Listing 39: Code to read the dataset for practice 11.

```
import netCDF4 as nc

ds = nc.Dataset('subset.nc')

print(ds)

```

```
print(ds.variables['precip'])
print(ds.variables['lat'])
print(ds.variables['lon'])
print(ds.variables['time'])
print(ds.variables['valid_yr_count'])
```

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF3_CLASSIC data model, file format NETCDF3):
    Conventions: CF-1.0
    curator: Dr. Jian-Jian Wang
ESSIC, University of Maryland College Park
College Park, MD  20742  USA
Phone: +1 301-405-4887
    description: http://eagle1.umd.edu/GPCP_ICDR/GPCPmonthlyV2.3.pdf
    citation: Adler, R.F., G.J. Huffman, A. Chang, R. Ferraro, P. Xie, J. Janowiak, B
Rudolf, U. Schneider, S. Curtis, D. Bolvin, A. Gruber, J. Susskind, P.
Arkin, 2003: The Version 2 Global Precipitation Climatology Project
(GPCP) Monthly Precipitation Analysis (1979 - Present). J. Hydrometeor.,
4(6), 1147-1167.
    title: GPCP Version 2.3 Combined Precipitation Dataset (Final)
    platform: NOAA POES (Polar Orbiting Environmental Satellites)
    source_obs: CDR RSS SSMI/SSMIS Tbs over ocean
CDR SSMI/SSMIS rainrates over land (Ferraro)
Geo-IR (Xie) calibrated by SSMI/SSMIS rainrates for sampling
TOVS/AIRS empirical precipitation estimates at higher latitudes
(ocean and land)
GPCC gauge analysis to bias correct satellite estimates over land and
merge with satellite based on sampling
OLR Precipitation Index (OPI) (Xie) used for period before 1988
    source: http://eagle1.umd.edu/GPCP_CDR/Monthly_Data/
    documentation: https://www.esrl.noaa.gov/psd/data/gridded/data.gpcp.html
    source_documentation: http://eagle1.umd.edu/GPCP_ICDR/
    version: V2.3
    Acknowledgement:

    contributor_name: Robert Adler    University of Maryland
George Huffman  NASA Goddard Space Flight Center
David Bolvin    NASA Goddard Space Flight Center/SSAI
Eric Nelkin     NASA Goddard Space Flight Center/SSAI
Udo Schneider   GPCC, Deutscher Wetterdienst
Andreas Becker  GPCC, Deutscher Wetterdienst
Long Chiu       George Mason University
Mathew Sapiano  University of Maryland
Pingping Xie    Climate Prediction Center, NWS, NOAA
Ralph Ferraro   NESDIS, NOAA
Jian-Jian Wang  University of Maryland
Guojun Gu       University of Maryland
    history: Created 2016/11/08 by doMonthLTM
```

```
    References: https://www.esrl.noaa.gov/psd/data/gridded/data.gpcp.html
    dataset_title: Global Precipitation Climatology Project (GPCP) Monthly Analysis P
    not_missing_threshold_percent: minimum 3% values input to have non-missing output
    dimensions(sizes): lon(144), lat(72), time(12), nbnds(2)
    variables(dimensions): float32 lat(lat), float32 lon(lon), float64 time(time), fl
    groups:
<class 'netCDF4._netCDF4.Variable'>
float32 precip(time, lat, lon)
    long_name: Long Term Mean Average Monthly Rate of Precipitation
    valid_range: [  0. 100.]
    units: mm/day
    add_offset: 0.0
    scale_factor: 1.0
    missing_value: -9.96921e+36
    precision: 32767
    least_significant_digit: 2
    var_desc: Precipitation
    dataset: GPCP Version 2.3 Combined Precipitation Dataset
    level_desc: Surface
    statistic: Long Term Mean
    parent_stat: Mean
    actual_range: [5.6044193e-04 3.0161072e+01]
unlimited dimensions:
current shape = (12, 72, 144)
filling on, default _FillValue of 9.969209968386869e+36 used
<class 'netCDF4._netCDF4.Variable'>
float32 lat(lat)
    units: degrees_north
    actual_range: [ 88.75 -88.75]
    long_name: Latitude
    standard_name: latitude
    axis: Y
unlimited dimensions:
current shape = (72,)
filling on, default _FillValue of 9.969209968386869e+36 used
<class 'netCDF4._netCDF4.Variable'>
float32 lon(lon)
    units: degrees_east
    long_name: Longitude
    actual_range: [  1.25 358.75]
    standard_name: longitude
    axis: X
unlimited dimensions:
current shape = (144,)
filling on, default _FillValue of 9.969209968386869e+36 used
<class 'netCDF4._netCDF4.Variable'>
float64 time(time)
    units: days since 1800-1-1 00:00:0.0
```

```
    long_name: Time
    delta_t: 0000-01-00 00:00:00
    avg_period: 0030-00-00 00:00:00
    prev_avg_period: 0000-01-00 00:00:00
    standard_name: time
    axis: T
    actual_range: [-657073. -656739.]
    climatology: climatology_bounds
    climo_period: 1981/01/01 - 2010/12/31
    ltm_range: [66109. 77035.]
    interpreted_actual_range: 0001/01/01 00:00:00 - 0001/12/01 00:00:00
unlimited dimensions:
current shape = (12,)
filling on, default _FillValue of 9.969209968386869e+36 used
<class 'netCDF4._netCDF4.Variable'>
int16 valid_yr_count(time, lat, lon)
    long_name: count of non-missing values used in mean
    missing_value: 32767
    add_offset: 0.0
    scale_factor: 1.0
unlimited dimensions:
current shape = (12, 72, 144)
filling on, default _FillValue of -32767 used
```

As being seen, these are the information about the dataset. It contains five variables: long term mean average monthly rate of precipitation (mm/day) as precip, latitude (degrees_north), longitude (degrees_east), days since 1800-1-1 00:00:0.0, and count of non-missing values used in the mean as valid_yr_count. It has four dimensions: lon (144 points), lat (72 points), time (12 points), and nbnds (2 points). The precip variable has a valid range of 0 to 100 mm/day, a missing value of -9.96921e+36, and an actual range of 0.0005604419 to 30.16107 mm/day. The valid_yr_count variable has a missing value of 32767 and is used to indicate the count of non-missing values used in the mean calculation.

In the following code, I calculate the mean precipitation value across the time dimension, which results in a 2D array of average precipitation values and plot using ax.contourf() function.

Listing 40: Practice 11

```
1 import cartopy.feature as cfeature
2 import cartopy.crs as ccrs
3 import netCDF4 as nc
4 from netCDF4 import Dataset
5 import matplotlib.pyplot as plt
6 import numpy as np
7
8 fh = Dataset('subset.nc', 'r')
9
```

```
lat = fh.variables['lat'][:].data
lon = fh.variables['lon'][:].data
d_times = nc.num2date(fh.variables['time'][:], fh.variables['time'].units)

precip = fh.variables['precip'][:].data
preci_mean = np.mean(precip[:, :, :], axis=0)

fig = plt.figure(figsize=(15, 5))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
im = ax.contourf(lon, lat, preci_mean, levels=100, cmap='Blues', transform=ccrs.PlateCarree())
ax.add_feature(cfeature.COASTLINE, edgecolor='black', zorder=1)
cbar = fig.colorbar(im, ax=ax, label='Long term mean monthly precipitation (mm/day)', extend='both')

plt.show()
```

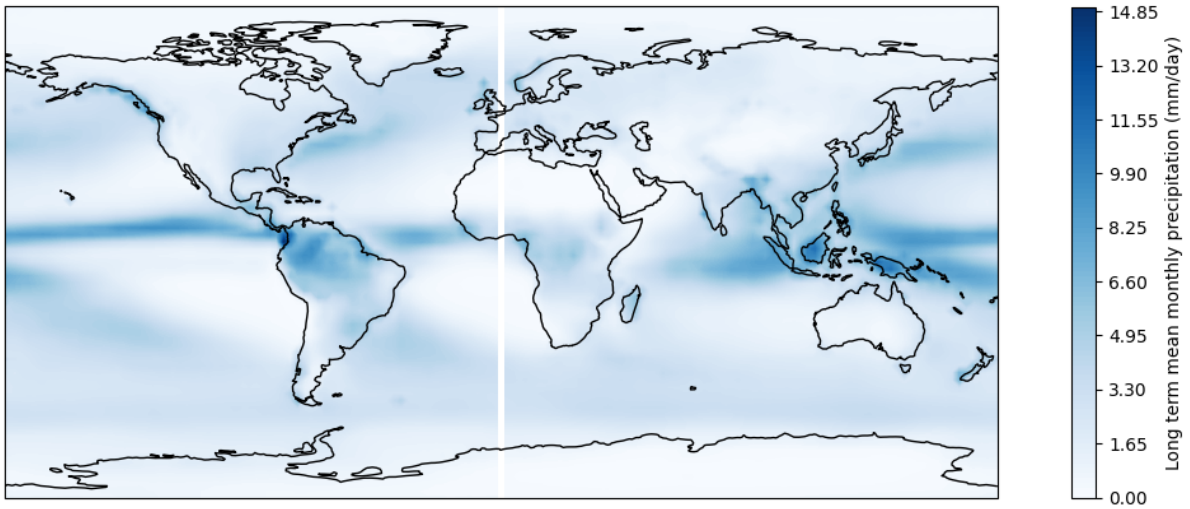

Figure 37: Long term mean monthly precipitation during 2003 - 2004.

As being seen, the darker blue represents higher mean monthly precipitation, with the highest mean precipitation shown being around 14.85mm/day, while the lighter blue represents lower mean monthly precipitation, with the lowest mean precipitation shown being 0mm/day.

And by looking at the graph, some characteristics of MJO can be concluded:

- The MJO's influence is most prominent within the equatorial region.
- The highest mean monthly precipitation values are being seen around western Pacific Ocean, the Maritime Continent, and over the Indian Ocean.
- The mean precipitation at other regions in the world are low.

2.12 Thai Binh rainfall

Given the Thai Binh rainfall data.

This is what the file looks like when opened on my laptop.

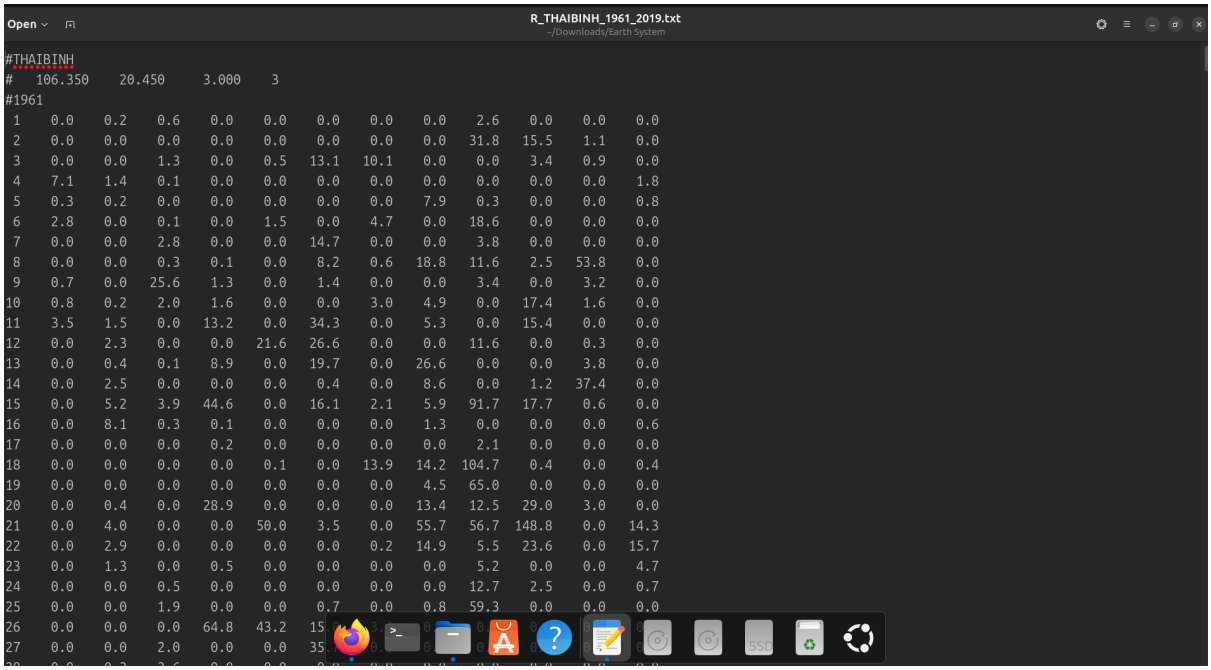

Figure 38: Dataset for practice 12.

And when opened with Python.

Listing 41: Code to read the dataset for practice 12

```
with open('R_THAIBINH_1961_2019.txt', 'r') as file:
    lines = file.readlines()

    for _ in range(2):
        lines.pop(0)

    for line in lines:
        print(line.strip())
```

	1961											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	0.0	0.2	0.6	0.0	0.0	0.0	0.0	0.0	2.6	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	31.8	15.5	1.1	0.0
3	0.0	0.0	1.3	0.0	0.5	13.1	10.1	0.0	0.0	3.4	0.9	0.0
4	7.1	1.4	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.8
5	0.3	0.2	0.0	0.0	0.0	0.0	0.0	7.9	0.3	0.0	0.0	0.8
6	2.8	0.0	0.1	0.0	1.5	0.0	4.7	0.0	18.6	0.0	0.0	0.0
7	0.0	0.0	2.8	0.0	0.0	14.7	0.0	0.0	3.8	0.0	0.0	0.0
8	0.0	0.0	0.3	0.1	0.0	8.2	0.6	18.8	11.6	2.5	53.8	0.0

9	0.7	0.0	25.6	1.3	0.0	1.4	0.0	0.0	3.4	0.0	3.2	0.0
10	0.8	0.2	2.0	1.6	0.0	0.0	3.0	4.9	0.0	17.4	1.6	0.0
11	3.5	1.5	0.0	13.2	0.0	34.3	0.0	5.3	0.0	15.4	0.0	0.0
12	0.0	2.3	0.0	0.0	21.6	26.6	0.0	0.0	11.6	0.0	0.3	0.0
13	0.0	0.4	0.1	8.9	0.0	19.7	0.0	26.6	0.0	0.0	3.8	0.0
14	0.0	2.5	0.0	0.0	0.0	0.4	0.0	8.6	0.0	1.2	37.4	0.0
15	0.0	5.2	3.9	44.6	0.0	16.1	2.1	5.9	91.7	17.7	0.6	0.0
16	0.0	8.1	0.3	0.1	0.0	0.0	0.0	1.3	0.0	0.0	0.0	0.6
17	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	2.1	0.0	0.0	0.0
18	0.0	0.0	0.0	0.0	0.1	0.0	13.9	14.2	104.7	0.4	0.0	0.4
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.5	65.0	0.0	0.0	0.0
20	0.0	0.4	0.0	28.9	0.0	0.0	0.0	13.4	12.5	29.0	3.0	0.0
21	0.0	4.0	0.0	0.0	50.0	3.5	0.0	55.7	56.7	148.8	0.0	14.3
22	0.0	2.9	0.0	0.0	0.0	0.0	0.2	14.9	5.5	23.6	0.0	15.7
23	0.0	1.3	0.0	0.5	0.0	0.0	0.0	0.0	5.2	0.0	0.0	4.7
24	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	12.7	2.5	0.0	0.7
25	0.0	0.0	1.9	0.0	0.0	0.7	0.0	0.8	59.3	0.0	0.0	0.0
26	0.0	0.0	0.0	64.8	43.2	15.8	3.4	0.0	0.0	0.0	0.5	0.0
27	0.0	0.0	2.0	0.0	0.0	35.7	0.0	0.0	0.0	0.0	0.0	0.4
28	0.0	0.3	3.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	-99.0	0.0	0.0	4.1	36.8	0.0	30.0	0.0	3.1	0.0	0.0
30	0.0	-99.0	1.5	13.7	0.0	0.0	1.3	34.0	0.0	0.8	3.0	0.5
31	4.3	-99.0	0.0	-99.0	0.0	-99.0	5.1	17.0	-99.0	0.0	-99.0	0.0

Table 5: Dataset read for practice 12.

As being seen, this file contains rainfall data at a latitude of 20.450°N and a longitude of 106.350°E, which is Thai Binh. The data covers from 1961 to 2019, with the year shown at the start of each section, indicated by . After that, there are 31 rows, which indicates 31 days in a month, and each row containing 12 numeric values, which indicates 12 months in a year.

2.12.1 Write a python program to read the daily rainfall data at THAIBINH station.

In the following code, I skip the first two rows since the information is unnecessary. The algorithm then iterates through each line in the file, if the line starts with "", it is the year. If the line does not start with "", it is then extracts the day and the values for each month. There are some -99.0 data points in the file, so I assume these are missing values so I assign them as NaN values. The extracted data is then stored in a pandas DataFrame with columns for 'Year', 'Month', 'Day', and 'Rainfall'. For each year, I create a 3x4 grid of subplots, corresponding to 12 months, using plt.subplots() function. For each month, the algorithm extracts the relevant data from the DataFrame and plots the daily rainfall values on the corresponding subplot, in which x-axis represents the day, and y-axis represents the amount of rainfall.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
4
5  file_path = 'R_THAIBINH_1961_2019.txt'
6
7  def read_and_parse_data(file_path):
8      with open(file_path, 'r') as file:
9          lines = file.readlines()
10
11     lines = lines[2:]
12
13     data = []
14     current_year = None
15
16     for line in lines:
17         line = line.strip()
18         if line.startswith('#'):
19             current_year = int(line[1:])
20         elif current_year:
21             values = line.split()
22             day = int(values[0])
23             for month in range(1, 13):
24                 if len(values) > month:
25                     value = float(values[month])
26                     if value == -99.0:
27                         value = np.nan
28                     data.append([current_year, month, day, value
                           ])
29
30     df = pd.DataFrame(data, columns=['Year', 'Month', 'Day', '
           Rainfall'])
31     return df
32
33 def plot_daily_rainfall(dataframe):
34     unique_years = dataframe['Year'].unique()
35     month_names = [
36         'January', 'February', 'March', 'April', 'May', 'June',
37         'July', 'August', 'September', 'October', 'November', '
               December'
38     ]
39
40     for year in unique_years:
41         df_year = dataframe[dataframe['Year'] == year]
42
43         fig, axs = plt.subplots(3, 4, figsize=(20, 15))
44         fig.suptitle(f'{year}')
45
46         x_min = df_year['Day'].min()
47         x_max = df_year['Day'].max()
48         y_min = df_year['Rainfall'].min()
49         y_max = df_year['Rainfall'].max()
50
51         for month in range(1, 13):
```

```
            month_data = df_year[df_year['Month'] == month]
            ax = axs[(month-1) // 4, (month-1) % 4]
            ax.plot(month_data['Day'], month_data['Rainfall'],
                color='deeppink')
            ax.set_title(month_names[month-1])
            ax.set_xlabel('Day')
            ax.set_ylabel('Rainfall')

            ax.set_xlim(x_min, x_max)
            ax.set_ylim(y_min, y_max)

        plt.tight_layout(rect=[0, 0, 1, 0.96])
        plt.show()

df = read_and_parse_data(file_path)
plot_daily_rainfall(df)
```

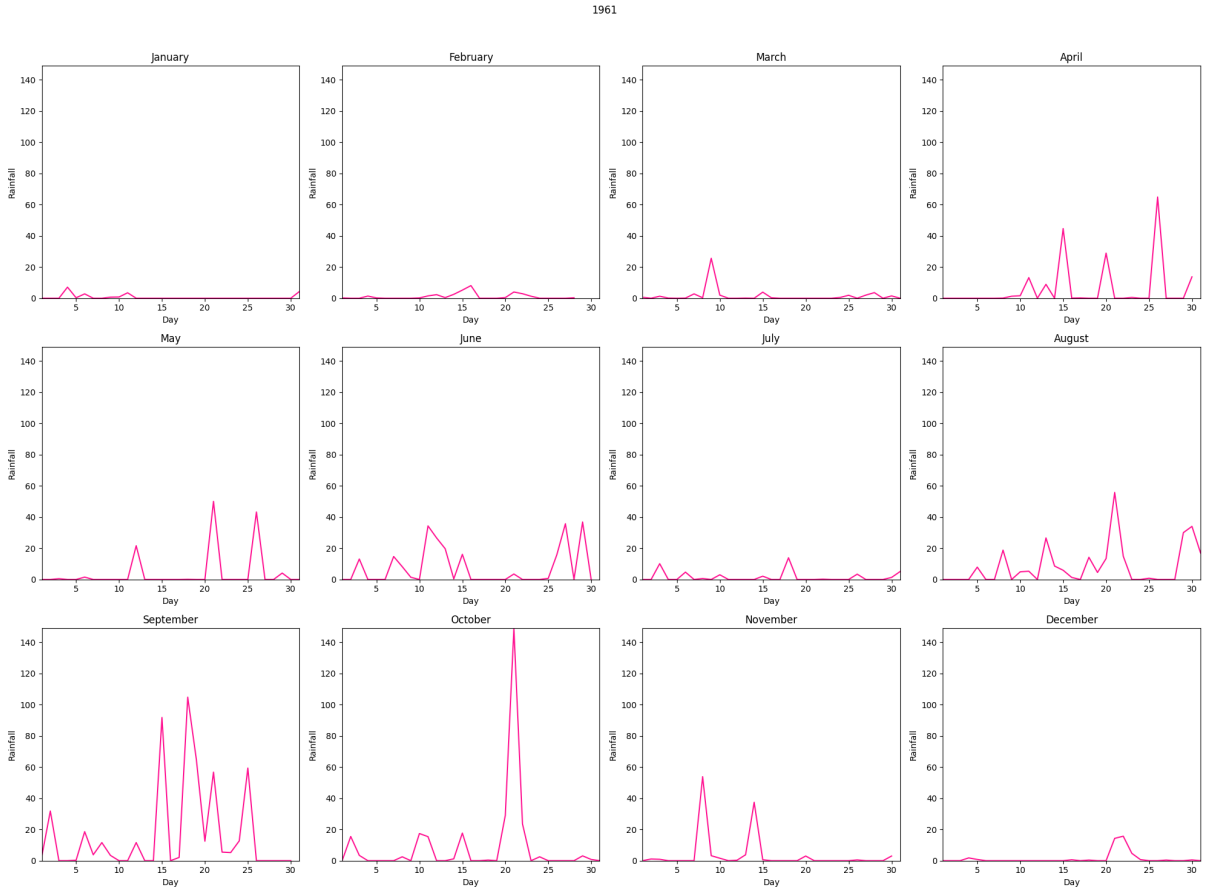

Figure 39: Daily rainfall data at Thai Binh in 1961.

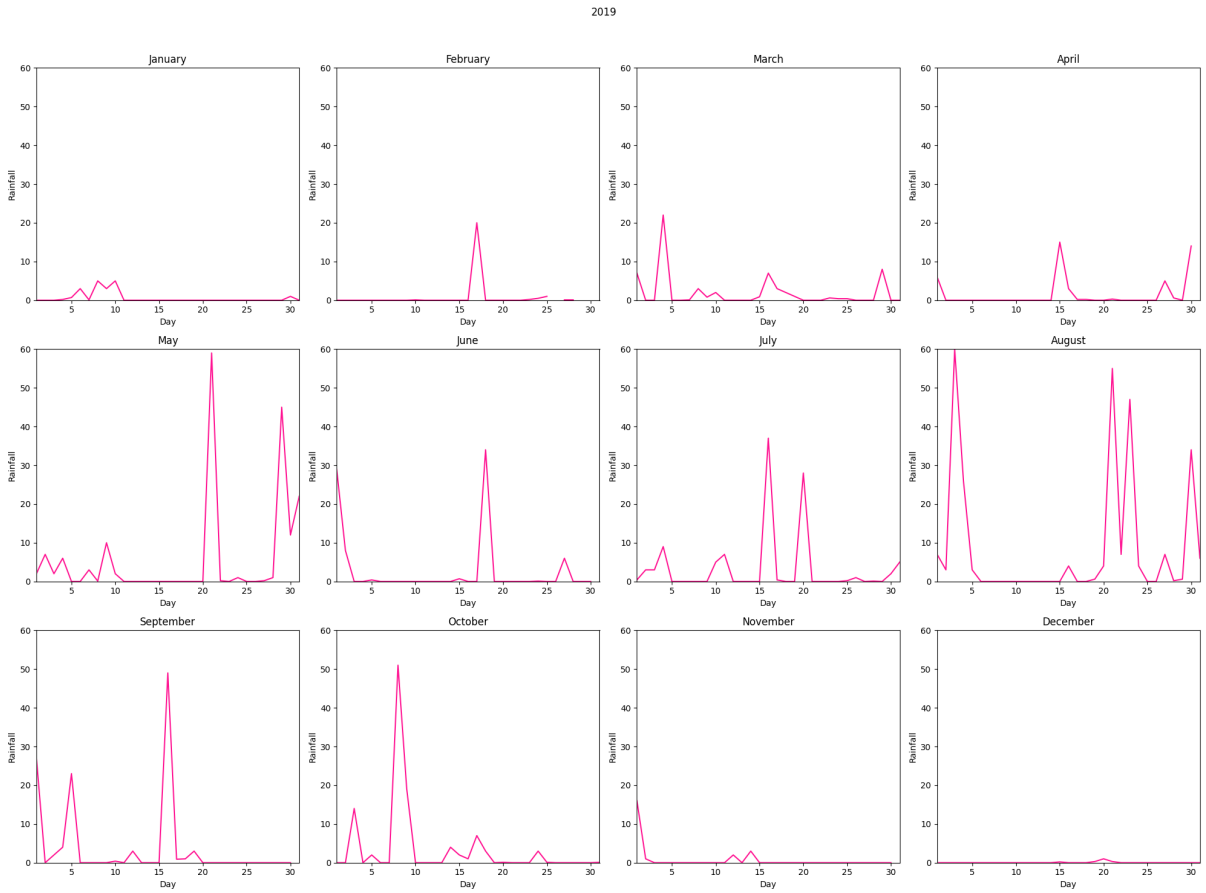

Figure 40: Daily rainfall data at Thai Binh in 2019.

This code took my laptop 3 minutes to plot all the daily rainfall data from 1961 to 2019. But since I cannot put all the graphs for 59 years here, I just provide the plots for the years 1961 and 2019 as shown above. As being seen, the overall seasonal patterns stay the same with the wettest months occurring during the summer and early fall and the driest months being in the winter and early spring. In 2019, the peak rainfall levels seem to be higher, with several months experiencing daily rainfall exceeding 40-50 mm, whereas the 1961 graph shows relatively lower peak rainfall, generally below 40 mm.

2.12.2 Estimate RX1day for each year.

RX1day, or the annual maximum 1-day precipitation, is a climate indicator that measures the maximum amount of precipitation recorded over a 24-hour period in a given year. In the following code, for each year, the maximum precipitation value is tracked by updating the max_precipitation_per_year dictionary with the maximum value seen so far and then print them out.

Listing 42: Practice 12.2

```
file_path = 'R_THAIBINH_1961_2019.txt'

rainfall_data = []
current_year = None
max_precipitation_per_year = {}
```

```
6
7  with open(file_path, 'r') as file:
8      next(file)
9      next(file)
10     for line in file:
11         line = line.strip()
12         if line.startswith('#'):
13             current_year = int(line[1:])
14             max_precipitation_per_year[current_year] = 0.0
15         else:
16             values = [float(value) for value in line.split()]
17             values.insert(0, current_year)
18             rainfall_data.append(values)
19             max_precipitation_per_year[current_year] = max(
                   max_precipitation_per_year[current_year], max(
                   values[1:]))
20
21 for year, rx1day in max_precipitation_per_year.items():
22     print(f"Year: {year}, RX1day: {rx1day} mm")
```

```
Year: 1961, RX1day: 148.8 mm
Year: 1962, RX1day: 127.5 mm
Year: 1963, RX1day: 294.9 mm
Year: 1964, RX1day: 290.7 mm
Year: 1965, RX1day: 134.1 mm
Year: 1966, RX1day: 209.9 mm
Year: 1967, RX1day: 140.0 mm
Year: 1968, RX1day: 164.7 mm
Year: 1969, RX1day: 82.3 mm
Year: 1970, RX1day: 131.1 mm
Year: 1971, RX1day: 144.4 mm
Year: 1972, RX1day: 100.9 mm
Year: 1973, RX1day: 227.5 mm
Year: 1974, RX1day: 189.8 mm
Year: 1975, RX1day: 253.6 mm
Year: 1976, RX1day: 122.7 mm
Year: 1977, RX1day: 182.8 mm
Year: 1978, RX1day: 187.5 mm
Year: 1979, RX1day: 206.1 mm
Year: 1980, RX1day: 210.5 mm
Year: 1981, RX1day: 100.2 mm
Year: 1982, RX1day: 215.8 mm
Year: 1983, RX1day: 167.8 mm
Year: 1984, RX1day: 135.5 mm
Year: 1985, RX1day: 188.5 mm
Year: 1986, RX1day: 70.3 mm
Year: 1987, RX1day: 64.7 mm
Year: 1988, RX1day: 123.9 mm
Year: 1989, RX1day: 172.0 mm
```

```
Year: 1990, RX1day: 300.3 mm
Year: 1991, RX1day: 143.7 mm
Year: 1992, RX1day: 113.7 mm
Year: 1993, RX1day: 81.2 mm
Year: 1994, RX1day: 169.4 mm
Year: 1995, RX1day: 100.8 mm
Year: 1996, RX1day: 145.8 mm
Year: 1997, RX1day: 62.4 mm
Year: 1998, RX1day: 192.2 mm
Year: 1999, RX1day: 86.0 mm
Year: 2000, RX1day: 91.4 mm
Year: 2001, RX1day: 115.1 mm
Year: 2002, RX1day: 122.1 mm
Year: 2003, RX1day: 512.3 mm
Year: 2004, RX1day: 104.2 mm
Year: 2005, RX1day: 289.7 mm
Year: 2006, RX1day: 126.0 mm
Year: 2007, RX1day: 72.7 mm
Year: 2008, RX1day: 98.3 mm
Year: 2009, RX1day: 114.5 mm
Year: 2010, RX1day: 100.4 mm
Year: 2011, RX1day: 194.9 mm
Year: 2012, RX1day: 393.0 mm
Year: 2013, RX1day: 104.0 mm
Year: 2014, RX1day: 137.0 mm
Year: 2015, RX1day: 182.0 mm
Year: 2016, RX1day: 199.0 mm
Year: 2017, RX1day: 93.0 mm
Year: 2018, RX1day: 106.0 mm
Year: 2019, RX1day: 60.0 mm
```

As being seen, RX1day values ranges from lowest of 60.0 mm in 2019 to highest of 512.3 mm in 2003. This means that Thai Binh experiences a wide range of extreme precipitation events, with some years being particularly wet and others being relatively dry. There are some extremely high RX1day values being seen, such as 512.3 mm in 2003, 294.9 mm in 1963, 290.7 mm in 1964, and 393 mm in 2012. These extreme events can be associated with risks of flooding and other weather hazards.

2.12.3 Plot the distribution of RX1day.

I use the max_precipitation_per_year dictionary in the previous problem to extract all the RX1day values into a list and create a histogram plot, in which the x-axis is RX1day (mm) and y-axis is frequency.

Listing 43: Practice 12.3

```
1 import matplotlib.pyplot as plt
2
3 rx1day_values = list(max_precipitation_per_year.values())
4
```

```
plt.figure()
plt.hist(rx1day_values, bins=100, color = 'deeppink')
plt.xlabel('RX1day (mm)')
plt.ylabel('Frequency')

plt.grid(True)
plt.show()
```

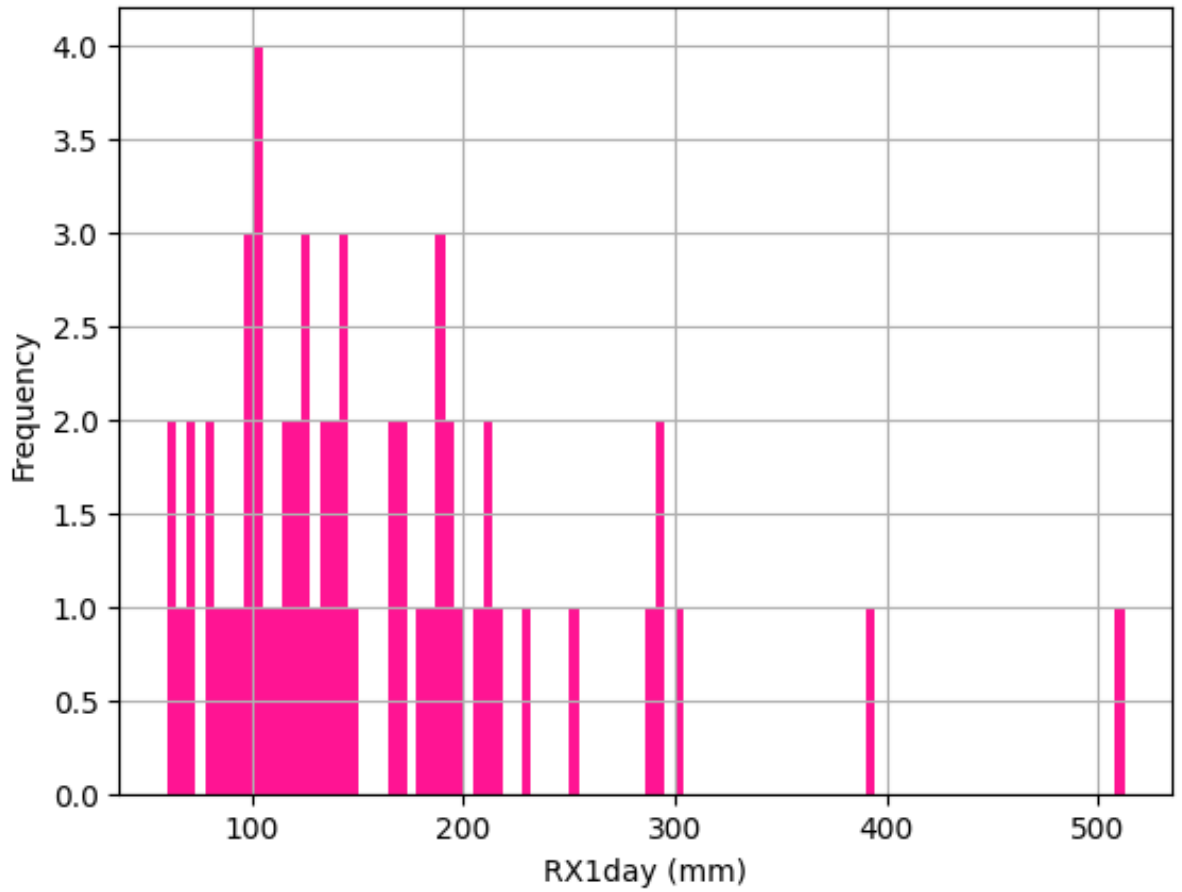

Figure 41: Frequency of RX1day from 1961 to 2019.

As being seen, the RX1day values range from around 100 mm to 500 mm, with the majority of the values below 300 mm. There is a peak around the 200-300 mm range, which means that this is the most common range of RX1day.

2.12.4 Calculate the return value of RX1day for the return period 5 years, 10years, 20 years, 100 years.

In the following code, I let the algorithm iterates through the return periods and for each return period, it calculates the corresponding rank within the sorted rx1day_values list. The rank is calculated as $\frac{n+1}{T}$ in which n is the length of the rx1day_values list and T is the return period. Then I use the interp() function to interpolate the return value corresponding to the calculated rank and store them in the return_values list and finally print them out.

```
import numpy as np

rx1day_values = list(max_precipitation_per_year.values())

rx1day_values.sort()

return_periods = [5, 10, 20, 100]
return_values = []

for return_period in return_periods:
    rank = (len(rx1day_values) + 1) / return_period

    return_value = np.interp(rank, np.arange(1, len(rx1day_values
        ) + 1), rx1day_values)
    return_values.append(return_value)

for i in range(len(return_periods)):
    print(f"Return period {return_periods[i]} years: {
        return_values[i]} mm")
```

```
Return period 5 years: 100.2 mm
Return period 10 years: 81.2 mm
Return period 20 years: 64.7 mm
Return period 100 years: 60.0 mm
```

2.12.5 Do similarly for the rainfall amounts of the heavy rainfall days (days with rainfall ≥ 50mm).

In the following code, i define a new function calculate_return_values and in the function, the algorithm filter out the heavy rainfall days (50 mm) and store the rainfall values in the heavy_rainfall list. The algorithm then loops through the return_periods and calculate the return value for each period using the np.interp() function and print them out.

Listing 44: Practice 12.5

```
def calculate_return_values(dataframe):
    heavy_rainfall = dataframe[dataframe['Rainfall'] >= 50]['
        Rainfall'].tolist()
    heavy_rainfall.sort(reverse=True)

    return_periods = [5, 10, 20, 100]
    return_values = []

    for return_period in return_periods:
        rank = (len(heavy_rainfall) + 1) / return_period
        return_value = np.interp(rank, np.arange(1, len(
            heavy_rainfall) + 1), heavy_rainfall)
        return_values.append(return_value)

    return return_values
```

```
14
15 return_values = calculate_return_values(df)
16
17 for i in range(len(return_periods)):
18     print(f"Return period {return_periods[i]} years: {
          return_values[i]:.2f} mm")
```

```
Return period 5 years: 101.22 mm
Return period 10 years: 135.59 mm
Return period 20 years: 182.76 mm
Return period 100 years: 290.69 mm
```

2.13 Daily 2m-temperature data at Thai Binh station.

Temperature at 2 meters (T2m) us used to describe the air temperature measured at a height of 2 meters above the ground. This measurement is used because it is typically above the influence of ground effects but close enough to represent the air temperature experienced by people and the environment.

This is what the file looks like when opened on my laptop.

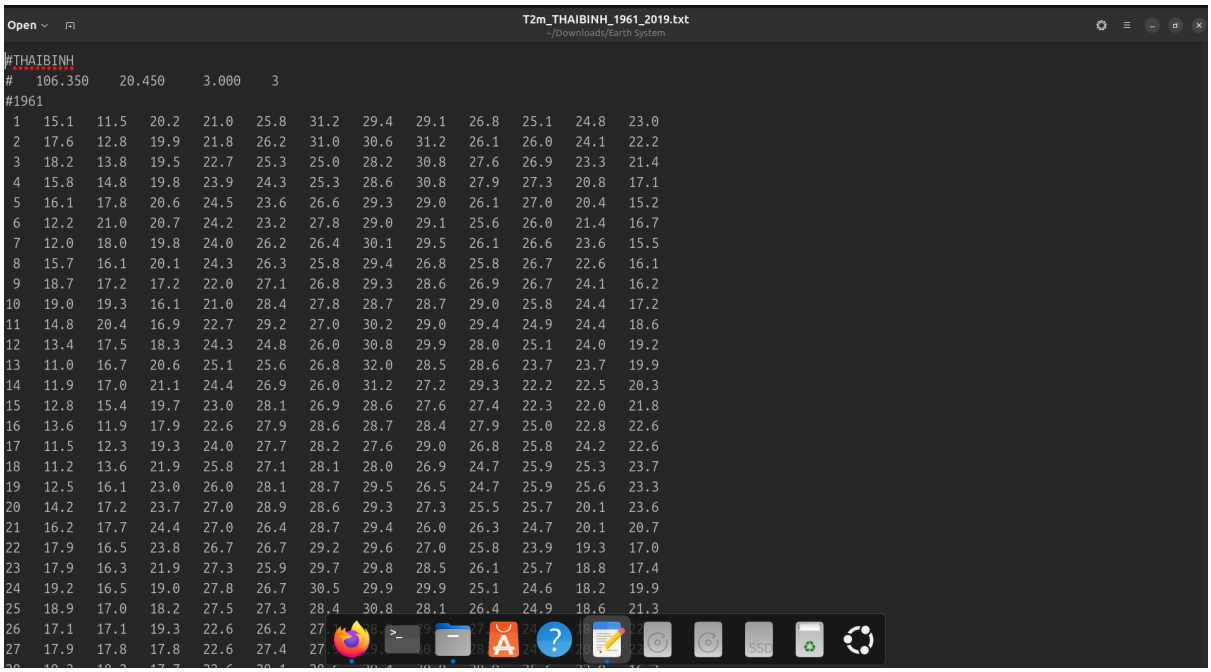

Figure 42: Dataset for practice 13.

As being seen, this file contains T2m data at a latitude of 20.450°N and a longitude of 106.350°E, which is Thai Binh. The data covers from 1961 to 2019, with the year shown at the start of each section, indicated by . After that, there are 31 rows, which indicates 31 days in a month, and each row containing 12 numeric values, which indicates 12 months in a year.

2.13.1 Plot annual mean of T2m.

In the following code, I skip the first two rows since the information is unnecessary. The algorithm then iterates through the remaining lines in the file, if the line starts with , it is stored in the current_year variable, if the line does not start with , it is splitted into individual values and convertted to floating-point numbers and appended to the data list. The first value is the day, and the remaining values are the monthly temperatures. I use the groupby() function to calculate the mean of the temperature for each year and store the results in the annual_mean variable and then plot.

Listing 45: Practice 13.1

```
1 import pandas as pd
```

```
import matplotlib.pyplot as plt

data = []
with open('T2m_THAIBINH_1961_2019.txt', 'r') as file:

    next(file)
    next(file)

    current_year = None

    for line in file:
        line = line.strip()

        if line.startswith('#'):
            current_year = int(line.split()[0][1:])
            continue
        values = [float(x) for x in line.split()]
        day = values[0]
        monthly_temps = values[1:]

        for month, temp in enumerate(monthly_temps):
            data.append([current_year, month + 1, day, temp])

df = pd.DataFrame(data, columns=['year', 'month', 'day', '
   temperature'])

annual_mean = df.groupby('year')['temperature'].mean().values

years = list(range(1961, 2020))

plt.figure()

plt.plot(years, annual_mean, color = 'deeppink')
plt.xlabel('Year')
plt.ylabel('Annual mean of T2m')

plt.grid()
plt.show()
```

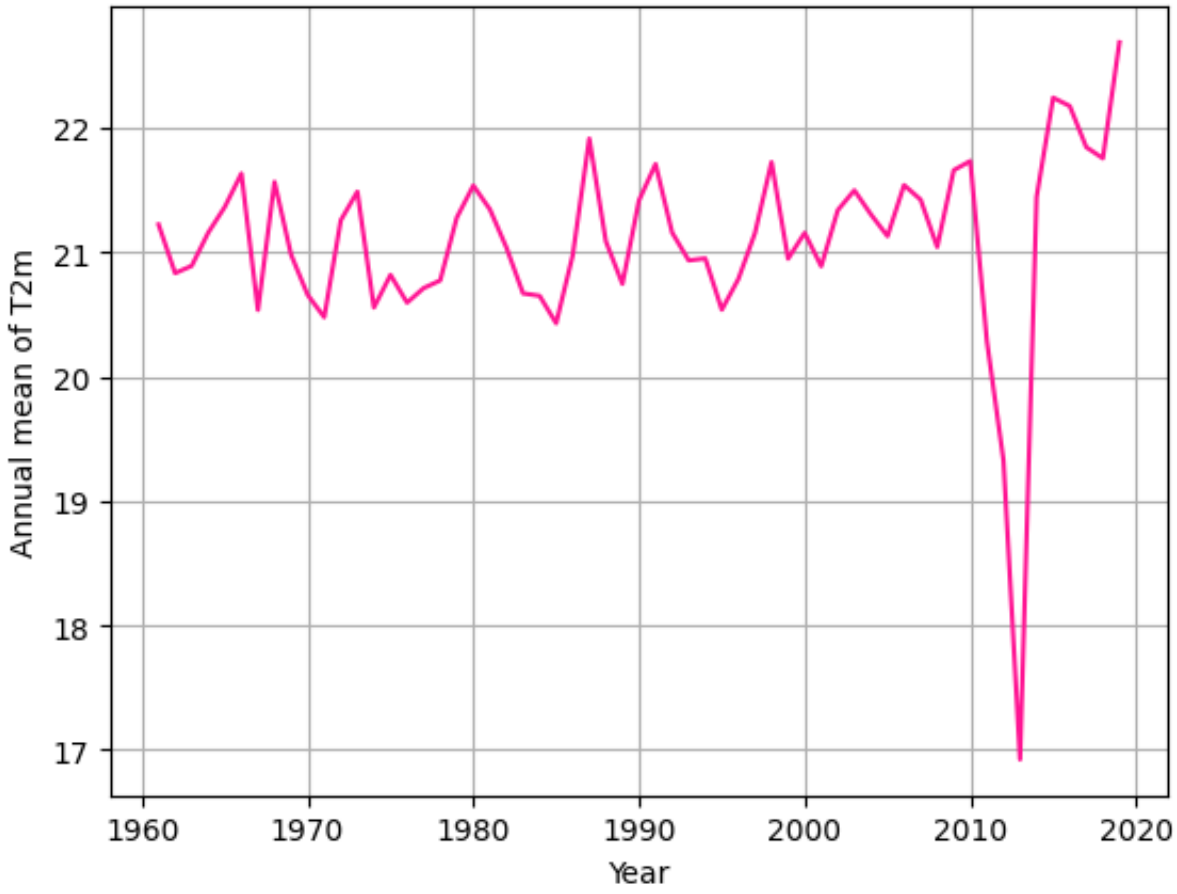

Figure 43: Annual mean of T2m from 1961 to 2019.

As being seen, from 1961 to around 2010, the temperature is quite stable, range from 20 °C to 22 °C with minor fluctuations. A significant drop in temperature is being seen around 2010 with the lowest value of 17 °C. Post-2010, there is a noticeable increase in the mean annual temperature, reaching the highest point in the graph by 2019.

2.13.2 Plot the linear trend of the annual mean of T2m.

For this problem, I use the np.polyfit() function to calculate the linear trend and plot it in the color blue, while the annual mean temperature still stay in the color pink.

Listing 46: Practice 13.2

```
slope, intercept = np.polyfit(years, annual_mean, 1)
trend_line = [slope * year + intercept for year in years]

plt.figure()
plt.plot(years, annual_mean, label='Annual Mean', color = '
   deeppink')
plt.plot(years, trend_line, label='Linear Trend', color = 'blue')
plt.xlabel('Year')
plt.ylabel('Annual mean T2m')
plt.grid()
plt.legend()
```

```
11 plt.show()
```

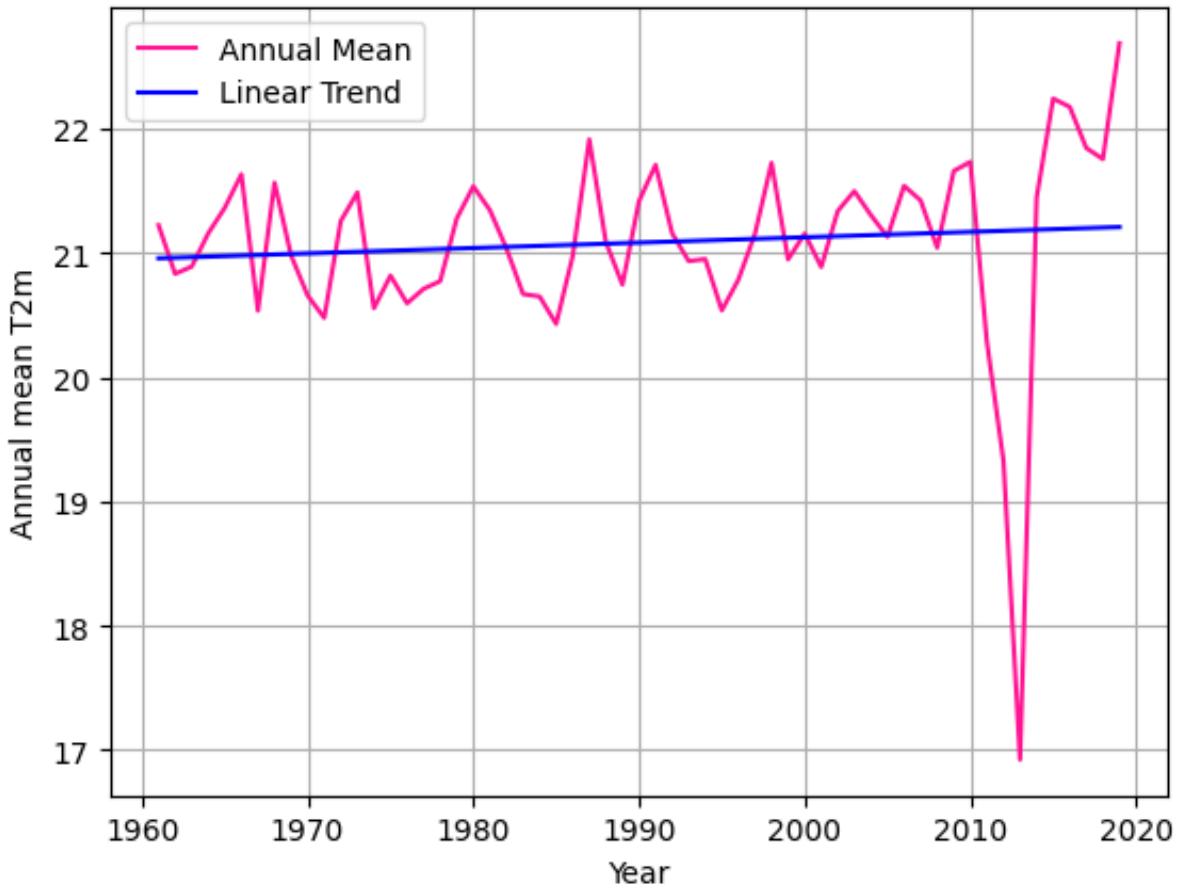

Figure 44: Linear trend of the annual mean of T2m from 1961 to 2019.

As being seen, the linear trend line shows a slight upward slope, indicating an increase in the mean annual temperature, which means there is a long term warming trend in Thai Binh. Warmer temperatures can affect crop yields, water availability, and increase the frequency of heat-related illnesses.

2.13.3 Plot the Kendall trend of the annual mean of T2m.

The Kendall trend is calculated by:

$$y = \tau(x - x_0) + y_0$$

in which y is the predicted value of the dependent variable at a given value of the independent variable x, τ is the Kendall's tau correlation coefficient, x is the independent variable, x_0 is the reference value of the independent variable, y_0 is the value of the dependent variable at the reference year x_0.

In the following code, I use the kendalltau() function to calculate it and set the starting point of the trend line at the first year's annual mean temperature value and then plot it out in the color blue.

Listing 47: Practice 13.3

```
from scipy.stats import kendalltau

tau, p_value = kendalltau(years, annual_mean)
trend_line = [tau * (year - 1961) + annual_mean[0] for year in
    years]

plt.figure()
plt.plot(years, annual_mean, label='Annual Mean Temperature',
    color = 'deeppink')
plt.plot(years, trend_line, label='Kendall Trend', color = 'blue'
    )
plt.xlabel('Year')
plt.ylabel('Annual mean T2m')
plt.grid()
plt.legend()
plt.show()
```

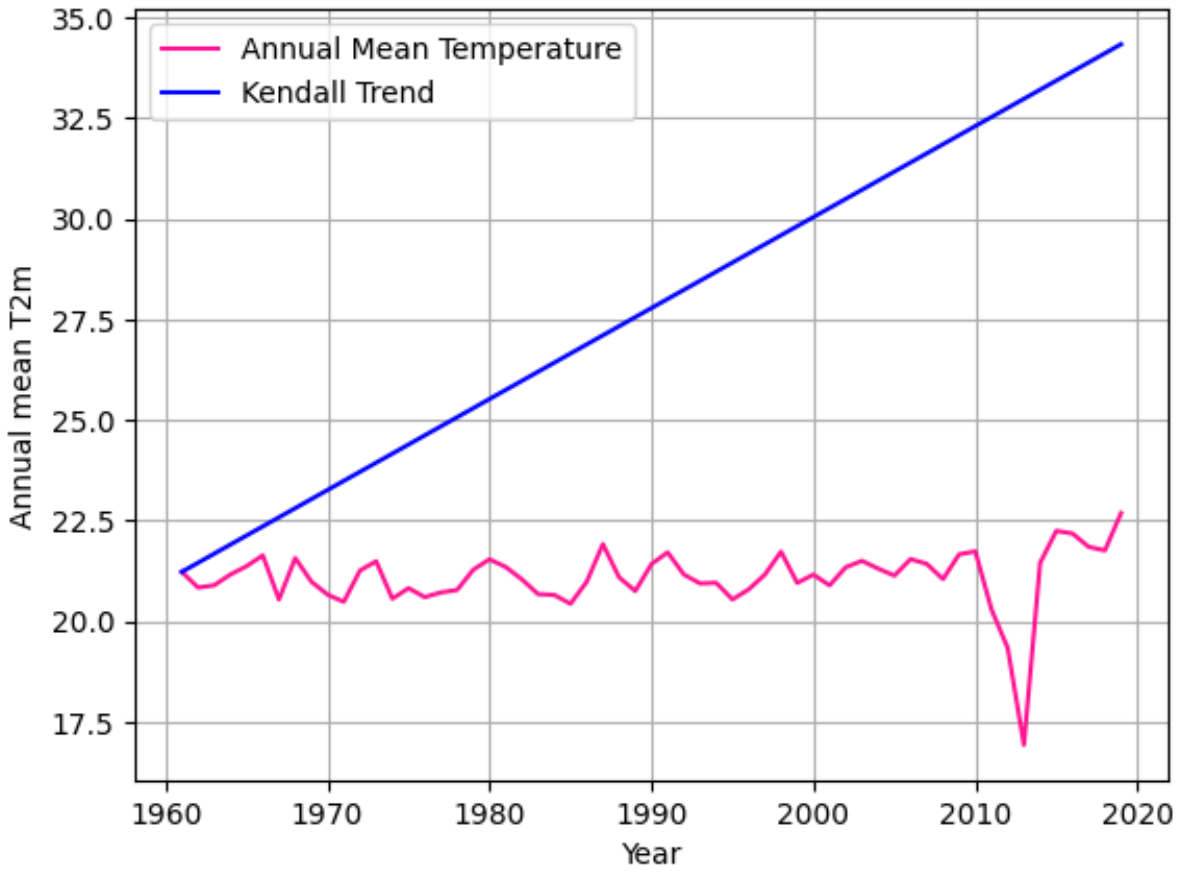

Figure 45: Kendall trend of the annual mean of T2m from 1961 to 2019.

As being seen, the slope of the Kendall trend line indicates an overall increasing trend in the annual mean temperature, suggesting a steady rise in temperature over the 60-year period. While the annual mean temperature values fluctuate above and below the Kendall trend line, it provides a more realistic representation of the long term trend in the data.

2.13.4 Plot a similar figure but for T2m in January.

For this problem, I let the algorithm selects all the temperature values from the DataFrame df where the month column is equal to 1, which represents January, and then iterates over the years and plot them out, with the x-axis being the years and the y-axis being the mean temperature.

Listing 48: Practice 13.4

```
january_temps = df[df['month'] == 1]['temperature']

january_mean_temps = [january_temps[df['year'] == year].mean()
   for year in years]

plt.figure()
plt.plot(years, january_mean_temps, color = 'deeppink')
plt.xlabel('Year')
plt.ylabel('Mean T2m')
plt.ylim(10, 35)
plt.grid()
plt.show()
```

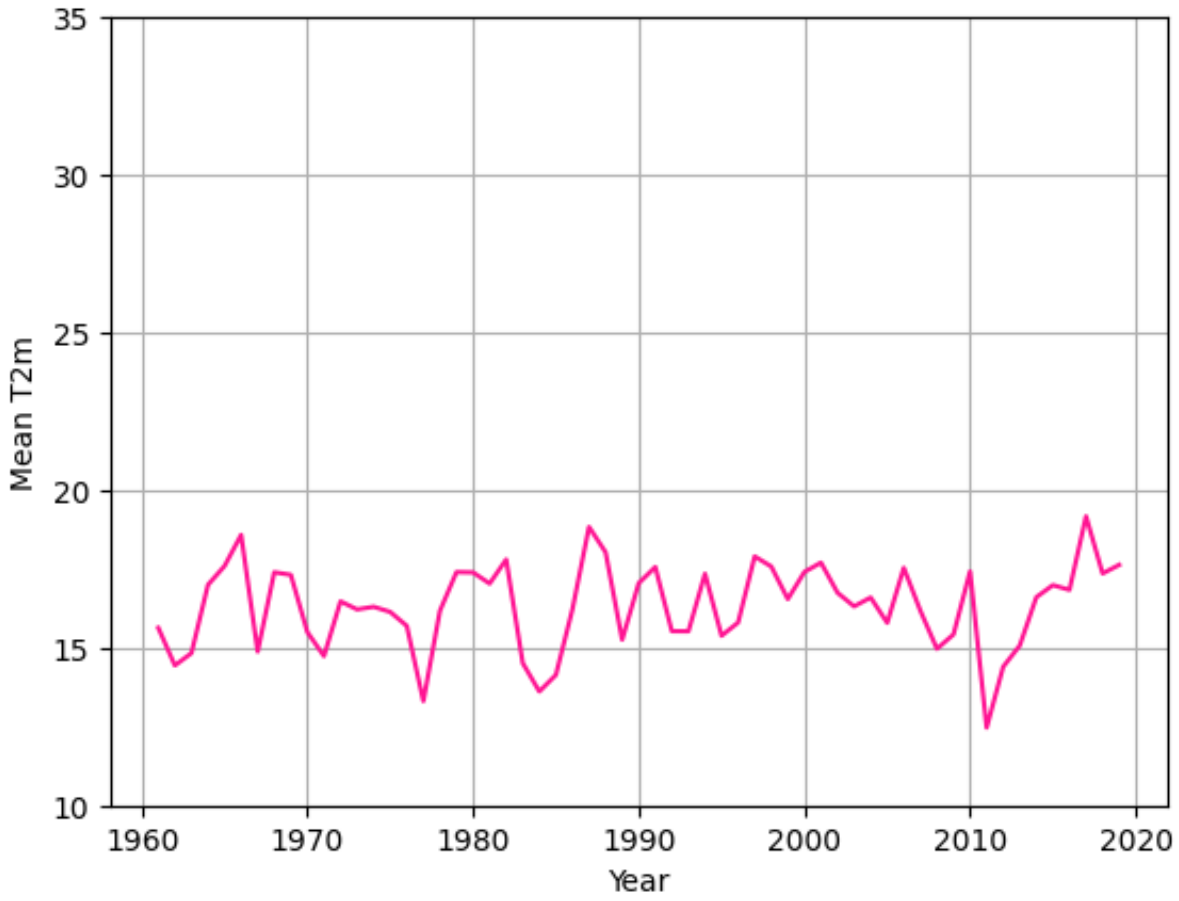

Figure 46: Annual mean of T2m in January from 1961 to 2019.

As being seen, the mean temperature in January every years range in between 10 °C

and 20 °C. The overall pattern seems to fluctuate around a stable mean with no obvious upward or downward trend. From around 2010 onwards, there seems to be a slight increase in the mean temperatures, with several years registering above the long-term average, which could mean a recent warming trend at Thai Binh in the recent years.

2.13.5 Plot a similar figure but for T2m in July.

For this problem, I use the same algorithm as the previous one but change month 1 to month 7, which represents July.

Listing 49: Practice 13.5

```
july_temps = df[df['month'] == 7]['temperature']

july_mean_temps = [july_temps[df['year'] == year].mean() for year
    in years]

plt.figure()
plt.plot(years, july_mean_temps, color = 'deeppink')
plt.xlabel('Year')
plt.ylabel('Mean T2m')
plt.ylim(10, 35)
plt.grid()
plt.show()
```

As being seen, the mean temperature in July every years range in between 28 °C and 31 °C. The trend seems relatively stable with a slight upward tendency, which means that the mean temperature in July over the past decades have remained quite stable. The stability in July temperatures is beneficial for agriculture and ecosystem stability. Compared to January when there is high variability and no clear long-term trend, the mean temperature in July seems to be more stable with a slight upward trend towards the end.

2.13.6 Plot a similar figure for rainfall.

To plot the annual mean of rainfall, I use the dataset from the previous problem which contains the rainfall data in Thai Binh from 1961 to 2019. After skipping the first two lines, the algorithm process the remaining lines and converts them in to pandas DataFrame with columns Year, Month, Day, and Rainfall. For each year, I sum all the rainfall values for that year to have the annual rainfall amount. Finally, the algorithm plot with the x-axis being the years and the y-axis being the rainfall amount in mm.

Listing 50: Pracitce 13.6

```
import pandas as pd
import matplotlib.pyplot as plt

file_path = 'R_THAIBINH_1961_2019.txt'

def read_and_parse_data(file_path):
    with open(file_path, 'r') as file:
```

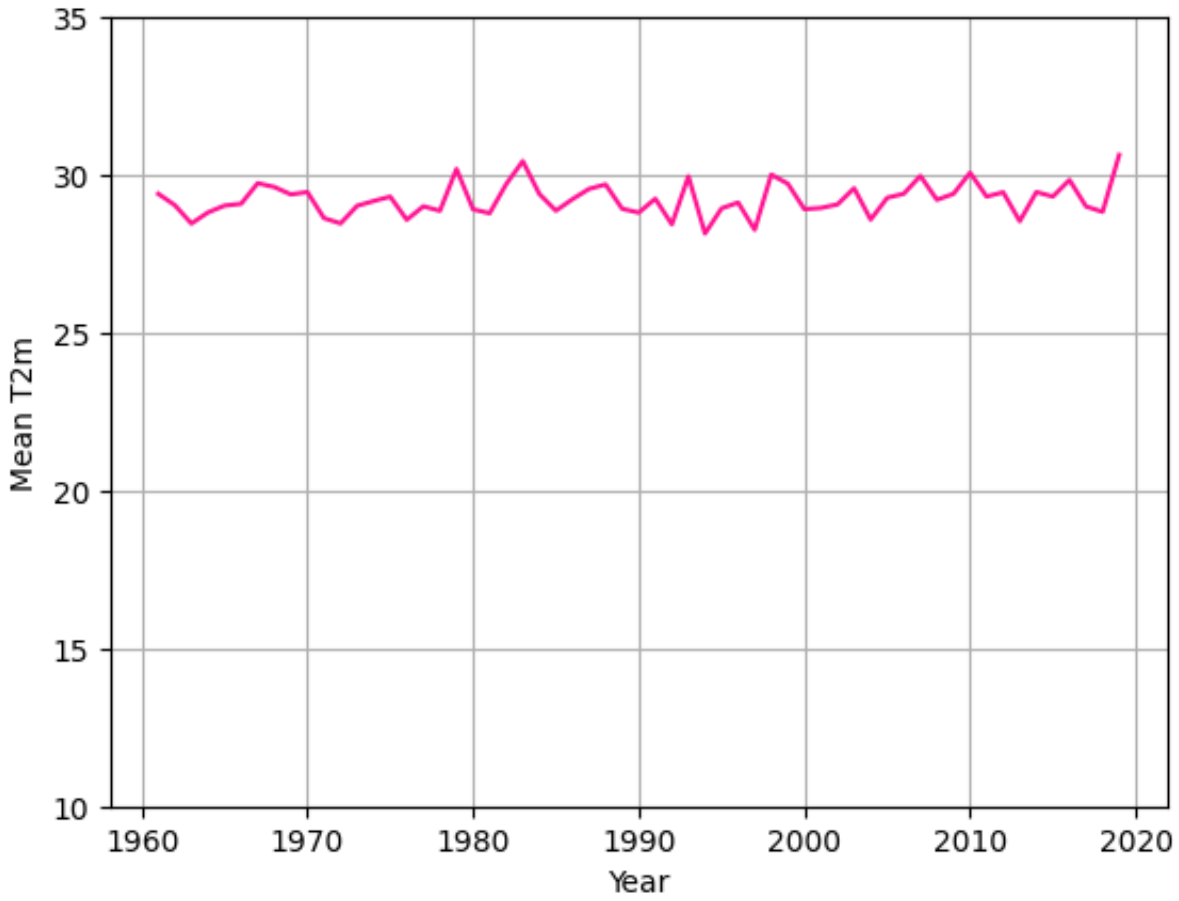

Figure 47: Annual mean of T2m in July from 1961 to 2019.

```
        lines = file.readlines()

lines = lines[2:]

data = []
current_year = None

for line in lines:
    line = line.strip()
    if line.startswith('#'):
        current_year = int(line[1:])
    elif current_year:
        values = line.split()
        day = int(values[0])
        for month in range(1, 13):
            if len(values) > month:
                value = float(values[month])
                if value == -99.0:
                    value = float('nan')
                data.append([current_year, month, day, value
                    ])

```

```
    df = pd.DataFrame(data, columns=['Year', 'Month', 'Day', '
        Rainfall'])
    return df

def plot_annual_rainfall(dataframe):
    unique_years = dataframe['Year'].unique()

    annual_rainfall = []
    for year in unique_years:
        df_year = dataframe[dataframe['Year'] == year]
        annual_rainfall.append([year, df_year['Rainfall'].sum()])

    annual_rainfall_df = pd.DataFrame(annual_rainfall, columns=['
        Year', 'Annual Rainfall'])

    plt.figure()
    plt.plot(annual_rainfall_df['Year'], annual_rainfall_df['
        Annual Rainfall'], color='deeppink')
    plt.xlabel('Year')
    plt.ylabel('Annual Rainfall (mm)')
    plt.grid()
    plt.show()

df = read_and_parse_data(file_path)
plot_annual_rainfall(df)
```

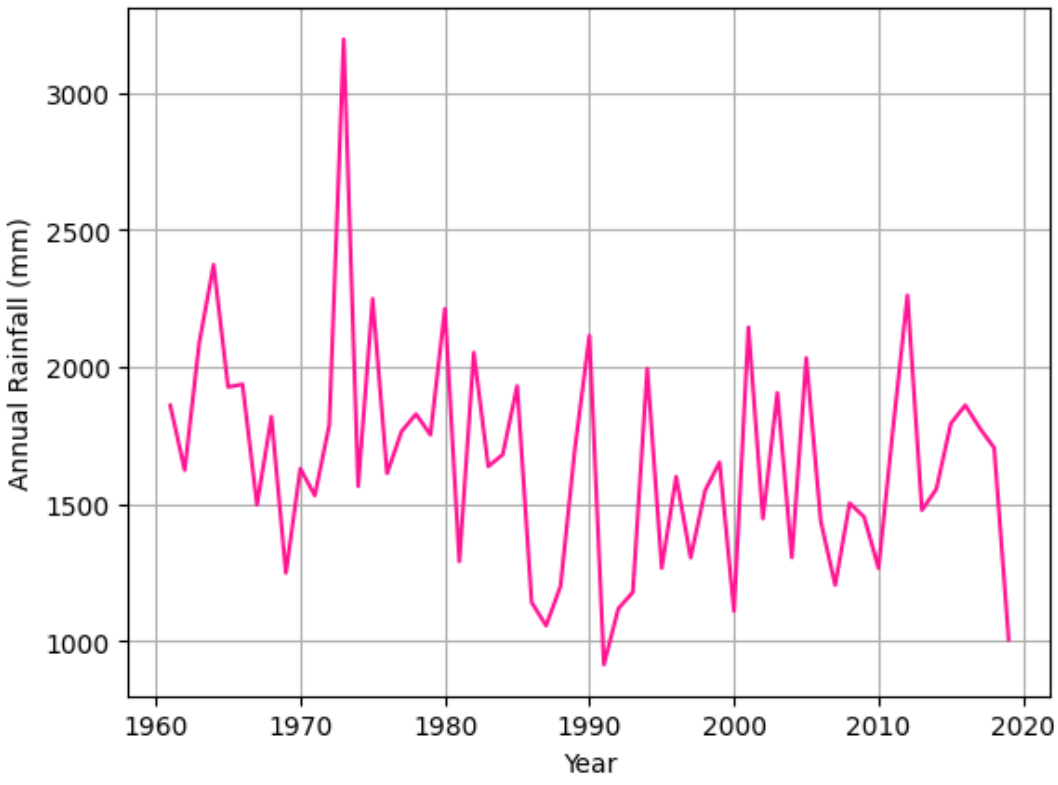

Figure 48: Annual mean of rainfall from 1961 to 2019.

As being seen, the annual mean fluctuates between below 1000 mm to over 3000 mm and there is no clear long term upward or downward trend. The peak, which represents annual rainfall with more than 3000m, is being seen in the early 1970s. On the other hand, in the early 1980s and the late 2010s, the annual rainfall amount drops to around 1000 mm. The significant peaks and troughs could be influenced by regional climatic events or human activities such as changes in land use, deforestation, etc.

3 Conclusion

This is the end of my report. To be honest, I am not really a person who is interested in weather and climate, but this course has exceeded my expectations in every way. I've learned a lot of things during solving, coding the problems and writing this report. The struggle I had while coding is way more harder than I expected, but the time I spent for it has been very valuable, it helped me realize there were a lot of things I thought I knew, but actually didn't. I've put a lot of my effort into this report, and I really enjoyed it. There are still many things I don't fully understand yet. But I'm very thankful for everything I've learned and for this course, which is one of the greatest first steps on my longer journey ahead.

Appendix

List of Figures

List of Listings

List of Tables