University of Science and Technology of Hanoi
Department of Space and Application

Introduction to Earth System

Practice Report

Basics of Earth System using Python

Author:
Nguyen Thi Yen Binh
binhnty.bi12-060@st.usth.edu.vn

Lecturer:
Assoc. Prof. Ngo Duc Thanh
ngo-duc.thanh@usth.edu.vn

October 1, 2023

Contents

List of Figures

List of Tables

1 Introduction

This report presents the solutions and explanation for all the problems and homework in the course Introduction to Earth System at USTH given by professor Ngo Duc Thanh.

Introduction to Earth System is a course designed to provide a comprehensive understanding of the intricate interactions and dynamics of the Earth System. The Earth System encompasses a vast array of interconnected components, including the lithosphere, atmosphere, hydrosphere, biosphere and even human component. By studying these interrelationship, we gain valuable insights into complex processes shaping our planet.

Understanding Earth System science allows us to comprehend the mechanisms driving natural phenomena such as climate change, weather patterns, ecosystem dynamics, and geological processes. By analyzing and interpreting data collected from various sources, we can make predictions, develop sustainable practices, and try to mitigate the potential impacts of environmental changes.

Throughout this report, we explore the applications of Python for data analysis and visualization in the context of Earth System, which truly entertain for me personally. The tools I use mostly are package `NumPy` for handling arrays and matrices, package `pandas` for data manipulation and analysis, module `pyplot` in package `matplotlib` and `scienceplots` for data visualization, module `stats` in package `scipy` for statistics-related problems, and some other packages for specific data format like `NetCDF4`. The codes are written on Jupyter Lab of Anaconda environment. The report is written in LaTeX. Not all code here are worked out by my self alone, some of them are a modified versions based on the original code of my professor Thanh.

2 Carbon Dioxide

1. Go to the following site to download the monthly CO2 data measured at Mauna Loa from March 1958:

`https://scrippsco2.ucsd.edu/data/atmospheric_co2/primary_mlo_co2_record.html`,

use Excel and/or other softwares that you can to answer the following questions:

- Where is the Mauna Loa Observatory?
- Comment about the seasonal cycle of the concentration of CO_2?
- Comment about the trend of CO2 during 1958-present?

2. Download another set of CO2 data observed at the South Pole

`https://scrippsco2.ucsd.edu/data/atmospheric_co2/spo.html`

Plot and compare with the data at Mauna Loa.

Python packages and plot style set-up

In this code, `numpy` is used for working with array-like data, `matplotlib` is used for data visualization, `pandas` is used for file-reading, and `scienceplots` is used for defining the style for the plots.

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

Listing 1: Code for importing packages

Pre-setting the style of matplotlib's figure with major, minor ticks and grid.

```
plt.style.use(['science','notebook','grid'])
matplotlib.rc('xtick', labelsize='12') #setting the size for labels of xtick
matplotlib.rc('ytick', labelsize='12') #setting the size for labels of ytick
```

Listing 2: Code for setting the style for figure

Data Reading

```
df_ml = pd.read_csv('monthly_in_situ_co2_mlo.csv',
                    skiprows=60, delimiter=',', header=None,
                    parse_dates={'Date':[0, 1]},
```

```
                    na_values=[-99.99],
                   )
df_ml = df_ml.iloc[:, [0,7,8]]
df_ml = df_ml.rename(columns={8: 'co2', 9: 'fixed co2'})
```

Listing 3: Code for reading data of CO_2 concentration in Mauna Loa Observation

```
df_sp = pd.read_csv('spo_station_data_set/monthly_merge_co2_spo.csv',
                    skiprows=60, delimiter=',', header=None,
                    parse_dates={'Date':[0, 1]},
                    na_values=[-99.99],
                   )
df_sp = df_sp.iloc[:, [0,7,8]]
df_sp = df_sp.rename(columns={8: 'co2', 9: 'fixed co2'})
```

Listing 4: Code for reading data of CO_2 concentration in South Pole Observation

2.1 Solution and Code

2.1.1 Location

The Mauna Loa Observatory is situated on the island of Hawaii, specifically on the renowned volcano, Mauna Loa, which is one of the five volcanoes forming the Big Island of Hawaii. Located in the Northern Hemisphere, the Mauna Loa Observatory can be found at an approximate latitude of 19.5362° N and a longitude of approximately 155.5763° W.

On the other hand, the South Pole Observatory is located at the geographic South Pole, situated on the continent of Antarctica. The exact latitude and longitude coordinates of the South Pole Observatory are approximately 90° S and 0° E, which lie in the Southern Hemisphere.

These two observatories, situated in contrasting hemispheres, provide valuable insights into the concentration of atmospheric CO2, the seasonal cycle related to the latitude.

2.1.2 Seasonal Cycle

```
year = 2021 # Can chose any year from 1958 to 2023
# Query data from chosen year
df_ml_sc = df_ml[(df_ml['Date'] >= f'{year}-01-01') & (df_ml['Date'] <= f'{
    year}-12-01')]
df_sp_sc = df_sp[(df_sp['Date'] >= f'{year}-01-01') & (df_sp['Date'] <= f'{
    year}-12-01')]

# Plot
fig, ax1 = plt.subplots(figsize=(12,10), sharey=True)
ax1.plot(df_ml_sc['Date'], df_ml_sc['co2'], 'o-', color = 'darkred', label =
     'Mauna Loa CO$_2$', alpha = 0.5)
```

```
ax1.plot(df_sp_sc['Date'], df_sp_sc['co2'], 'o-',color = 'navy', label = '
    South Pole CO$_2$', alpha = 0.5)

# Labels
ax1.set_xlabel("Date")
ax1.set_ylabel("CO$_2$ (ppm)")

plt.title(f"Seasonal Cycle at Mauna Loa and South Pole in {year}")
plt.legend()
plt.show()
```

Listing 5: Code for plotting the seasonal cycle in the year 2021

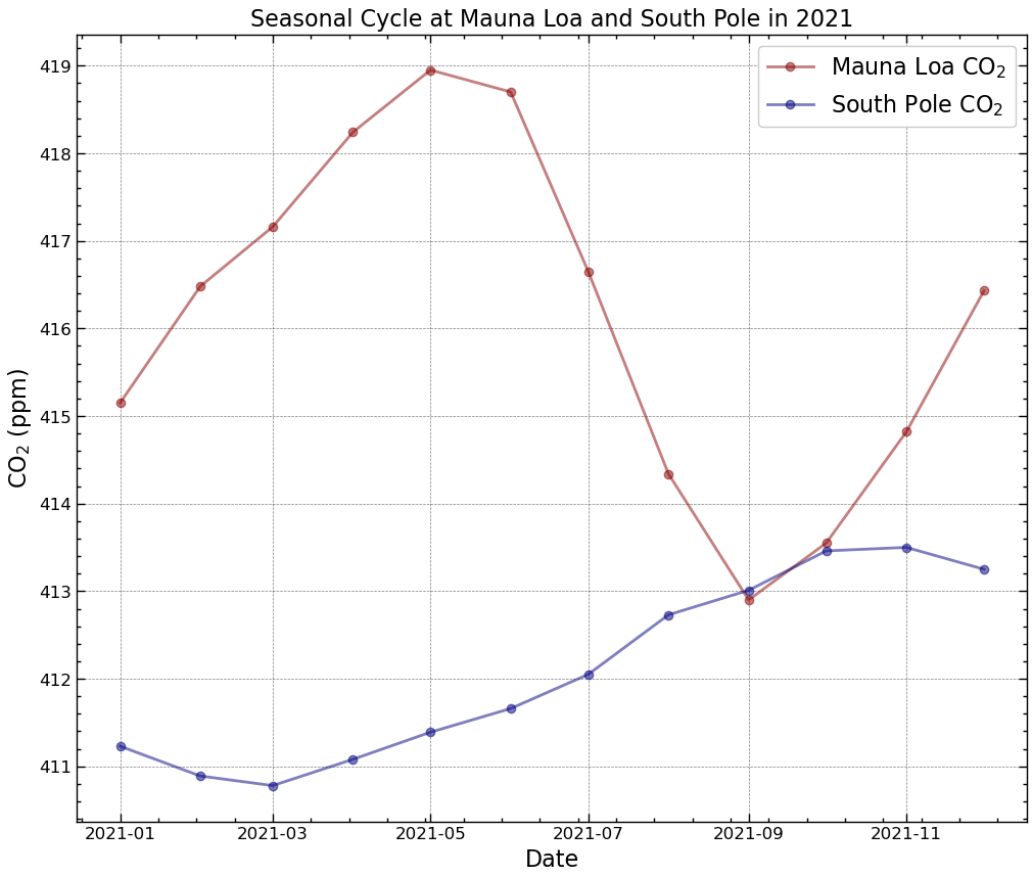

Figure 1: Seasonal Cycle of CO_2 at Mauna Loa and South Pole in 2021

Figure 1 shows the monthly records of CO_2 concentration in Mauna Loa and South Pole in the year 2021 in parts per million (ppm). The difference in the seasonal cycle at the two observator can be observed.

The Mauna Loa Observatory, located in the northern hemisphere, experiences the typical seasonal cycle in this hemisphere. During the summer months, the Earth's tilt causes the Sun's ray to strike northern hemisphere more directly, resulting in longer daylight hours and warmer temperature. In contrast, during the winter months, the Earth tilts away from the Sun leading to shorter days and colder temperatures.

On the other hand, the South Pole Observatory, located in southern hemisphere, experiences an opposite seasonal cycle, which can be seen in figure 1. That's becasue when it is summer in northern hemisphere, it is winter in the southern hemisphere, and vice versa.

The difference in the strength of seasonal cycle at different latitude are driven by the photosynthetic activity of plants. In the spring and summer, plants consume more CO_2 from the atmosphere and use it as a carbon source for growth and reproduction in photosynthesis, while in the winter, plants save energy by decreasing photosynthesis. Therefore, the regions with more plants will experience larger fluctuations which explain for the amplitude of the cycle we obtained.

2.1.3 Trend

```
fig, ax1 = plt.subplots(figsize=(12,10), sharey=True)
ax1.plot(df_ml['Date'], df_ml['co2'], color = 'darkred', label = 'Mauna Loa
    CO$_2$', alpha = 0.5)
ax1.plot(df_ml['Date'], df_ml['fixed co2'], color = 'red', linewidth = 1,
    label = "Mauna Loa CO$_2$ without seasonal cycle")

ax1.plot(df_sp['Date'], df_sp['co2'], color = 'darkblue', alpha = 0.7, label
     = 'South Pole CO$_2$')
ax1.plot(df_sp['Date'], df_sp['fixed co2'], color = 'blue', linewidth = 1,
    label = "South Pole CO$_2$ without seasonal cycle")

# Labels
ax1.set_xlabel("Date")
ax1.set_ylabel("CO$_2$ (ppm)")

ax1.legend()

# Titles
ax1.set_title("Mauna Loa and South Pole CO$_2$")
# Adjust the spacing between subplots
plt.savefig('figures/co2_ML_SP')
# plt.show()
```

Listing 6: Code for ploting the CO_2 concentration of Mauna Loa and South Pole with and without the seasonal cycle

Regarding the trend of CO_2 concentration in this period, if we do not consider about the

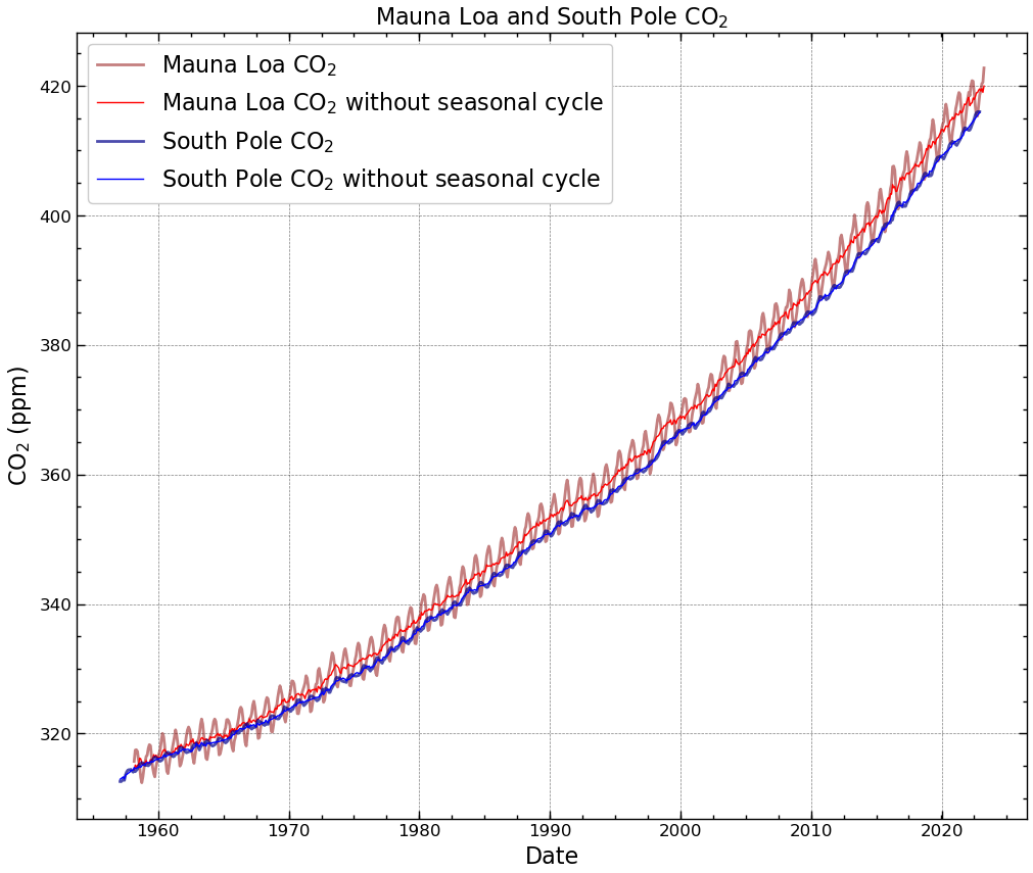

Figure 2: The CO_2 concentration at Mauna Loa Observation and South Pole Observation

seasonal cycle, the long-term trend of rising carbon dioxide are clearly shown in both Mauna Loa and South Pole Observatory, even though the CO_2 level is a little bit higher in Mauna Loa.

This long-term trend is driven mainly by the human activities; from the beginning of Industrial Revolution since 18^{th} century, fossil fuels like coal and oil containing the carbon that plants pulled out of the atmosphere through photosynthesis over millions of years are burn intensively. As we can see from the figure, since the middle of 20^{th} century the annual emissions from burning fossil fuels have increased from approximately 315 ppm in 1960 to 420 ppm in 2022.

3 Basics of Python

1. Input your name from the keyboard and print it again to the screen
2. Plot sin(x)
3. Read n real numbers from the keyboard and calculate their mean, median, variance, standard deviation
4. Enter a number n from the keyboard. Check if n is a Fibonacci number
5. Enter a number n from the keyboard. Check if n is a PRIME number

3.1 Input and output

```
print("Please input your name: ")
x = input()
print("My name is", x)
```

Listing 7: Code for taking input as string and print out the result

Note: Whatever you enter as input, the input function converts it into a string.

The result is given as follow

```
Please input your name:
 Binh
My name is Binh
```

3.2 Sin function

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

x = np.linspace(0,10)
plt.plot(x, np.sin(x))
plt.xlabel("x")
plt.ylabel("sin(x)")
#plt.savefig('figures/sin')
plt.show()
```

Listing 8: Code for plotting the sin fuction

Just the basic code for sin function.

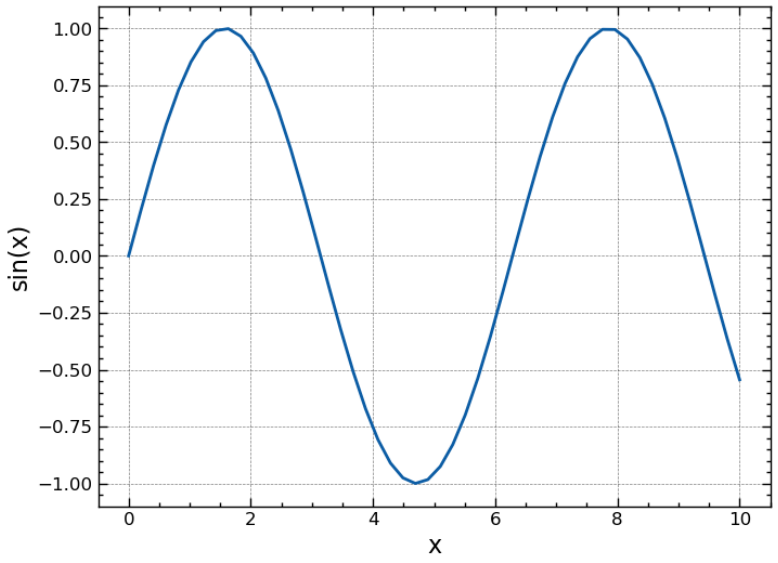

Figure 3: Sine function

3.3 Basic Statistics

The standard input function in Python always return a string, so we just simply split the string array gotten and convert all of them into desired number format, in my case I simply chose integer, but we can choose float or complex number depending on your problem.

```
numbers = input("Enter numbers separated by spaces: ")
my_array = numbers.split() # Split the number input
my_array = [int(num) for num in my_array] # Create a list of number
```

```
Enter numbers separated by spaces:  1 4 6 7
```

Mean

The mean of a set of random variables is the expectation value of that set based on the distribution of the values. The formula of mean of a dataset x_1, x_2, ..., x_n:

$$\text{mean} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + ... + x_n}{n} = E[X] = \mu \tag{1}$$

Median

The median of a dataset is the value separating the higher half from lower half when the data is arranged in order or the 'central position' of the dataset. Given an ordered dataset x_1, x_2,

..., x_n:

$$\text{median} = \begin{cases} x_{\frac{n+1}{2}} & \text{if n odd} \\ \dfrac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} & \text{if n even} \end{cases} \tag{2}$$

Variance and standard deviation

Variance and standard deviation both use to evaluate the dispersion of the data from its mean; however, standard deviation has the same dimension with the data making it more useful for comparison. The variance of a dataset x_1, x_2, ..., x_n:

$$\text{variance} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{3}$$

The standard deviation of dataset x_1, x_2, ..., x_n:

$$\text{standard deviation} = \sqrt{\text{variance}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{4}$$

We can calculate the mean, median, variance and standard deviation based solely on the definition:

```
# Calculate the mean
mean = sum(my_array) / len(my_array)

# Calculate the median
sorted_array = sorted(my_array)
mid = len(sorted_array) // 2
median = sorted_array[mid] if len(sorted_array) % 2 != 0 else (sorted_array[
    mid - 1] + sorted_array[mid]) / 2

# Calculate the variance
squared_diff = [(num - mean) ** 2 for num in my_array]
variance = sum(squared_diff) / len(my_array)

# Calculate the standard deviation
std_deviation = variance ** 0.5

print('Median: ', median)
print('Mean: ', mean)
print('Variance', variance)
print('Standard deviation', std_deviation)
```

```
Median:  5.0
Mean:  4.5
Variance 5.25
Standard deviation 2.29128784747792
```

Or simply use the package in Python like `NumPy`, both ways give the same result:

```
# Using numpy
print('Median: ', np.median(my_array))
print('Mean: ', np.mean(my_array))
print('Variance', np.var(my_array))
print('Standard deviation', np.std(my_array))
```

```
Median:  5.0
Mean:  4.5
Variance 5.25
Standard deviation 2.29128784747792
```

3.4 Fibonacci

Fibonacci number is the number belonging to a sequence with the rule that the following number is the sum of the preceding numbers:

$$x_{n+2} = x_{n+1} + x_n \tag{5}$$

The starting numbers are 0 and 1, so the sequence goes like this: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,... Therefore, to check whether a number belongs to the Fibonacci sequence or not, the most simple way is to create a list continuously add the fibonacci number starting from the ground while the sum is still lower than the number we need to check, then we check if the number is equal to last number in our list.

```
k = int(input())
x = [1, 1]
while x[-1] < k:
    p = x[-1] + x[-2]
    x.append(p)

if x[-1] == k:
    print(f'{k} is a fibonanci number.')
else:
    print(f'{k} is not a fibonanci number.')
```

```
 8
8 is a fibonanci number.

 15
15 is not a fibonanci number.
```

3.5 Prime number

Prime numbers are the natural numbers that can only divided by 1 or itself. Therefore, to check if a number is a prime number or not, we just need to divide the input number n by the number in the range from 1 to $\sqrt{n}$, and create a *count* variable to store the number of times the remainder is equal 0. If $count = 1$, then we have a prime number.

```
k = int(input())
c = 0
for i in range(1,round(np.sqrt(k))):
    if ((k%i) == 0):
        c = c+1
if c == 1:
    print(f"{k} is the prime number.")
else:
    print(f"{k} is not the prime number.")
```

```
 23
23 is the prime number.
```

```
 15
15 is not the prime number.
```

4 Energy Balance Model (EBM)

4.1 Planck function

A blackbody is an idealized physical object that can absorb the electromagnetic radiation at all wavelengths. Although a true black body does not exist in pratice, many objects like the Sun, to a good approximation, can be considered as black body over a certain ranges of wavelengths. Planck function, which formulated by the German physicist Max Planck in 1900, describes the distribution of electromagnetic radiation emitted by a black body in thermal equilibrium at a given temperature. The equation is expressed as follow:

$$B(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1} \tag{6}$$

From this equation, we can prove the Wien displacement's law by taking $\frac{\partial B(\lambda, T)}{\partial \lambda} = 0$ giving the following expression:

$$\frac{2hc^2}{\lambda^5}[-5(1 - e^{-\frac{hc}{\lambda k_B T}}) + \frac{hc}{\lambda k_B T}]\frac{e^{-\frac{hc}{\lambda k_B T}}}{(e^{-\frac{hc}{\lambda k_B T}} - 1)^2} = 0$$

Therefore:

$$\frac{\alpha}{\lambda} = 5(1 - e^{\frac{-\alpha}{\lambda}})$$

where $\alpha = \frac{hc}{k_B T}$.

Let $\frac{\alpha}{\lambda} = 5 - \epsilon$, replacing to the equation above:

$$5 - \epsilon = 5 - 5e^{-5+\epsilon}$$

We can solve the equation with the approximation (note that this solution only works because ϵ is close to 0):

$$\epsilon \approx 5e^{-5} = 0.0337$$

As $\frac{\alpha}{\lambda} = 5 - 0.0337 = 4.9663$, the Wien displacement's law can be written as:

$$\lambda_{max} = \frac{hc}{4.9663 k_B} \frac{1}{T} = \frac{2898.9 \times 10^{-6} mK}{T} \tag{7}$$

```
import astropy.constants as c
import numpy as np
import matplotlib.pyplot as plt
import scienceplots
import copy
import matplotlib

plt.style.use(['science','notebook','grid'])
matplotlib.rc('xtick', labelsize='12')
matplotlib.rc('ytick', labelsize='12')

# Constants
h = c.h.value
c_ = c.c.value
k = c.k_B.value

def planck_f(lamb, T = 6000):
    const = 2*h*c_**2/np.power(lamb, 5)
    d = np.exp(h*c_/(lamb*k*T)) - 1
    return const/d

def wien_f(T):
    lambda_max = 2.897771955e-3 / T
    return lambda_max

# Initial parameters
T = np.array([6000, 5000, 4000, 3000])
lamb = np.arange(1e-9, 3e-6, 1e-9)

# Plot
fig, ax1 = plt.subplots(1, figsize=(12,8))
for i in range(4):
    ax1.plot(lamb, planck_f(lamb, T[i]), label = f'Temperature of {T[i]} K')
T_x = np.arange(1000,6500)
ax1.plot(wien_f(T_x), planck_f(wien_f(T_x), T_x), linestyle= '--', color = '
    black', label = 'Wien\'s displacement law')
ax1.set_xlabel("$\lambda$ (m)")  # Set x-axis label with units
ax1.set_ylabel("$B(\lambda, T)$ (W m$^{-2}$ sr$^{-1}$ m$^{-1}$)")  # Set y-
    axis label with units
ax1.set_title("Planck function")

plt.legend()

#plt.savefig('figures/planck_function')
plt.show()
```

Listing 9: Code for ploting the Planck function and Wien's displacement law

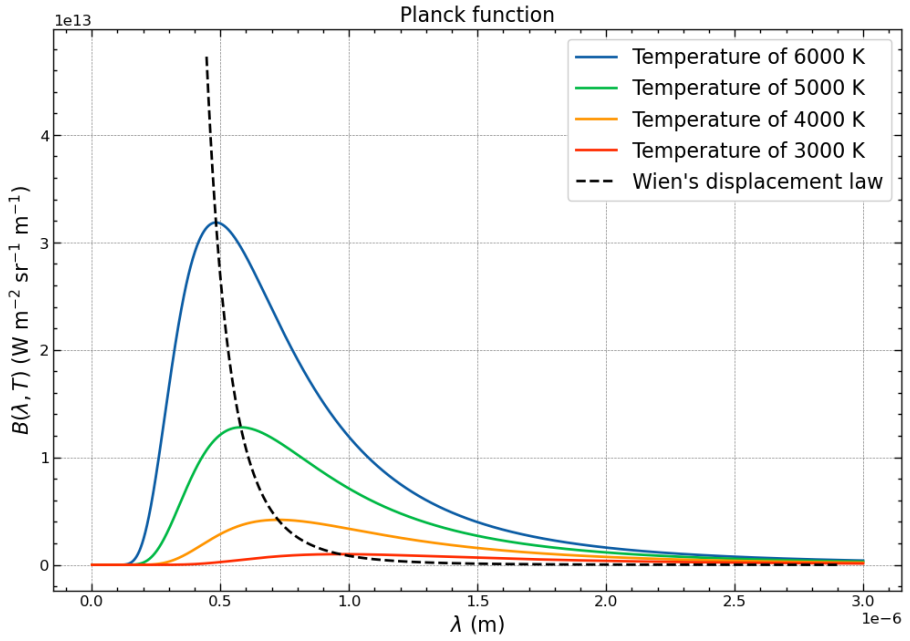

Figure 4: Planck function and Wien law

4.2 EBM

Energy Balance Models (EBM) are simple models of the Earth's climate (Budyko, 1969; Sellers, 1969). This model simply focuses on balance between the incoming energy (the Sun) and outgoing energy of the Earth, providing the information about surface temperature on the planetary scale and how it is affected by environmental factors such as solar isolation, albedo, atmosphere, etc. The most famous models are EBM 0D and EBM 1D.

4.2.1 Zero Dimensional (0-D) EBM

This is the most simple EBM Model where the Earth is treated as a mathematical point in space with zero dimension. In this 0-D EBM, we solve only for the balance between the incoming and outgoing sources of energy and radiation at the surface of the Earth.

We assume that the incoming energy from the Sun reaches a portion of the Earth that can be approximated as a flat disk with the radius R of the Earth, a part of this radiation is reflected by the clouds, aerosol and atmospheric gases with is parameterized by the albedo α:

$$E_{in} = \pi R^2 (1 - \alpha) S \tag{8}$$

With the assumption that the Earth is a black body and the energy is radiated out all over the surface of the Earth, we can express the long-wave emission of the Earth as:

$$E_{out} = 4\pi R^2 \sigma T_e^4 \tag{9}$$

where $\sigma = 5.6704 \times 10^{-8} \mathrm{W/m^2 \cdot K}$ is the Stefan-Boltzmann constant. The balance of the energy can be expressed as:

$$E_{in} = E_{out} \tag{10}$$

Solving the equation, we can obtain the expression for the mean temperature of the Earth as follow:

$$T_e = \sqrt[4]{\frac{(1 - \alpha) S}{4\sigma}} \tag{11}$$

The model can become more accurate if we account for the effect of the atmosphere. If the atmosphere of the planet contains gases that absorb more radiation then the green house effect happens which trap more heat and increase the overall temperature of the planet. We can parameterize this effect by modify the Earth as a "gray" body by taking into account the emissivity of the atmosphere ϵ:

$$\epsilon \sigma T_s^4 = \sigma T_e^4 \tag{12}$$

or by express this effect by the greenhouse increment ΔT:

$$T_e = T_s + \Delta T \tag{13}$$

Practice #1

Given:

- S: Solar constant, for the Earth, S=1370 Wm^{-2}
- α: planetary albedo (~ 0.3)
- σ: Stefan-Boltzmann constant ($5.6696 \times 10^{-8}\,\mathrm{Wm}^{-2}\mathrm{K}^4$).

Planet	Distance (AU)	Albedo	Average T_s (°C)	Atmosphere
Venus	0.723	0.76	425	95 atm, 96% CO_2
Earth	1.000	0.32	15	0.02 atm, N_2, O_2, Trace H_2O, CO_2
Mars	1.524	0.16	-50	0.02 atm, 95% CO_2
Europa	5.203	0.64	-145	no atmospher

Table 1: Planetary Properties

Write a program to:

1. Estimate the incoming solar flux for those planets
2. Estimate their blackbody equivalent temperature
3. Plot: Te versus Solar fluxes à comment on the obtained results

Solution and code

The luminosity of the Sun is a constant $L = 4\pi R^2 F$. Using the solar isolation at the Earth and the distance R of each planets provided by table 1, we can deduce the solar flux of each planet by this expression:

$$F = \frac{L}{4\pi R^2} \tag{14}$$

Have the solar flux, now we can use method in section 4.2.1 to calculate the planetary temperature and compare with the surface temperature in table 1.

```
planets = ['Venus', 'Earth', 'Mars', 'Europa']

dis = np.array([0.723, 1, 1.524, 5.203]) # in the unit of AU
albedo = np.array([0.76, 0.32, 0.16, 0.64])
temp = (np.array([425, 15, -50, -145]) + 273) # in the unit of celcius
sig = c.sigma_sb.value # Stefan-Boltzman constant
S_earth = 1370 # Solar flux of Earth

# Solar luminosity
A = S_earth * dis[1]**2
def solar_flux_function(d):
    return A / d**2
```

```
13
14 # Solar flux for each planet
15 F = [solar_flux_function(d) for d in dis]
16
17 def black_temp(albedo, S, epsilon = 1):
18     return np.power((1-albedo)*S/(4*sig*epsilon), 1/4)
19
20 # equivalent temperature of blackbody
21 T_e = []
22 for i in range(len(F)):
23     T_e.append(black_temp(albedo[i], F[i]))
24
25 # Surface temperature including greenhouse increment
26 T_s = np.array([425, 15, -50, -145]) + 273
27
28 # Greenhouse effect increment
29 gh_incre = T_s - T_e
```

Listing 10: Code for calculate equivalent temperature of planetary blackbodies

```
1 for index, temp in enumerate(T_e):
2     print('The blackbody equivalent temperature of', planets[index], f'is {
    temp:.2f}', 'K')
```

```
The blackbody equivalent temperature of Venus is 229.48 K
The blackbody equivalent temperature of Earth is 253.16 K
The blackbody equivalent temperature of Mars is 216.19 K
The blackbody equivalent temperature of Europa is 94.67 K
```

```
1 for index, temp in enumerate(T_s):
2     print('The surface temperature of', planets[index], f'is {temp:.2f}', 'K
    ')
```

```
The surface temperature of Venus is 698.00 K
The surface temperature of Earth is 288.00 K
The surface temperature of Mars is 223.00 K
The surface temperature of Europa is 128.00 K
```

```
1 for index, temp in enumerate(gh_incre):
2     print('The increment temperature of', planets[index], f'is {temp:.2f}',
    'K')
```

```
The increment temperature of Venus is 468.52 K
The increment temperature of Earth is 34.84 K
The increment temperature of Mars is 6.81 K
The increment temperature of Europa is 33.33 K
```

Without greenhouse effect:
On the x-axis, the solar flux represents the distance from the Sun to the corresponding celestial

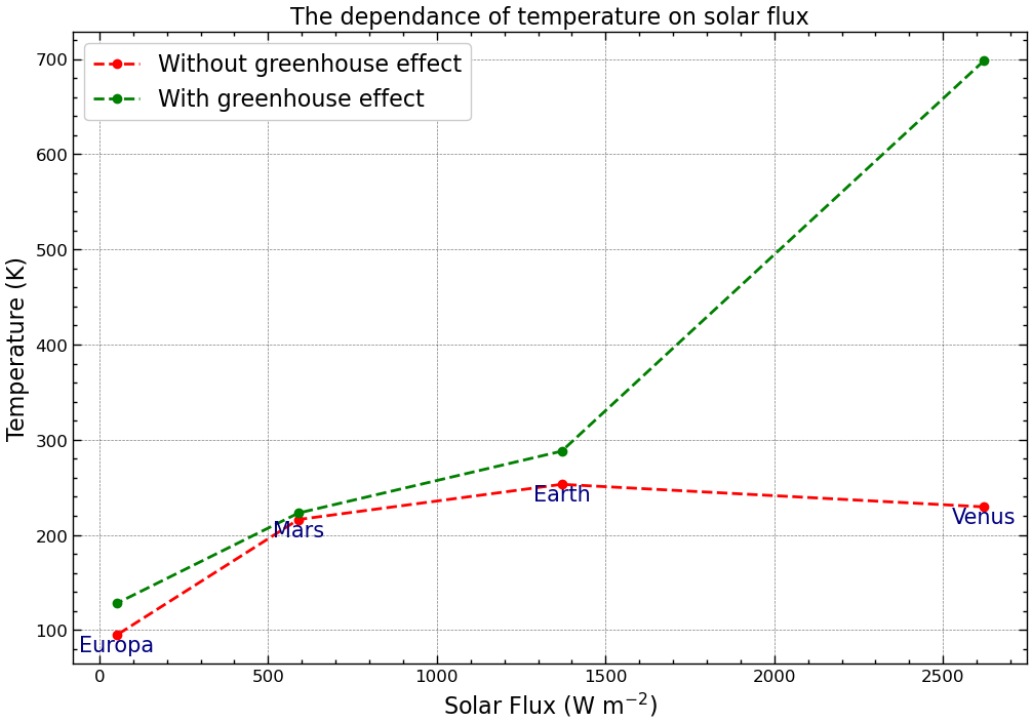

Figure 5: Equivalent temperature and Surface temperature of Europa, Mars, Earth and Venus

objects. Intuitively, one might assume that the planets closer to the Sun would be hotter. However, taking into account the planetary albedo, which measures the ability of the planet to reflect solar radiation, Earth has the highest temperature of approximately 253 K (-20°C) because of the lower albedo. The temperature of Venus is lower than Earth at around -44°C due to its almost complete cloud coverage and consequently high planetary albedo. Mars, with no cloud and farther away from Earth, has a lower temperature of around -50°C. Europa, one of Jupiter's moons, has no clouds and is far from the Sun, resulting in low temperatures suitable for the formation of ice. Hence, Europa has a relatively high albedo, leading to a positive feedback loop that further lowers its surface temperature to -179°C.

With greenhouse effect

The atmosphere plays a significant role in determining the surface temperature of a planet. An atmosphere rich in greenhouse gases such as CO_2 and CH_4 effectively traps longwave radiation emitted by the planet, leading to an increase in surface temperature. In Figure 5, we can observe the greenhouse effect nearly tripling the temperature of Venus in the model due to the fact that Venus have a thick atmosphere with 96% of CO_2. Earth's temperature calculated by EBM 0D would be too cold compared with our current condition but a greenhouse effect that contributes to an increase in temperature to 15 ^{o}C. Mars experiences a relatively small increment of approximately 6.81 K, while Europa sees an increment of around 33.33 K.

Practice #2

Given:

- The solar constant, $S = 1368$ W/m^2.
- The Stefan-Boltzmann constant, $\sigma = 5.67 \times 10^{-8}$ W/(m^2 K^4).

The outgoing longwave radiation of the planet is given by $\epsilon\sigma T_s^4$, where ϵ is the emissivity of the atmosphere and T_s is the surface temperature.

1. α=0,32 , write a python program to plot the dependence of Ts on ϵ
2. ϵ=0.66, write a python program to plot the dependence of Ts on α
3. *Write a python program to plot the dependence of Ts on both α and ϵ

Solution and Code: We know that surface temperature can be approximated in the form:

$$T_s \propto (\frac{1-\alpha}{\epsilon})^{1/4} \tag{15}$$

where α is the planetary albedo which is the reflectivity of the planet and ϵ is the emissivity of atmosphere.

If we have a high planetary albedo, it means that less and less solar radiation can reach the earth, thus when the albedo (α) approaches 1 (high albedo), the term $(1-\alpha)$ approaches 0. This results in a smaller denominator in the equation, which leads to a smaller value for the whole term $\left(\frac{1-\alpha}{\epsilon}\right)^{1/4}$. Consequently, a higher albedo causes a more rapid decrease in the surface temperature (T_s).

On the other hand, the emissivity represents the ability of atmosphere to let the longwave radiation go through. As the emissivity (ϵ) increases, the term $\left(\frac{1-\alpha}{\epsilon}\right)^{1/4}$ decreases, leading to a lower value for the whole equation. Therefore, a higher emissivity contributes to a slower decrease in the surface temperature.

```
# Surface temperature in EBM 0D
def Ts_function(epsilon, alpha, S_0 = 1368):
    T_s = ((1-alpha)*S_0/(4*sig*epsilon))**(1/4)
    return T_s

# Array of atmospheric emissivity and planetary albedo
epsilon = np.linspace(0.01,1,100)
alpha = np.linspace(0.01,1,100)

# Plot
fig, (ax, ax1) = plt.subplots(1,2,figsize=(20,6))

# sensitivity over emissivity
```

```
alpha_1 = 0.32; alpha_2 = 0.66 # Fix values

ax.plot(epsilon, Ts_function(epsilon, alpha_1), color = 'navy', label = r'$\
    alpha =$' + f' {alpha_1}')
ax.plot(epsilon, Ts_function(epsilon, alpha_2), color = 'darkred', label=r'$
    \alpha =$' + f' {alpha_2}')
ax.set_xlabel('Emissivity of the atmostphere $\epsilon$')
ax.set_ylabel('Surface Temperature T$_s$ (K)')
ax.set_title('Dependance of Surface Temperature on the emissivity')

ax.legend()

# sensitivity over albedo
epsilon_1 = 0.3; epsilon_2 = 0.7

ax1.plot(alpha, Ts_function(epsilon_1, alpha), color = 'navy', label = r'$\
    epsilon =$' + f' {epsilon_1}')
ax1.plot(alpha, Ts_function(epsilon_2, alpha), color = 'darkred', label = r'
    $\epsilon =$' + f' {epsilon_2}')
ax1.set_xlabel(r'Planetary albedo $\alpha$')
ax1.set_ylabel(r'Surface Temperature T$_s$ (K)')
ax1.set_title('Dependance of Surface Temperature on the planetary albedo')

ax1.legend()
plt.show()
```

Listing 11: Code for check the sensitivity of surface temperature on epsilon and alpha respectively

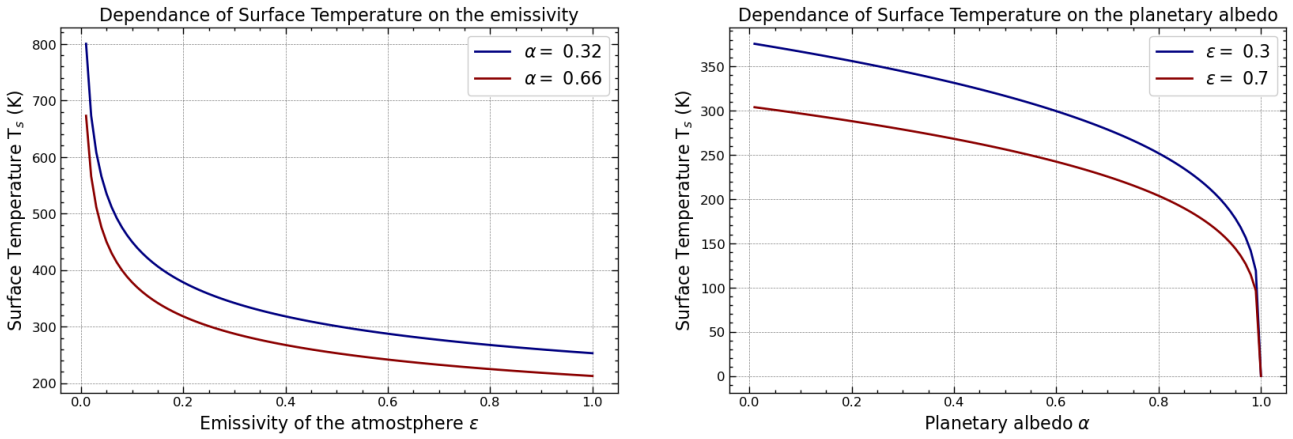

Figure 6: Sensitivity of Surface temperature on individual components

```
from matplotlib import cm
from matplotlib.ticker import LinearLocator

```

```
epsilon = np.linspace(0.01,1,100)
alpha = np.linspace(0.01,1,100)
epsilon_mesh, alpha_mesh = np.meshgrid(epsilon, alpha)
Ts = Ts_function(epsilon_mesh, alpha_mesh)

fig = plt.figure(figsize = (10,8))
ax = fig.add_subplot(111, projection = '3d')

# Plot the surface.
surf = ax.plot_surface(epsilon_mesh, alpha_mesh, Ts, cmap=cm.jet, alpha =
    0.7)

# Set labels and title
ax.set_ylabel(r'$\alpha$')
ax.set_xlabel(r'$\epsilon$')
ax.set_zlabel('Surface Temperature (Ts)')
ax.set_title('Surface Temperature as a function of Alpha and Epsilon')

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=12)

plt.show()
```

Listing 12: Code for checking the sensitivity of surface temperature on both epsilon and alpha

As we can guess, the highest surface temperature occurs when both planetary albedo and atmospheric emissivity are close to zero, and vice versa. At the lower range of temperature the planetary albedo is dominant, if the albedo is too high the temperature will be really low. On the other hand, the emissivity of the atmosphere. The result is shown in figure 7.

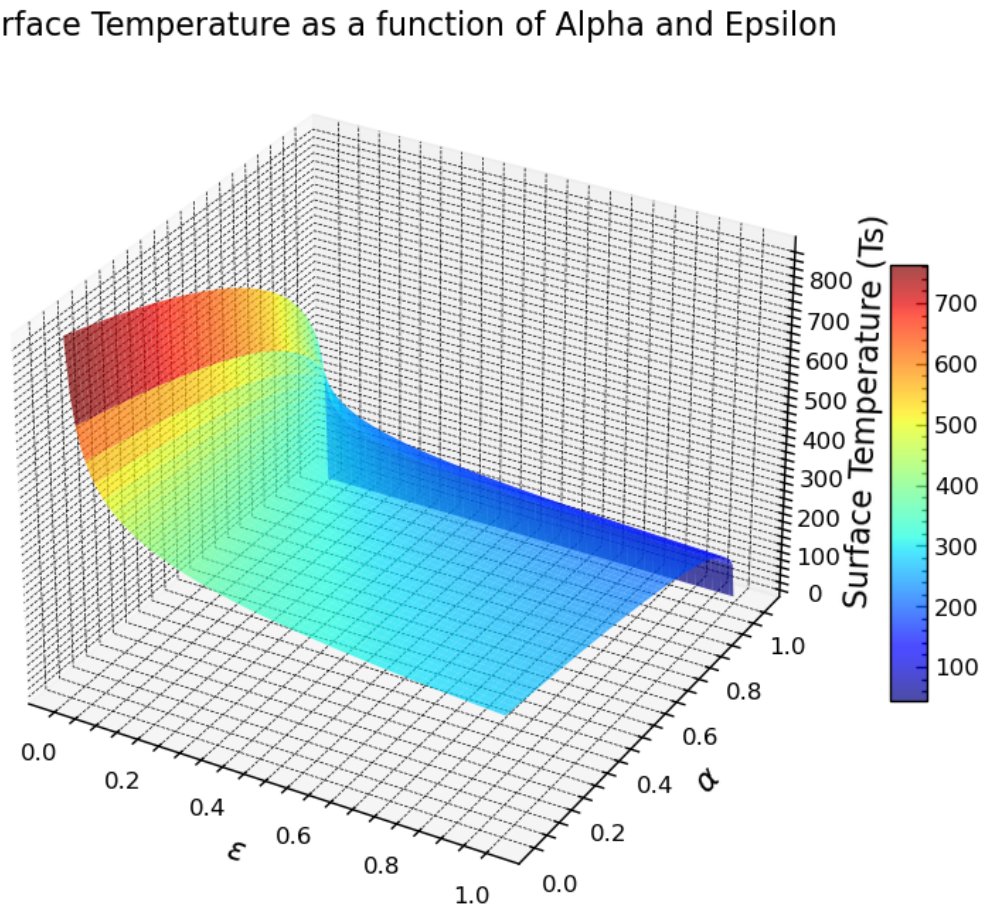

Figure 7: Sensitivity of Surface temperature on both components

Practice #3

Planetary albedo depends on surface temperature. Let's assume the following relations:

$$\alpha(T_i) = \begin{cases} \alpha_{ice} & \text{if } T_i \leq T_{ice} \\ \alpha_{land} & \text{if } T_i > T_{land} \\ \alpha_{ice} + (\alpha_{land} - \alpha_{ice}) \times \dfrac{T_i - T_{ice}}{T_{land} - T_{ice}} & \text{if } T_{ice} < T \leq T_{land} \end{cases} \tag{16}$$

where $\alpha_{ice} = 0.6$; $\alpha_{land} = 0.32$, $T_{ice} = -10°C$ is the temperature where the Earth becomes a Snowball; $T_{land} = 10°C$ is the temperature where the Earth remains in the nowadays state.

1. Write a program to estimate the equilibrium temperature (T_s) 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 (S_0) remains unchanged at $S_0 = 1368\,\mathrm{W/m^2}$.

2. Write a program to estimate the equilibrium surface temperature (T_s) with a changing solar flux S (S/S_0 varies 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.

Solution and Code:

This problem reflects the practical situation that when the temperature is low enough for formation of the ice, the global albedo increases, because while water has relatively low albedo, ice is opposite. There is a temperature which is warm enough so that no ice forms and in between this range the albedo is approximated by a linear relation.

In this case, while the albedo depends on the temperature, the temperature also depends on albedo, forming a feedback loop until the equilibrium temperature is reached, in which the temperature and albedo stabilize and , at this point, the energy balance is achieved.

In this problem, we use the equation 11 for EBM 0D model to examine the ice-albedo feedback mechanism with different initial temperature. The initial temperature defines the initial states of the presence of ice on the planet, subsequently affect the albedo and temperature.

```
# Initial Condition
alpha_ice = 0.6
alpha_land = 0.32
epsilon = 0.62
T_ice = -10
T_land = 10
S0 = 1368

# planetary albedo
def alpha_condition(Ts, T_ice = T_ice, T_land = T_land, alpha_ice =
    alpha_ice, alpha_land = alpha_land):
     if Ts < T_ice:
         alpha = alpha_ice
     elif Ts > T_land:
         alpha = alpha_land
     else:
         alpha = alpha_ice + (alpha_land - alpha_ice)*(Ts - T_ice)/(T_land -
    T_ice)
     return alpha

# Surface temperature function in EBM 0D
def Ts_function(epsilon, alpha, S_0 = 1368):
     T_s = ((1-alpha)*S_0/(4*sig*epsilon))**(1/4)
     return T_s

# equilibrium temperature function
def T_function(T, S = S0):
     k = 0
     while k == 0:
```

```
28          T_ = T
29          T = Ts_function(epsilon, alpha_condition(T), S) - 273
30          if np.abs(T - T_) <= 0.001:
31              k = 1
32      return T
33
34 T = np.arange(3,14,0.5)[::-1] # initial temperature range of interest
35 T_eq = [T_function(t) for t in T] # Equilibrium temperature
36
37 # Plot
38 fig, ax = plt.subplots()
39 ax.plot(T, T_eq, 'ro')
40 ax.set_xlabel("intial temperature")
41 ax.set_ylabel("Equilibrium temperature")
42 ax.set_title("Dependance of equilibrium temperature on initial temperature")
43 plt.show()
```

Listing 13: Code for obtaining the equilibrium temperature based on EBM 0D model

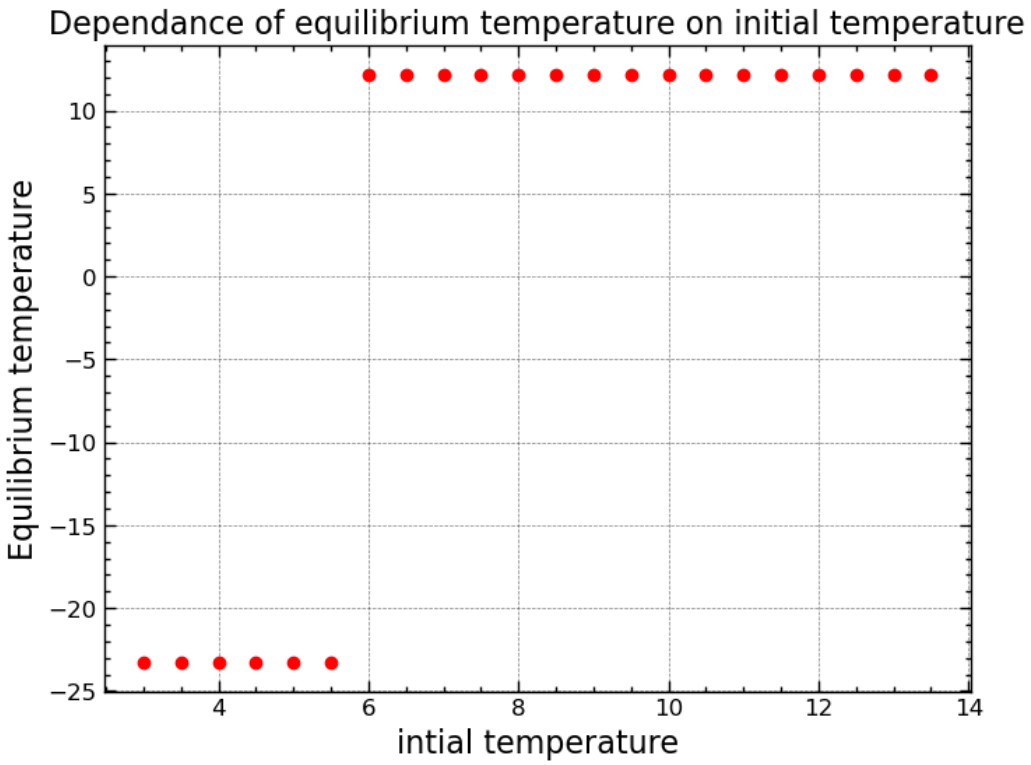

Figure 8: Equilibrium temperature of EBM 0D Model

The results we obtained shown in figure 8 is interesting that when the inital temperature lower than 6 °C, the system reaches the equilibrium quite low at -23.24 °C, while the higher

temperature gives the equilibrium temperature at 12.20 °C. This sudden 'jump' of about 35 °C is so called *tipping point*, a kind of critical point where the system undergoes a notable shift in its behavior. Tipping points are usually arised in a non-linear dynamics and indicate a abrupt changes or transitions in the system. Personally, I think this phenomenon is somehow associated with the butterfly effect, which suggests that small changes in initial conditions can result in disproportionate and potentially irreversible shifts in the system's behavior.

This phenomenon is concerning because if we exceed a certain threshold, the temperature can increase dramatically, leading to severe consequences.

Now, what happen if the solar flux varies?

```
# Solar flux range and initial temperature
S = np.arange(0.2, 2.05, 0.05)*S0
T_ini = np.arange(-15, 15)[::-1]

# Equilibrium temperature based on solar flux and initial temperaute
T_eq_So = np.array([[T_function(t, s) for t in T_ini] for s in S])

# Plot
T_ini_mesh, S_mesh = np.meshgrid(T_ini, S)
# T_eq_So_mesh = T_function(T_ini_mesh, S_mesh)

fig = plt.figure(figsize = (12,10))
ax = fig.add_subplot(111, projection = '3d')

# Plot the surface.
surf = ax.plot_surface(T_ini_mesh, S_mesh, T_eq_So, cmap=cm.jet)

# Set labels and title
ax.set_xlabel('Inital Temperature (K)')
ax.set_ylabel(r'Solar Flux (W m$^{-2}$)')
ax.set_zlabel('Equilibrium Temperature (K)')
ax.set_title('Equilibrium Temperature as a function of inital temperature
    and solar flux')

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=12)

plt.show()
```

Listing 14: Code for plotting the equilibrium temperature of EBM 0D with variation of solar flux

```
fig, ax = plt.subplots(figsize=(20,25))
for i in range(T_eq_So.shape[0]):
    ax.plot(T_ini, T_eq_So[i, :], 'o-', label = f'S$_0$ = {S[i]:.1f}')
ax.set_xlabel("intial temperature")
```

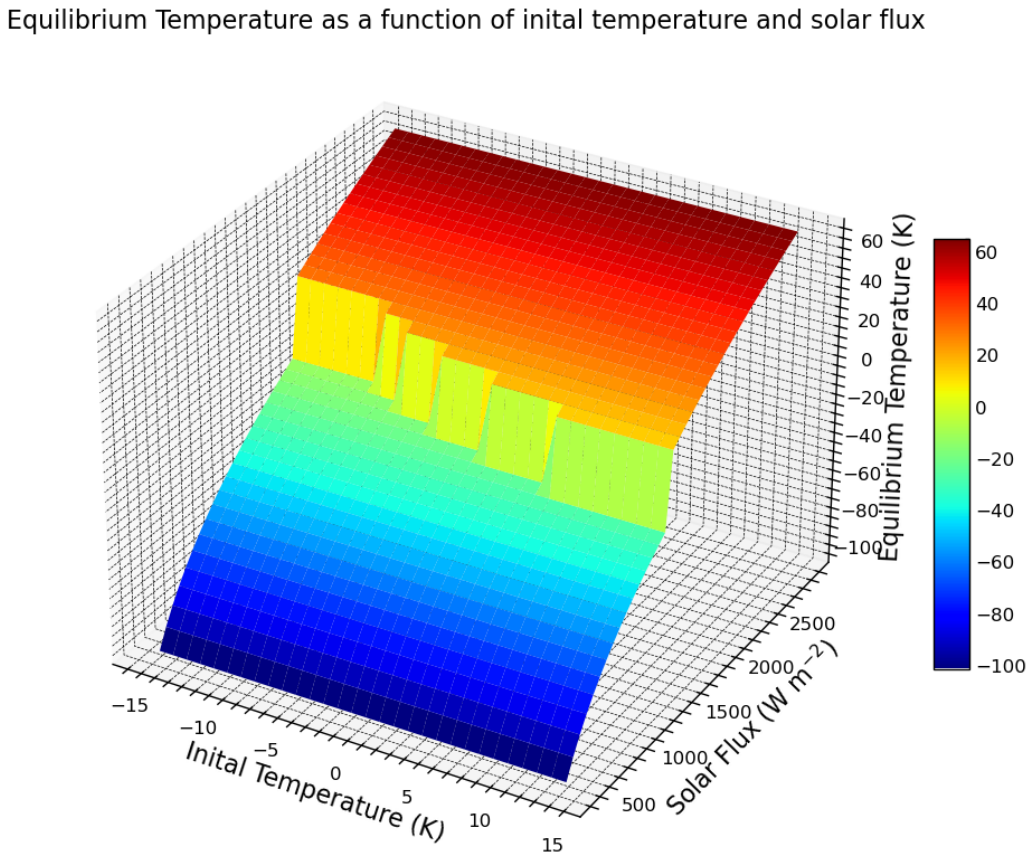

Figure 9: Equilibrium Temperature of EBM 0D with the variation of solar flux

```
ax.set_ylabel("Equilibrium temperature")
plt.legend()

plt.show()
```

From Figure 9 and Figure 10, we observe that tipping points occur within a specific range of initial temperatures (-15°C to 15°C) and solar flux values (approximately 1300 W/m^2 to 1600 W/m^2). It is evident that the occurrence of tipping points is influenced by both the initial temperature and solar flux.

Additionally, when analyzing the effect of solar flux alone in Figure 9, we observe that tipping points eventually occur regardless of the initial temperature, although the presence of a lower initial temperature delays this process.

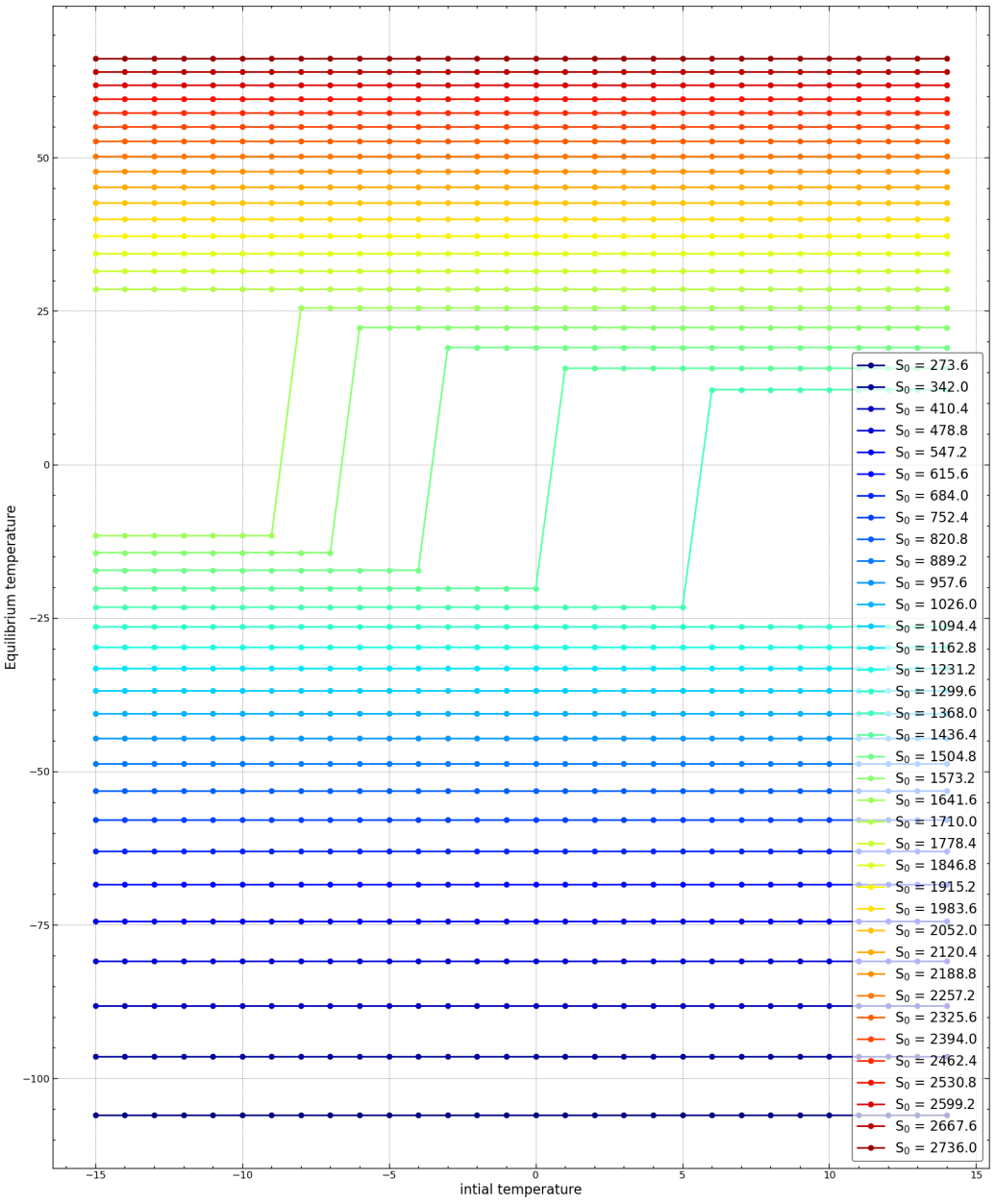

Figure 10: Equilibrium temperature with variation of solar flux on two dimension

4.2.2 One dimensional (1D) EBM

In this EBM model, we include one spatial dimension that is to say, the Earth's surface is divided into different latitudinal bands with different temperature, each referred to as by the index i. The balance model between the incoming and outgoing radiation is governed by the

following equation:

$$S_i(1 - \alpha(T_i)) = R(T_i) + \Delta F_{transport}(T_i) \tag{17}$$

where T_i is the mean surface temperature in the band i (^{o}C), S_i is the mean annual radiation incident (Wm^{-2}), α_i is the albedo for the latitude band i, $R(T_i)$ is the long-wave energy emitted by the Earth and $\Delta F_{transport}$ is the energy exchanged between band i and the surrounding band.

The emitted energy $R(T_i)$ can be evaluated using Stefan-Boltzmann equation for black body radiation which is proportional to T^4. Since the temperature range of interest is relatively small (between 250 and 300 K), $R(T_i)$ can be approximated using a linear relation function of the temperature T$_i$, we can prove this using Taylor series expansion around the global mean temperature T_0:

$$R(T) = R(T_0) + (T - T_0)R'(T_0) + (T - T_0)^2 R''(T_0) + ...$$

Since the temperature range is small the high-order of the component $T - T_0$ can be approximated to 0 which leave us the expression:

$$R(T) = R(T_0) + (T - T_0)R'(T_0)$$

Now, let's substitute the values of $R(T_0)$ and $R'(T_0)$ using the original equation:

$$R(T) = \sigma T_0^4 + 4\sigma T_0^3(T - T_0)$$

Simplifying the expression and grouping the term:

$$R(T) = -3\sigma T_0^4 + 4\sigma T_0^3 T$$

Now we can rewrite the equation in this form:

$$R(T_i) = A + BT_i \tag{18}$$

where A (W m^{-2}) and B (W m^{-2} ^{o}C^{-1}) are empirically determined constants accounting for greenhouse effect of clouds, vater vapour and CO_2.

The rate of transport of energy $\Delta F_{transport}(T_i)$ is defined by the difference between the zonal temperature T_i and the global mean temperature T_0:

$$F(T_i) = k_i(T_i - T_0) \tag{19}$$

where k_i is the transport coefficient (W m^{-2} ^{o}C^{-1}).

Combining three equation 17, 18 and 19 gives:

$$T_i = \frac{S_i(1 - \alpha(T_i)) + k_i T_0 - a}{b + k_i} \tag{20}$$

The condition for α_i is proposed in this following expression:

$$\alpha(T_i) = \begin{cases} \alpha_{ice} & \text{if } T_i \leq T_{ice} \\ \alpha_{land} & \text{if } T_i > T_{land} \\ \alpha_{ice} \times \dfrac{T_{land} - T_i}{T_{land} - T_{ice}} + \alpha_{land} \times \dfrac{T_i - T_{ice}}{T_{land} - T_{ice}} & \text{if } T_{ice} < T \leq T_{land} \end{cases} \tag{21}$$

where T_{land} is the temperature that no ice cover on the planet and T_{ice} is the temperature that all ice cover on the planet.

Practice

Given the initial conditions and constants for the EBM-1D model:

1. Make a complete code for the EBM-1D.
2. Estimate the value of the solar flux (S) so that the Earth will be entirely covered by ice.
3. The parameter K could vary. For example, Budyko (1969) set $K_t = 3.81$, while Warren and Schneider (1979) set $K_t = 3.74$. Investigate the sensitivity of the model with K.
4. The parameters A and B can vary. For example, Budyko (1969) set $A = 202\,\mathrm{W/m}$ and $B = 1.45\,\mathrm{W/m^2\,^\circ C}$, while Cess (1976) set $A = 212\,\mathrm{W/m}$ and $B = 1.6\,\mathrm{W/m^2\,^\circ C}$. Investigate the sensitivity of the model with B. What are the physical meanings behind these parameters?

Practice #1: EBM 1D

The initial portion of the code, up to line 27, was provided by Professor Thanh, while the remaining work involved completing the implementation of the EBM 1D model. In this model, the mean albedo is calculated as the average of all zonal albedo values. By using the mean albedo, we approximate the mean temperature using the EBM 0D equation (refer to equation 11). The initial temperatures are set to the global mean temperature obtained from the previous calculations.

In the EBM 1D model (refer to equation 17), with a constant solar flux, the local albedo and global temperature interact to determine the local temperature of each latitudinal zone. This local temperature, in turn, affects the local albedo (condition 21), leading to a feedback loop that influences the global temperature. The program continues to iterate until an energy balance is achieved, ensuring that the model's calculations align with the observed energy balance of the Earth.

```
1 # Constants
2 kt = 3.81; A = 204; B = 2.17
3
```

```
# EBM 1D
def EBM_1d(Si, alpha_i, T_global, kt = kt, A = A, B = B):
    R_in = Si*(1 - alpha_i)
    Ti = (R_in + kt*T_global - A)/(B + kt)
    return Ti

ny = 9 #number of latitude bands
temp_c1 = -10 # Completely covered by ice
temp_c2 = 0 # No ice cover
alb_ice_free = np.array([0.23, 0.24, 0.25, 0.29, 0.35, 0.40, 0.46, 0.50,
   0.50])
alb_ice = 0.62

So = 1368
sol_frac = np.array([0.30475 ,0.29725 ,0.28 ,0.25525 ,0.223 ,0.1925 ,0.156
   ,0.13275 ,0.125])
sol_flux=So*sol_frac #Solar flux for bands

# Sum area
dp = np.pi / (2*ny)
phi = np.array([(0.5 + j - 1) * dp for j in range(0,ny)])
area = np.sin(phi + dp/2) - np.sin(phi - dp/2)
sum_area = np.sum(area)

# Initial albedo
alb = np.copy(alb_ice_free) # Define inital albedo with the assumption that
   the earth is not covered by ice

# Global mean temperature
alb_mean = np.mean(alb)/sum_area
T_mean = (So*(1-alb_mean)/4 - A)/B

# Equilibrium temperature
temp = np.ones(ny)*T_mean
temp_pre = np.ones(ny)*T_mean
temp_mean_pre = T_mean
loop = 0

stable_check = 1
epsilon = 0.001
while abs(stable_check) > epsilon:
    stable_check = 0
    temp = EBM_1d(sol_flux, alb, temp_mean_pre)
    if np.any(temp > temp_c2):
        alb[temp > temp_c2] = alb_ice_free[temp > temp_c2]
    if np.any(temp <= temp_c1):
        alb[temp <= temp_c1] = alb_ice
    else:
        alb[(temp <= temp_c2) & (temp > temp_c1)] = alb_ice*(temp_c2 - temp
```

```
    [(temp <= temp_c2) & (temp > temp_c1)])/(temp_c2 - temp_c1) +
    alb_ice_free[(temp <= temp_c2) & (temp > temp_c1)]*(temp[(temp <= temp_c2
    ) & (temp > temp_c1)] - temp_c1)/(temp_c2 - temp_c1)
     T_mean = np.sum(temp * area)
     stable_check = np.sum(temp - temp_pre)

     temp_pre = np.copy(temp)
     temp_mean_pre = T_mean / np.sum(area)
     loop = loop + 1

# Plot
plt.plot(sol_flux, temp, 'ro-')
plt.xlabel('Solar Flux')
plt.ylabel('Temperature')
plt.show()
```

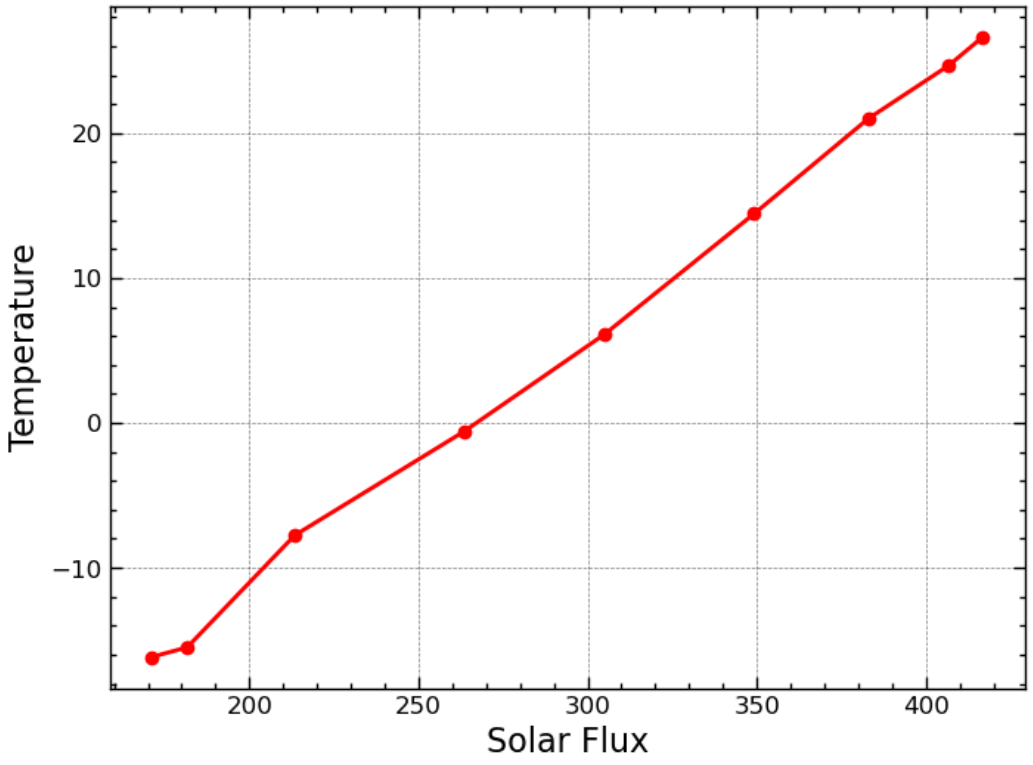

Figure 11: Equilibrium Temperature of Model EBM 1D

Figure 11 shows the relationship between solar flux and equilibrium temperature at different latitudes on Earth. The graph suggests that there is a proportional increase in local equilibrium temperature as the solar flux rises, covering a range of approximately 42°C from -16°C to 26°C.

The observed increased trend can be attributed to the model of energy balance in the Earth's climate system. In the EBM 1D model, solar flux represents the incoming energy from the Sun, which is a primary driver of Earth's climate. As the solar flux intensifies, more energy is available to warm the Earth's surface and atmosphere, decreasing the zonal albedo. Consequently, temperatures rise across the globe.

Practice #2: Solar flux for Earth becoming a complete snowball

In this part, we intends to find the amount of solar flux needed so that feedback loop will eventually turn the Earth to a completely icy planet. To do this, we run the EBM 1D model just like in the previous section with different values of solar flux. The condition for the whole Earth to turn into icy is that all the zones have the equilibrium temperature lower than 0°C.

In order to do this job, i created a EBM 1D function, that take the solar flux as input and return an array of the equilibrium temperature for each latitudinal band. Then by varying the value of solar flux, and setting the condition for all T ¡ 0°C, i found the needed solar flux for turning the Earth to a snowball.

```
# Create a range of solar isolation
So_range = np.arange(1115, 1120,0.5)

# Define function for EBM1d_equilibrium temperature
def EBM1d_equi_fucntion(S, kt = kt, ini_alb = alb_ice_free, A = A, B = B):
    ### Solar flux
    ini_alb = np.array([0.23, 0.24, 0.25, 0.29, 0.35, 0.40, 0.46, 0.50,
   0.50])
    sol_frac = np.array([0.30475 ,0.29725 ,0.28 ,0.25525 ,0.223 ,0.1925
   ,0.156 ,0.13275 ,0.125])
    sol_flux = S*sol_frac #Solar flux for bands
    ### Global mean temperature

    alb_mean = np.mean(ini_alb)/sum_area
    T_mean = (S*(1-alb_mean)/4 - A)/B

    ## Define matrix for storing values of all bands
    temp = np.ones(ny)*T_mean
    temp_pre = np.ones(ny)*T_mean
    temp_mean_pre = T_mean
    loop = 0

    ## Boundary condition
    stable_check = 1
    epsilon = 0.001

    alb = np.copy(ini_alb)
    ## Loop
    while abs(stable_check) > epsilon:
        stable_check = 0
        temp = EBM_1d(sol_flux, alb, temp_mean_pre, kt = kt)
        ## Albedo condition
        if np.any(temp > temp_c2):
            alb[temp > temp_c2] = alb_ice_free[temp > temp_c2]
        if np.any(temp <= temp_c1):
            alb[temp <= temp_c1] = alb_ice
```

```
        else:
            alb[(temp <= temp_c2) & (temp > temp_c1)] = alb_ice*(temp_c2 -
    temp[(temp <= temp_c2) & (temp > temp_c1)])/(temp_c2 - temp_c1) +
    alb_ice_free[(temp <= temp_c2) & (temp > temp_c1)]*(temp[(temp <= temp_c2
    ) & (temp > temp_c1)] - temp_c1)/(temp_c2 - temp_c1)
        T_mean = np.sum(temp * area)
        stable_check = np.sum(temp - temp_pre)

        temp_pre = np.copy(temp)
        temp_mean_pre = T_mean / np.sum(area)
        loop = loop + 1
    return temp

# Plot
plt.figure(figsize=(15, 10))

for s in So_range:
    temp_s = EBM1d_equi_fucntion(s)
    if np.all(temp_s < 0):
        plt.plot(sol_flux,temp_s, 'o--',label = f'S_ice = {s}', color = '
    gray')
    else:
        plt.plot(sol_flux,temp_s, 'o--',label = f'S_no_ice = {s}', color = '
    green')
    plt.xlabel("Solar Flux")
    plt.ylabel(r"Temperature ($^o$C)")

ax = plt.gca()
ax.legend(loc = 'upper left')
# plt.subplots_adjust(right=0.7)
plt.savefig('figures/ebm1d_sol_allice.png')
# plt.show()
```

Listing 15: Code for the EBM 1D model with changing solar flux

From the figure 12, we can observe that when S = 1118 W m^{-2} at about 82% the solar flux of the Earth at this moment (S = 1368 W m^{-2}), the equilibrium temperatures in all zone are lower than 0°C, thus the Earth is now covered completely in ice.

Practice #3: Sensitivity of the model

With k: In this problem, we will check the sensitivity of the model with the changing of k. k is the transport coefficient (refer to equation 19) which implies the linear relationship of the energy transport from each band with the difference in the zonal temperature and global temperature. In another words, k associates with the process of lateral heat advection which have a tendency to warm the regions that are colder than the global temperature and cool the regions that are hotter than the global temperature.

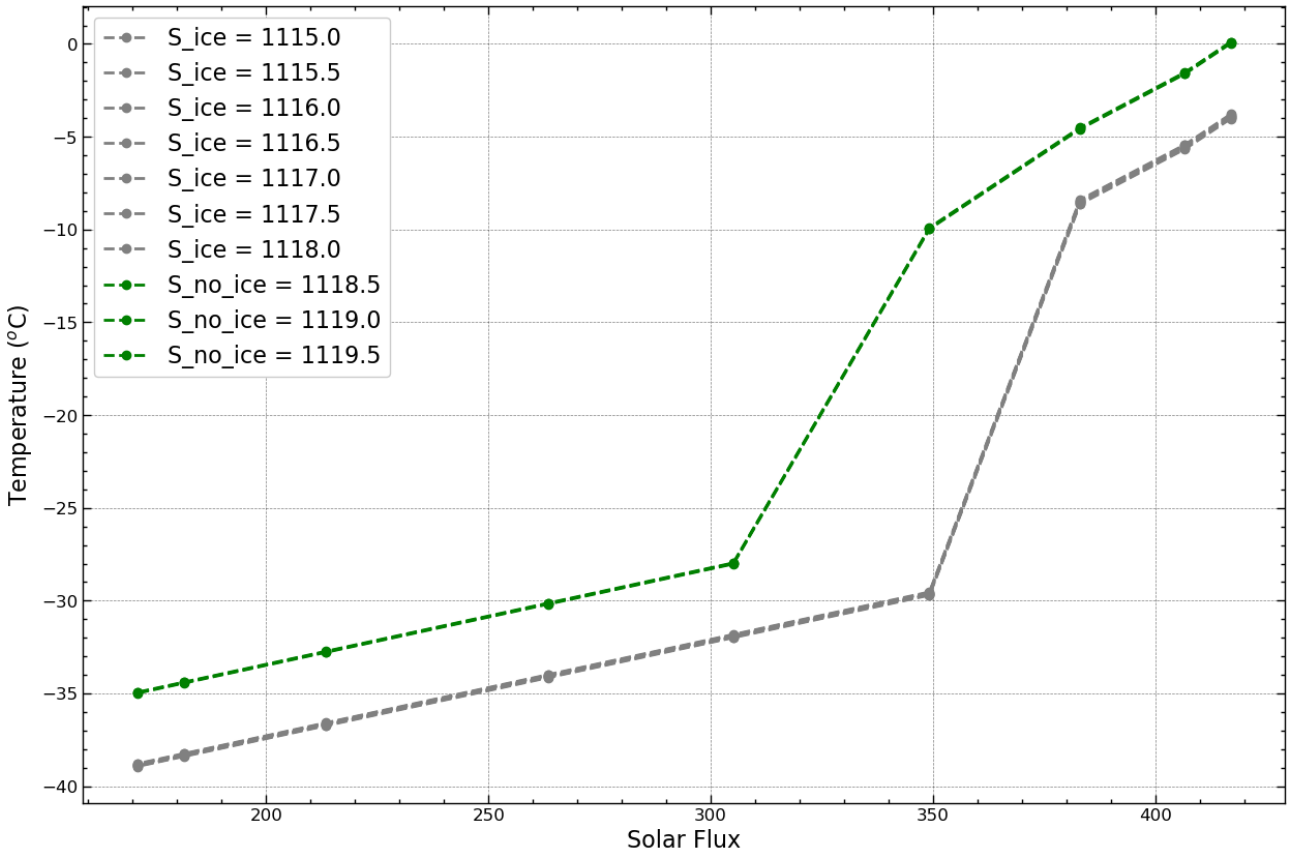

Figure 12: The value of the solar flux so that the Earth will be entirely covered by ice

```
# Create a range of kt
kt_sen = np.arange(2,5,0.5)
kt_Budyko = 3.81
kt_Schneider = 3.74
# Plot
fig, ax = plt.subplots(figsize = (12,10))
for k in range(len(kt_sen)):
    T_k = EBM1d_equi_fucntion(S0, kt_sen[k])
    plt.plot(sol_flux, T_k, 'o-', label = f'k = {kt_sen[k]}')

plt.plot(sol_flux, EBM1d_equi_fucntion(S0, kt_Budyko), 'o--', color = 'black
    ',label = f'Budyko k = {kt_Budyko}')
plt.plot(sol_flux, EBM1d_equi_fucntion(S0, kt_Schneider), 'o--', color = '
    deeppink', label = f'Schneider k = {kt_Schneider}')
ax.set_xlabel("Solar Flux")
ax.set_ylabel("Temperature")
ax.set_title("The sensitivity of the model with K")
```

```
16 plt.legend()
17
18 plt.savefig('figures/ebm1d_sens_k.png')
19 # plt.show()
```

Listing 16: Code for exmamine the sensitivity of EBM 1D model to k

Figure 13 shows result of the sensitivity of the EBM 1D model on k, we can see that the increasing value of k leading, we can observe that the model is quite sensitive to the change of k. The higher the k value, the lower the slope of temperature variation with solar flux. This phenomenon happens because increasing k implies that there is a more pronounced response of the model to keep the temperature balance, leading to smaller variation in the temperature between each latitudinal band.

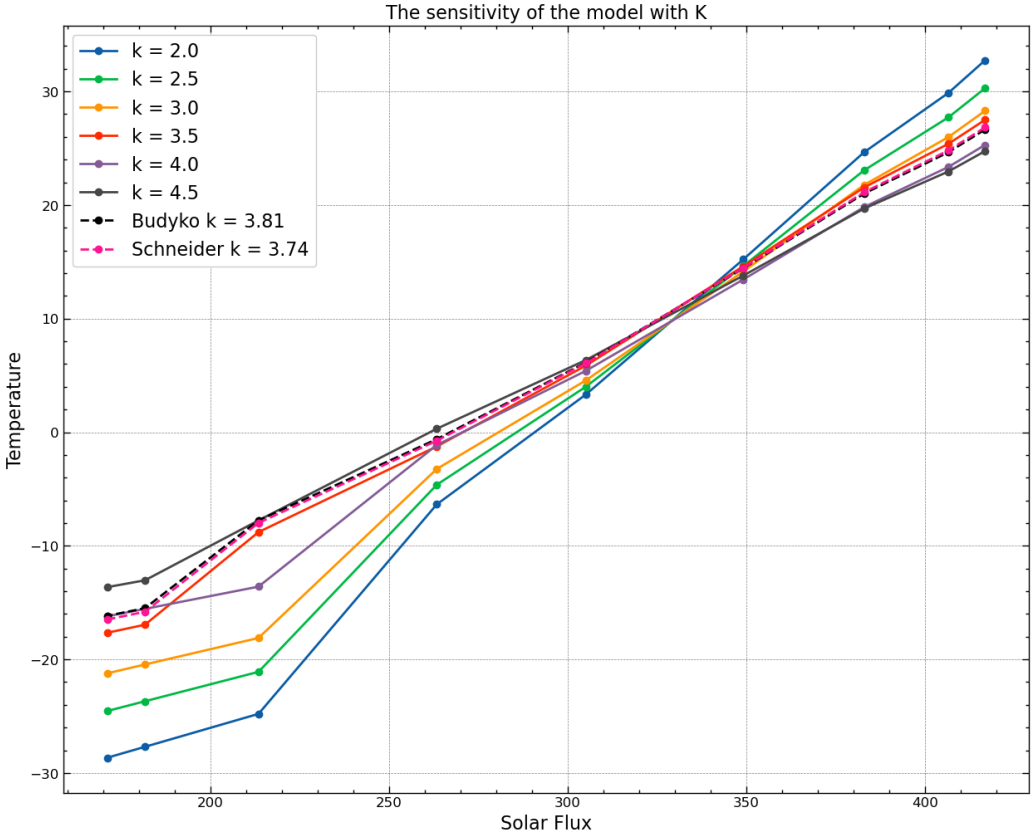

Figure 13: The sensivity of the model with variation of k

With B: In this problem, we will check the sensitivity of the model with the changing of B, B is the coefficient presented in equation 18 associating with the process of releasing the infrared radiation of the Earth including the heat trapping effect of greenhouse gases.

```
# Create a range of B
B_sen = np.arange(1,3, 0.3)
A_Bydyko = 202; B_Bydyko = 1.45
A_Cess = 212; B_Cess = 1.6
# Plot
fig, ax = plt.subplots(figsize = (12,8))
for i in range(len(B_sen)):
    T_b = EBM1d_equi_fucntion(S0 , B = B_sen[i])
    plt.plot(sol_flux, T_b, 'o-', label = f'A = {A}, B = {B_sen[i]:.1f}',
   color = color_palette(i/len(B_sen)))

plt.plot(sol_flux, EBM1d_equi_fucntion(S0, A = A_Bydyko, B = B_Bydyko), 'o--
   ', color = 'deeppink',label = f'Bydyko A = {A_Bydyko}, B = {B_Bydyko}')
plt.plot(sol_flux, EBM1d_equi_fucntion(S0, A = A_Cess, B = B_Cess), 'o--',
   color = 'black',label = f'Cess A = {A_Cess}, B = {B_Cess}')
ax.set_xlabel("Solar Flux")
ax.set_ylabel("Temperature")
plt.legend()

plt.savefig('figures/ebm1d_sens_B.png')
# plt.show()
```

Listing 17: Code for exmamine the sensitivity of EBM 1D model to B

Figure 14 illustrates the sensitivity of the EBM 1D model with the change of B. I guessed that the lower the B value means that less energy released to space, so the planet should be little bit hotter. However, there is no significant different between the Bydyko and Cess values. Additionally, as we can see that the model is not really sensitive with the changing of B. Although, it's a little bit counter-intuitive, the Earth in EBM 1D seems to find a way to reach to the same equilibrium value of temperature although there is some.

Let's try to see whether the change of A value can cause a shift to the equilibrium temperature in EBM 1D model.

```
# Create a range of A
A_sen = np.arange(200, 1000, 100)
A_Bydyko = 202; B_Bydyko = 1.45
A_Cess = 212; B_Cess = 1.6
# Plot
fig, ax = plt.subplots(figsize = (15,12))
for i in range(len(A_sen)):
    T_a = EBM1d_equi_fucntion(S0 , A = A_sen[i])
    plt.plot(sol_flux, T_a, 'o-', label = f'A = {A_sen[i]}, B = {B}', color
   = color_palette(i/len(A_sen)))

plt.plot(sol_flux, EBM1d_equi_fucntion(S0, A = A_Bydyko, B = B_Bydyko), 'o--
   ', color = 'deeppink',label = f'Bydyko A = {A_Bydyko}, B = {B_Bydyko}')
```

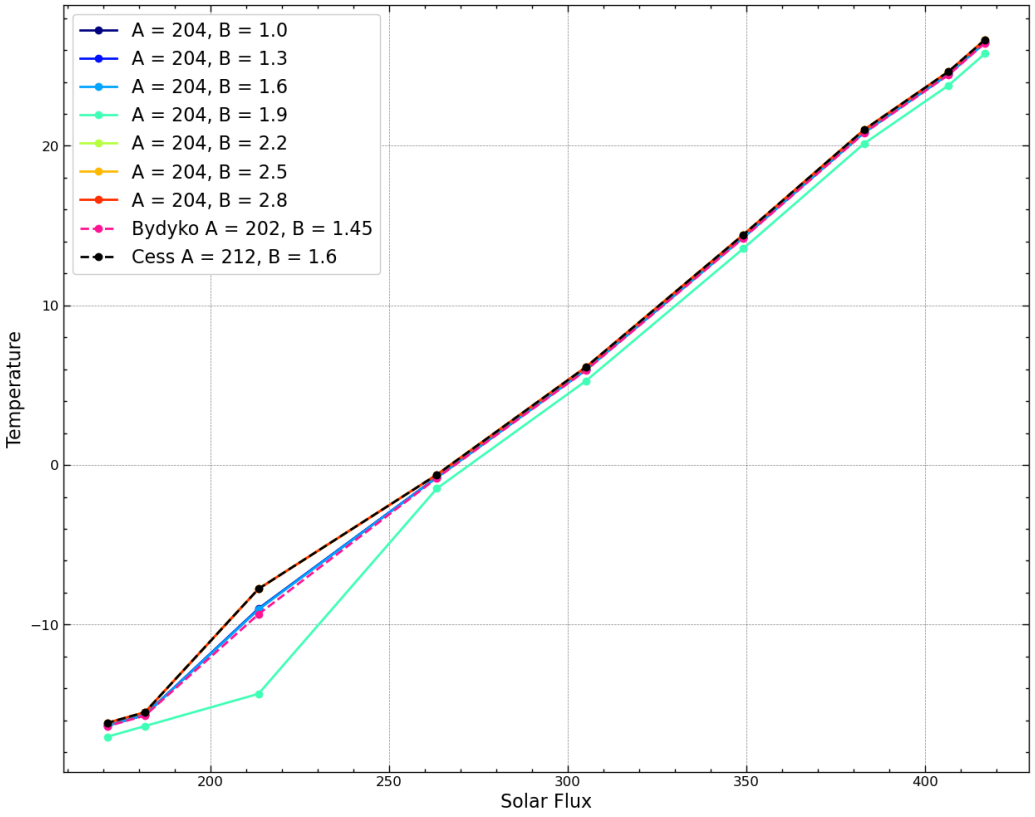

Figure 14: The sensivity of the model with variation of B

```
plt.plot(sol_flux, EBM1d_equi_fucntion(S0, A = A_Cess, B = B_Cess), 'o--',
    color = 'black',label = f'Cess A = {A_Cess}, B = {B_Cess}')
ax.set_xlabel("Solar Flux")
ax.set_ylabel("Temperature")
plt.legend(loc = 'upper left')

plt.savefig('figures/ebm1d_sens_A.png')
# plt.show()
```

Figure 15 show the sensitivity of the EBM 1D model with the change of A. We can see that the model is sensitive with A to some degree. The increase in A does indeed lower the equilibrium temperature of Earth, but it has to exceed a certain threshold.

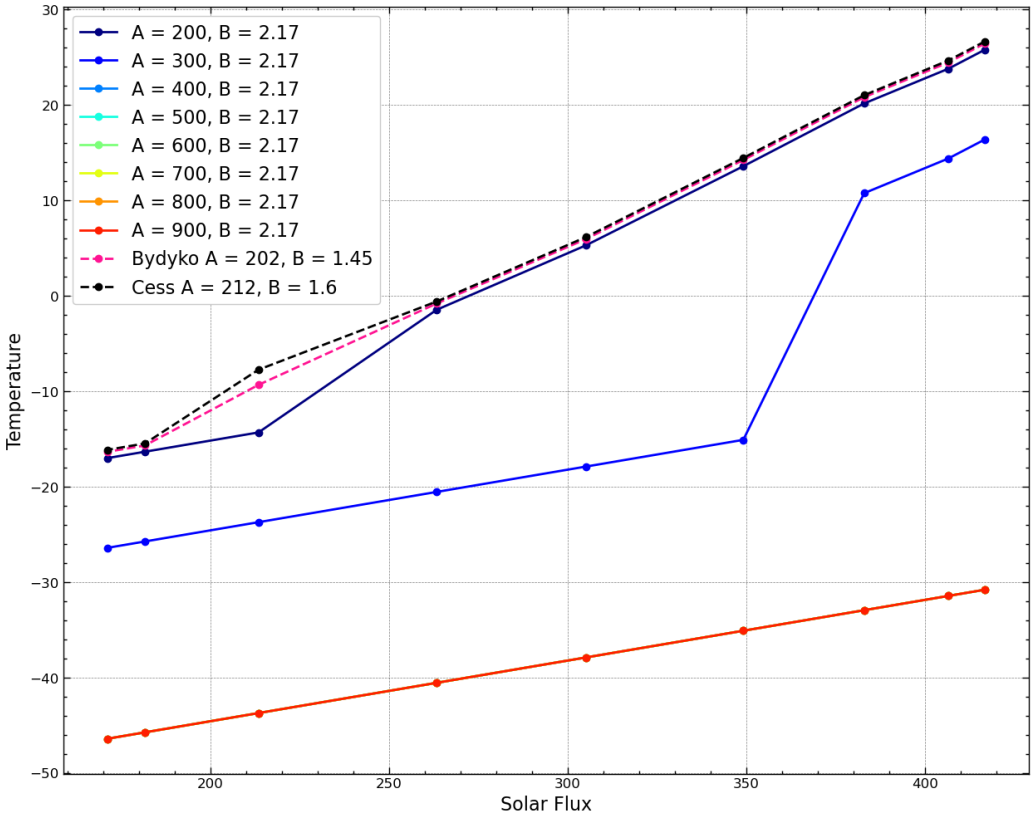

Figure 15: The sensivity of the model with variation of A

5 Daisy World

5.1 Gaia Hypothesis

The Gaia hypothesis, proposed by James Lovelock in 1970s, postulates that the Earth's surface is maintained in a habitable state by self-regulating feedback mechanisms involving organisms tightly coupled to their environment (Encyclopedia of Atmospheric Sciences, 2003). This hypothesis was suggested in order to explain for the Faint Sun paradox which states that the early Earth should have been too cold to support liquid water, given the Sun's lower luminosity in the past. However, the situation is differentThe Gaia hypothesis suggests that life on Earth has played a crucial role in regulating the planet's climate and maintaining conditions suitable for life.

5.2 Daisy World

Daisy World is a model of an imaginary planet developed by James Lovelock and his companion Andrew Watson. It aims to explain the feedback and homeostasis that regulate the

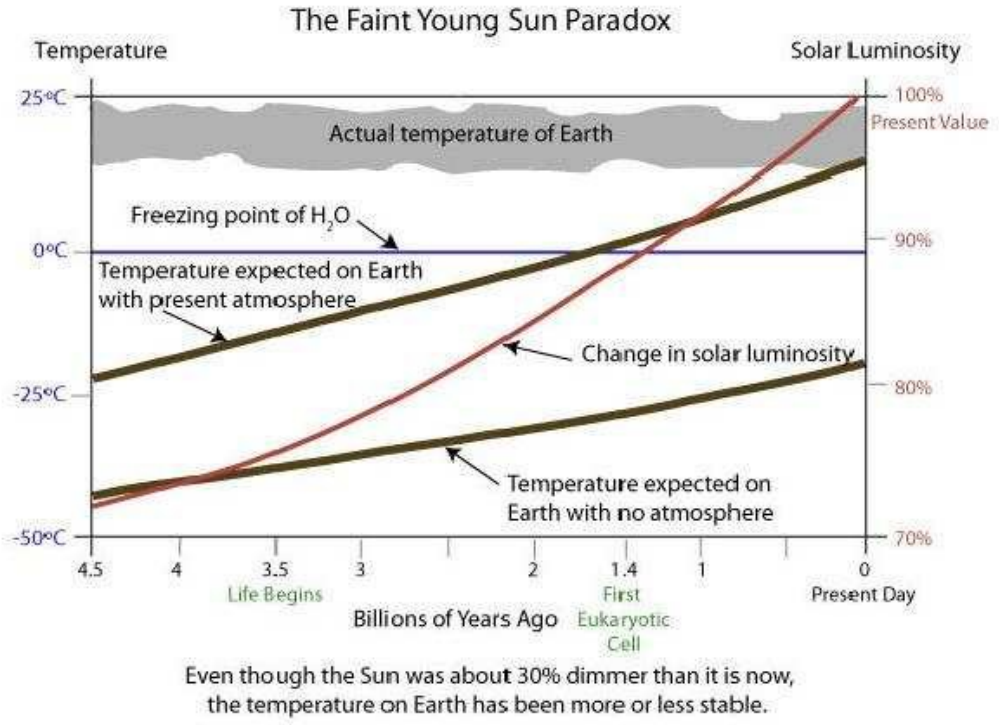

Figure 16: Faint Sun Paradox

environmental conditions on a planetary scale, as proposed by the Gaia Hypothesis.

Daisy World is depicted as a flat disk with no seasonality in climate, a transparent atmosphere, no clouds, and no greenhouse effect. The planet is inhabited by two species of daisies: black and

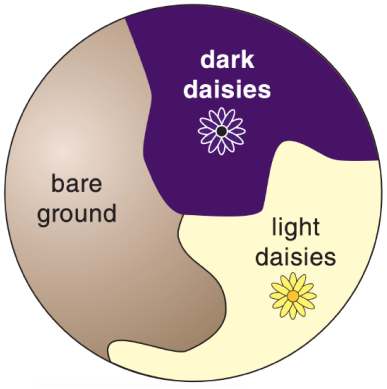

Figure 17: Daisy World illustration

white. These daisies are identical in every respect except for the color of their flowers, which

affects their albedo. The black daisies have a low albedo, absorbing more energy and warming up the planet. In contrast, the white daisies have a high albedo, reflecting more energy and cooling down the planet. As a result, the growth rates of the two species vary based on the intensity of sunlight.

5.3 Mathematical model of Daisy World

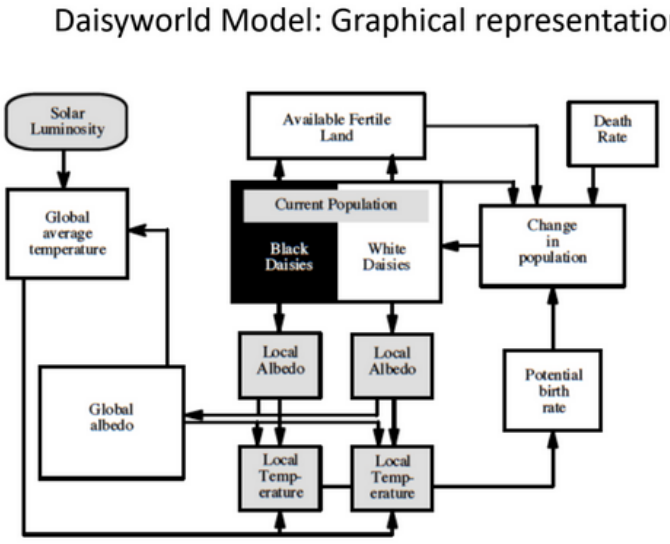

Figure 18: Graphical Representation of Daisy World Model

1. The amount of fertile land available for daisy growth:

$$x = 1 - w - b \tag{22}$$

where x is the amount of available land and w and b are the areas of white and black daisies, respectively.

2. The planetary albedo:

$$\alpha_p = w\alpha_w + b\alpha_b + x\alpha_g \tag{23}$$

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

3. The planetary temperature:

$$T_e = \frac{S(1-\alpha_p)}{4\epsilon\sigma} - 273.15 \tag{24}$$

where T_e is the planetary temperature, S is the solar constant, ϵ is the emissivity, and σ is the Stefan-Boltzmann constant.

4. The local temperature for populations of black and white daisies:

$$T_b = T_e + q(\alpha_p - \alpha_b) \tag{25}$$

$$T_w = T_e + q(\alpha_p - \alpha_w) \tag{26}$$

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

5. The growth rate of the populations of black and white daisies:

$$f_b = \max(0, 1 - k(T_0 - T_b)^2) \tag{27}$$

$$f_w = \max(0, 1 - k(T_0 - T_w)^2) \tag{28}$$

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.

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

$$\frac{db}{dt} = b(xf_b - d) \tag{29}$$

$$\frac{dw}{dt} = w(xf_w - d) \tag{30}$$

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

7. The new area of black and white daisies:

$$b = b + \frac{db}{dt} \tag{31}$$

$$w = w + \frac{dw}{dt} \tag{32}$$

5.4 Practice

1. Plot the growth rate function f_b and f_w versus temperature:

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

2. Set $S = S_0 = 1000$ and at the initial stage $b = w = 0.2$. Find the equilibrium values of b and w.

3. S/S_0 varies from 0.4 to 1.6:
   (a) Plot the equilibrium daisy black/white area versus S/S_0.
   (b) Plot the planetary temperature versus S/S_0.
4. Suppose that Daisy World has only one species, black daisy. How does the behavior of the system change?

5.4.1 Growth rate versus temperature

I define a range of temperature to examine the growth rate function of the daisies. Because two daisies are similar except for their albedo, so their growth rate should be identical.

```
### 1 Growth rate function
Tb = np.arange(-5, 50, 0.1)
Tw = np.arange(-5, 50, 0.1)
k = 0.003265
T0 = 22.5

f_b = np.maximum(0, 1 - k * (T0 - Tb)**2)
f_w = np.maximum(0, 1 - k * (T0 - Tw)**2)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(18,6), sharey=True)
ax1.plot(Tb, f_b, color='black')
ax1.set_xlabel("Temperature")
ax1.set_ylabel("Growth Rate")
ax1.set_title("Black Daisy")

ax2.plot(Tw, f_w, color='Gold')
ax2.set_xlabel("Temperature")
ax2.set_title("White Daisy")
plt.tight_layout(w_pad=0)

plt.show()
```

Listing 18: Code for growth rate of daisies

Figure 19 shows the result, just as we guess, the two plots for black and white daisies are identical. The growth rate of the daisies follows a parabolic curve, indicating the temperature range in which daisies can thrive. This temperature range extends from approximately 5°C to 40°C, with the highest growth rate occurring at their optimal temperature, $T_0 = 22.5$°C.

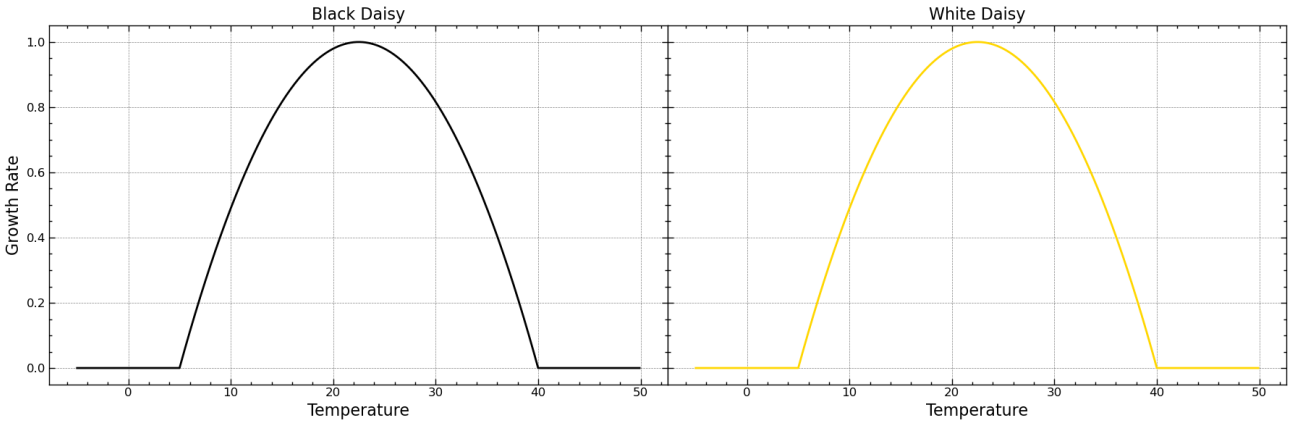

Figure 19: Growth rate of black and white daisy

5.4.2 Equilibrium value of b and w at S = 1000 W m^{-2}

At a fixed solar flux S = 1000 W m^{-2}, we define a low albedo of 0.25 for black daisy and high albedo of 0.75 for white daisy, and a medium albedo of 0.5 for bare ground. We define the initial condition of the area coverage of the daisy world, this will affect the global albedo which in turns define the global temperature. The global temperature now change the local temperature deciding the growth rate of the daisies on planet. This change in growth rate leads to modification in the current area coverage of the daisy world, forming a feedback loop. This process proceeds until the equilibrium between the area coverage of black and white daisies is achieved.

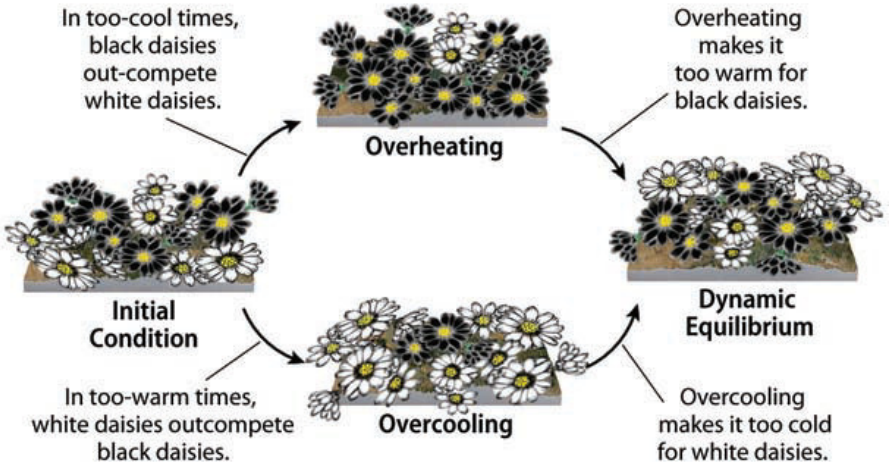

Figure 20: Daisy feedback mechanism

```
#### Constants
alb_black = 0.25
alb_white = 0.75
alb_ground = 0.5
S0 = 1000
epsilon = 0.3
q = 20
k = 0.003265
T0 = 22.5
d = 0.3  # Death ratio
sigma = 5.6704e-8

#### Initial conditions
white_daisy = 0.2
black_daisy = 0.2
global_albedo = white_daisy * alb_white + black_daisy * alb_black + (1 -
    black_daisy - white_daisy) * alb_ground
eps = 1
while eps > 1e-8:
    available_land = 1 - white_daisy - black_daisy
    global_albedo = white_daisy * alb_white + black_daisy * alb_black +
   available_land * alb_ground
    Te = np.power((1 - global_albedo) * S0 / (sigma), 1 / 4) - 273.15
    Tb = Te + q * (global_albedo - alb_black)
    Tw = Te + q * (global_albedo - alb_white)
    f_b = max(0, 1 - k * (T0 - Tb)**2)
    f_w = max(0, 1 - k * (T0 - Tw)**2)
    db = black_daisy * (available_land * f_b - d)
    dw = white_daisy * (available_land * f_w - d)
    black_daisy = black_daisy + db
    white_daisy = white_daisy + dw
    # Convergence check
    eps = np.abs(db + dw)

# Equilibrium value
print("Black Daisy Area:", black_daisy)
print("White Daisy Area:", white_daisy)
```

```
Black Daisy Area: 0.18556264788266139
White Daisy Area: 0.4877733688485004
```

The result of this problem is that the black daisy covers about 20% and white daisy covers about 50% of the planet.

5.4.3 Daisy World model with variation of solar flux

In this problem, we evaluate the behavior of the Daisy World model with the changing solar flux. The idea for working out this code came from previous work by Do Quoc Trong, my senior

in our department. The first point to solve this problem is that we start the initial condition at our chosen point, then keep increasing (or decreasing) the solar flux and use the equilibrium value of black and white daisy for the next value of solar flux. This approach reflects the real situation and help us examine how the population of daisies change with the variation of solar isolation. The second point is that we need to start the simulation from a stable condition instead running from the lowest value of solar flux which is unstable. Therefore, I divided into two different parts at S = 1000 W m^{-2}, one run forward with increasing solar flux and one run backward with decreasing solar flux.

```
# Solar flux array with the ratio S/S0 of (0,4 to 1.6)
solar_flux_1 = np.arange(1, 1.6, 0.01) * S0  # Forward
solar_flux_2 = np.arange(1, 0.39, -0.01) * S0  # Reverse
solar_flux = np.concatenate((solar_flux_1, solar_flux_2))

# Inital conditions
initial_white_fraction = 0.2
initial_black_fraction = 0.2
global_albedo_no_daisy = (
    initial_white_fraction * alb_white
    + initial_black_fraction * alb_black
    + (1 - initial_black_fraction - initial_white_fraction) * alb_ground
)
b_eq = []
w_eq = []
global_temperature = []
global_temperature_no_daisy = []

# Main program
for s in solar_flux:
    if s == S0:
        white_daisy = initial_white_fraction
        black_daisy = initial_black_fraction
    else:
        pass
    eps = 1
    while eps > 1e-8:
        available_land = 1 - white_daisy - black_daisy
        global_albedo = (
            white_daisy * alb_white
            + black_daisy * alb_black
            + available_land * alb_ground
        )
        Te = np.power((1 - global_albedo) * s / (sigma), 1 / 4) - 273.15
        Tb = Te + q * (global_albedo - alb_black)
        Tw = Te + q * (global_albedo - alb_white)
        f_b = max(0, 1 - k * (T0 - Tb) ** 2)
        f_w = max(0, 1 - k * (T0 - Tw) ** 2)
        db = black_daisy * (available_land * f_b - d)
```

```
        dw = white_daisy * (available_land * f_w - d)
        black_daisy = black_daisy + db
        white_daisy = white_daisy + dw
        # Convergence check
        eps = np.abs(db + dw)
    # Equilibrium value
    global_temperature.append(Te)
    b_eq.append(black_daisy)
    w_eq.append(white_daisy)

    # No daisy
    Te_no_daisy = np.power((1 - global_albedo_no_daisy) * s / sigma, 1 / 4)
    - 273.15
    global_temperature_no_daisy.append(Te_no_daisy)

# Put all the value in the order of increasing solar flux for plotting
solar_flux_plot = np.concatenate((np.flip(solar_flux_2), solar_flux_1))
b_eq_plot = np.hstack((np.flip(np.array(b_eq[len(solar_flux_1):])), np.array
    (b_eq[:len(solar_flux_1)])))
w_eq_plot = np.hstack((np.flip(np.array(w_eq[len(solar_flux_1):])), np.array
    (w_eq[:len(solar_flux_1)])))
no_daisy_plot = 1 - b_eq_plot - w_eq_plot
Te_no_daisy_plot = np.hstack(
    (np.flip(np.array(global_temperature_no_daisy[len(solar_flux_1):])), np.
    array(global_temperature_no_daisy[:len(solar_flux_1)]))
)
Te_daisy_plot = np.hstack((np.flip(np.array(global_temperature[len(
    solar_flux_1):])), np.array(global_temperature[:len(solar_flux_1)])))

# Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

ax1.plot(solar_flux_plot, no_daisy_plot, "g--", label="No daisy")
ax1.plot(solar_flux_plot, b_eq_plot, color="black", label="Black daisy")
ax1.plot(solar_flux_plot, w_eq_plot, color="gold", label="White daisy")
ax1.set_title("Area Coverage")
ax1.set_ylabel("Area Ratio")
ax1.set_xlabel("Solar Flux Ratio")
ax1.legend()

ax2.plot(solar_flux_plot, Te_no_daisy_plot, "g--", label="No daisy")
ax2.plot(solar_flux_plot, Te_daisy_plot, "r", label="With daisy")
ax2.set_title("Temperature")
ax2.set_ylabel("Temperature")
ax2.set_xlabel("Solar Flux Ratio")
ax2.legend()

fig.suptitle("Daisy World Model", size=20)
plt.tight_layout()
```

```
84 plt.show()
```

Listing 19: Code for Daisy World with variation of solar flux

In the figure 21, we can observe the opposite behavior of the black and white daisy in response to the change of solar flux.

As the two species of daisy have their existing temperature, if the planet is too cold (low value of solar flux) or too hot (high value of solar flux), the species will die out. We can see that the existing range of black daisy going from around 550 to 1250 W m^{-2}, while the existing range of white daisy is little bit higher, from around 680 to 1450 W m^{-2}. Additionally, I have tried to initiate the daisy world at different solar flux and the range that give the same result as figure 21 is from about 650 to 1150 W m^{-2} which is the range that *both black and white exists at the same time*! Thus, this condition define **the stable condition** for launching the Daisy World model, if we begins the simulation outside of this range, the simulation will get unstable and give off very "weird" result.

Generally, the population of black and white daisies are self-modified in order to stabilize the global temperature, as we can see in the Temperature in figure 21. Normally, the solar flux increases continously, however, the appearance of black and white daisy will help keeping a **stable temperature** of about 20-23°C (the favourable temperature for daisies to thrive) in the existing range of the daisies. The self-regulation mechanism can also observed in the Area Coverage in figure 21. The black daisy with low albedo is dominant when the solar flux is low to warm up the planet. As the solar flux increases, the black daisy population decrease while the white daisy population with high albedo thrives to cool down the planet to the favourable temperature for their growth.

At the extreme condition where the solar flux is too low, all white daisy die out and black daisy population reaches their maximum at about 70% of the area before decreases and vanish. The similar situation happens when the solar flux is too high, all black daisy disappear, white daisy thrive at about 70% and then disappear. Also, observing the green dash line of no daisy, we realize not matter how the solar flux changes, in the existing range, white and black daisies in this model can only cover approximately 70% in total.

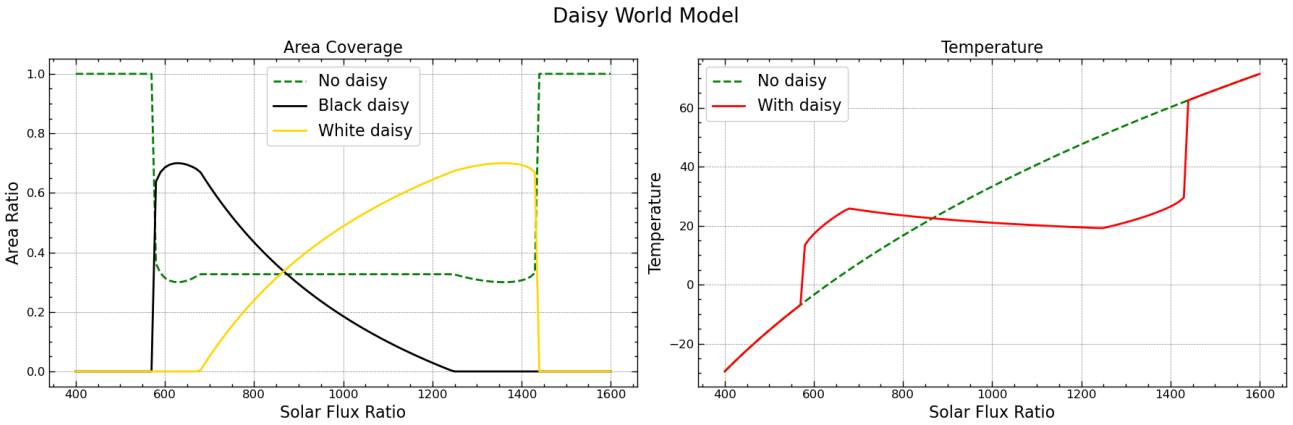

Figure 21: Daisy World Model

5.4.4 Daisy World model with lonely black daisy

Now, what if there is a epidemic that wipe out all the white daisy, what will happen to our Daisy World?

```
# Initial value
initial_black_fraction = 0.2
global_albedo_no_daisy = initial_black_fraction * alb_black + (1 -
    initial_black_fraction) * alb_ground
b_eq = []
global_temperature = []
global_temperature_no_daisy = []

# Main program
for s in solar_flux:
    if s == 1000:
        black_daisy = initial_black_fraction
    else:
        pass

    eps = 1
    while eps > 1e-6:
        available_land = 1 - black_daisy
        global_albedo = black_daisy * alb_black + available_land *
    alb_ground

        Te = np.power((1 - global_albedo) * s / sigma, 1 / 4) - 273.15
        Tb = Te + q * (global_albedo - alb_black)

        f_b = max(0, 1 - k * (T0 - Tb) ** 2)
        db = black_daisy * (available_land * f_b - d)
        black_daisy = black_daisy + db
```

```
26
27          # Convergence check
28          eps = np.abs(db)
29
30      # Equilibrium value
31      global_temperature.append(Te)
32      b_eq.append(black_daisy)
33
34      # No daisy
35      Te_no_daisy = np.power((1 - alb_ground) * s / sigma, 1 / 4) - 273.15
36      global_temperature_no_daisy.append(Te_no_daisy)
37
38  # Put all the value in the order of increasing solar flux for plotting
39  b_eq_plot = np.hstack((np.flip(np.array(b_eq[len(solar_flux_1):])), np.array
        (b_eq[:len(solar_flux_1)])))
40  no_daisy_plot = 1 - b_eq_plot
41  Te_no_daisy_plot = np.hstack((np.flip(np.array(global_temperature_no_daisy[
        len(solar_flux_1):])), np.array(global_temperature_no_daisy[:len(
        solar_flux_1)])))
42  Te_daisy_plot = np.hstack((np.flip(np.array(global_temperature[len(
        solar_flux_1):])), np.array(global_temperature[:len(solar_flux_1)])))
43
44  # Plot
45  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
46
47  ax1.plot(solar_flux_plot, no_daisy_plot, 'g--', label='No daisy')
48  ax1.plot(solar_flux_plot, b_eq_plot, color='black', label='Black daisy')
49  ax1.set_title("Area Coverage")
50  ax1.set_ylabel('Area Ratio')
51  ax1.set_xlabel('Solar Flux Ratio')
52  ax1.legend()
53
54  ax2.plot(solar_flux_plot, Te_no_daisy_plot, 'g--', label='No daisy')
55  ax2.plot(solar_flux_plot, Te_daisy_plot, 'r', label='With daisy')
56  ax2.set_title("Temperature")
57  ax2.set_ylabel('Temperature')
58  ax2.set_xlabel('Solar Flux Ratio')
59  ax2.legend()
60
61  fig.suptitle('Daisy World Model (Only Black Daisy)', size=20)
62  plt.tight_layout()
63  plt.show()
```

Listing 20: Code of Daisy World with only black daisy

Figure 22 indicate the situation where the planet is only left with black daisy which still regulate the temperature (a little higher than the favourable temperature) at of the world at the low solar flux. However, the species dies out at about 950 W m^{-2} faster than in the world with two

species.

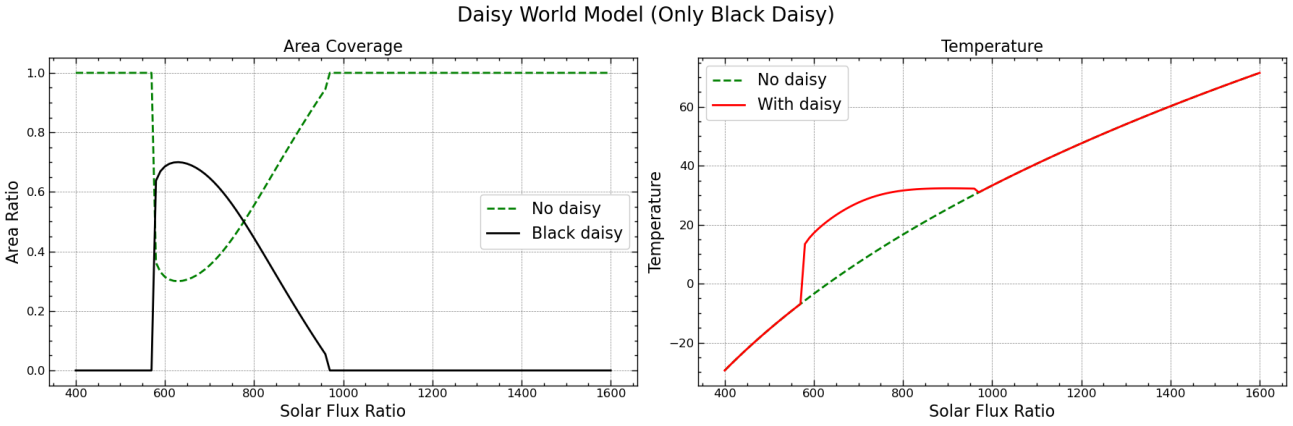

Figure 22: Daisy World with only black daisy

5.4.5 Daisy World model with lonely white daisy

Similarly, what if only white daisy survives in this world?

```
# Initial value
initial_white_fraction = 0.2
global_albedo_no_daisy = initial_white_fraction * alb_white + (1 -
    initial_white_fraction) * alb_ground
w_eq = []
global_temperature = []
global_temperature_no_daisy = []

# Main program
for s in solar_flux:
    if s == 1000:
        white_daisy = initial_white_fraction
    else:
        pass

    eps = 1
    while eps > 1e-6:
        available_land = 1 - white_daisy
        global_albedo = white_daisy * alb_white + available_land *
    alb_ground

        Te = np.power((1 - global_albedo) * s / sigma, 1 / 4) - 273.15
        Tw = Te + q * (global_albedo - alb_white)

        f_w = max(0, 1 - k * (T0 - Tw) ** 2)
        dw = white_daisy * (available_land * f_w - d)
        white_daisy = white_daisy + dw

        # Convergence check
        eps = np.abs(dw)

    # Equilibrium value
    global_temperature.append(Te)
    w_eq.append(white_daisy)

    # No daisy
    Te_no_daisy = np.power((1 - alb_ground) * s / sigma, 1 / 4) - 273.15
    global_temperature_no_daisy.append(Te_no_daisy)

# Put all the value in the order of increasing solar flux for plotting
w_eq_plot = np.hstack((np.flip(np.array(w_eq[len(solar_flux_1):])), np.array
    (w_eq[:len(solar_flux_1)])))
no_daisy_plot = 1 - w_eq_plot
Te_no_daisy_plot = np.hstack((np.flip(np.array(global_temperature_no_daisy[
    len(solar_flux_1):])), np.array(global_temperature_no_daisy[:len(
```

```
    solar_flux_1)])))
Te_daisy_plot = np.hstack((np.flip(np.array(global_temperature[len(
    solar_flux_1):])), np.array(global_temperature[:len(solar_flux_1)])))

# Plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

ax1.plot(solar_flux_plot, no_daisy_plot, 'g--', label='No daisy')
ax1.plot(solar_flux_plot, w_eq_plot, color='gold', label='White daisy')
ax1.set_title("Area Coverage")
ax1.set_ylabel('Area Ratio')
ax1.set_xlabel('Solar Flux Ratio')
ax1.legend()

ax2.plot(solar_flux_plot, Te_no_daisy_plot, 'g--', label='No daisy')
ax2.plot(solar_flux_plot, Te_daisy_plot, 'r', label='With daisy')
ax2.set_title("Temperature")
ax2.set_ylabel('Temperature')
ax2.set_xlabel('Solar Flux Ratio')
ax2.legend()

fig.suptitle('Daisy World Model (Only White Daisy)', size=20)
plt.tight_layout()
plt.savefig('figures/daisyworld_onlywhite')
# plt.show()
```

Figure 23 illustrate the behavior of white daisy in the world without black daisy. Similar to previous case, white daisy still help regulate the temperature of the planet at the high solar flux. Their existing range is also shorter than in two species world.

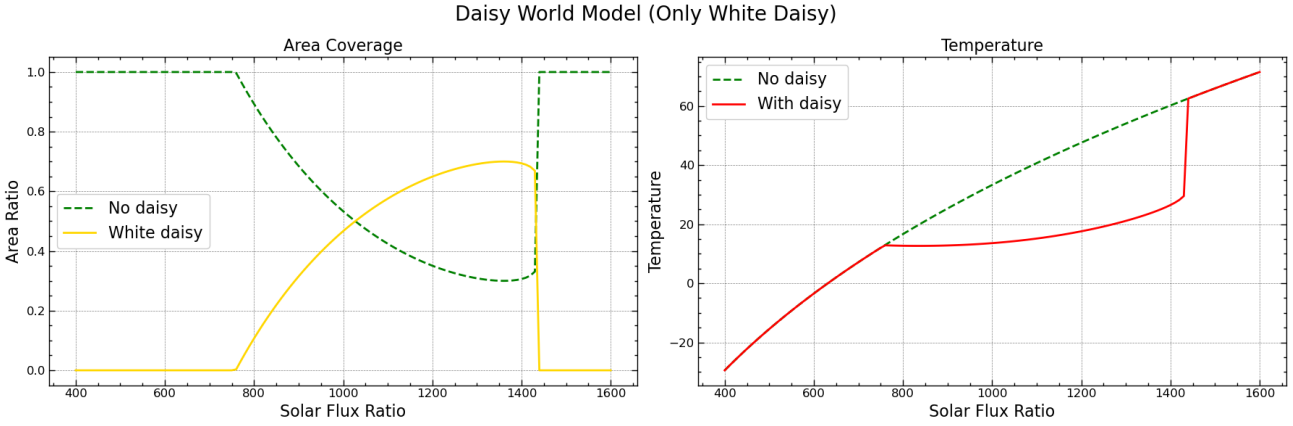

Figure 23: Daisy World with only white daisy

6 Sea level rise

Sea level rise is considered one of the major consequences of climate change, which can lead to various adverse effects such as coastal flooding, erosion, and the loss of valuable ecosystems like wetlands and mangroves.

This rise in sea level is primarily attributed to the global warming, as higher global temperatures cause the expansion of the upper layers of the ocean and the melting of glaciers and ice sheets.

Therefore, the measurement of sea level rise can, in turn, provide the crucial evident and predictions of global warming and climate change. The sea level rise data in our practice was taken using satellite altimeter technique to measure the height of the ocean surface relative to a reference point.

6.1 Practice

1. Download the altimetry data from https://www.star.nesdis.noaa.gov/socd/lsa/SeaLevelRise/slr/slr_sla_gbl_keep_txj1j2_90.csv.
2. Plot a figure from the data above to represent the change of sea level since 1992.

The data of Sea level rise comes from a series of satellite missions that started with TOPEX/-Poseidon (T/P) in 1992 and continued with Jason-1 (2001-2013), Jason-2 (2008 - 2019), and Jason-3 (2016 - present) estimate global mean sea level every 10 days with uncertainty of 3-4 mm.

Due to the combination of data from multiple satellites, there are some missing values in the dataset. To address this, the missing points in the data are filled using the forward fill method, which involves propagating the last valid observation forward to the next valid point, ensuring a continuous and consistent time series. Finally, we just simply need to remove unnecessary values to get a comprehensive figure.

```
# Data reading
df_slr = pd.read_csv('slr_sla_gbl_keep_txj1j2_90.csv',
                     skiprows=6, delimiter=',', header=None,
                     # na_values=[-99.99],
                    )

# Fill the NaN values with using forward method
df_slr_ = df_slr.fillna(method='ffill')

# Locate the unnecessary values
discont = []
for i in range(1,5):
    arr = np.where(df_slr_[i] == np.asarray(df_slr_[i])[-1])[0]
```

```
    for i in range(len(arr) - 1, -1, -1):
        if arr[i] != arr[i - 1] + 1:
            discont.append(arr[i])

# Mask the unnecessary values and plot
for i in range(1,5):
    df_slr_[i][discont[i-1]:] = np.nan
    plt.plot(df_slr_[0], df_slr_[i])

plt.xlabel("Year")
plt.ylabel("Sea level rise")

plt.show()
```

Figure 24 shows the rise of sea level (mm) over the period from 1992 to 2022. Overall, the sea has risen at about 8-10 cm in 30 years, and the rising rate is increasing. Over the course of the 20th century from 1992 to 2002, the rate is about 2 mm per year. Ten year after that from 2002 to 2012, it was about 2.5 mm per year. Over the past 10 years, the rate has increased to 4 mm per year. The rise of this rate is concerning because it is happening faster and faster due to human activities.

Furthermore, the annual cycle of sea level rise exhibits distinct peaks and valleys, which can be attributed to the seasonal movement of water between the oceans and land. During winter in the northern hemisphere, rainfall and snowfall increase, leading to an accumulation of water on land. This water takes time to runoff and eventually returns to the oceans. As a result, we observe a rise and fall in sea level each year, typically amounting to around 1 centimeter. The amplitude of this cycle may vary slightly during El Niño and La Niña years, with a slightly higher or lower sea level rise depending on the prevailing climate patterns.

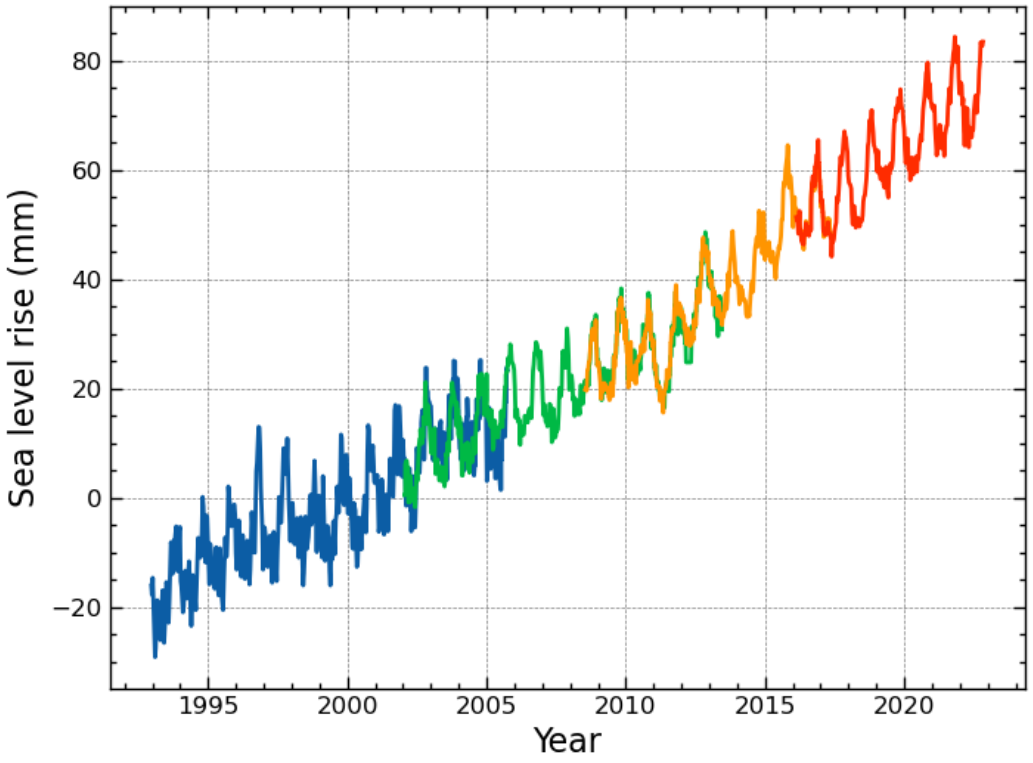

Figure 24: Sea level rise

7 Sea Surface Temperature with NetCDF

Sea surface temperature (SST) refers to the temperature of the upper layer of the ocean, which is in direct contact with the atmosphere. It is a crucial parameter to monitor as it is influenced by various factors like solar radiation, wind, and ocean currents.

Understanding and tracking changes in SST are vital for comprehending and addressing the effects of climate change and other environmental pressures. Furthermore, changes in global SST can influence climate patterns, such as the well known El Niño and La Niña events. On a smaller scale, ocean temperatures influence the formation of tropical cyclones by difference of temperature generating high and low pressure areas.

The dataset we used for this practice is monthly-mean data of Sea Surface Temperature written in NetCDF format (a file format for storing multidimensional scientific variables), provided by National Oceanic and Atmospheric Administration.

7.1 Practice

1. Download 'sst.mnmean.nc' from https://www.esrl.noaa.gov/psd/data/gridded/data.noaa.oisst.v2.html.
2. Read some basic information from the file.
3. Plot the average Sea Surface Temperature (SST) over the oceans.
4. Plot the SST departures.

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

import cartopy.crs as ccrs
import cartopy.feature as cfeature

import cftime
import matplotlib.units as munits
from matplotlib.dates import ConciseDateConverter
munits.registry[cftime.DatetimeGregorian] = ConciseDateConverter()
```

Listing 21: Code for necessary packages: NetCDF4 for reading files and Cartopy for plotting world map

```
# File reading
file = 'sst.mnmean.nc'
fh = Dataset(file, 'r')

# Data
d_times=nc.num2date(fh.variables['time'][:],fh.variables['time'].units)
sst = fh.variables['sst'][:].data # shape is time, lat, lon as shown above

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

Listing 22: Code for data reading and extracting from NetCDF files

7.1.1 Average Sea Surface Temperature

The sea surface temperature data is represented in a data cube format, as shown in Figure 25, which consists of three dimensions: time, latitude and longitude. Each point in the cube represents a specific location on the Earth's surface and a particular time period.

To obtain the average sea surface temperature, we calculate the mean temperature values across the time axis, resulting in a single average value for each location. This provides us with a comprehensive overview of the average temperature distribution across the sea surface.

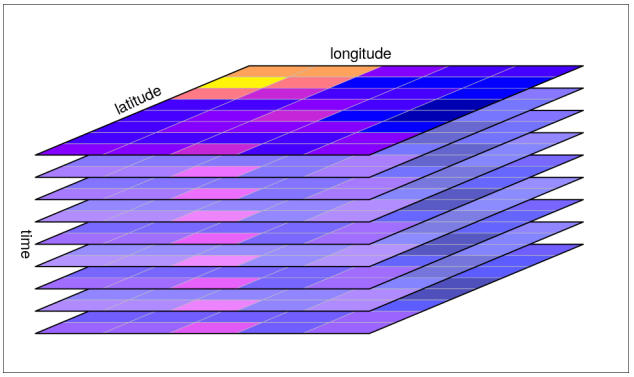

Figure 25: Data cube

```python
# Average SST
sst_mean = np.mean(sst, axis=0)

# Create a figure and axes
fig = plt.figure(figsize=(16,8))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude
    =180))

# Plot the sea water temperature array using imshow
cmap_reversed = plt.cm.get_cmap('RdYlBu').reversed()
im = ax.contourf(long, lat, sst_mean, cmap=cmap_reversed, transform=ccrs.
    PlateCarree(), levels = 15)

# Add land and coastline features
ax.add_feature(cfeature.COASTLINE, edgecolor='black')
ax.add_feature(cfeature.LAND, edgecolor='black', zorder = 1)
ax.add_feature(cfeature.BORDERS, linestyle=':')

# Add a colorbar
cbar = fig.colorbar(im, ax=ax, orientation='horizontal', label='Sea Water
    Temperature', shrink = 0.7, aspect=30)

# Set the title
ax.set_title('Sea Water Temperature')

# Show the plot
plt.tight_layout()
plt.savefig("figures/SST")
# plt.show()
```

Listing 23: Code for Average sea surface temperature

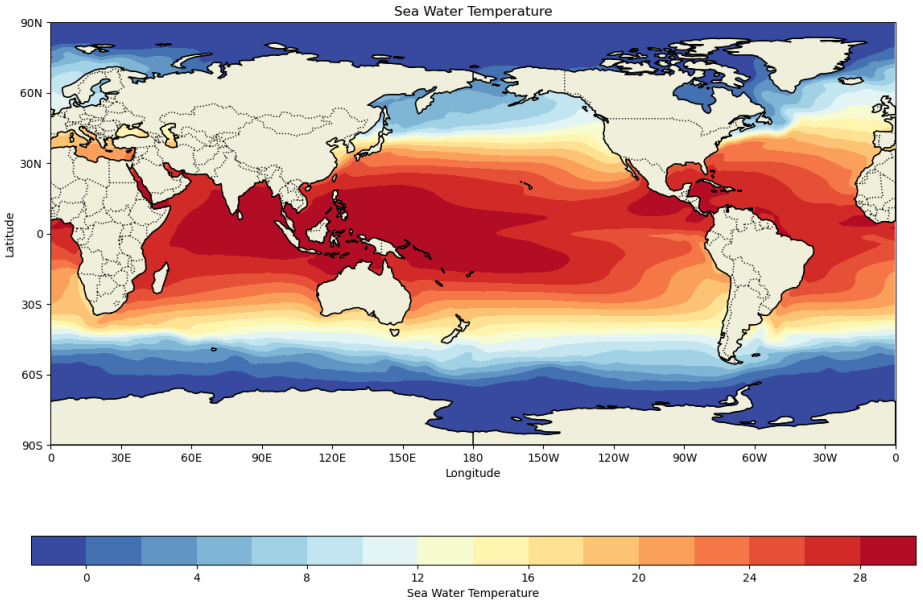

Figure 26: Average Sea Surface Temperature

Figure 26 illustrates the average sea surface temperature. As the equatorial regions get the high amount of solar flux, the average SST at this region are very high at about 30°C, and then forming a decreasing gradient of temperature toward the two poles reaching the lowest temperature of about -2°C

7.1.2 Sea Surface Temperature Departure

Sea surface temperature departure (anomaly) refers to the deviation from average temperature conditions. In this particular analysis, we calculate the localized departure from longitudinal average sea temperature in order to to reveal the horizontal movements of the ocean current.

```
# SST departure
sst_average = np.mean(sst, axis=(2))
sst_departure = sst - np.expand_dims(sst_average, axis=(2))

max_abs_value = np.max(np.abs(sst_departure))

# Create a figure and axes
fig = plt.figure(figsize=(16,8))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude
    =180))

# Plot the sea water temperature array using imshow
im = ax.contourf(long, lat, np.mean(sst_departure, axis=0), cmap =
    cmap_reversed, transform=ccrs.PlateCarree(), levels = 20, vmin = -6, vmax
     = 7)

# Add land and coastline features
ax.add_feature(cfeature.COASTLINE, edgecolor='black')
ax.add_feature(cfeature.LAND, edgecolor='black', zorder = 1)
ax.add_feature(cfeature.BORDERS, linestyle=':')

# Add a colorbar
cbar = fig.colorbar(im, ax=ax, orientation='horizontal', label='SST Anomaly'
    , shrink = 0.7, aspect=30)

# Set the title
ax.set_title('SST Departure')

# Show the plot
plt.tight_layout()
plt.savefig("figures/SST_departure")
# plt.show()
```

Listing 24: Sea surface temperature departure

Figure 27 illustrates the sea surface temperature anomaly over the longitude. The range of the anomaly is either higher than average about 7.2 °C or lower than the average about 8 °C. The most noticeable detail in the figure is the high temperature anomaly dominating on the East side of the Pacific Ocean, while the low temperature anomaly concentrates on the West side of Pacific Ocean.

The figure indicates the underlying dynamics of ocean currents driven by the prevailing winds.

Trade winds push warm surface waters eastward in the Pacific Ocean, leading to the formation of an upwelling zone in the west. Upwelling involves the ascent of cold, nutrient-rich waters to the surface. Conversely, the arrival of warm waters causes a buildup of water mass, resulting in the sinking and downward movement of surface waters. These processes, combined with atmospheric movements influenced by the creation of high and low-pressure systems, give rise to a circulation pattern known as the Walker Cell.

These ocean temperature patterns are closely linked to the El nino-Southern Oscillation (ENSO) phenomenon which including two extreme events happens in 2-7 years cycle which disrupts the normal climate patterns, resulting to some extreme weather condition. These phenomenon will be analyzed closely in the Osciallltion section.

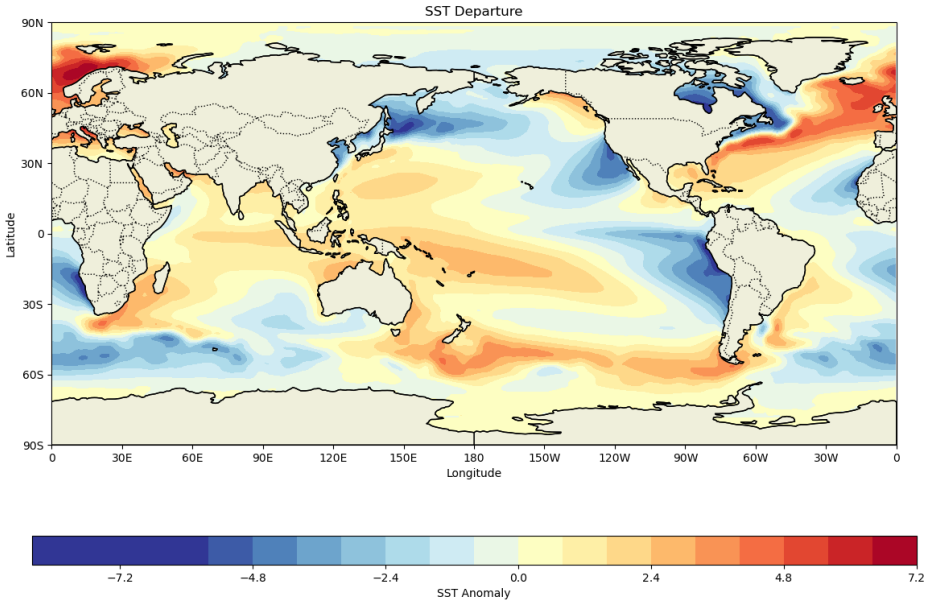

Figure 27: Sea Surface Temperature Departure

8 Statistics with rainfall amount and mean temperature

8.1 Practice 1

I. The daily 2m-temperature data at THAIBINH station can be downloaded at:

https://remosat.usth.edu.vn/Download/dat_THAIBINH/T2m_THAIBINH_1961_2019.txt

1. Write a Python program to read the daily temperature data at THAIBINH station.
2. Estimate monthly values for each year.
3. Plot the monthly-mean climatology at THAIBINH station.

We read the daily mean temperature data of Thai Binh station from 1961 to 2019, then for each year, we calculate the monthly-mean temperature and stor the data for further analysis.

```
def monthly_mean(file):
    df_tb = pd.read_csv(file, header=None, delim_whitespace=True, comment='#',na_values=[-99.99, -1])
    mean_mon = []
    for i in range(59):
        mean_mon.append(np.nanmean(df_tb.iloc[i*31:i*31+31, 1:], axis=0))
    mean_mon = np.asarray(mean_mon)
    return mean_mon
```

Listing 25: Code for defining the function for file reading and monthly-mean calculating

```
mean_mon_T = monthly_mean('T2m_THAIBINH_1961_2019.txt')
mean_YofM_T = np.mean(mean_mon, axis=0)
plt.figure(figsize=(20, 10))
color_palette = plt.get_cmap('jet')
trend = np.zeros(12)
for i in range(12):
    # Calculate linear regression parameters
    regression_line = np.polyfit(np.arange(1961, 2020), mean_mon_T.T[i], 1)
    plt.plot(np.arange(1961, 2020), np.polyval(regression_line, np.arange(1961, 2020)), '--', color=color_palette(i/12), linewidth=1, alpha = 0.7)

    plt.plot(np.arange(1961,2020), mean_mon_T.T[i], 'o-', label = months[i], markersize = 4, color = color_palette(i/12))


plt.xlabel("Years")
plt.ylabel(r"Temperature ($^o$C)")
plt.legend(framealpha = 0.7, edgecolor = 'black')

# plt.savefig('figures/meanT_MoY_1.png')
plt.show()
```

Listing 26: Code for calculating and plotting monthly-mean temperature of Thai Binh in period 1961-2019

Figure 28 presents the time series data of monthly-mean temperatures spanning the period from 1961 to 2019. A visual examination of the figure reveals that the summer months, including June, July, and August, consistently exhibit the highest temperatures. The temperature range during these months is relatively narrow, with values oscillating between 27-30 °C. In contrast, the winter months, particularly December and January, showcase the lowest temperatures, typically ranging from around 12-20 °C. Moreover, there is noticeable variability in temperature between each year during the winter season, indicating wider fluctuations.

I have also incorporated trend lines for each year in the plot, represented by the dash lines. These trend lines provide a visual representation of the overall increasing trend for each month. The presence of upward-sloping trend lines indicates a consistent warming process in recent years.

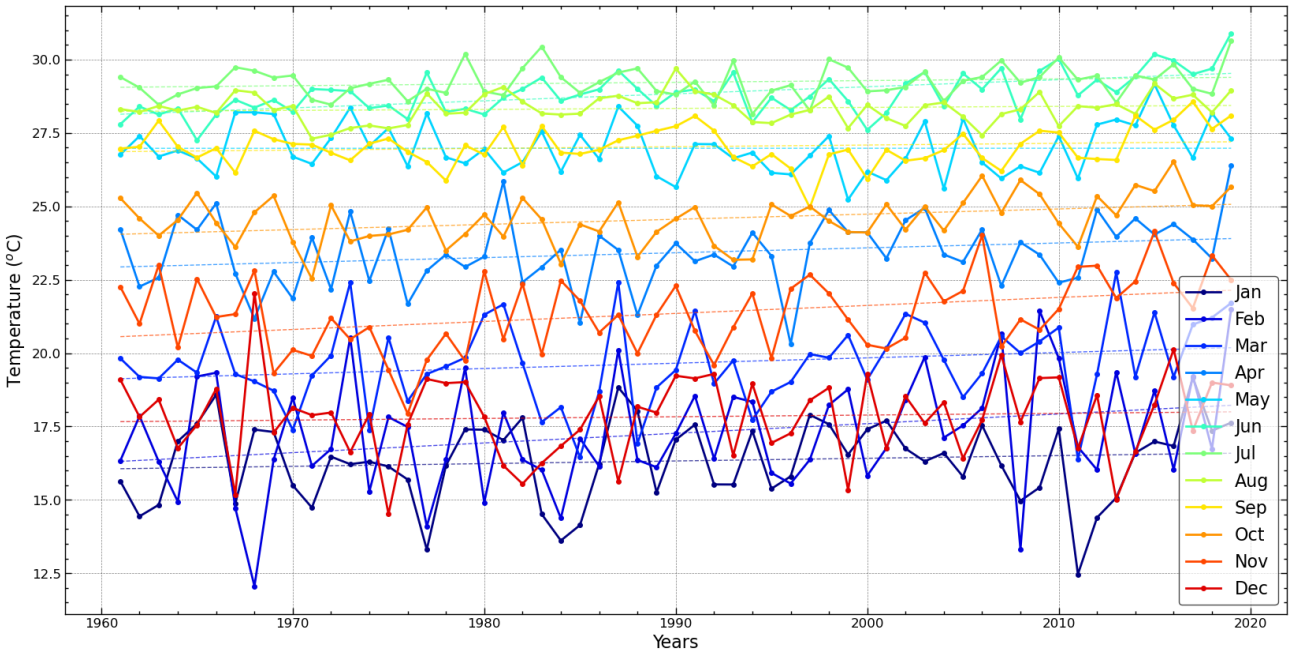

Figure 28: Monthly-mean temperature from 1961 to 2019

```
plt.figure(figsize=(12,8))

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', '
    Oct', 'Nov', 'Dec']
plt.plot(np.arange(1,13), mean_mon.T, alpha = 0.1, color = 'gray')
plt.plot(np.arange(1,13), mean_YofM, 'o--',color = 'red')

plt.xticks(np.arange(1, 13), months)  # Set the tick locations and labels
plt.xlabel("Months")
plt.ylabel(r"Temperature ($^o$C)")
plt.show()
```

Listing 27: Code for plotting the seasonal trend of monthly-mean temperature

Figure 29 displays the seasonal cycle of temperature throughout a year, obtained by averaging the temperature data over a 59-year period. The temperature cycle exhibits distinct patterns, with the lowest temperatures occurring in January at approximately 16°C. As the months progress, the temperature gradually rises, reaching its peak in July at around 29°C. Following July, the temperature begins to decline, resulting in decreasing temperatures for the remainder of the year.

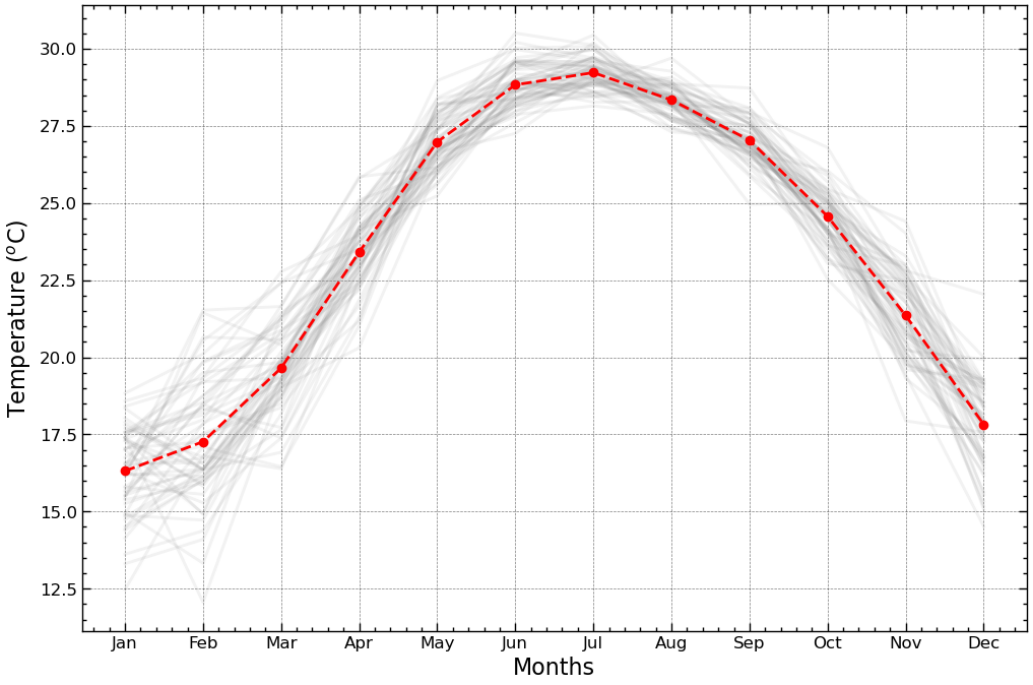

Figure 29: Seasonal trend of monthly-mean temperature

II. Download the daily rainfall data at THAIBINH station:

https://remosat.usth.edu.vn/Download/dat_THAIBINH/R_THAIBINH_1961_2019.txt

1. Repeat the above tasks for rainfall.
2. Homework: Estimate statistics for each month (mean, median, variance, standard deviation, 10th percentile, 90th percentile).
3. Homework: Compute at least 5 extreme indices from daily data at THAIBINH station.

Monthly-mean rainfall amount at Thai Binh in 1961-2019 period

```
mean_mon_R = monthly_mean('R_THAIBINH_1961_2019.txt')
mean_YofM_R = np.mean(mean_mon_R, axis=0)

plt.figure(figsize=(20, 10))
for i in range(12):
    plt.plot(np.arange(1961,2020), mean_mon.T[i], 'o-', label = months[i],
   markersize = 4, color = color_palette(i/12))

plt.xlabel("Years")
plt.ylabel(r"Rainfall amount (mm)")
plt.legend(framealpha = 0.7, edgecolor = 'black')

plt.show()
```

Listing 28: Code for monthly-mean rainfall amount at Thai Binh in 1961-2019

In this study, a similar analysis was done on rainfall data in Thai Binh spanning from 1961 to 2019. Figure 30 presents the results, showcasing the monthly-mean rainfall amounts over the 59-year period. The findings reveal that the highest rainfall levels occur during the summer and autumn months (August, September and October). This can be attributed to the elevated temperatures during this time, which promote increased evaporation and subsequently higher precipitation rates. Conversely, the overall trend indicates lower rainfall amounts during the winter months (December, January, February).

The figure displays several pronounced peaks, indicating the occurrence of extreme events such as tropical cyclones or heavy rainfalls during certain months. These peaks signify instances where unusually high rainfall amounts were recorded, likely associated with these extreme weather phenomena.

```
plt.figure(figsize=(12,8))

plt.plot(np.arange(1,13), mean_mon_R.T, alpha = 0.1, color = 'gray')
plt.plot(np.arange(1,13), mean_YofM_R, 'o--',color = 'red')

plt.xticks(np.arange(1, 13), months)  # Set the tick locations and labels
```

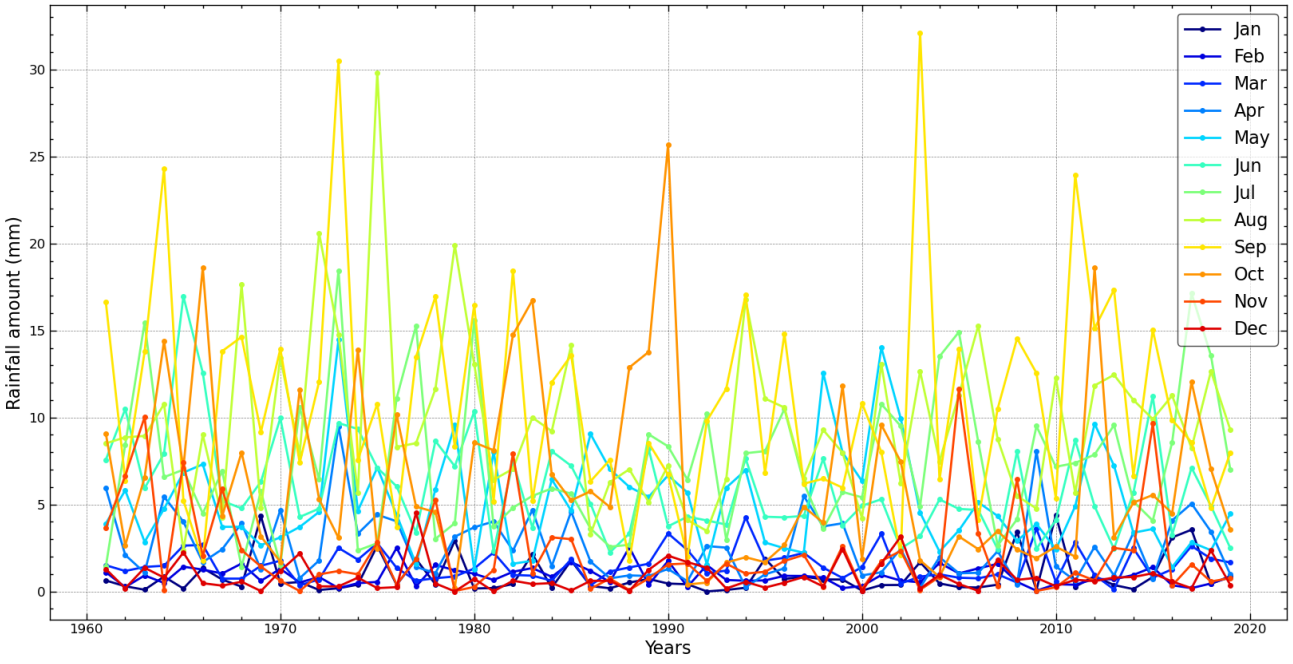

Figure 30: Monthly-mean rainfall at Thai Binh from 1961 to 2019

```
plt.xlabel("Months")
plt.ylabel(r"Rainfall amount (mm)")

plt.savefig('figures/mean_rain_MoY')
# plt.show()
```

Listing 29: Code for plotting seasonal trend of rainfall amount

The overall trend of rainfall amounts in Thai Binh is illustrated in Figure 31. The figure highlights a significant range of rainfall variability, particularly during the months of August, September, and October. This period coincides with the peak of the rainy season in Vietnam, characterized by heavy rainfall events and the potential for tropical cyclones leading to other natural hazard like flooding and hurricanes.

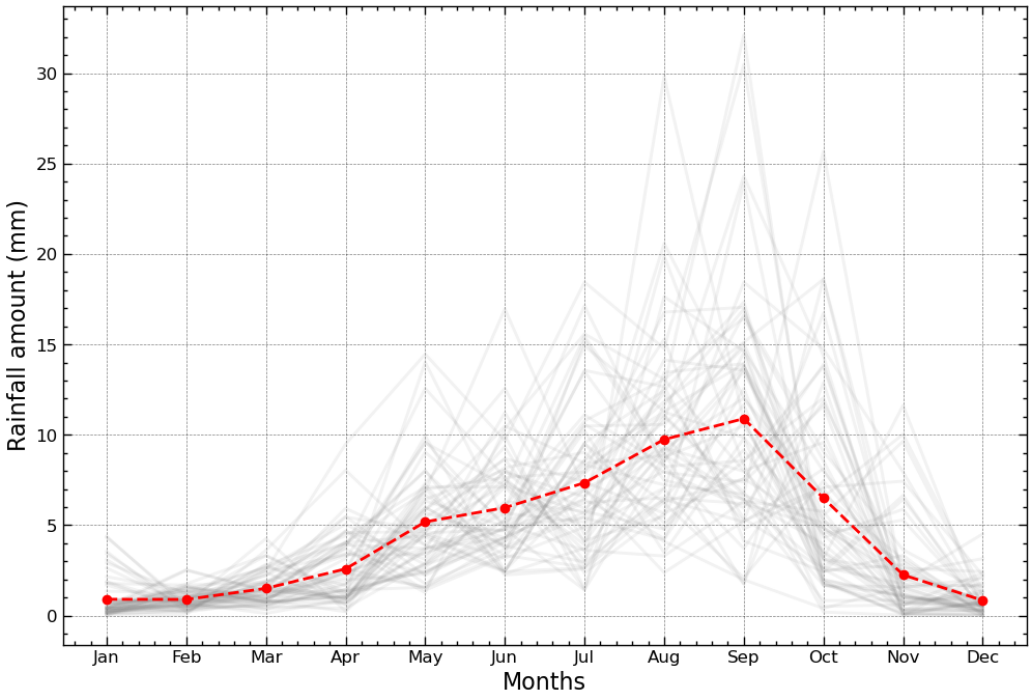

Figure 31: Seasonal rainfall amount at Thai Binh from 1961 to 2019

Statistics for each month

This practice is dedicated for calculation of the mean, median, variance, standard deviation, 10th percentile and 90th percentile of the rainfall amount spanning from 1961 to 2019.

```
pd.set_option('display.precision', 2)

mean_YofM = np.mean(mean_mon_R, axis=0)
median_YofM = np.median(mean_mon_R, axis=0)
var_YofM = np.var(mean_mon_R, axis=0)
std_YofM = np.std(mean_mon_R, axis=0)
percent_10 = np.percentile(mean_mon_R, 10, axis=0)
percent_90 = np.percentile(mean_mon_R, 90, axis=0)

stat_YofM = {'Mean':mean_YofM, 'Median': median_YofM, 'Var': var_YofM, 'std'
    : std_YofM, '10th percentile': percent_10, '90th percentile': percent_90}
df_stat = pd.DataFrame(data=stat_YofM, index=months)
```

The statistics give the same conclusions as I proposed before, the mean rainfall amount is highest in September with a huge uncertainty characterized by the standard deviation, the range from 10th to 90th percentile is about 13 mm which is quite huge.

Month	Mean	Median	Variance	Standard Deviation	10th Percentile	90th Percentile
Jan	0.90	0.46	1.40	1.18	0.09	2.25
Feb	0.89	0.81	0.29	0.54	0.28	1.59
Mar	1.50	1.35	0.79	0.89	0.61	2.69
Apr	2.58	2.10	3.75	1.94	0.74	4.83
May	5.20	4.64	8.65	2.94	2.13	9.15
Jun	5.95	5.05	8.34	2.89	2.46	9.73
Jul	7.32	6.44	17.32	4.16	2.67	14.91
Aug	9.73	8.94	22.80	4.78	4.67	14.87
Sep	10.86	9.17	40.89	6.39	4.06	17.11
Oct	6.47	4.85	28.71	5.36	1.69	13.99
Nov	2.24	1.23	7.06	2.66	0.12	6.48
Dec	0.84	0.61	0.73	0.85	0.05	2.06

Table 2: Statistics of rainfall amount for each month

Compute 5 extreme indices from daily data at Thai Binh station

With 5 extreme indices, we will use the data of rainfall amount in Thai Binh, because in order to calculate the extreme temperature indices, we need the temperature on days and nights or maximum daily temperature not the mean temperature. I chose 5 extreme precipitation indices that are fairly easy to compute:

- Very wet day precipitation: Annual total precipitation when daily precipitation $>$ 95th percentile
- Very heavy precipitation days: Annual count of days with daily precipitation $\geq$ 30 mm
- Maximum 1-day precipitation: Annual maximum 1-day precipitation total
- Annual total wet day precipitation: Annual total precipitation on wet days (daily precipitation $\geq$ 1 mm)
- Simple daily intensity: Annual total precipitation divided by the number of wet days (daily precipitation $\geq$ 1 mm)

```
df_tb_R = pd.read_csv('R_THAIBINH_1961_2019.txt', header=None,
    delim_whitespace=True, comment='#',na_values=[-99.99, -1])
indices = ['Very wet day precip', 'Very heavy precip days', 'Annual-total
    wet day precip','Maximum 1-day', 'Simple daily intensity']
years = np.arange(1961, 2020)

very_wet_day = np.empty(59)
very_heavy_days = np.empty(59)
anual_total_wet_day = np.empty(59)
```

```
maximum = np.empty(59)
simple_daily_intensity = np.empty(59)
for i in range(59):
    very_wet_day[i] = np.nansum(df_tb_R[i*31:(i+1)*31][df_tb_R[i*31:(i+1)*31] > np.nanpercentile(df_tb_R[i*31:(i+1)*31], 95)])
    very_heavy_days[i] = np.count_nonzero(df_tb_R[i*31:(i+1)*31] >= 30)
    anual_total_wet_day[i] = np.nansum(df_tb_R[i*31:(i+1)*31][df_tb_R[i*31:(i+1)*31] >= 1])
    maximum[i] = np.nanmax(df_tb_R[i*31:(i+1)*31])
    simple_daily_intensity[i] = np.nansum(df_tb_R[i*31:(i+1)*31])/np.count_nonzero(df_tb_R[i*31:(i+1)*31] >= 1)

extr_indices_dict = {indices[0]: very_wet_day, indices[1]: very_heavy_days, indices[2]: anual_total_wet_day, indices[3]: maximum, indices[4]: simple_daily_intensity}
extr_indices_df = pd.DataFrame(data=extr_indices_dict, index=years)
```

Listing 30: Code for computing 5 extreme precipitation indices

The result of extreme precipitation indices for each year is presented in table 3. The result is too long, so I just show the indices from 1990 to 2019.

Year	Very wet day precipitation	Very heavy precipitation days	Annual total wet day precipitation	Maximum 1-day precipitation	Simple daily precipitation intensity
1990	1601.8	18.0	2590.2	300.3	19.06
1991	748.8	7.0	1388.7	143.7	13.44
1992	888.9	13.0	1601.3	113.7	13.24
1993	816.7	13.0	1653.2	81.2	14.57
1994	1210.5	22.0	2466.0	169.4	18.05
1995	845.5	12.0	1733.0	100.8	14.69
1996	941.0	21.0	2070.7	145.8	17.33
1997	718.6	12.0	1778.1	62.4	13.54
1998	1064.9	18.0	2020.9	192.2	18.27
1999	1002.9	22.0	2130.1	86.0	16.41
2000	846.5	15.0	1582.1	91.4	14.22
2001	1220.4	25.0	2618.7	115.1	16.82
2002	1009.4	16.0	1917.1	122.1	16.20
2003	1526.7	18.0	2383.2	512.3	22.88
2004	922.6	12.0	1787.2	104.2	13.45
2005	1536.2	22.0	2505.8	289.7	21.08
2006	954.5	13.0	1915.2	126.0	15.36
2007	808.1	15.0	1684.1	72.7	14.41
2008	942.7	18.0	1983.5	98.3	15.87
2009	1071.0	17.0	1939.0	114.5	18.05
2010	901.7	12.0	1742.0	100.4	15.19
2011	1224.1	19.0	2251.2	194.9	18.17
2012	1461.2	25.0	2734.3	393.0	19.02
2013	979.0	17.0	1960.0	104.0	15.92
2014	956.0	18.0	2034.4	137.0	14.35
2015	1323.0	18.0	2273.0	182.0	19.75
2016	1263.0	23.0	2307.0	199.0	19.50
2017	1001.0	21.0	2256.0	93.0	16.73
2018	926.0	19.0	2187.9	106.0	16.68
2019	752.0	12.0	1460.0	60.0	12.92

Table 3: Precipitation Extreme Indices

8.2 Practice #2

Given the ThaiBinh rainfall data http://remosat.usth.edu.vn/~thanhnd/Download/dat_THAIBINH/

1. Estimate RX1day for each year.
2. Plot the distribution of RX1day.
3. Calculate the return value of RX1day for return periods of 5 years, 10 years, 20 years, and 100 years.
4. *Repeat the above steps for the rainfall amounts of heavy rainfall days (days with rainfall* $\geq$ *50mm).*

```
file = 'R_THAIBINH_1961_2019.txt'
df_tb = pd.read_csv(file, header=None, delim_whitespace=True, skiprows=3,
    na_values=[-99.99, -1])
len_year = 2019 - 1961
for i in range(len_year):
    df_tb.drop(31+32*i, inplace=True)
df_tb.astype('float')
df_tb.drop(0, axis=1, inplace=True)

year = np.arange(1961, 2020)
```

Listing 31: Code for reading rainfall amount data at Thai Binh station from 1961 to 2019

We use the Pandas library to read the daily precipitation data at Thai Binh station spanning 59 years from 1961 to 2019. we can remove all unnecessary lines that contain the year information, and set the data frame's data type to 'float'. As a result, we now have data for each year, where each year consists of 31 lines. To calculate the precipitation for a specific period, such as 1 year or 5 years, we can simply multiply the corresponding number of years by 31.

Highest one day precipitation amount (RX1day)

```
RX1 = np.empty(len_year + 1)
for i in range(len_year + 1):
    RX1[i] = np.nanmax(np.asarray(df_tb.iloc[i*31: (i+1)*31]), axis=(0,1))

plt.figure(figsize = (12,8))
plt.bar(year, RX1, color = 'cornflowerblue')
plt.xlabel("Year")
plt.ylabel("Rainfall amount (mm)")

plt.title("RX1 days")
plt.savefig('figures/RX1days_1')
# plt.show()
```

Listing 32: Code for RX1 day in 1-year period

Figure 32 illustrate the highest daily precipitation amount in each year. Normally, the values fall into the range of 100-200 mm. However, there are some years with really high RX1day index, the highest is about 512.3 mm in year 2003 and 393 mm in 2012. In order to calculate

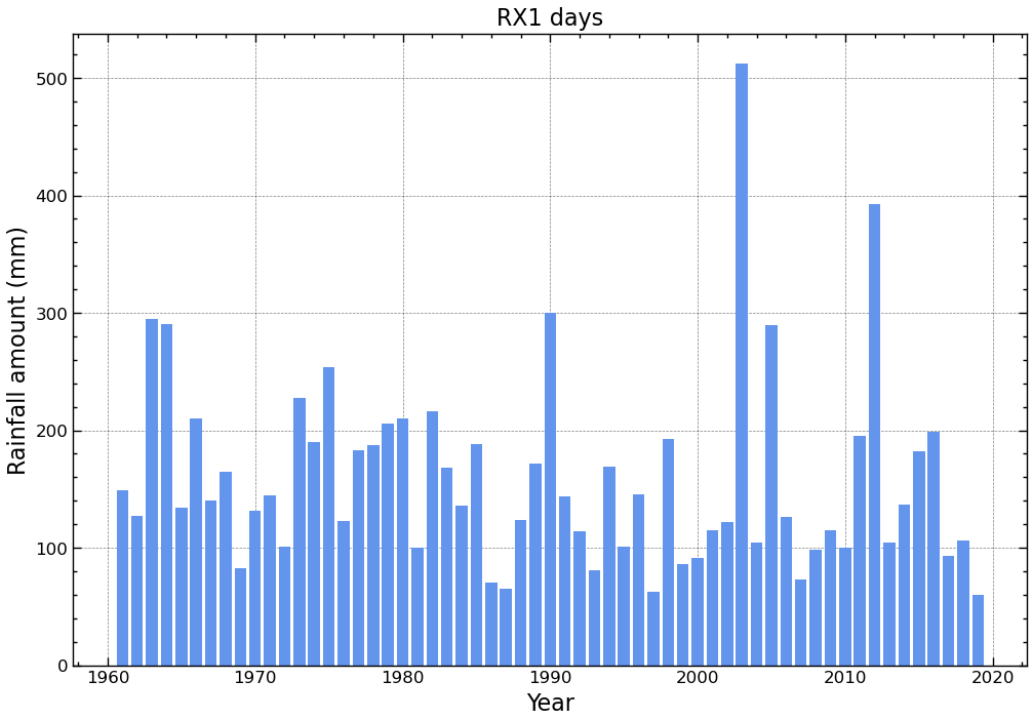

Figure 32: RX1 days in 1-year period

the RX1day index for 5-year, 10-year and 20-year, we create a function which takes the period as an input and calculates the number of years based on the length of the data. It initializes empty arrays for storing the RX1day values and corresponding years. Then, it iterates over the number of years and calculates the maximum value using np.nanmax for the selected period. Finally, it returns the arrays of years and RX1day values. The 100-year return value is just the value of the whole 59-year course.

```
def RX1_function(period):
    num_year = (len_year + 1)//period
    RX1 = np.empty(num_year)
    year = np.empty(num_year)
    for i in range(num_year):
        RX1[i] = np.nanmax(np.asarray(df_tb.iloc[i*31*5: (i+1)*31*5]), axis
   =(0,1))
```

```
        year[i] = 1961 + i*period
    return year, RX1

year_5, RX1_5 = RX1_function(5)
year_10, RX1_10 = RX1_function(10)
year_20, RX1_20 = RX1_function(20)
RX1_100 = np.nanmax(df_tb)

fig, ax = plt.subplots(2,2,figsize = (20, 15))

ax[0,0].bar(year_5, RX1_5, color = 'limegreen')
ax[0,0].set_xlabel("Year")
ax[0,0].set_ylabel("rainfall amount (mm)")
ax[0,0].set_title("RX1 days (5-year period)")

ax[0,1].bar(year_10, RX1_10, color = 'gold')
ax[0,1].set_xlabel("Year")
ax[0,1].set_title("RX1 days (10-year period)")

ax[1,0].bar(year_20, RX1_20, color = 'red')
ax[1,0].set_xlabel("Year")
ax[1,0].set_ylabel("rainfall amount (mm)")
ax[1,0].set_title("RX1 days (20-year period)")

ax[1,1].bar('1961', RX1_100, color = 'cornflowerblue')
ax[1,1].set_xlim([-2, 2])
ax[1,1].set_xlabel("Year")
ax[1,1].set_title("RX1 days (1-year period)")

plt.tight_layout()
plt.savefig('figures/RX1days')

# plt.show()
```

Listing 33: Code for return value of RX1day

Then we display the results of different periods in different subplots in figure 33.

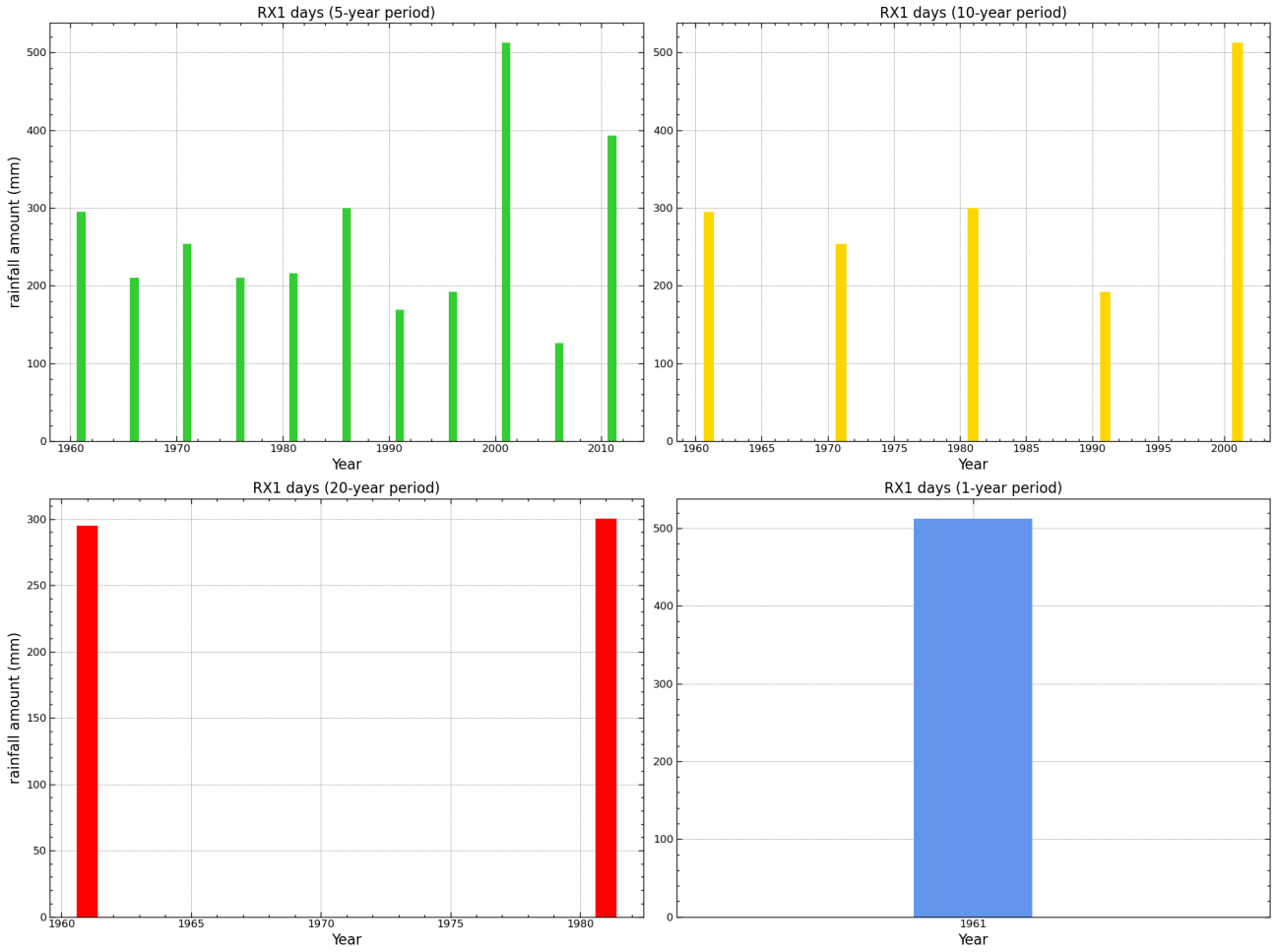

Figure 33: RX1 days in 1-year, 5-year, 10-year and 20-year periods

The Generalized Extreme Value (GEV) distribution

The Generalized Extreme Value (GEV) distribution is commonly used to model the maximum or minimum values (block maxima) of a given variable over a certain time period. In the context of the RX1day index, which represents the maximum 1-day rainfall, the GEV distribution can be employed to analyze and characterize extreme rainfall events.

The cumulative distribution function (CDF) of the block maxima z in the GEV distribution is defined as:

$$G(z) = \exp\left[-\left(1 + \frac{\xi(z-\mu)}{\sigma}\right)^{-1/\xi}\right] \tag{33}$$

This distribution has three parameters:

1. Location parameter (μ): This parameter represents the shifting or displacement of the distribution along the x-axis. It indicates the average value or the location of the extreme events.
2. Scale parameter (σ): The scale parameter controls the spread or variability of the distribution. It determines the magnitude of the extreme events.
3. Shape parameter (ξ): The shape parameter characterizes the shape of the distribution. If $\xi > 0$, it follows a Fréchet distribution, indicating a heavy-tailed distribution with a longer upper tail. If $\xi < 0$, it follows a Weibull distribution, indicating a lighter-tailed distribution.

By estimating these three parameters from the RX1day index data, we can gain insights into the characteristics of extreme rainfall events, including their average intensity, variability, and tail behavior. This information is valuable for understanding the risks associated with heavy rainfall events and for designing appropriate measures to manage and mitigate their impacts.

We plot a histogram of RX1day to visualize the distribution. To fit a Generalized Extreme Value (GEV) distribution to this data, we can use the `genextreme` module in the `scipy` package.

```
from scipy.stats import genextreme as gev

file = 'R_THAIBINH_1961_2019.txt'
df_tb = pd.read_csv(file, header=None, delim_whitespace=True, skiprows=3,
    na_values=[-99.99, -1])
len_year = 2019 - 1961
for i in range(len_year):
    df_tb.drop(31+32*i, inplace=True)
df_tb.astype('float')
df_tb.drop(0, axis=1, inplace=True)

RX1 = np.empty(len_year + 1)
for i in range(len_year + 1):
    RX1[i] = np.nanmax(np.asarray(df_tb.iloc[i*31: (i+1)*31]), axis=(0,1))

shape, loc, scale = gev.fit(RX1)

fig, ax = plt.subplots(figsize = (12,8))
n, bins, patches = ax.hist(RX1, bins=50, density=True, alpha=0.6, label='
    Histogram of RX1')
x = np.linspace(0, bins[-1], 100)
pdf = gev.pdf(x, shape, loc, scale)
ax.plot(x, pdf, 'ro', label=f'GEV ({shape:.2f}, {loc:.2f}, {scale:.2f})')
ax.set_xlabel('X-axis label')
ax.set_ylabel('Density')
ax.set_title('Histogram with Fitted GEV Distribution')
```

```
25 ax.legend()
26
27 plt.show()
```

Listing 34: Code for GEV Distribution

The figure 34 shows the distribution of RX1day following the Generalised Extreme Value (GEV) distribution. The fitting curve suggests that the shape parameter, denoted as ξ, is less than 0, indicating that the distribution follows a Weibull distribution.

The mean of the highest rainfall amount per year is approximately 120 mm, with a scale factor of about 48.86 mm. Notably, there are some high anomaly values observed at rainfall amounts of 100 mm, 200 mm, and 300 mm.

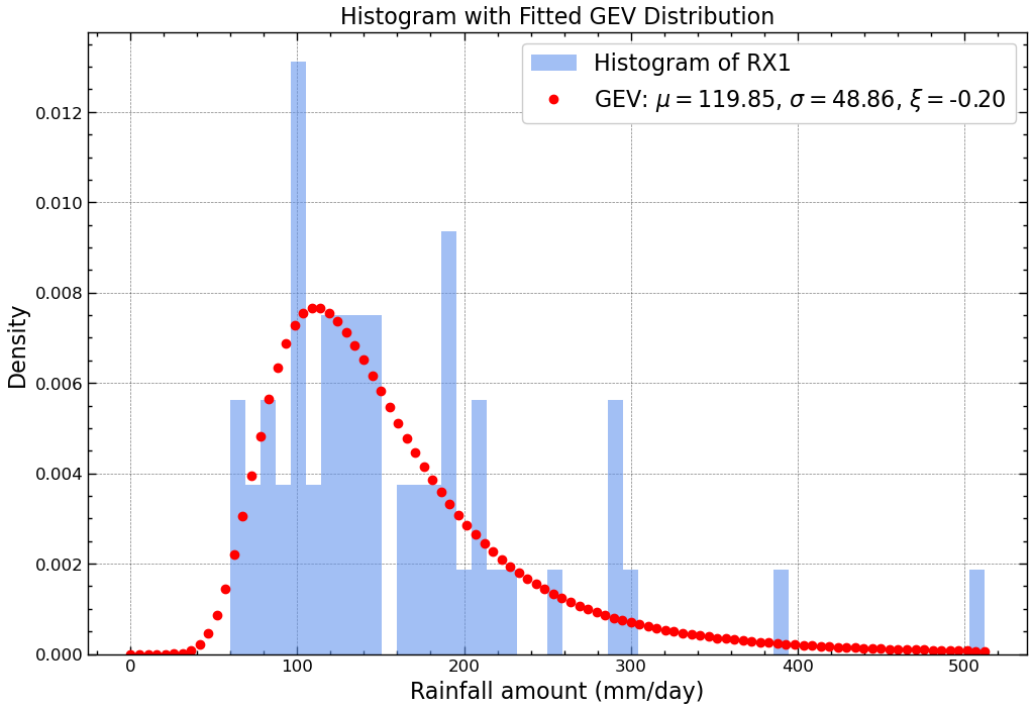

Figure 34: GEV distribution

Days with heavy rainfall $\geq$ 50 mm (R50 days)

In this practice, the focus is on counting the number of heavy rainfall days in a year. The process involves extracting all the dates with a rainfall amount greater than or equal to 50 mm and subsequently counting the occurrences.

To accomplish this, we filter the dataset to retain only the dates with rainfall amounts of 50 mm or higher. Once the filtering is complete, we can count the number of remaining records, which corresponds to the number of heavy rainfall days in a year.

```
from scipy.stats import genpareto
R50 = df_tb.values[df_tb > 50]

R50_year = np.empty(59)
for i in range(59):
    R50_year[i] = np.count_nonzero(df_tb[i*31:(i+1)*31] > 50)

plt.figure(figsize = (12,8))
plt.bar(year, R50_year, color = 'cornflowerblue')
plt.xlabel("Year")
plt.ylabel("R50 days")

plt.title("R50 days")
plt.savefig('figures/R50days_1')
# plt.show()
```

Listing 35: Code for R50 days each year

Figure 35 displays the R50day values for each year, representing the number of heavy rainfall days. On average, there are approximately 5 to 12 days of heavy rainfall per year. However, in certain year such as 1981, the number of heavy rainfall days go up to 20.

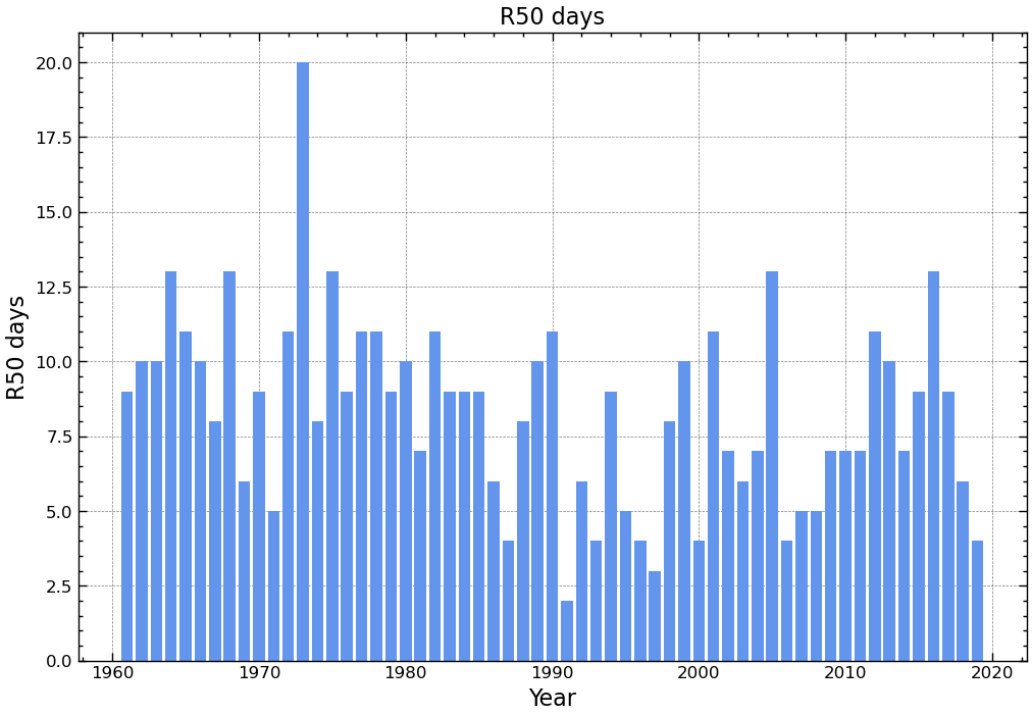

Figure 35: R50 days for each year

The calculation of R50day in the period of 5-year, 10-year, 20-year and 100-year is the same with what we do with RX1day.

```
def R50_function(period):
    num_year = (len_year + 1)//period
    year = np.empty(num_year)
    R50_year = np.empty(num_year)
    for i in range(num_year):
        R50_year[i] = np.count_nonzero(df_tb[i*31*period:(i+1)*31*period] >
   50)
        year[i] = 1961 + i*period
    return year, R50_year

year_5, R50_5 = R50_function(5)
year_10, R50_10 = R50_function(10)
year_20, R50_20 = R50_function(20)
R50_100 = np.count_nonzero(df_tb > 50)

fig, ax = plt.subplots(2,2,figsize = (20, 15))

ax[0,0].bar(year_5, R50_5, color = 'limegreen')
ax[0,0].set_xlabel("Year")
ax[0,0].set_ylabel("R50 days")
```

```
ax[0,0].set_title("RX1 days (5-year period)")

ax[0,1].bar(year_10, R50_10, color = 'gold')
ax[0,1].set_xlabel("Year")
ax[0,1].set_ylabel("R50 days")
ax[0,1].set_title("RX1 days (10-year period)")

ax[1,0].bar(year_20, R50_20, color = 'red')
ax[1,0].set_xlabel("Year")
ax[1,0].set_ylabel("R50 days")
ax[1,0].set_title("RX1 days (20-year period)")

ax[1,1].bar('1961-2019' ,R50_100, color = 'cornflowerblue', width=0.5)
ax[1, 1].set_xlim([-1.5, 1.5])
ax[1,1].set_xlabel("Year")
ax[1,1].set_ylabel("R50 days")
ax[1,1].set_title("RX1 days (100-year period)")

plt.tight_layout()
plt.savefig('figures/R50days')

# plt.show()
```

Listing 36: Code for return value of R50 for 5-year, 10-year, 20-year and 100-year period

Figure 36 illustrates the R50day values over different periods. We can see that the R50day increases with the increasing of period's time, in 1-year period we only have maximum of 20 days, we have more than 50 days in 5-year period, more than 100 days in 10-year period, more than 200 days in 20-year period and nearly 500 days in the whole 59-year period.

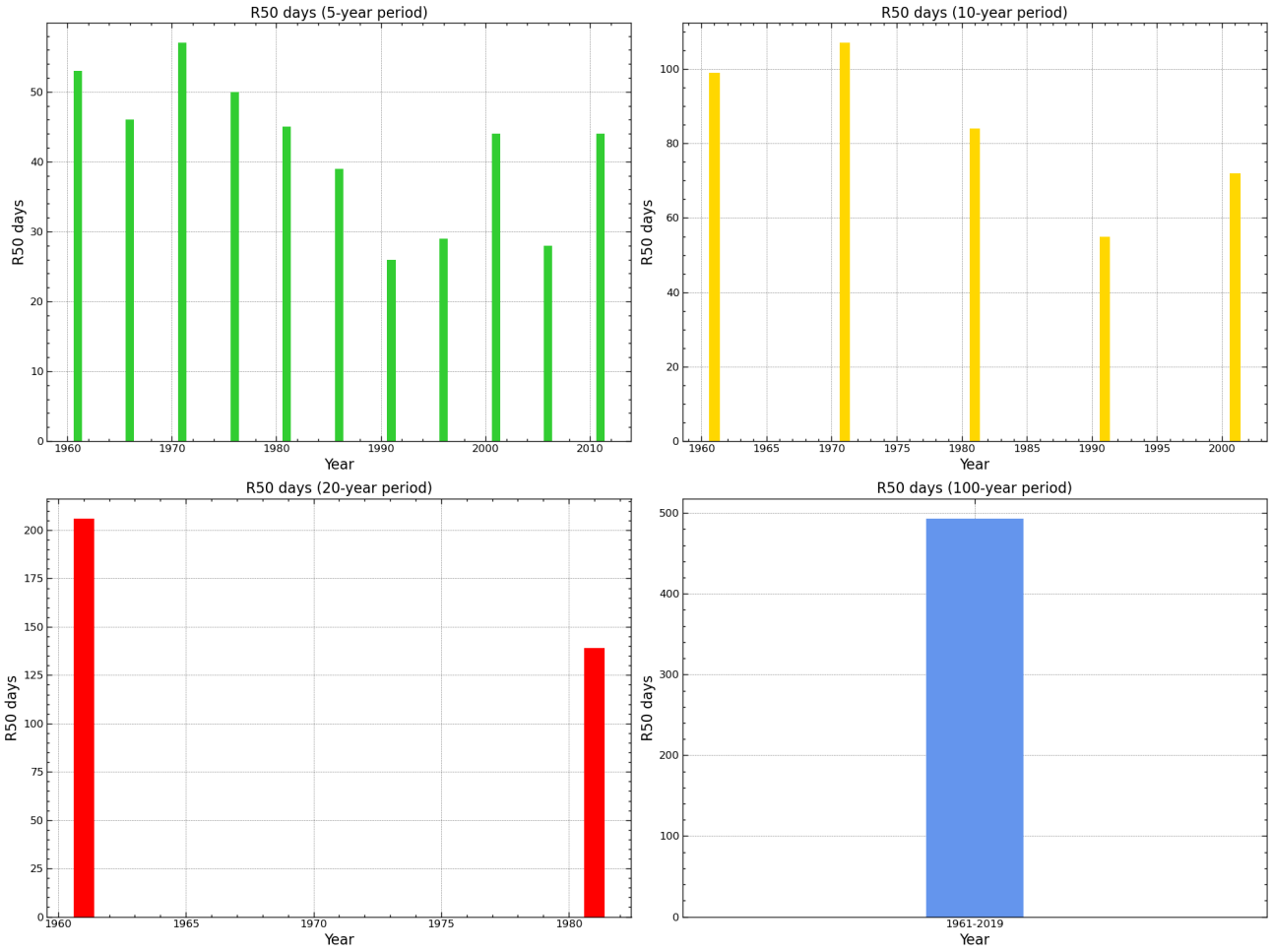

Figure 36: R50 days in 1-year, 5-year, 10-year and 20-year periods

Generalized Parento Distribution

The Generalized Pareto (GP) distribution is commonly used to model threshold exceedances or maxima in various events. In the context of the R50day index, which represents the number of heavy rainfall days, the GP distribution can be employed to analyze and characterize extreme rainfall events above a certain threshold.

The equation of the cumulative distribution function (CDF) for the Generalized Pareto (GP) distribution:

$$H(x) = 1 - \left(1 + \frac{(x-\mu)}{\sigma}\right)^{-\xi} \tag{34}$$

Threshold parameter (μ): This parameter represents the minimum threshold value above which extreme events are considered. It determines the starting point for analyzing extreme events.

Scale parameter (σ): The scale parameter controls the spread or variability of the distribution. It quantifies the magnitude of extreme events relative to the threshold.

Shape parameter (ξ): The shape parameter characterizes the tail behavior of the distribution. If $\xi > 0$, it follows a Pareto distribution, indicating a heavy-tailed distribution where a few extreme events contribute significantly to the overall outcomes. If $\xi < 0$, it follows a Beta distribution, representing a lighter-tailed distribution. When ξ approaches 0, it converges to an exponential distribution.

```
from scipy.stats import genpareto

R50 = df_tb.values[df_tb > 50]

fig, ax = plt.subplots(figsize = (12,8))
n, bins, patches = ax.hist(R50, bins=60, density=True, alpha=0.6, label='
    Histogram of R50', color = 'cornflowerblue')

shape, loc, scale = genpareto.fit(R50, floc = 50)
x = np.linspace(5, bins[-1], 100)
pdf = genpareto.pdf(x, shape, loc, scale)
ax.plot(x, pdf, 'ro', label=f'Pareto: ' + r'$\mu =$' f'{loc:.2f}, '+ r'$\
    sigma =$' f'{scale:.2f}, ' + r'$\xi =$' + f'{shape:.2f}')
ax.set_xlabel('Rainfall amount (mm/day)')
ax.set_ylabel('Density')
ax.set_title('Histogram with Fitted Pareto Distribution')
ax.legend()
plt.savefig('figures/pareto')
# plt.show()
```

Listing 37: Code for Generalized Parento distribution

Figure 37 shows the distribution of 500 heavy rainfall days which following the Generalized Parento distribution, with three parameters $\mu = 50$, $\sigma = 28.34$ and $\xi = 0.22$.

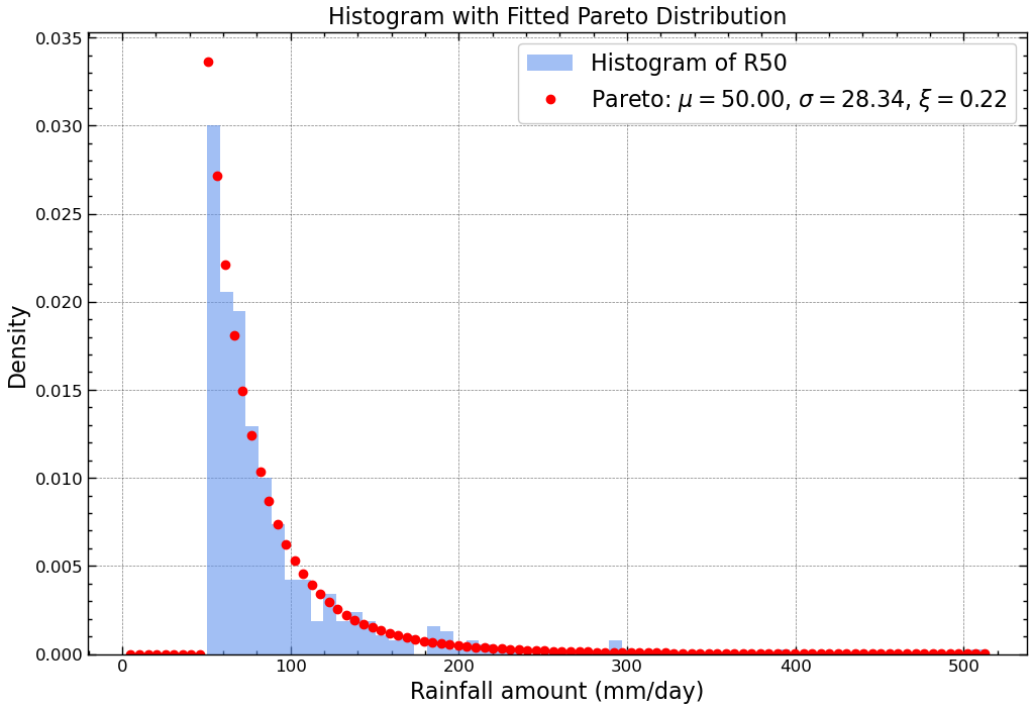

Figure 37: Generalized Pareto distribution

9 Climate Oscillation

Climate oscillations are recurring patterns of climate variability that occur on different time scale. They are driven by complex interactions between the atmosphere, oceans, and other factors. These oscillations influence regional and global climate patterns, affecting weather, temperature, and precipitation.

Question 1 (ENSO):

- Download the monthly NOAA Optimum Interpolation (OI) Sea Surface Temperature (SST) V2 dataset sst.mnmean.nc from https://www.esrl.noaa.gov/psd/data/gridded/data.noaa.oisst.v2.html
- From the SST data, calculate the Nino3 index and identify the El Niño and La Niña years for the period 1981-present.
  - Niño3 area: 5S-5N and 150W-90W.
  - ENSO criteria: five consecutive 3-month running mean SST anomalies exceed the threshold (0.5°C)

Question 2 (QBO:

Download QBO data from http://www.geo.fu-berlin.de/en/met/ag/strat/produkte/qbo/

- From the data downloaded, prove that QBO exists
- Identify QBO's characteristics from the data

9.1 The El Niño-Southern Oscillation (ENSO)

The El Niño-Southern Oscillation (ENSO) is a prominent climate oscillation that occurs in the tropical Pacific Ocean. ENSO is characterized by two extreme phases: El Nino (Spanish for "little boy") and La Nina (Spanish for "little girl"). These phases occur irregularly and typically span over 3-7 years. El Nino refers to the unusual warming of the central and eastern tropical Pacific Ocean, leading to changes in atmospheric circulation and weather patterns worldwide. La Nina, on the other hand, represents the opposite phase with cooler than average sea surface temperatures in the same region.

During El Nino, the trade winds weaken, and warm water accumulates in the central and eastern Pacific. This alters the atmospheric pressure patterns, affecting the distribution of rainfall and temperatures across the globe. El Nino events are associated with increased rainfall in the eastern Pacific, droughts in Southeast Asia and Australia, and altered weather patterns in various regions.

In contrast, during La Nina, the trade winds strengthen, pushing warm surface waters to the western Pacific. This enhances upwelling of cold, nutrient-rich waters along the coasts of South America, leading to cooler sea surface temperatures. La Nina events are often associated with increased rainfall in Southeast Asia and Australia, while other regions may experience drier conditions.

To quantify El Niño and La Niña events, we can utilize monthly Sea Surface Temperature (SST) data as described in Section 7. The provided code in that section can be used to read the SST data from files. By calculating the Nino 3 and Nino 3.4 indices over the period from 1981 to the present, we can identify El Niño and La Niña years.

The equation for calculating Sea Surface Temperature Anomaly (SSTA) is:

$$\text{SSTA} = \text{SST} - \text{SST}_{avg} \tag{35}$$

Where:

SSTA represents the Sea Surface Temperature Anomaly
SST refers to the observed sea surface temperature at a specific location and time
SST_{avg} is the long-term average sea surface temperature for that location

The Nino 3 and Nino 3.4 indices are derived from the SST data and capture the SST anomalies in specific regions. The Nino 3 index focuses on the area bounded by 5°S-5°N and 150°W-90°W

where SST changes are the highest during El Nino. The Nino 3.4 index covers a slightly larger region bounded by 5°S-5°N and 170°W-120°W where SST changes are important for the shift of rainfall area.

```
# Calulate NINO index, & plot the results
latidx_nino3 = (lat >=-5. ) & (lat <=5. )
lonidx_nino3 = (long >=210. ) & (long <=270. )
latidx_nino34 = (lat >=-5. ) & (lat <=5. )
lonidx_nino34 = (long >=190. ) & (long <=240. )

sst_3 = sst[:, latidx_nino3][..., lonidx_nino3]
sst_34 = sst[:, latidx_nino34][..., lonidx_nino34]
#to get the mean values over lon/lat axis
nino3= np.mean(sst_3,axis=(1,2))-np.mean(sst_3)
nino34= np.mean(sst_34,axis=(1,2))-np.mean(sst_34)

plt.figure(1,figsize=(12,6))
plt.plot(d_times, nino3, color = "green", linewidth = 2, label = "Nino 3")
plt.plot(d_times, nino34, color = "blue", linewidth = 3, label = "Nino 3.4")
plt.xlabel('Year')
plt.ylabel('Sea Surface Temperature Anomaly (oC)')
plt.legend(loc='upper left', prop = {'size':10})
plt.title('SSTA over Nino3 and Nino3.4')

plt.show()
```

Listing 38: Code for sea surface temperature anomaly over Nino 3 and Nino 3.4 regions

Figure 38 depicts the sea surface temperature anomaly (SSTA) in the Nino 3 and Nino 3.4 regions. Both regions exhibit a comparable oscillation pattern, indicating a similar variability in sea surface temperature. However, the SSTA in the Nino 3 region tends to have higher magnitudes compared to the Nino 3.4 region.

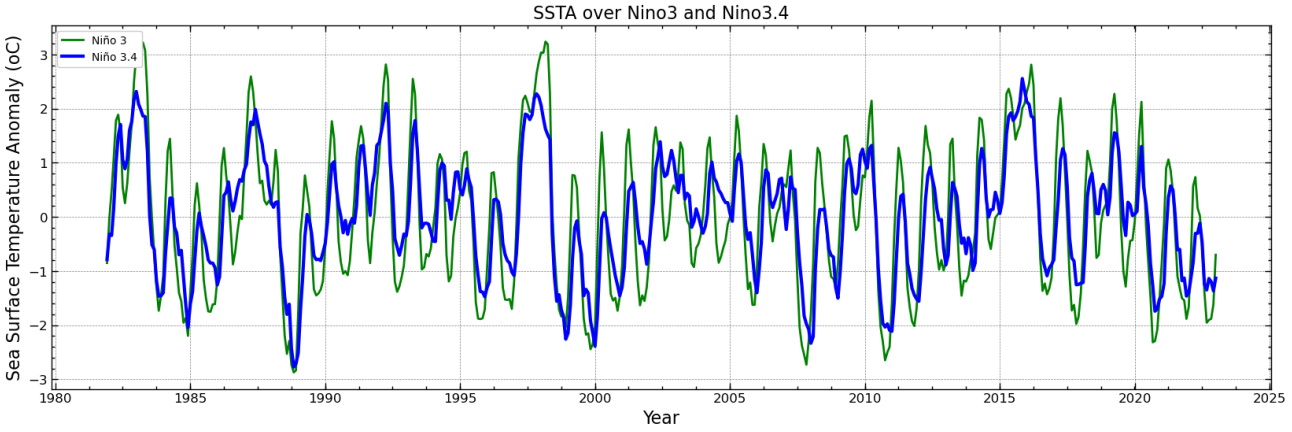

Figure 38: Sea Surface Temperature Anomaly over Nino 3 and Nino 3.4 region

To identify El Nino and La Nina events, we can apply the ENSO criteria. This involves masking the time periods where there are three consecutive days with sea surface temperature (SST) anomalies higher (for El Nino) or lower (for La Nina) than the average SST. We creates boolean masks using a convolution operation and checking if the boolean sum of three consecutive values satisfies the threshold condition for identifying El Niño and La Niña events based on the Nino3 and Nino3.4 index.

By doing so, we can visualize these events using colors. In this case, red color can be used to represent El Nino events, while blue color can be used to represent events occurring in both the Nino 3 and Nino 3.4 regions for comparison.

```
# Define El Nino and La Nina thresholds
el_nino_threshold = 0.5
la_nina_threshold = -0.5

# Create a boolean mask indicating El Nino events
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))

# Create a boolean mask indicating La Nina events
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))

# Plot the results
plt.figure(1, figsize=(20, 3))
plt.plot(d_times, nino3, color="green", linewidth=2, label="Nino 3")
plt.fill_between(d_times, nino3, where=el_nino_mask, color="red", alpha=0.5, label="El Ni o")
plt.fill_between(d_times, nino3, where=la_nina_mask, color="blue", alpha
```

```
    =0.5, label="La Ni a")
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Sea Surface Temperature Anomaly ( C )', fontsize = 8)
plt.legend(loc='upper left', prop={'size': 10}, framealpha = 0.7)
plt.title('El nino and La nina using Nino3 criteria')
plt.show()
```

Listing 39: Code for detecting ENSO over Nino 3

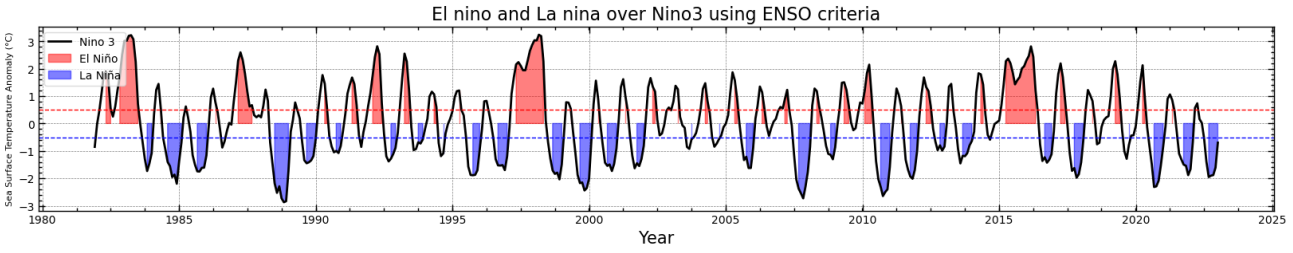

Figure 39: El nino and La nina over Nino3 using ENSO criteria

```
# Create a boolean mask indicating El Nino events
el_nino_mask = np.convolve(nino34 >= el_nino_threshold, np.ones(5), mode='
    valid') == 5
el_nino_mask = np.concatenate((np.zeros(4, dtype=bool), el_nino_mask))

# Create a boolean mask indicating La Nina events
la_nina_mask = np.convolve(nino34 <= la_nina_threshold, np.ones(5), mode='
    valid') == 5
la_nina_mask = np.concatenate((np.zeros(4, dtype=bool), la_nina_mask))

# Plot the results
plt.figure(1, figsize=(20, 3))
plt.plot(d_times, nino34, color="black", linewidth=2, label="Nino 3")
plt.fill_between(d_times, nino34, where=el_nino_mask, color="red", alpha
    =0.5, label="El Ni o")
plt.fill_between(d_times, nino34, where=la_nina_mask, color="blue", alpha
    =0.5, label="La Ni a")
plt.xlabel('Year', fontsize = 15)
plt.ylabel('Sea Surface Temperature Anomaly ( C )', fontsize = 8)
plt.legend(loc='upper left', prop={'size': 10}, framealpha = 0.7)
plt.title('El nino and La nina over nino34 using ENSO criteria')
plt.show()
```

Listing 40: Code for detecting ENSO over Nino 3.4

The El Niño and La Niña events over the Nino 3 and Nino 3.4 regions from 1980 to the present are depicted in Figure 39 and Figure 40 using the ENSO criteria of 3 consecutive days. These

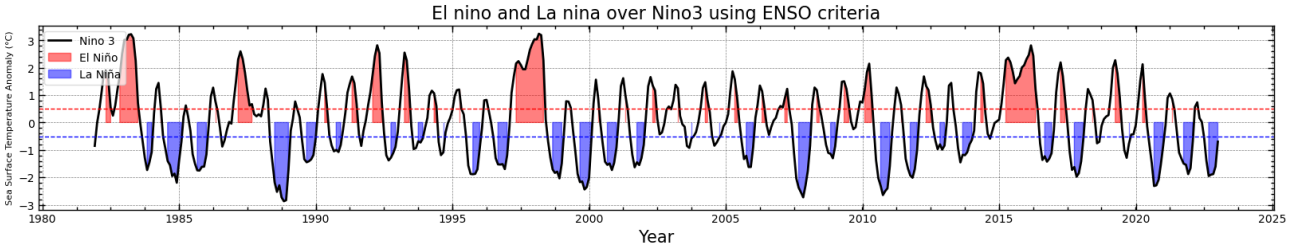

Figure 40: El nino and La nina over Nino34 using ENSO criteria

figures illustrate a strong correlation between the occurrence of El Niño and La Niña events in both regions.

In addition, ENSO events typically transpire approximately once or twice a year. El Niño events tend to occur in clusters with intervals of around 3-4 years, sometimes featuring multiple smaller El Niño events in succession. Similarly, La Niña events also exhibit multiple consecutive smaller events.

9.2 Quasi-Biennial Oscillation (QBO)

The Quasi-Biennial Oscillation (QBO) is a meteorological phenomenon characterized by the alternating reversal of high-altitude atmospheric winds in the tropics. The winds change direction from east to west and then reverse back to eastward, completing a cycle over a period of approximately two years or 28-29 months.

We first download the data from the Internet using `curl` on terminal or "!" in Jupyter Lab.

```
! curl -o qbo.dat https://www.geo.fu-berlin.de/met/ag/strat/produkte/qbo/qbo
   .dat
! curl -o singapore.dat https://www.geo.fu-berlin.de/met/ag/strat/produkte/
   qbo/singapore.dat
```

Listing 41: Code for downloading the QBO data from the Internet

The data enconding the location with different number 91700 for Canton Island, 43599 for Gan/Maledives and 48694 & 48698 for Singapore. Moreover the date is written in YYMM format, we can use some mathematical tricks to extract the year and month from data (the way to use it is depends on data, I can use it here because the year in 20th century is higher than 53).

```
# Data reading
qbo = pd.read_fwf('qbo.dat', skiprows = 9, header = None)

# Extract the time from qbo table
time = np.asarray(qbo[1])
```

```
y = time // 100
year_1 = y[y >= 53] + 1900
year_2 = y[y < 53] + 2000
year = np.hstack((year_1, year_2))
month = time % 100
# Create datetime series
datetime_series = pd.to_datetime({'year': year, 'month': month, 'day': 1})

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

Listing 42: Code for QBO data reading

We use for loop to plot all zonal wind speed at different pressure (or altitude).

```
fig, ax = plt.subplots(7, 1, figsize=(20, 20), sharex=True, sharey=True)

pressure_levels = ['70hPa', '50hPa', '40hPa', '30hPa', '20hPa', '15hPa', '10hPa']

for i, level in enumerate(pressure_levels):
    ax[i].plot(datetime_series, df_qbo[level], color='red')
    ax[i].set_ylabel("Velocity (m/s)")
    ax[i].set_title("QBO at {} hPa".format(level))

plt.xlabel("Date")
plt.savefig('figures/qbo')
# plt.show()
```

Listing 43: Code for plotting QBO according to altitude

Figure 41 displays the wind speed at various pressure levels. It is evident that the oscillation is less pronounced at higher altitudes, particularly at 70hPa, while it gradually intensifies as we move downwards. The distinct two-year cycle of the Quasi-Biennial Oscillation (QBO) is clearly observed across all pressure bands, making it easily to identify.

We read the Singapore file and remove all the unnecessary lines between the data, and then plot the contour map and transpose to obtain the data with time series on the x axis and inverse the pressure gradient on the y axis. The wind speed are color-coded.

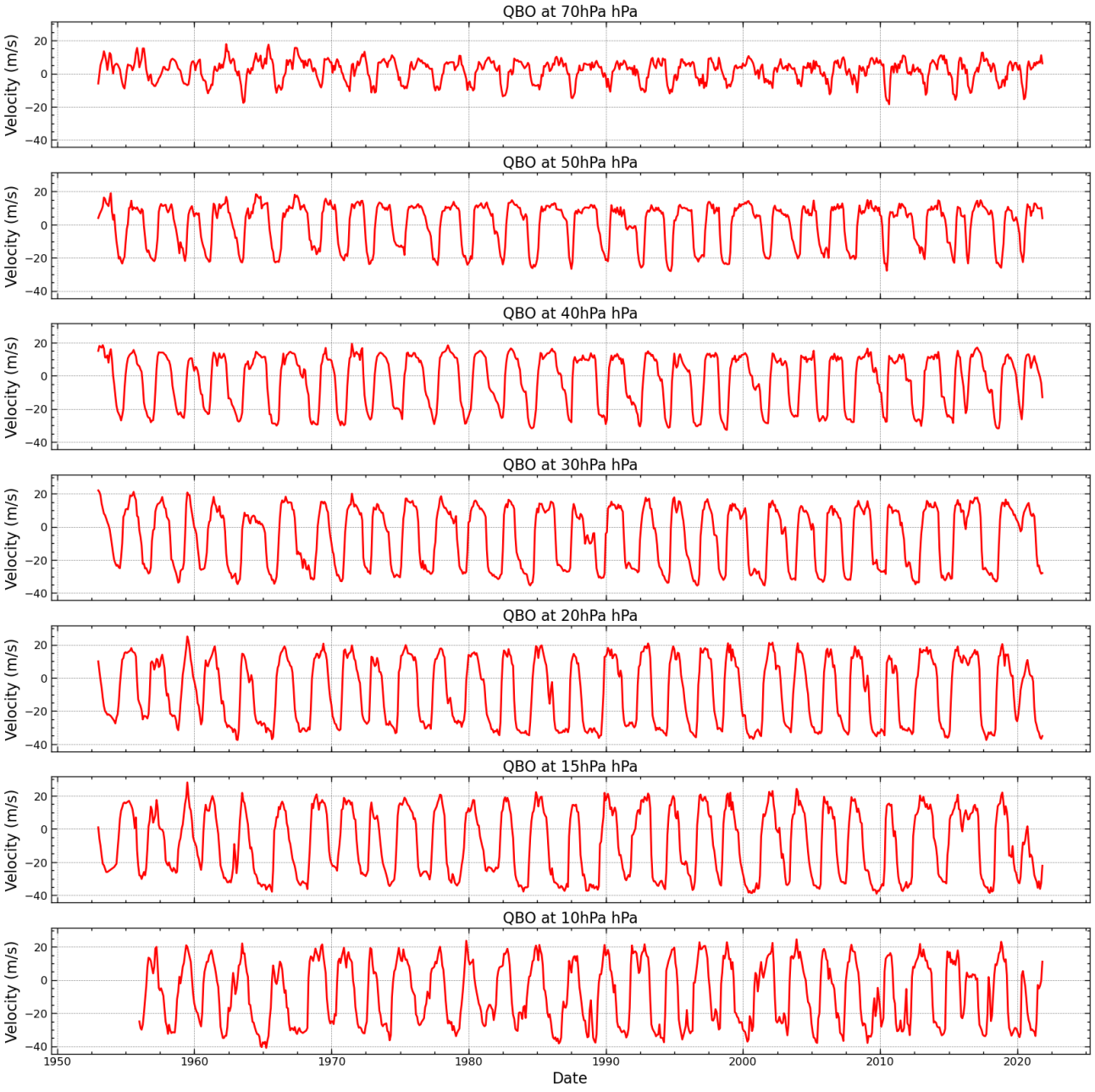

Figure 41: QBO

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

data = data[data[0] < 1900]
```

```
indices = np.where(data[0].values == 10)[0]
filtered_data = [data.iloc[indices[i]: indices[i+1]].values for i in range(
    len(indices) - 1)]

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

fig = plt.figure(figsize=(24, 3))
y = np.arange(1987, 1997, 10/126)
pa = np. arange (10 ,100 , 7.5)
plt.contourf(y, pa, data[:14*9].transpose(), cmap='jet')

plt.gca().invert_yaxis()  # Invert the y-axis
plt.colorbar(label = 'Wind speed (m/s)')
plt.xlabel("Year")
plt.ylabel("Pressure (Pa)")
plt.tight_layout()
plt.show()
```

Listing 44: Code for plotting QBO at Singapore from 1987 to 1997

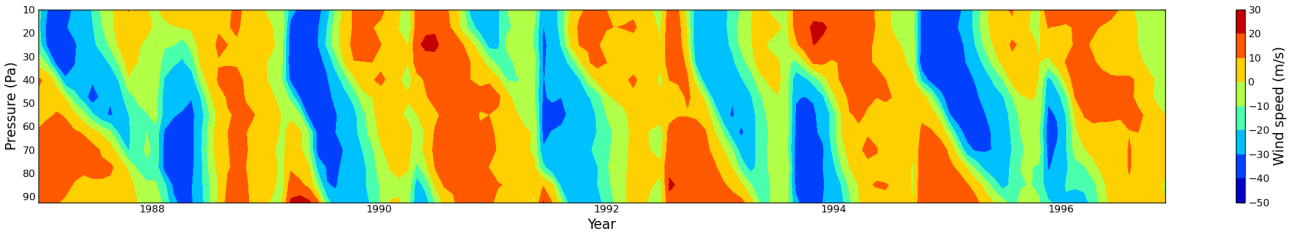

Figure 42: QBO in period 1987-1997

```
fig = plt.figure(figsize=(24, 3))
y = np.arange(1997, 2007, 10/150)
pa = np. arange (10 ,100 ,7.5)
plt.contourf(y, pa, data[14*9:14*9 + 15*10].transpose(), cmap='jet')

plt.gca().invert_yaxis()  # Invert the y-axis
plt.colorbar(label = 'Wind speed (m/s)')
plt.xlabel("Year")
plt.ylabel("Pressure (Pa)")
plt.tight_layout()
plt.show()
```

Listing 45: Code for plotting QBO at Singapore from 1997 to 2007

From figure 42 and 43, we can confirm these characteristics of QBO:

- Periods range from 20 to 36 months with an average of 28 months

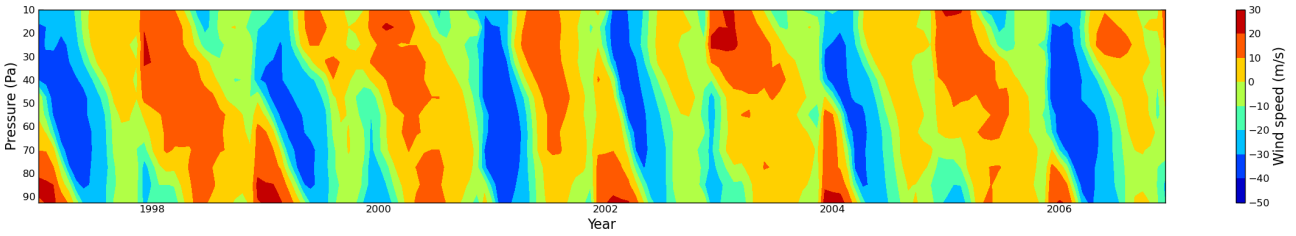

Figure 43: QBO in period 1997-2007

- The amplitude of the easterlies is usually stronger ($\sim$ 2 times) than the amplitude of the westerlies
- The QBO signal propagates downward over time from about 10hPa ($\sim$ 30km) to 100hPa ($\sim$ 16km) or lower
- The downward propagating speed is about 1km / month
- At the top of the vertical QBO region, easterlies dominate while at the bottom of this region, the westerlies more usually occur.
- The westerlies propagate downward faster than the esterlies
- QBO amplitude decreases as the altitude decreases. The maximum amplitude of 40-50 m/s is usually observed around 20mb ($\sim$ 25km).
- The transition from the westerlies to the easterlies usually slows down between 30-50mb ($\sim$ 20-23km)
- There are significant fluctuations in QBO periods and amplitudes.

10 Madden-Julian Oscillation (MJO)

The Madden-Julian Oscillation (MJO) is an atmospheric phenomenon characterized by the eastward propagation of large-scale patterns of rainfall and atmospheric circulation in the tropics with period of about 30 to 60 days. It is named after Roland Madden and Paul Julian, who first identified this phenomenon in the early 1970s.

I found a monthly data file of precipitation and daily data file of precipitation. I calculated the mean precipitation over the course of two years 1979 and 1980

```
file = 'precip.mon.mean.nc'
fh = Dataset(file, 'r')
lat = fh.variables['lat'][:].data
lon = fh.variables['lon'][:].data
d_times=nc.num2date(fh.variables['time'][:],fh.variables['time'].units)
```

```
6
7  precip = fh.variables['precip'][:].data
8
9  # Average SST
10 preci_mean = np.mean(precip[:24, :, :], axis=0)
11
12 # Create a figure and axes
13 fig = plt.figure(figsize=(14,7))
14 ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude
       =180))
15
16 # Plot the sea water temperature array using imshow
17 cmap_reversed = plt.cm.get_cmap('RdYlBu').reversed()
18
19 im = ax.contourf(lon, lat, preci_mean, cmap=cmap_reversed, transform=ccrs.
       PlateCarree(), levels = 15)
20
21 # Add land and coastline features
22 ax.add_feature(cfeature.COASTLINE, edgecolor='black')
23 ax.add_feature(cfeature.LAND, edgecolor='black', zorder = 1)
24 ax.add_feature(cfeature.BORDERS, linestyle=':')
25
26 # Add a colorbar
27 cbar = fig.colorbar(im, ax=ax, orientation='horizontal', label='Amount of
       rainfall (mm)', shrink = 0.7, aspect=30)
28
29 # Set the title
30 ax.set_title('Mean Precipitation')
31 ax.set_xticks([-180,-150,-120, -90, -60, -30, 0, 30, 60, 90, 120, 150, 180])
32 ax.set_xticklabels(['0', '30E', '60E', '90E', '120E', '150E', '180', '150W',
       '120W', '90W', '60W', '30W', '0'])
33
34 ax.set_yticks([-90, -60, -30, 0, 30, 60, 90])
35 ax.set_yticklabels(['90S', '60S', '30S', '0', '30N', '60N', '90N'])
36
37 ax.set_xlabel('Longitude')
38 ax.set_ylabel('Latitude')
39 # Show the plot
40 plt.tight_layout()
41 plt.savefig("figures/mean_preci")
42 # plt.show()
```

Figure 44 shows the mean precipitation in the period 1979-1980. One observation is that the mean precipitation is generally high in the regions of Asia and Australia. This indicates that these regions receive a relatively higher amount of rainfall on average compared to other parts of the world.

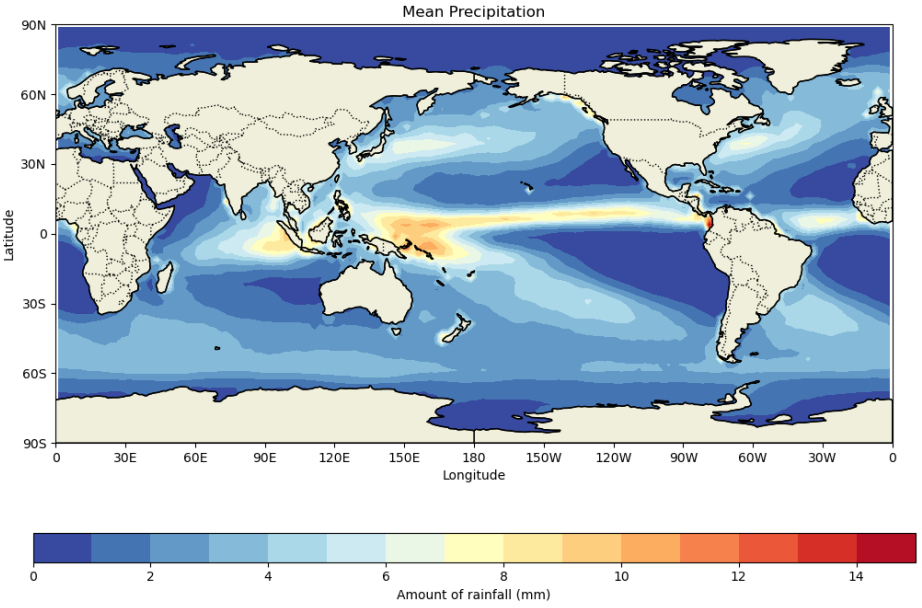

Figure 44: Precipitation Mean

We can also plot the precipitation anomaly, to see the deviations from the average or normal precipitation patterns along the longitude. This anomaly can be influenced by various atmospheric phenomena, including the MJO.

```
# Precipitation anomaly
preci_average = np.mean(precip[:24, :, :], axis=(2))
preci_ano = precip[:24, :, :] - np.expand_dims(preci_average, axis=(2))

# max_abs_value = np.min(np.abs(preci_ano))

# Create a figure and axes
fig = plt.figure(figsize=(16,8))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree(central_longitude
    =180))

# Plot the sea water temperature array using imshow
im = ax.contourf(lon, lat, np.mean(preci_ano, axis=0), cmap = cmap_reversed,
     transform=ccrs.PlateCarree(), levels = 20,
                  vmin = -4.8, vmax = 6
                 )

# Add land and coastline features
ax.add_feature(cfeature.COASTLINE, edgecolor='black')
ax.add_feature(cfeature.LAND, edgecolor='black', zorder = 1)
ax.add_feature(cfeature.BORDERS, linestyle=':')
```

```
20
21 # Add a colorbar
22 cbar = fig.colorbar(im, ax=ax, orientation='horizontal', label='
       Precipitation Anomaly', shrink = 0.7, aspect=30)
23
24 # Set the title
25 ax.set_title('Precipitation Anomaly')
26 ax.set_xticks([-180,-150,-120, -90, -60, -30, 0, 30, 60, 90, 120, 150, 180])
27 ax.set_xticklabels(['0', '30E', '60E', '90E', '120E', '150E', '180', '150W',
       '120W', '90W', '60W', '30W', '0'])
28
29 ax.set_yticks([-90, -60, -30, 0, 30, 60, 90])
30 ax.set_yticklabels(['90S', '60S', '30S', '0', '30N', '60N', '90N'])
31
32 ax.set_xlabel('Longitude')
33 ax.set_ylabel('Latitude')
34 # Show the plot
35 plt.tight_layout()
36 plt.savefig("figures/precipitation_abno")
37 # plt.show()
```

Figure 45 depicts the longitudinal precipitation anomaly over a two-year period, providing insights into the movement of the Madden-Julian Oscillation (MJO). In the Indian Ocean, we observe a negative anomaly in the western region and a positive anomaly in the eastern region, indicating the eastward propagation of the MJO. The positive anomaly suggests enhanced convection and increased rainfall in the eastern Indian Ocean, while the negative anomaly corresponds to suppressed convection and reduced rainfall in the western Indian Ocean.

In contrast, the pattern is reversed in the Pacific Ocean, with a positive anomaly in the western region and a negative anomaly in the eastern region. This signifies that the MJO weakens as it traverses the Pacific Ocean, leading to increased rainfall in the western Pacific and drier conditions in the eastern Pacific.

I also downloaded the daily data from https://rda.ucar.edu/datasets/ds728-7/ and try to calculate and plot the 5-day mean over the course of 2 months in January and February of 1979.

```
1 from datetime import datetime, timedelta
2
3 start_date = datetime(1997, 1, 1)
4 end_date = datetime(1997, 3, 21)
5 delta = timedelta(days=1)
6
7 date_list = []
8 current_date = start_date
```

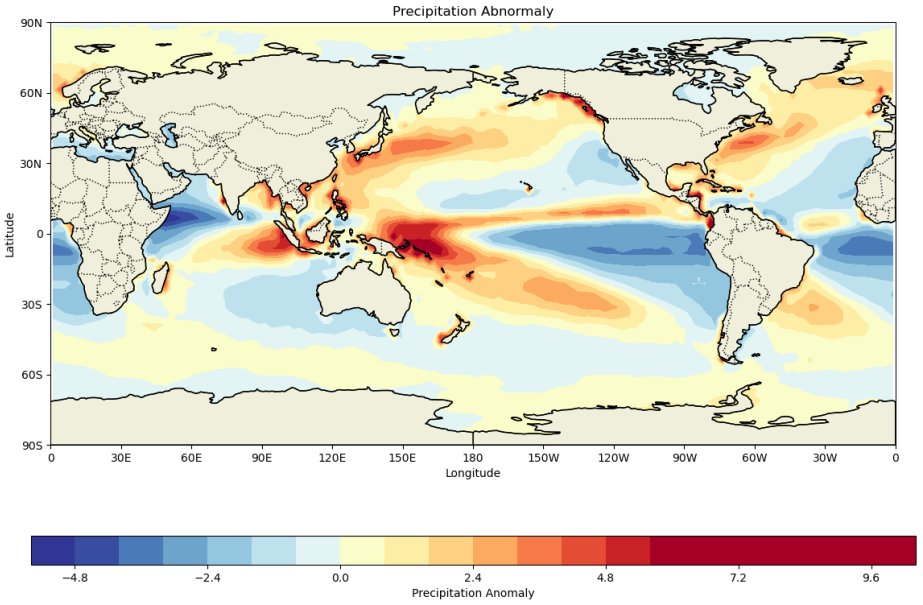

Figure 45: Precipitation Anomaly

```
9
10 while current_date <= end_date:
11     date_list.append(current_date.strftime('%Y%m%d'))
12     current_date += delta
13
14 names = [f'gpcp_v01r03_daily_d{d}.nc' for d in date_list]
15
16 data = {key:[] for key in date_list}
17 for i in range(len(names)):
18     data[date_list[i]] = Dataset(names[i], 'r')
19
20 lat = data['19970101'].variables['latitude'][:].data
21 lon = data['19970101'].variables['longitude'][:].data
22
23 precipitation = []
24 for day in date_list:
25     precipitation.append(data[day].variables['precip'][:].data[0,:,:])
26
27 p = []
28 for i in range(len(precipitation)//5):
29     p.append(np.mean(precipitation[i:(i+1)*5], axis = 0))
```

Listing 46: Code for daily data reading

```
1 fig, ax = plt.subplots(12, 1, figsize=(12,25), sharex=True, subplot_kw={'
```

```
    projection': ccrs.PlateCarree(central_longitude=180)})

cmap_reversed = plt.cm.get_cmap('RdYlBu').reversed()
# Plot the subplots
for i in range(12):
    ax[i].contourf(lon, lat, p[i], cmap=cmap_reversed, transform=ccrs.
    PlateCarree(), levels=15)
    ax[i].set_ylim(-30, 30)  # Set the latitude limits for each subplot

    # Add land and coastline features
    ax[i].add_feature(cfeature.COASTLINE, edgecolor='black')
    ax[i].add_feature(cfeature.LAND, edgecolor='black', zorder=1)
    ax[i].add_feature(cfeature.BORDERS, linestyle=':')

    # Set the y-axis label and tick labels for the first subplot
    if i % 2 == 0:
        ax[i].set_ylabel('Latitude')
        ax[i].set_yticks([-30, 0, 30])
        ax[i].set_yticklabels(['30S', '0', '30N'])

ax[-1].set_xlabel('Longitude')  # Set the x-axis label for the last subplot

# Create a separate axis for the colorbar
cbar_ax = fig.add_axes([0.2, 0.05, 0.6, 0.02])  # Adjust the position of the
    colorbar axis

# Plot the colorbar
cbar = fig.colorbar(ax[0].collections[0], cax=cbar_ax, orientation='
    horizontal', label='Precipitation')

# Adjust the layout
plt.subplots_adjust(hspace=0, wspace=0)

# Show the plot
plt.savefig('figures/mjo')
# plt.show()
```

As depicted in Figure 46, distinct patterns of the Madden-Julian Oscillation (MJO) can be observed in the Indian Ocean and the Pacific Ocean. In the Indian Ocean, the MJO starts to form and intensifies over the course of two months, with its strongest phase observed in the eastern side of the ocean. This is characterized by the development of convective clouds and enhanced atmospheric convection, indicating the presence of active MJO conditions.

In contrast, the Pacific Ocean exhibits a different behavior. The MJO in this region undergoes a shorter cycle, with cloud formation and intensification occurring over the span of one month, followed by a weakening phase in the subsequent month. It is worth noting that dur-

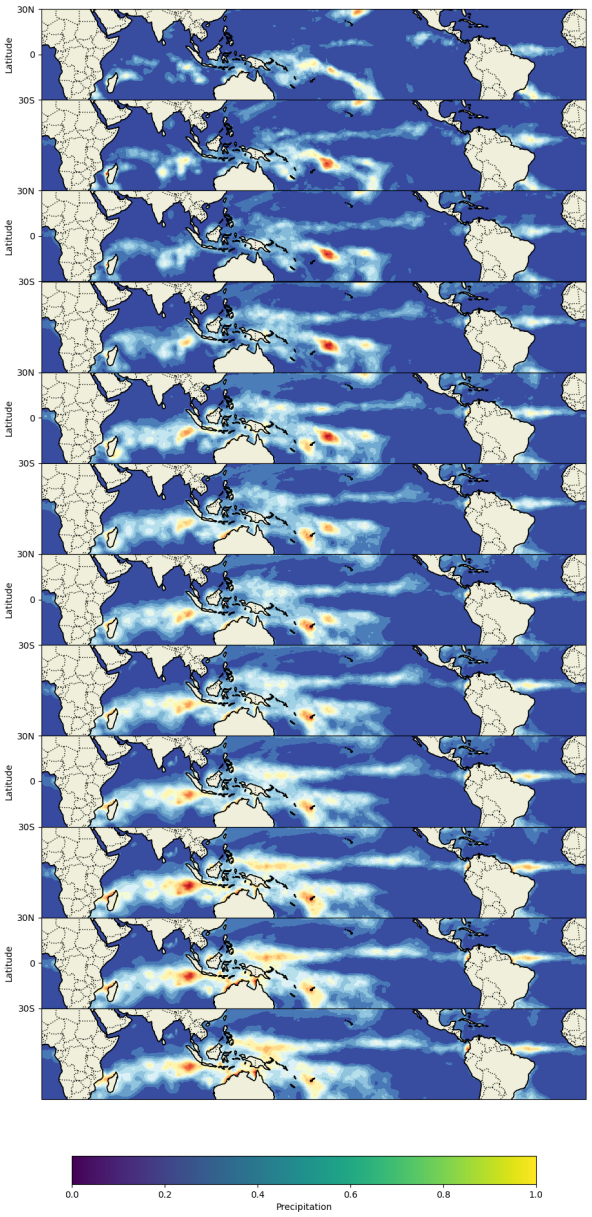

Figure 46: MJO

ing this weakening phase in the Pacific Ocean, the MJO in the Indian Ocean experiences an intensification, as evidenced by the strengthening of winds.

11 Conclusion

There comes a point when every journey must reach its destination, and this 100-page report marks the end of mine in this course. Although I never considered myself a weather and climate enthusiast, this course has exceeded my expectations in every way. I have gained knowledge not only in mathematics, physics, and programming, but also in the art of problem-solving. The time I spent struggling and grappling with problems has proven to be invaluable, as it deepened my understanding of the intricate complexities inherent in the interactions of various factors on our seemingly insignificant planet within the vastness of the universe. I have given my best effort within the time provided, and I found great joy in creating simulation and visualizations. While there are still some questions I am not fully understanding, I am grateful for everything I have learned and for the opportunity to be a part of this journey. Thank you for all.