Surface Hydrology

Yair Mau

Table of contents

About

Welcome to Surface Hydrology (71630) at the Hebrew University of Jerusalem. This is Yair Mau, your host for today. I am a senior lecturer at the Institute of Environmental Sciences, at the Faculty of Agriculture, Food and Environment, in Rehovot, Israel.

This website contains (almost) all the material you'll need for the course. If you find any mistakes, or have any comments, please email me.

The material here is not comprehensive and `does not` constitute a stand alone course in Surface Hydrology. This is only the support material for the actual presential course I give.

Syllabus

Course description

This is an introductory course in Surface Hydrology, dealing with some of the major processes in the hydrologic cycle: precipitation, evaporation and transpiration, infiltration, runoff generation and streamflow. The different topics will be treated using mathematical models and practical programming exercises.

Course aims

The course aims at giving the students a quantitative understanding of the main processes in the hydrologic cycle. We will characterize the hydrologic cycle and its fluxes through mass balance equations. The random nature of the various processes will be studied with statistics, time series analysis, return periods, extreme value distributions, etc. We will take a "hands-on approach", where students will actively engage with the material by analysing data and writing models using Python.

Learning outcomes

On successful completion of this module, students should be able to:

- Identify the various components of hydrologic budget and their interdependency.
- Describe the various processes in hydrology (precipitation, infiltration, evaporation, etc) in a mathematical language.
- Write computer code to analyze the statistics of hydrologic fluxes, and construct models of hydrological systems.

Books and other sources

- Dingman, S. L. (2015). Physical hydrology (3rd edition). Waveland press.
- Ward, A. D., & Trimble, S. W. (2003). Environmental hydrology. CRC Press.
- Brutsaert, W. (2005). Hydrology: An Introduction. Cambridge University Press.

Course evaluation

There will be some small projects during the semester, all worth 50% of the grade. A final and larger project (50% of the grade) will be due at the end of the semester. All projects will be done in Python (on Jupyter Notebooks).

Part I

Introduction

1 Water Cycle: Fluxes and Storage

1.1 How much water is there? Where?

https://youtu.be/2ObMyytxLz8

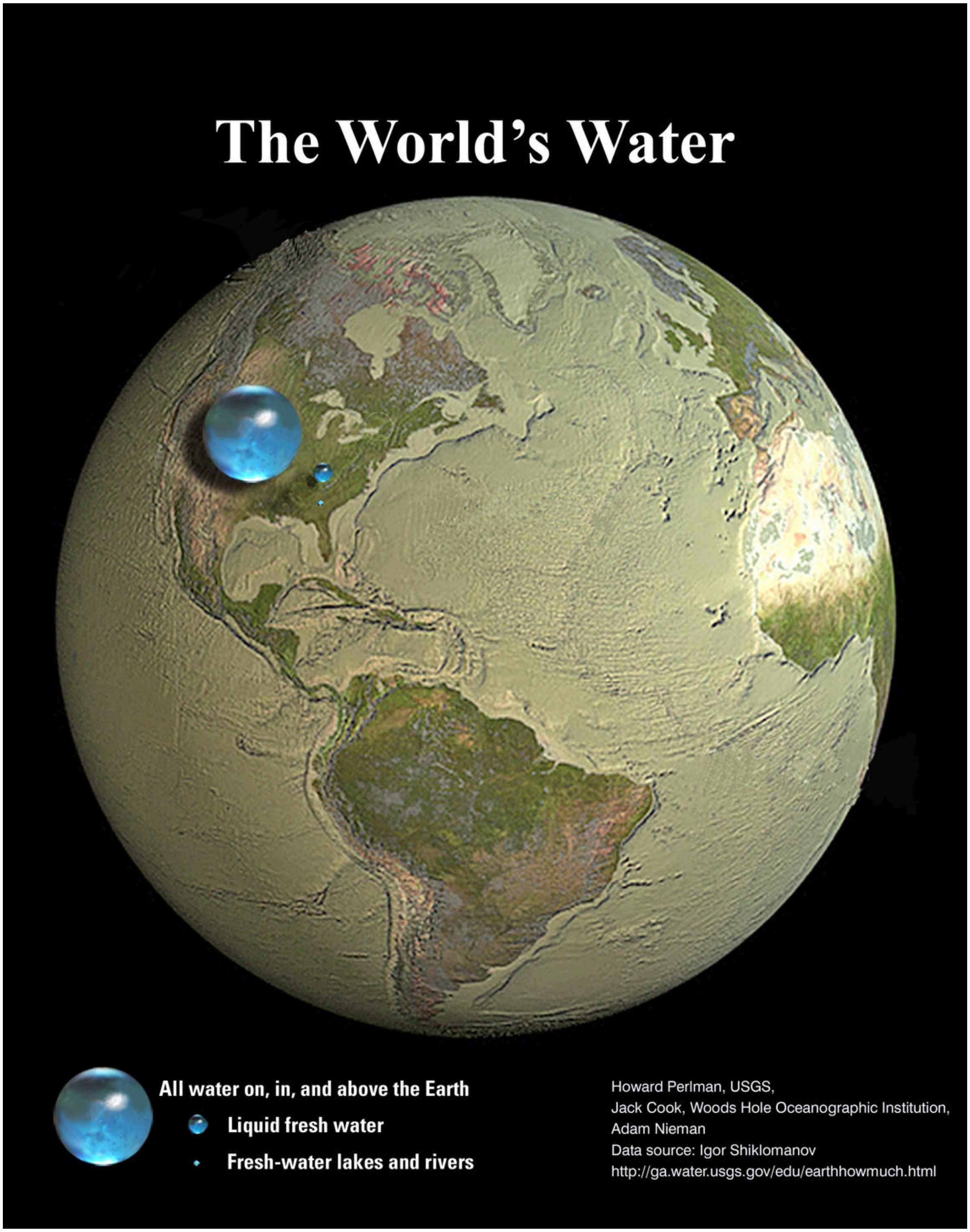


Figure 1.1: Source: Water Science School (2019c)

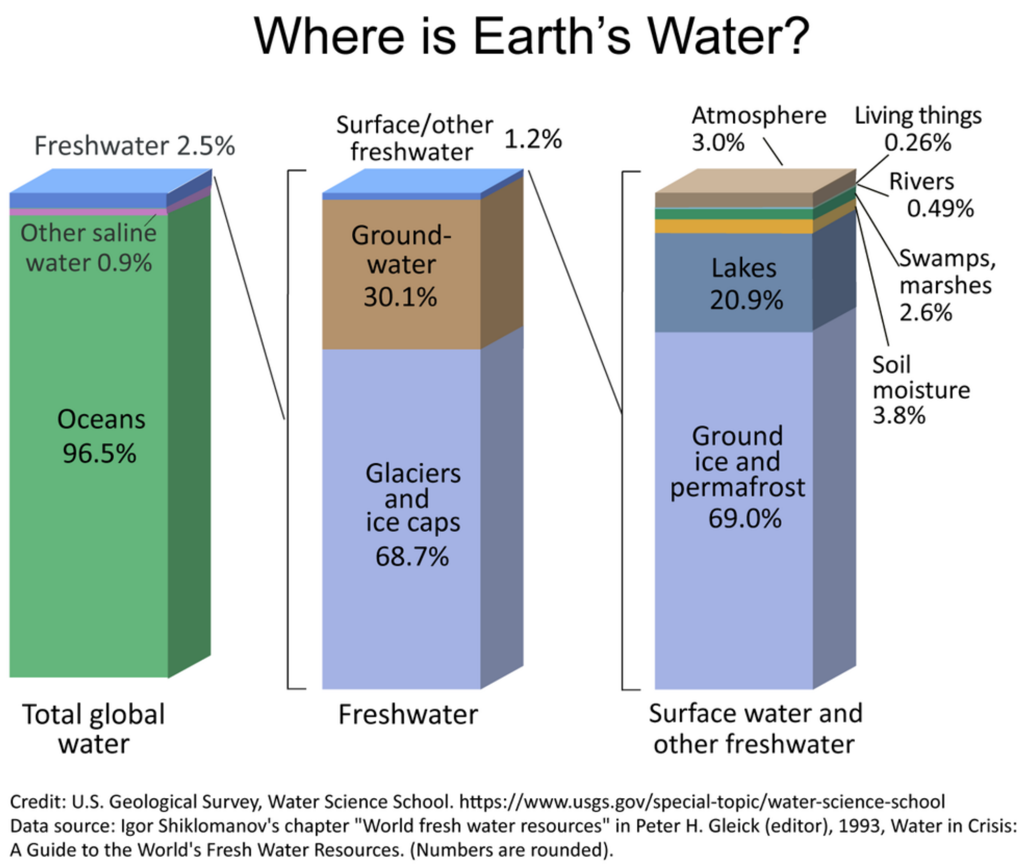


Figure 1.2: Source: Water Science School (2018)

1.2 The natural water cycle (2019)

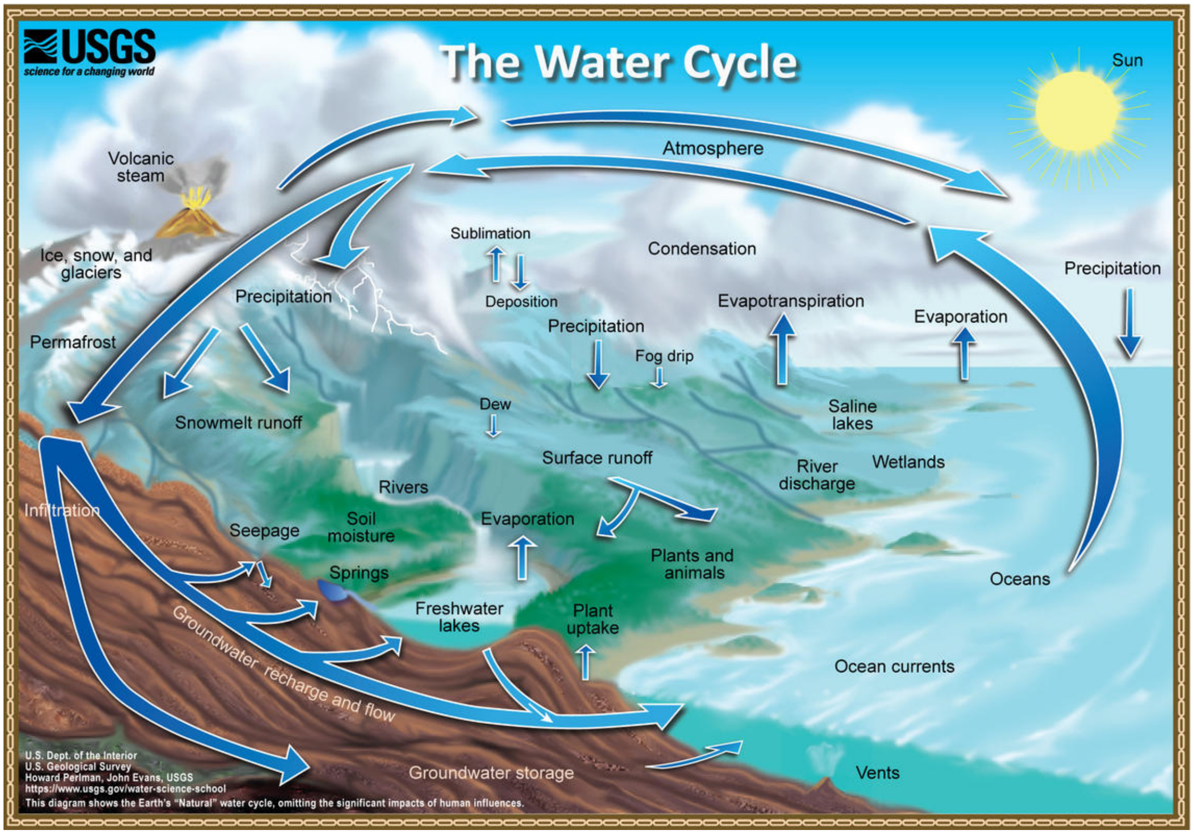


Figure 1.3: Source: Water Science School (2019g)

1.3 The new water cycle (2022)

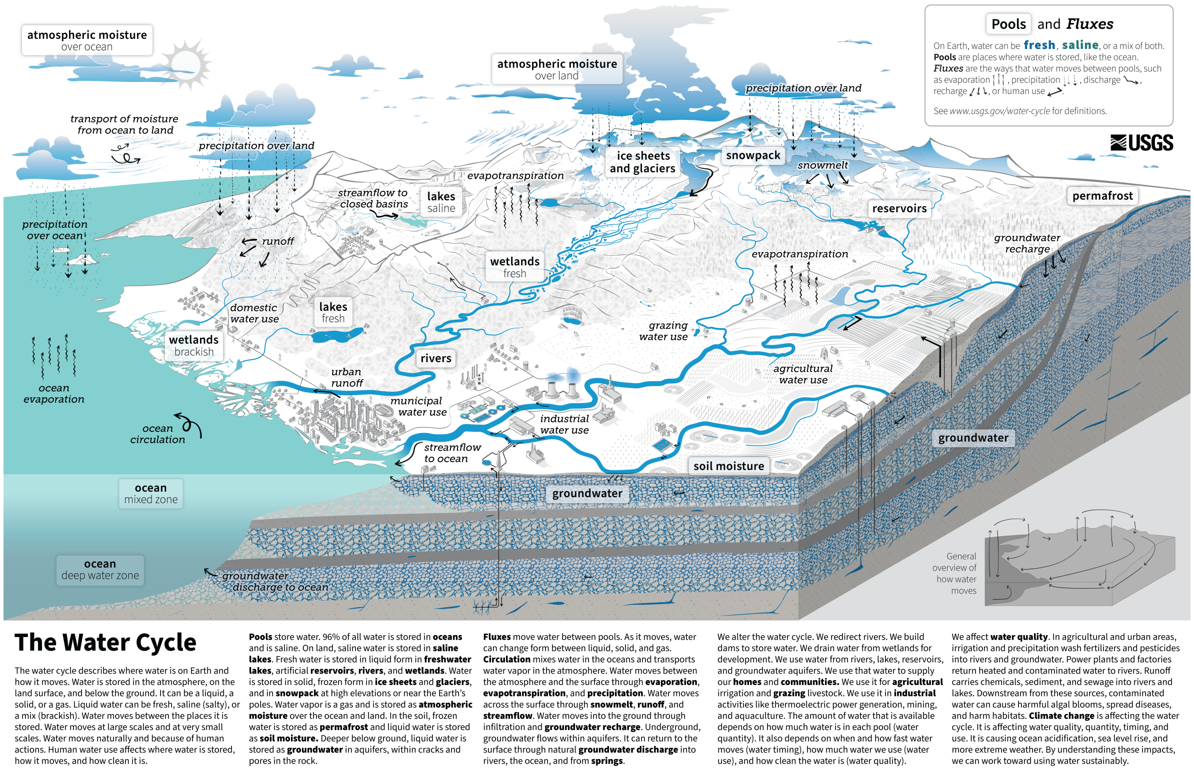


Figure 1.4: Source: Water Science School (2022)

Interactive chart: Pools and fluxes in the water cycle

1.4 Global water distribution

Table 1.1: Source: Water Science School (2018). **(Percents are rounded, so will not add to 100)**

Water source	Volume (km^3)	% of freshwater	% of total water
Oceans, Seas, & Bays	1,338,000,000	–	96.54
Ice caps, Glaciers, & Permanent Snow	24,064,000	68.7	1.74

Water source	Volume (km^3)	% of freshwater	% of total water
Groundwater	23,400,000	–	1.69
Fresh	10,530,000	30.1	0.76
Saline	12,870,000	–	0.93
Soil Moisture	16,500	0.05	0.001
Ground Ice & Permafrost	300,000	0.86	0.022
Lakes	176,400	–	0.013
Fresh	91,000	0.26	0.007
Saline	85,400	–	0.006
Atmosphere	12,900	0.04	0.001
Swamp Water	11,470	0.03	0.0008
Rivers	2,120	0.006	0.0002
Biological Water	1,120	0.003	0.0001

1.5 Energy drives the hydrologic cycle

From Margulis (2019)

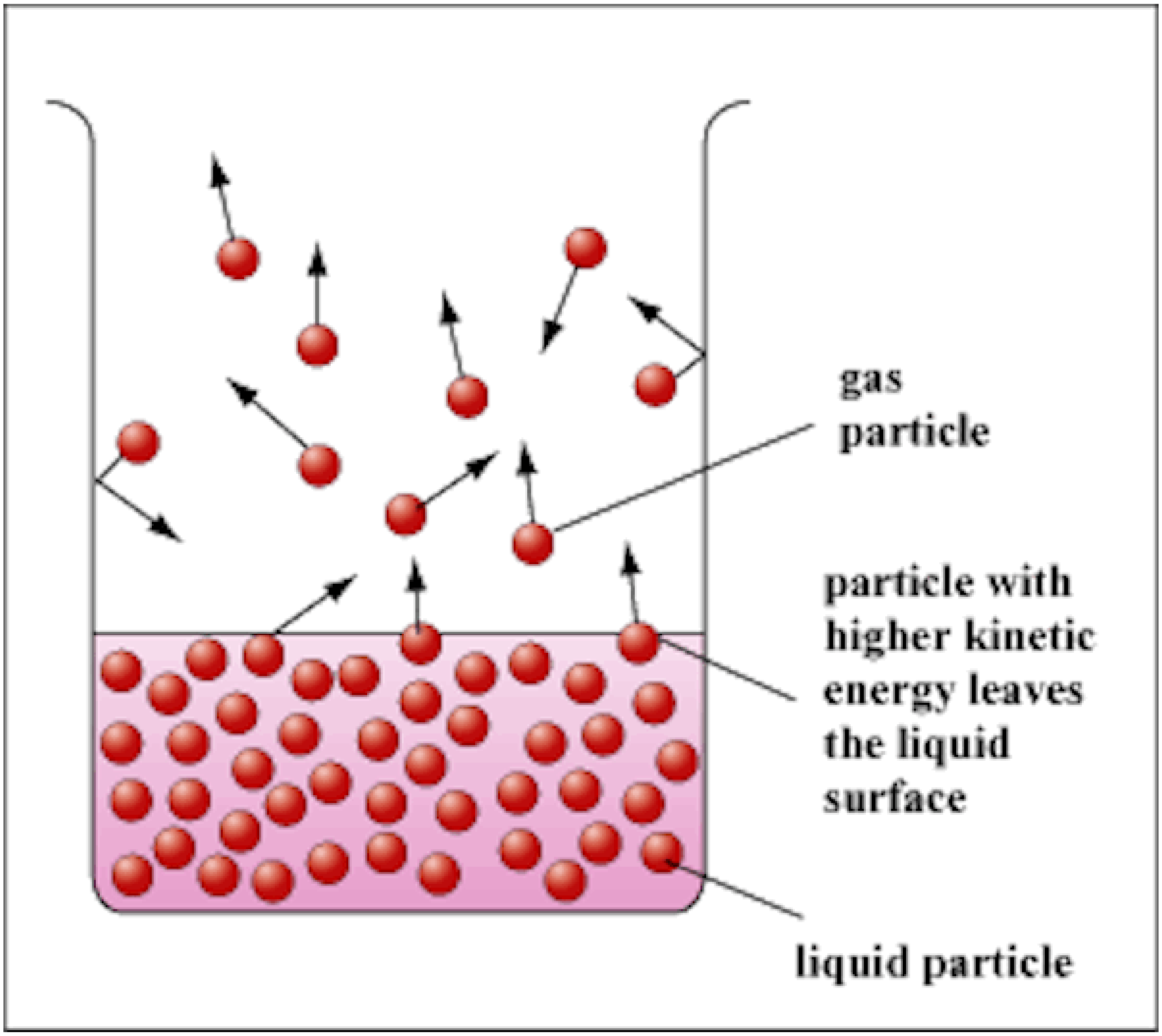
> A key aspect of the hydrologic cycle is the fact that it is driven by energy inputs (primarily from the sun). At the global scale, the system is essentially closed with respect to water; negligible water is entering or leaving the system. In other words, there is no external forcing in terms of a water flux. Systems with no external forcing will generally eventually come to an equilibrium state. So what makes the hydrologic cycle so dynamic? The solar radiative energy input, which is external to the system, drives the hydrologic cycle. Averaged over the globe, 342 W m^{-2} of solar radiative energy is being continuously input to the system at the top of the atmosphere. This energy input must be dissipated, and this is done, to a large extent, via the hydrologic cycle. Due to this fact, the study of hydrology is not isolated to the study of water storage and movement, but also must often include study of energy storage and movements.

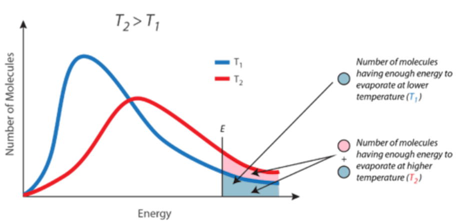
1.6 Components of the water cycle

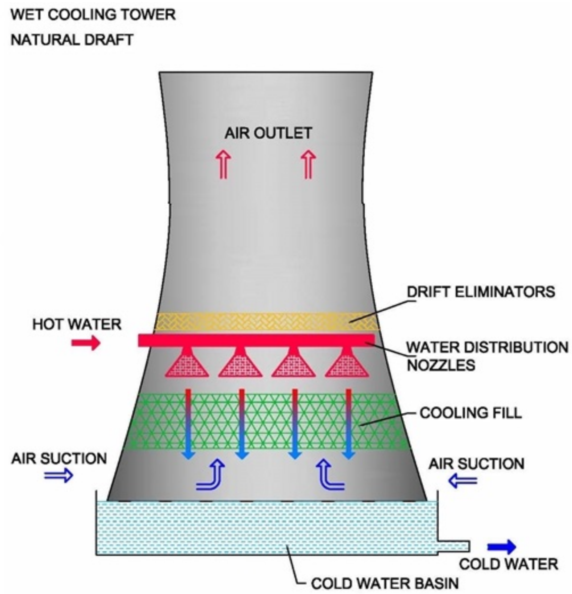
1.6.1 Water storage in oceans

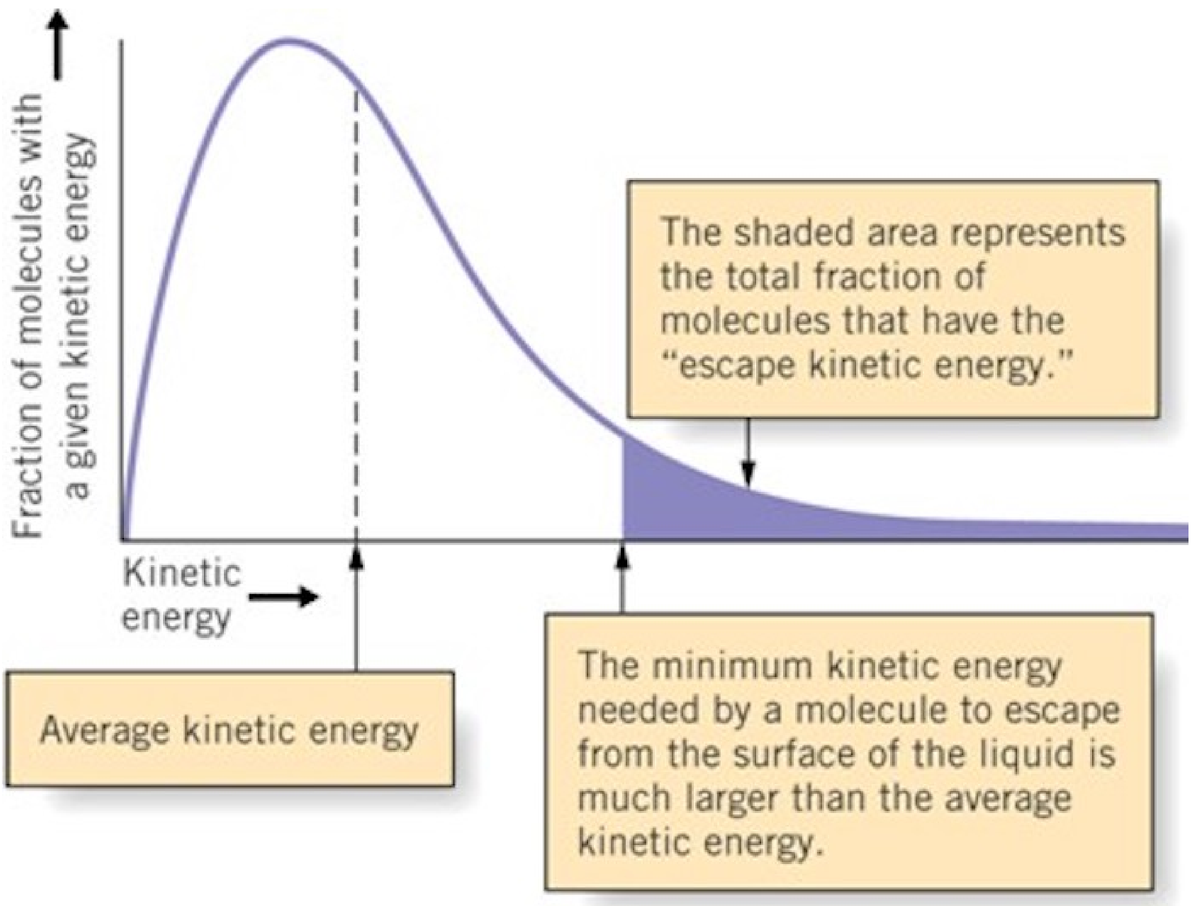
1.6.2 Evaporation / Sublimation

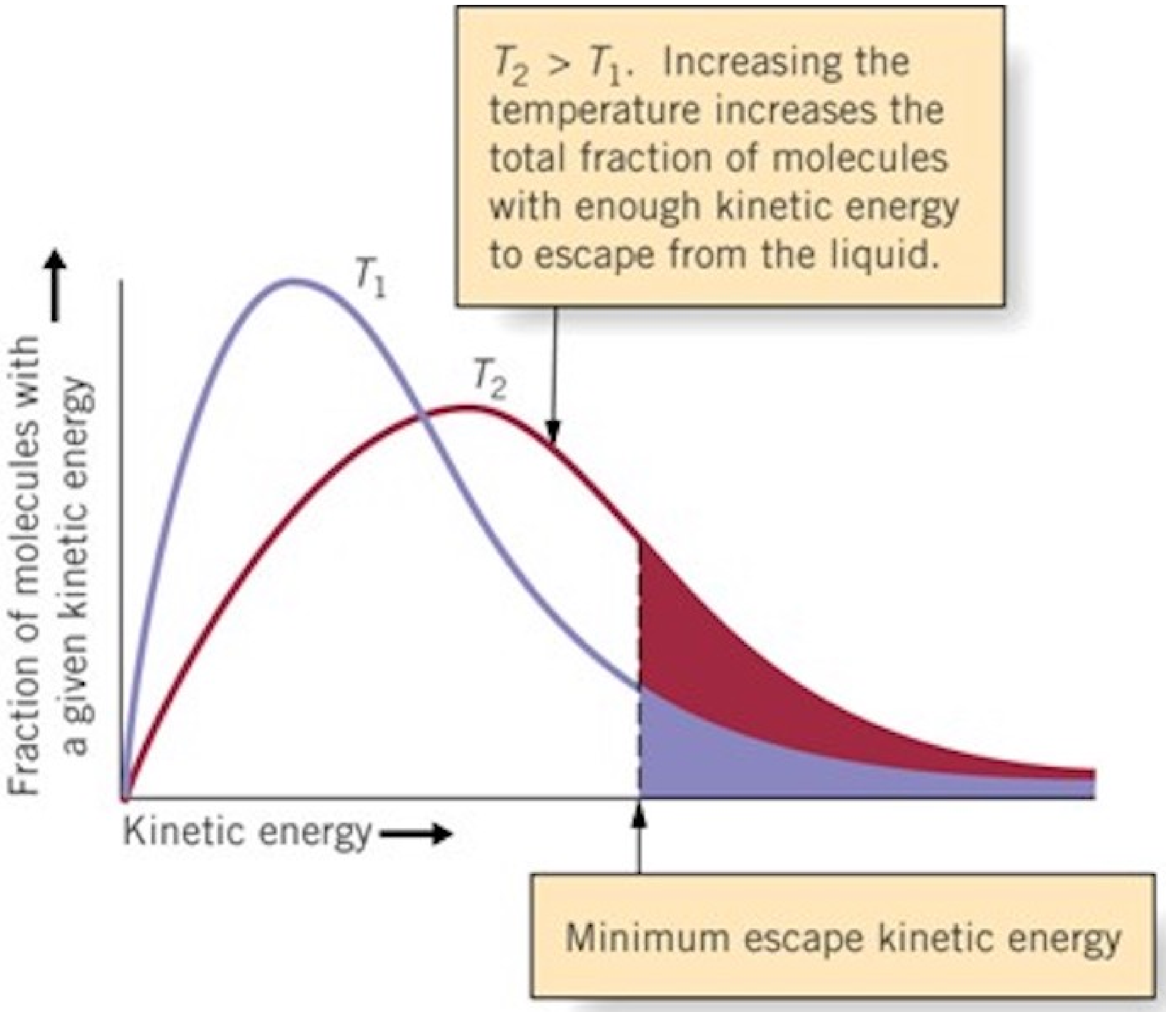
Evaporation $\longrightarrow$ cooling


gas particle
particle with higher kinetic energy leaves the liquid surface
liquid particle

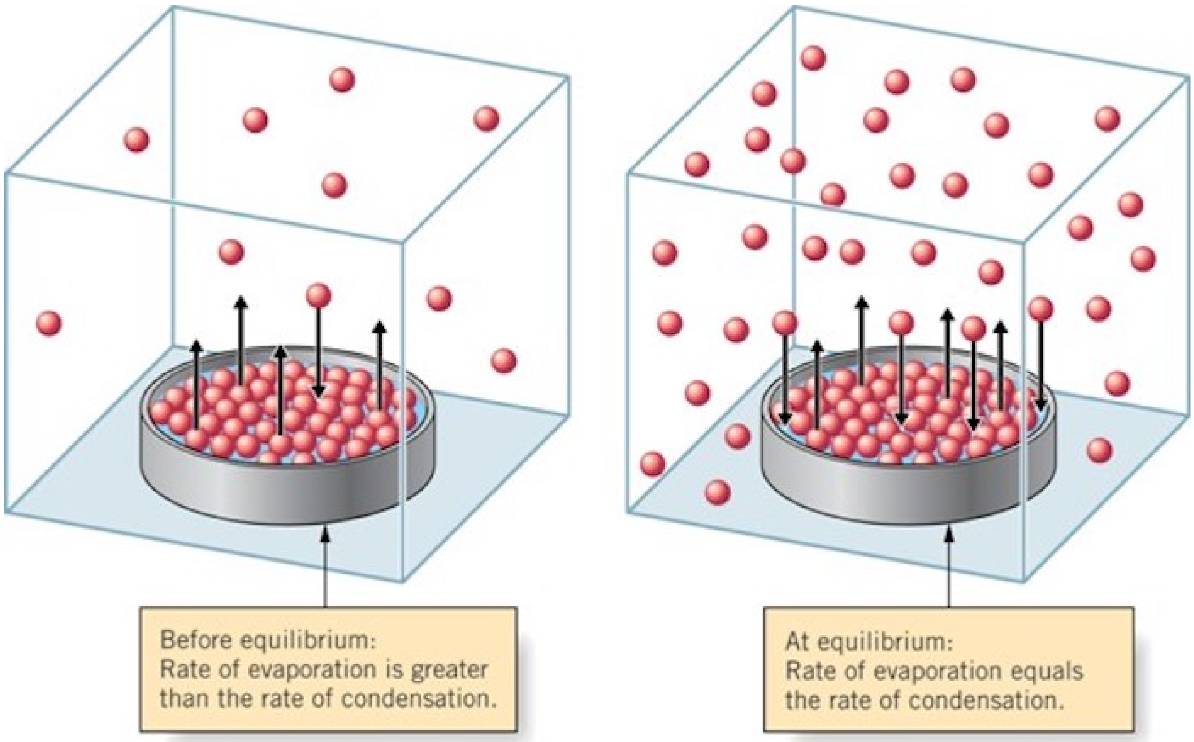

$T_2 > T_1$
Number of Molecules
Energy
T_1
T_2
E
Number of molecules having enough energy to evaporate at lower temperature (T_1)
Number of molecules having enough energy to evaporate at higher temperature (T_2)


WET COOLING TOWER
NATURAL DRAFT
AIR OUTLET
DRIFT ELIMINATORS
HOT WATER
WATER DISTRIBUTION NOZZLES
COOLING FILL
AIR SUCTION
AIR SUCTION
COLD WATER
COLD WATER BASIN


Fraction of molecules with a given kinetic energy
Kinetic energy
The shaded area represents the total fraction of molecules that have the "escape kinetic energy."
Average kinetic energy
The minimum kinetic energy needed by a molecule to escape from the surface of the liquid is much larger than the average kinetic energy.

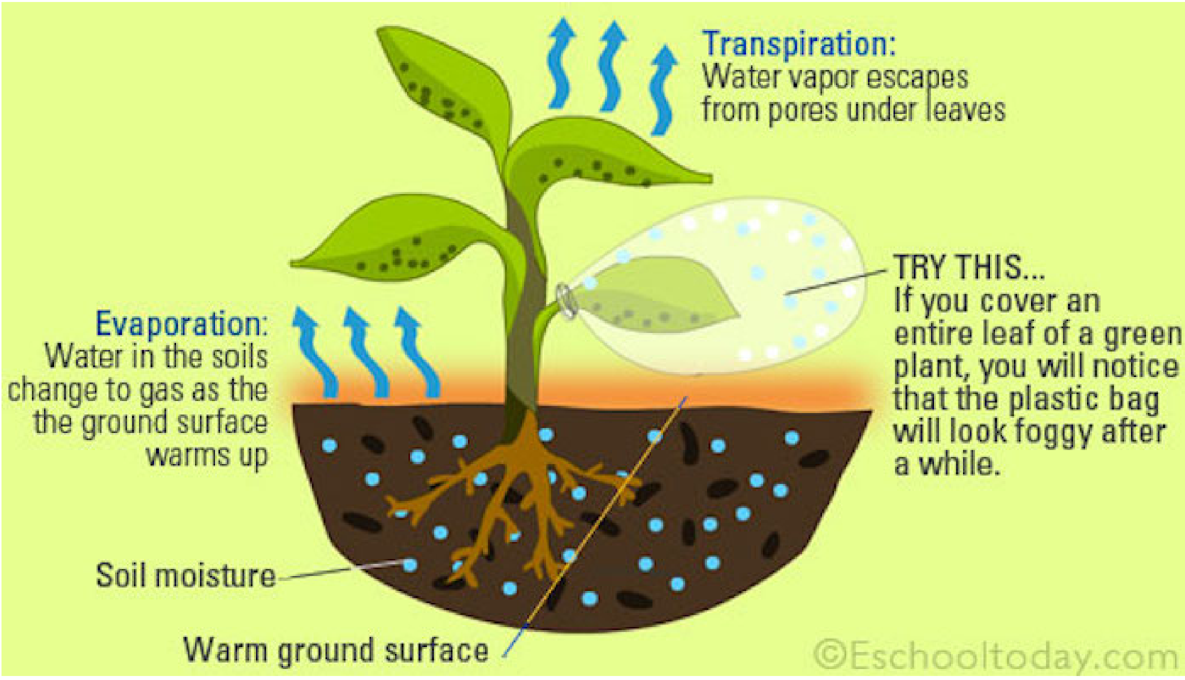


$T_2 > T_1$. Increasing the temperature increases the total fraction of molecules with enough kinetic energy to escape from the liquid.
T_1
T_2
Fraction of molecules with a given kinetic energy
Kinetic energy
Minimum escape kinetic energy


Before equilibrium:
Rate of evaporation is greater than the rate of condensation.
At equilibrium:
Rate of evaporation equals the rate of condensation.

(a) (b)

1.6.3 Evapotranspiration



1.6.4 Water storage in the atmosphere

Cumulonimbus cloud over Africa 

Picture of cumulonimbus taken from the International Space Station, over western Africa near the Senegal-Mali border.

If all of the water in the atmosphere rained down at once, it would only cover the globe to a depth of 2.5 centimeters.

$$
\begin{aligned}
\text{amount of water in the atmosphere} \quad & V = 12\,900\,\mathrm{km}^3 \\
\text{surface of Earth} \quad & S = 4\pi R^2; \quad R = 6371\,\mathrm{km} \\
& V = S \times h \\
\text{height} \quad & h = \frac{V}{S} \simeq 2.5\,\mathrm{cm}
\end{aligned}
$$

Try to calculate this yourself, and click on the button below to check how to do it.

```
# amount of water in the atmosphere
V = 12900 # km^3
# Earth's radius
R = 6371 # km
# surface of Earth = 4 pi R^2
S = 4 * 3.141592 * R**2
# Volume: V = S * h, therefore
# height
h = V / S # in km
h_cm = h * 1e5 # in cm
print(f"The height would be ~ {h_cm:.1f} cm")
```

```
The height would be ~ 2.5 cm
```

1.6.5 Condensation

1.6.6 Precipitation

Figure 1.5: Source: Water Science School (2019f)

Table 1.2: Source: Water Science School (2019e)

	Intensity (cm/h)	Median diameter (mm)	Velocity of fall (m/s)	Drops s^{-1} m^{-2}
Fog	0.013	0.01	0.003	67,425,000
Mist	0.005	0.1	0.21	27,000
Drizzle	0.025	0.96	4.1	151
Light rain	0.10	1.24	4.8	280
Moderate rain	0.38	1.60	5.7	495
Heavy rain	1.52	2.05	6.7	495
Excessive rain	4.06	2.40	7.3	818
Cloudburst	10.2	2.85	7.9	1,220

1.6.7 Water storage in ice and snow

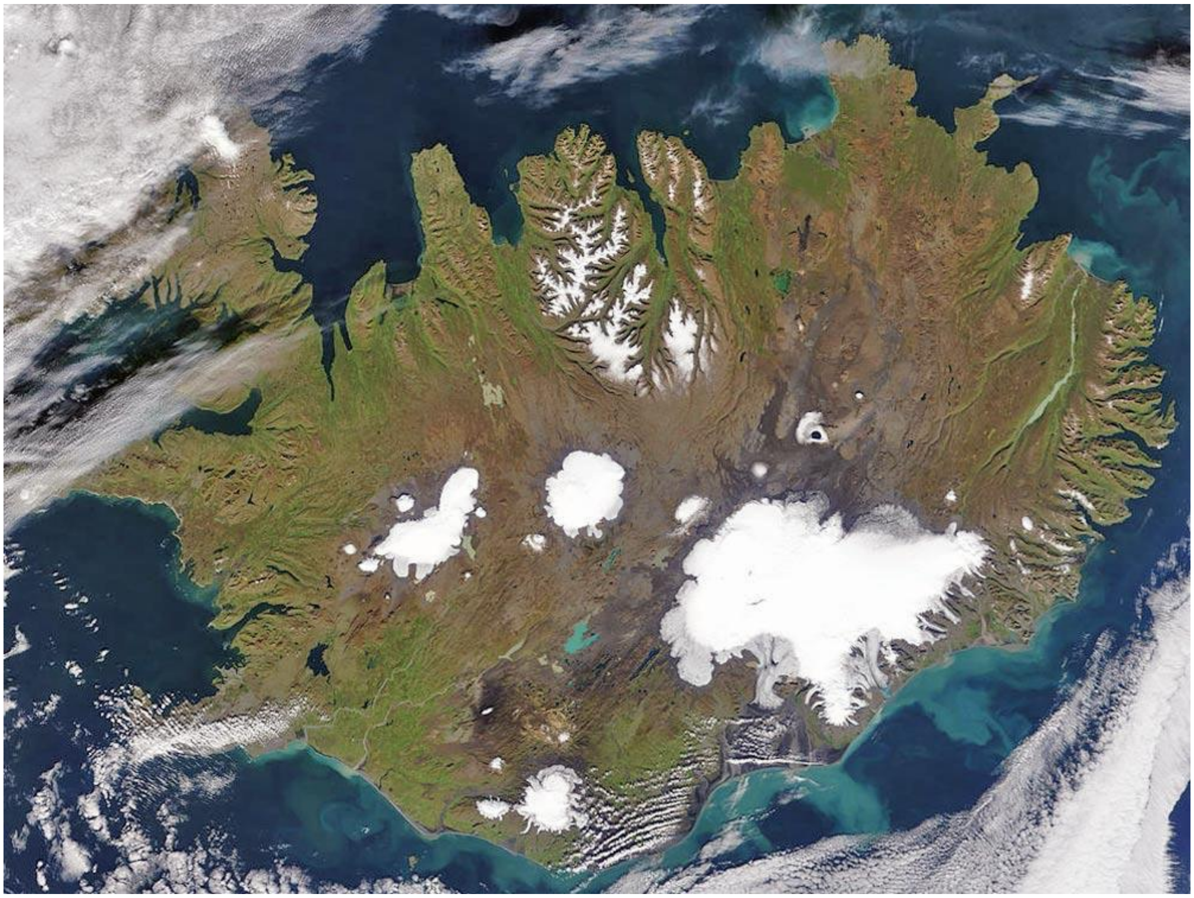


Figure 1.6: Source: Water Science School (2019d)

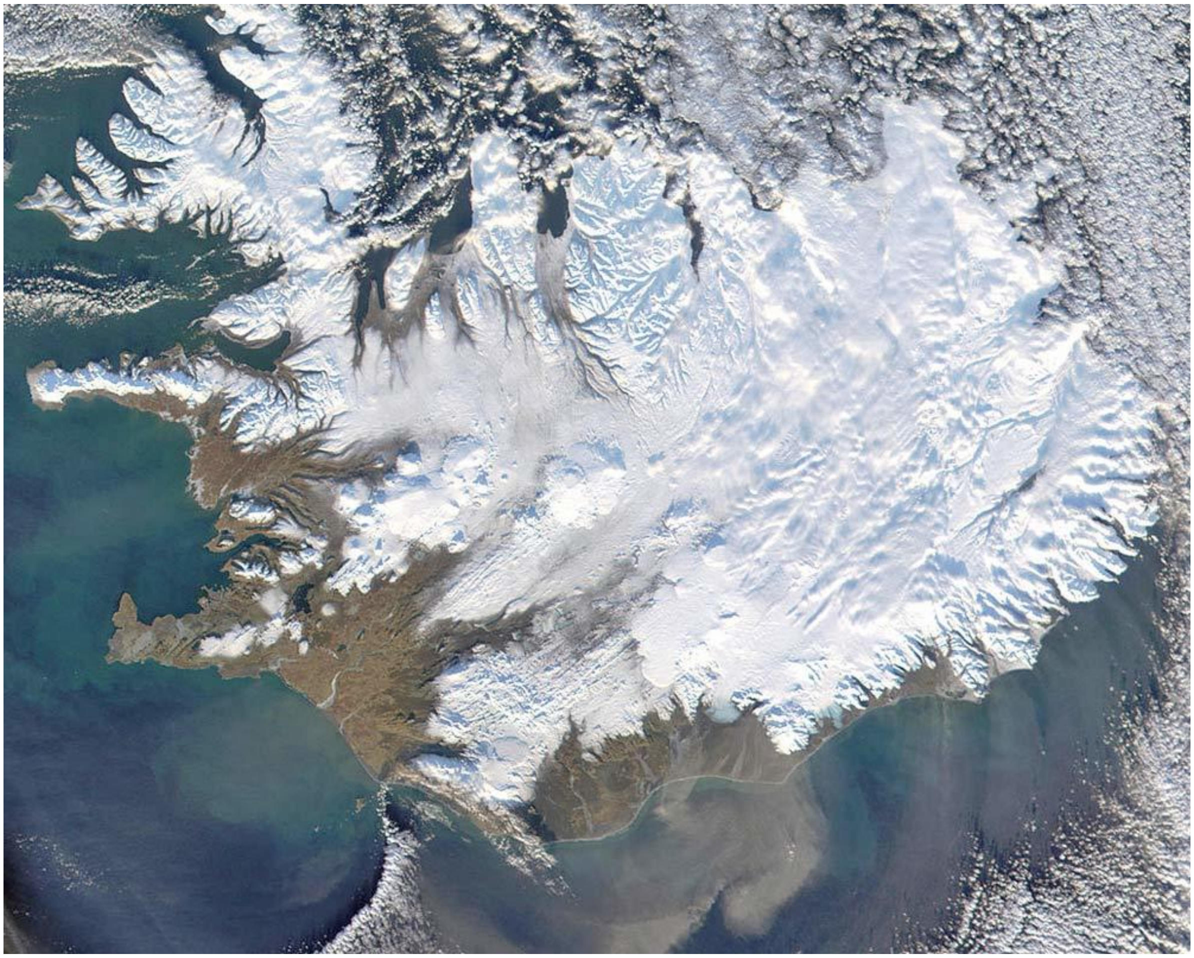


Figure 1.7: Source: Water Science School (2019d)

1.6.8 Snowmelt runoff to streams

1.6.9 Surface runoff

Figure 1.8: Source: (2020)

1.6.10 Streamflow

The Mississippi river basin is very large

The Amazon river basin is **Huge**

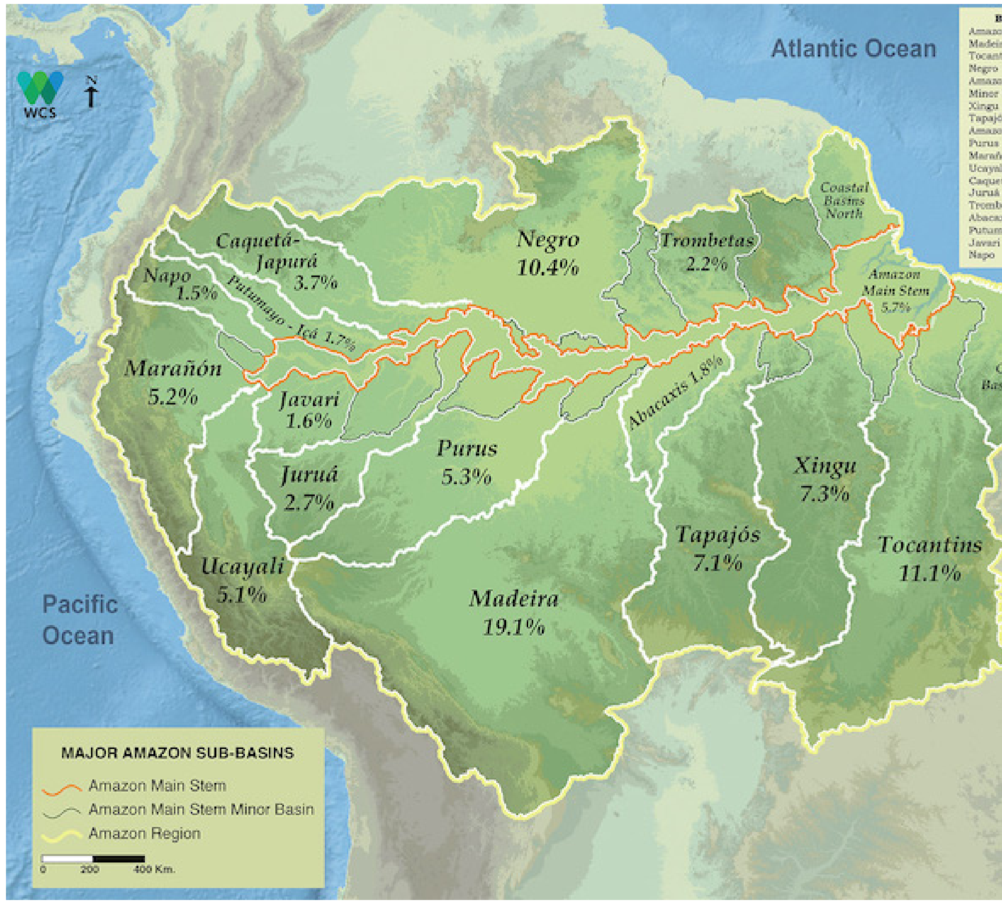


Figure 1.9: Source: Yair Mau

1.6.11 Lakes and rivers

Figure 1.10: Source: dreamstime (2022)

Lake Malawi

Figure 1.11: Source: Fiona Bruce (2015)

1.6.12 Infiltration

Figure 1.12: Source: Suma Groulx (2015)

1.6.13 Groundwater storage

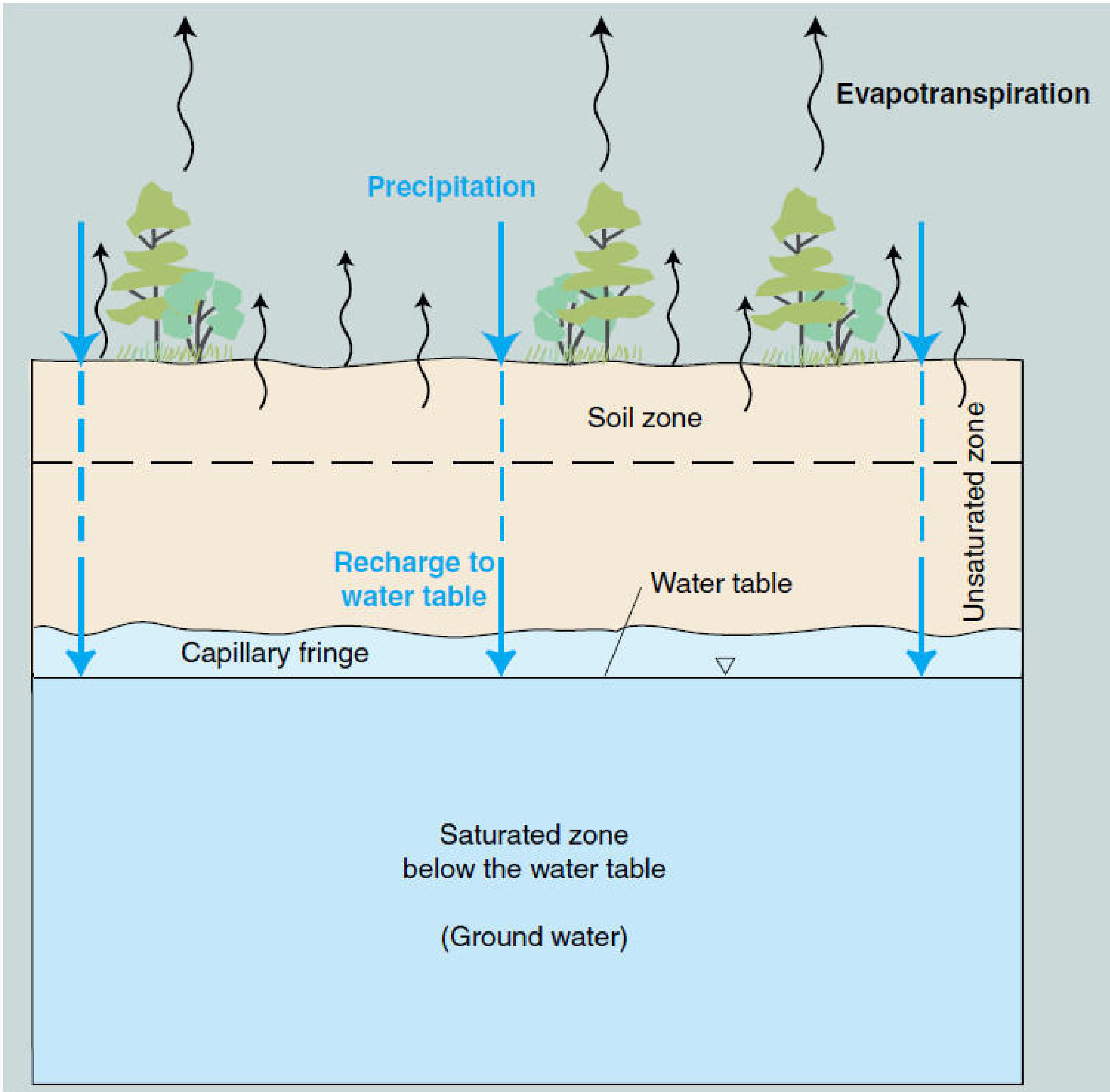


Figure 1.13: Source: Water Science School (2019b)

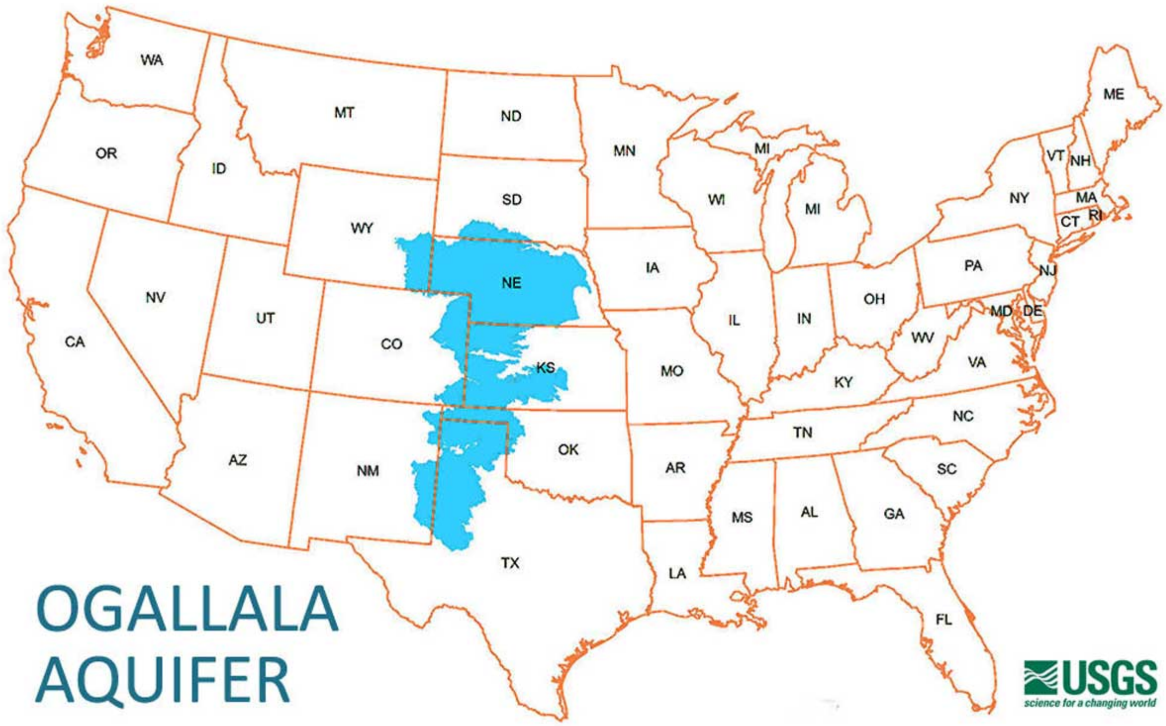


Figure 1.14: Source: (**modernfarmer?**)

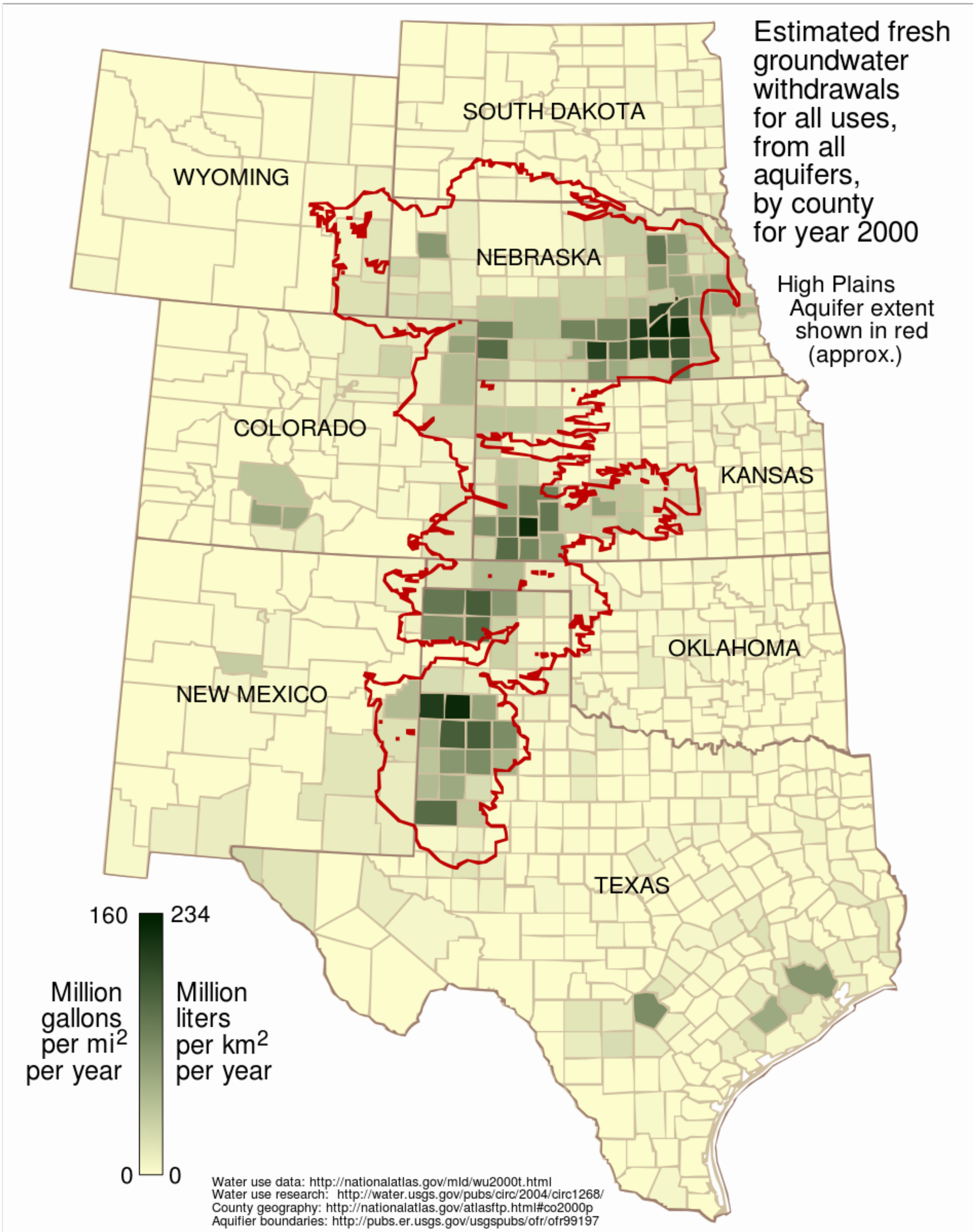


Figure 1.15: Source: (**ogallala1?**)

Center Pivot irrigation in Nebraska taps the Ogallala Aquifer.

1.6.14 Groundwater flow and discharge

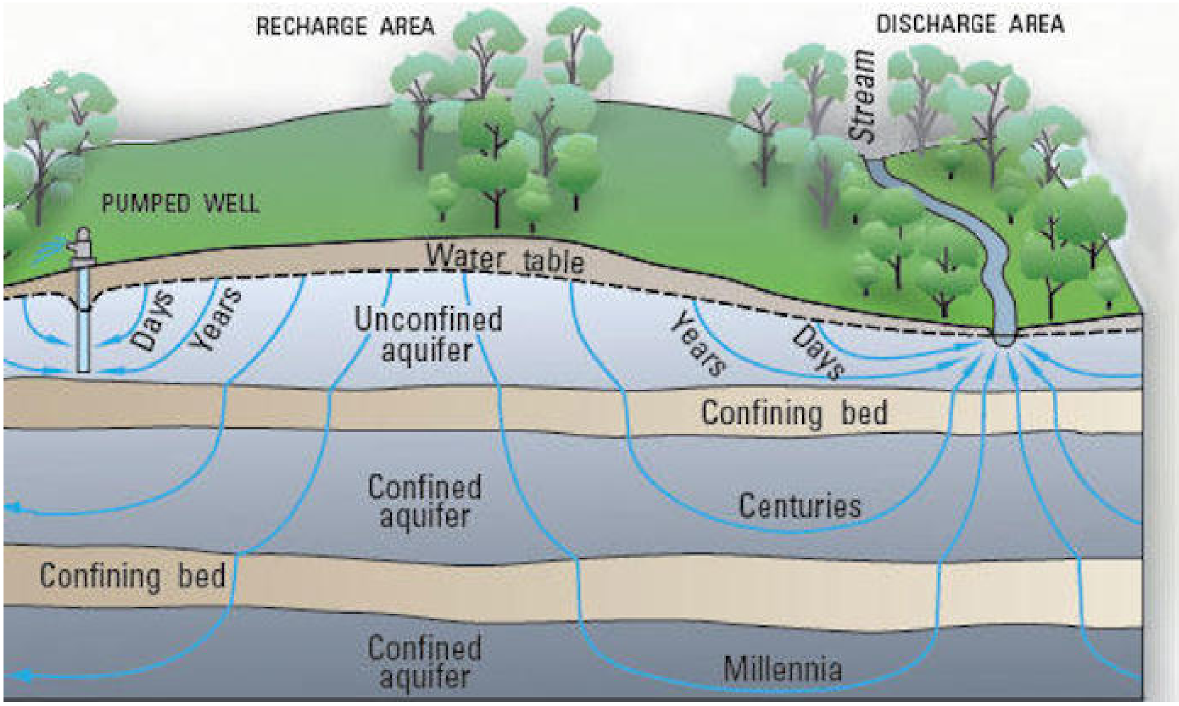


Figure 1.16: Source: Water Science School (2019a)

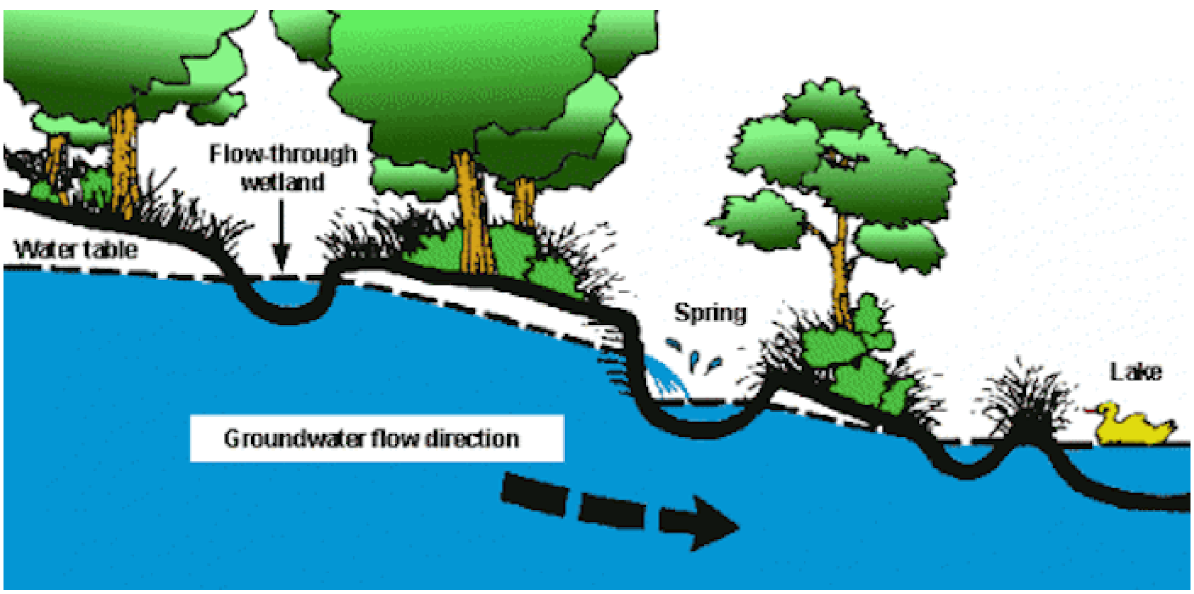


Figure 1.17: Source: Raymond, Lyle S. Jr. (1988)

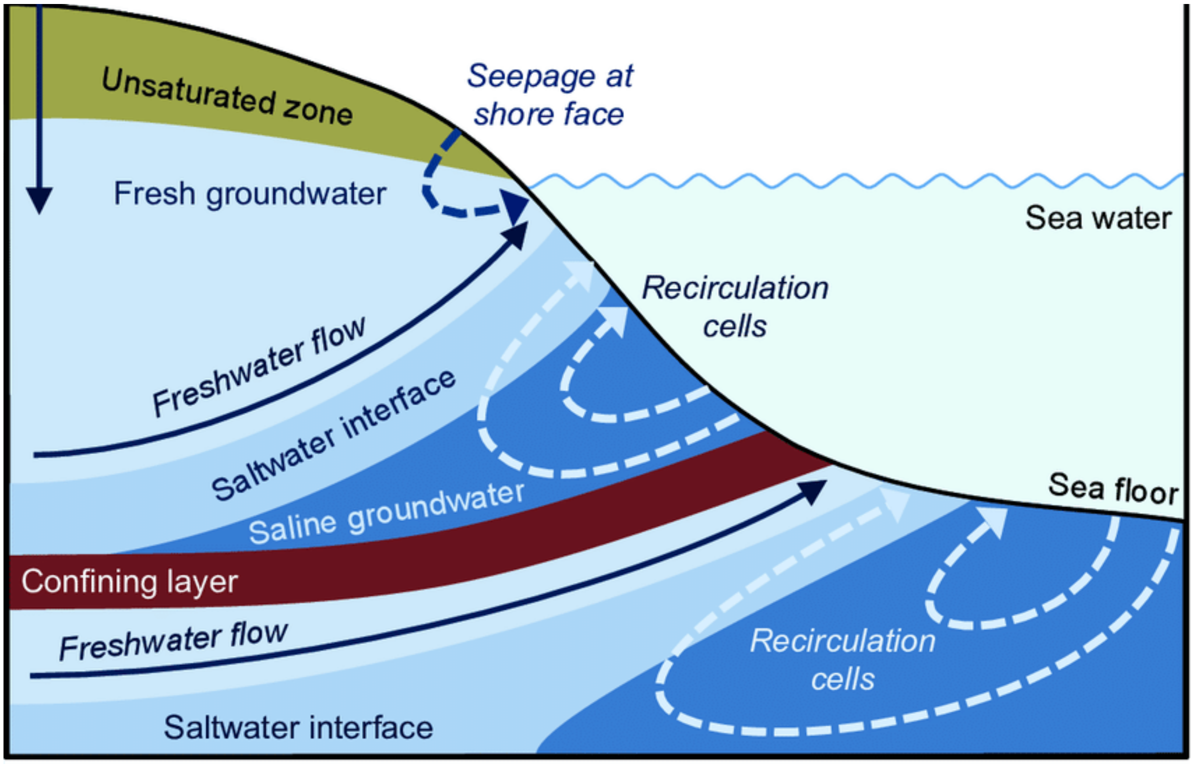


Figure 1.18: Source: Valentí Rodellas (1988)

1.6.15 Spring

Ein Gedi

Thousand Springs, Idaho 

2 Exercises

let's have fun plotting some data

2.1 download the data

1. Go to the Faculty of Agriculture's weather station.
2. Click on and download data for 1 September 2020 to 28 February 2021, with a 24h interval. Call it `data-sep2020-feb2021`
3. Open the .csv file with Excel, see how it looks like
4. If you can't download the data, just click here.

2.2 import packages

We need to import this data into python. First we import useful packages. **Type** (don't copy and paste) the following lines in the code cell below.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
```

2.3 import data with pandas

Import data from csv and put it in a pandas **dataframe** (a table). Make line 5 the header (column names)

```
df = pd.read_csv("data-sep2020-feb2021.csv",
                 skiprows=4,
                 encoding='latin1',
                 )
df
```

	Unnamed: 0	°C	°C.1	km/h	mm	mm.1
0	01/09/20	32.8	25.3	29.7	0.0	0.0
1	02/09/20	33.0	24.0	28.8	0.0	0.0
2	03/09/20	34.2	23.8	31.6	0.0	0.0
3	04/09/20	36.3	27.3	24.2	0.0	0.0
4	05/09/20	34.2	26.3	22.4	0.0	0.0
...	...	...	...	...	...	...
176	24/02/21	20.6	9.9	28.8	0.0	481.7
177	25/02/21	19.4	9.3	23.3	0.0	481.7
178	26/02/21	21.3	8.0	24.2	0.1	481.8
179	27/02/21	23.4	9.2	30.6	0.0	481.8
180	28/02/21	19.7	9.2	22.4	0.0	481.8

2.4 rename columns

rename the columns to:
`date, tmax, tmin, wind, rain24h, rain_cumulative`

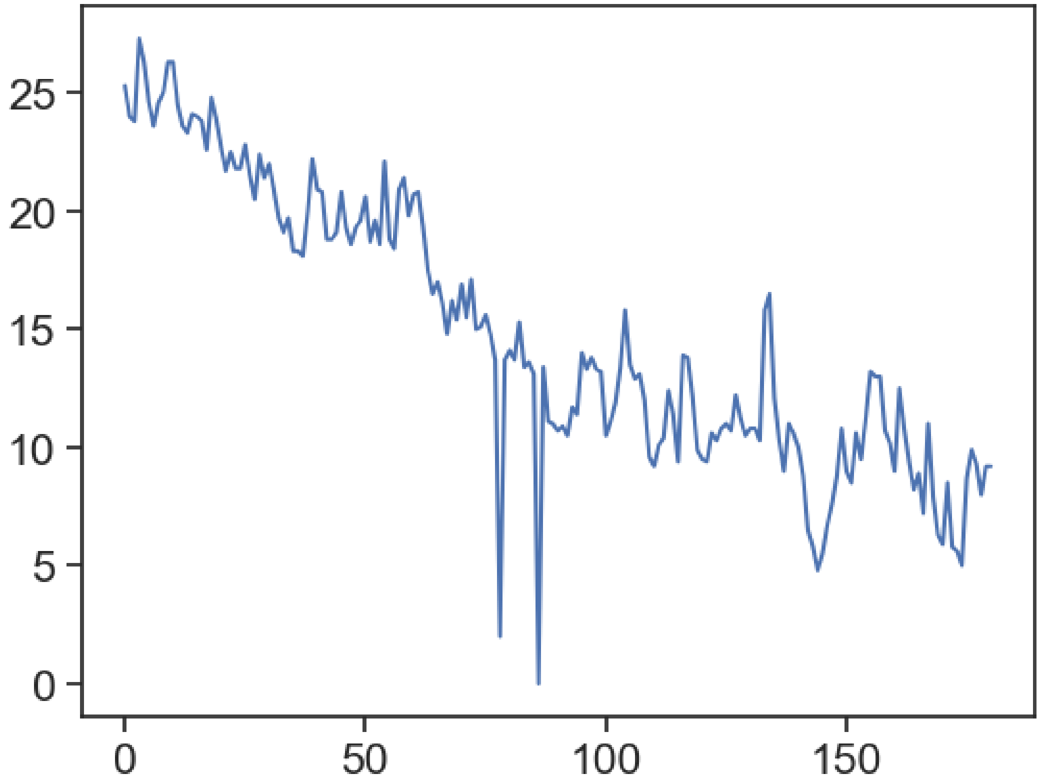
```
df.columns = ['date', 'tmax', 'tmin', 'wind', 'rain24h', 'rain_cumulative']
df
```

	date	tmax	tmin	wind	rain24h	rain_cumulative
0	01/09/20	32.8	25.3	29.7	0.0	0.0
1	02/09/20	33.0	24.0	28.8	0.0	0.0
2	03/09/20	34.2	23.8	31.6	0.0	0.0
3	04/09/20	36.3	27.3	24.2	0.0	0.0
4	05/09/20	34.2	26.3	22.4	0.0	0.0
...	...	...	...	...	...	...
176	24/02/21	20.6	9.9	28.8	0.0	481.7
177	25/02/21	19.4	9.3	23.3	0.0	481.7
178	26/02/21	21.3	8.0	24.2	0.1	481.8
179	27/02/21	23.4	9.2	30.6	0.0	481.8
180	28/02/21	19.7	9.2	22.4	0.0	481.8

2.5 a first plot!

plot the minimum temperature:

```
plt.plot(df['tmin'])
```



2.6 how to deal with dates

We want the dates to appear on the horizontal axis.
Interpret 'date' column as a pandas datetime, see how it looks different from before
before: 01/09/20
after: 2020-09-01

```
df['date'] = pd.to_datetime(df['date'], dayfirst=True)
df
```

	date	tmax	tmin	wind	rain24h	rain_cumulative
0	2020-09-01	32.8	25.3	29.7	0.0	0.0
1	2020-09-02	33.0	24.0	28.8	0.0	0.0
2	2020-09-03	34.2	23.8	31.6	0.0	0.0

	date	tmax	tmin	wind	rain24h	rain_cumulative
3	2020-09-04	36.3	27.3	24.2	0.0	0.0
4	2020-09-05	34.2	26.3	22.4	0.0	0.0
...	...	...	...	...	...	...
176	2021-02-24	20.6	9.9	28.8	0.0	481.7
177	2021-02-25	19.4	9.3	23.3	0.0	481.7
178	2021-02-26	21.3	8.0	24.2	0.1	481.8
179	2021-02-27	23.4	9.2	30.6	0.0	481.8
180	2021-02-28	19.7	9.2	22.4	0.0	481.8

2.6.1 date as dataframe index

Make 'date' the dataframe's index (leftmost column, but not really a column!)

```
df = df.set_index('date')
df
```

	tmax	tmin	wind	rain24h	rain_cumulative
date					
2020-09-01	32.8	25.3	29.7	0.0	0.0
2020-09-02	33.0	24.0	28.8	0.0	0.0
2020-09-03	34.2	23.8	31.6	0.0	0.0
2020-09-04	36.3	27.3	24.2	0.0	0.0
2020-09-05	34.2	26.3	22.4	0.0	0.0
...	...	...	...	...	...
2021-02-24	20.6	9.9	28.8	0.0	481.7
2021-02-25	19.4	9.3	23.3	0.0	481.7
2021-02-26	21.3	8.0	24.2	0.1	481.8
2021-02-27	23.4	9.2	30.6	0.0	481.8
2021-02-28	19.7	9.2	22.4	0.0	481.8

2.7 plot again, now with dates

Plot minimum temperature, now we have dates on the horizontal axis

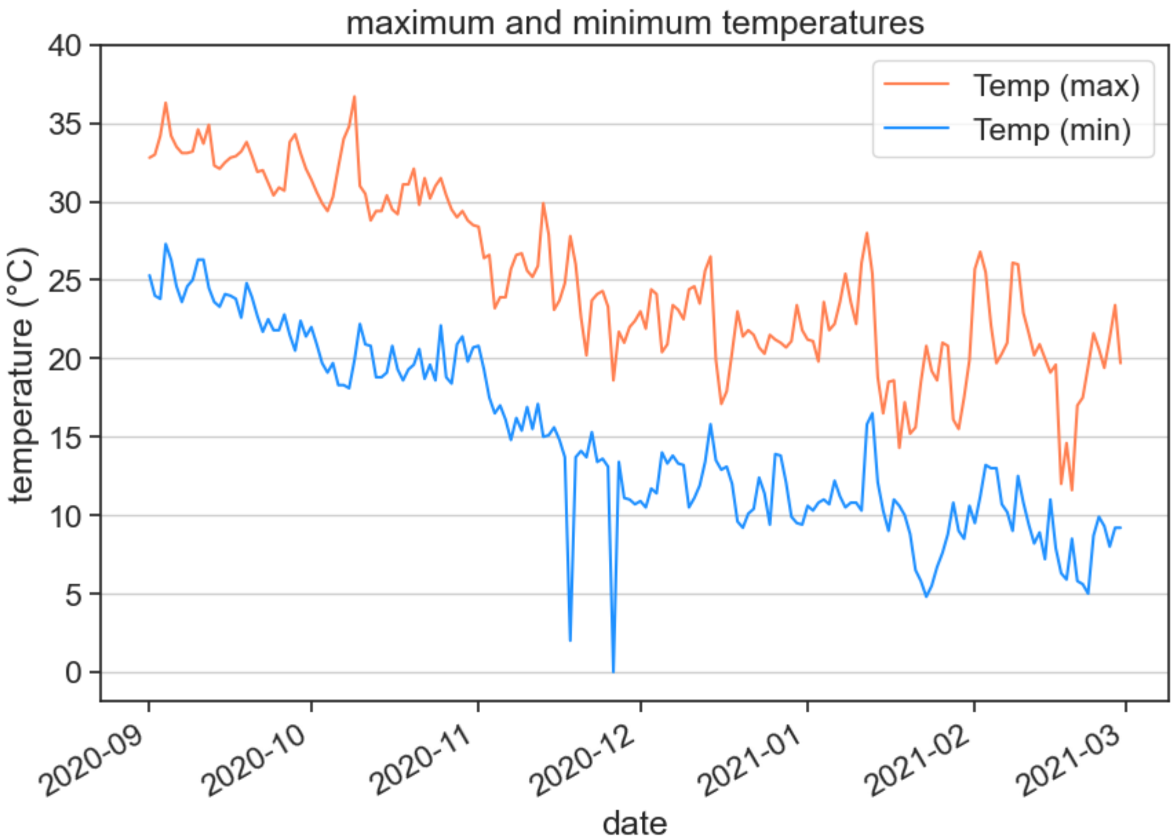
```
plt.plot(df['tmin'])
```

2.8 we're getting there! the graph could look better

Let's make the graph look better: labels, title, slanted dates, etc

```
# creates figure (the canvas) and the axis (rectangle where the plot sits)
fig, ax = plt.subplots(1, figsize=(10,7))
# two line plots
ax.plot(df['tmax'], color="coral", label="Temp (max)")
ax.plot(df['tmin'], color="dodgerblue", label="Temp (min)")
# axes labels and figure title
ax.set_xlabel('date')
ax.set_ylabel('temperature (°C)')
ax.set_title('maximum and minimum temperatures')
# some ticks adjustments
ax.set_yticks(np.arange(0,45,5))  # we can choose where to put ticks
ax.grid(axis='y')          # makes horizontal lines
plt.gcf().autofmt_xdate()  # makes slanted dates
# legend
```

```
ax.legend(loc='upper right')
# save png figure
plt.savefig("temp_max_min.png")
```



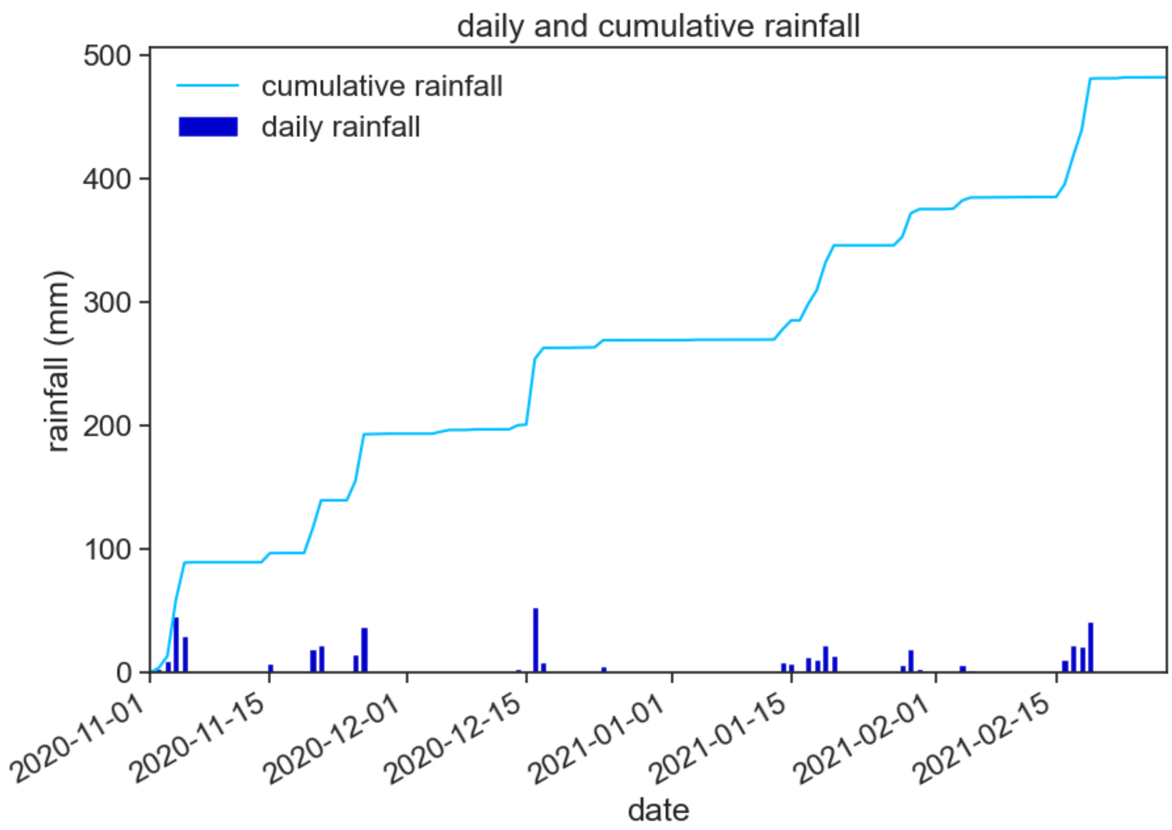
2.9 make the following figure

Use the following function to plot bars for daily rainfall

```
ax.bar(x_array, y_array)
```

Can you write yourself some lines of code that calculate the cumulative rainfall from the daily rainfall?

```
# creates figure (the canvas) and the axis (rectangle where the plot sits)
fig, ax = plt.subplots(1, figsize=(10,7))

# line and bar plots
ax.bar(df.index, df['rain24h'], color="mediumblue", label="daily rainfall")

# there are many ways of calculating the cumulative rain

# method 1, use a for loop:
# rain = df['rain24h'].to_numpy()
# cumulative = rain * 0
# for i in range(len(rain)):
#     cumulative[i] = np.sum(rain[:i])
# df['cumulative1'] = cumulative

# method 2, use list comprehension:
# rain = df['rain24h'].to_numpy()
# cumulative = [np.sum(rain[:i]) for i in range(len(rain))]
# df['cumulative2'] = cumulative

# method 3, use existing functions:
df['cumulative3'] = np.cumsum(df['rain24h'])

ax.plot(df['cumulative3'], color="deepskyblue", label="cumulative rainfall")
# compare our cumulative rainfall with the downloaded data
# ax.plot(df['rain_cumulative'], 'x')
# axes labels and figure title
ax.set(xlabel='date',
       ylabel='rainfall (mm)',
       title='daily and cumulative rainfall',
       xlim=pd.to_datetime(['2020-11-01','2021-02-28'])
      )
# some ticks adjustments
plt.gcf().autofmt_xdate()  # makes slanted dates
# legend
ax.legend(loc='upper left', frameon=False)
# save png figure
plt.savefig("cumulative_rainfall.png")
```



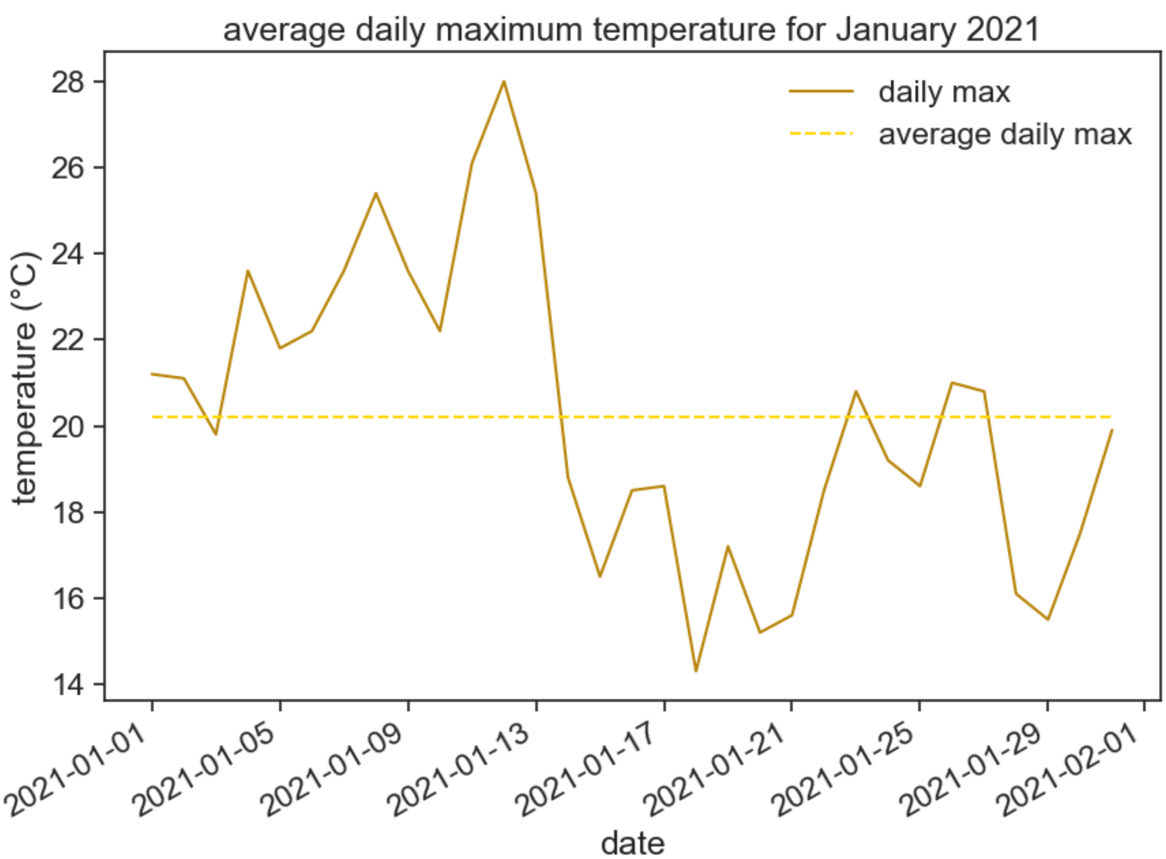
2.10 make another figure

In order to choose just a part of the time series, you can use the following:

```
start_date = '2021-01-01'
end_date = '2021-01-31'
january = df[start_date:end_date]
```

```
# creates figure (the canvas) and the axis (rectangle where the plot sits)
fig, ax = plt.subplots(1, figsize=(10,7))
# define date range
start_date = '2021-01-01'
end_date = '2021-01-31'
january = df.loc[start_date:end_date, 'tmax']
# plots
ax.plot(january, color="darkgoldenrod", label="daily max")
```

```
ax.plot(january*0 + january.mean(), color="gold", linestyle="--", label="average daily max")
# axes labels and figure title
ax.set_xlabel('date')
ax.set_ylabel('temperature (°C)')
ax.set_title('average daily maximum temperature for January 2021')
# some ticks adjustments
plt.gcf().autofmt_xdate()  # makes slanted dates
# legend
ax.legend(loc='upper right', frameon=False)
# save png figure
plt.savefig("average_max_temp.png")
```



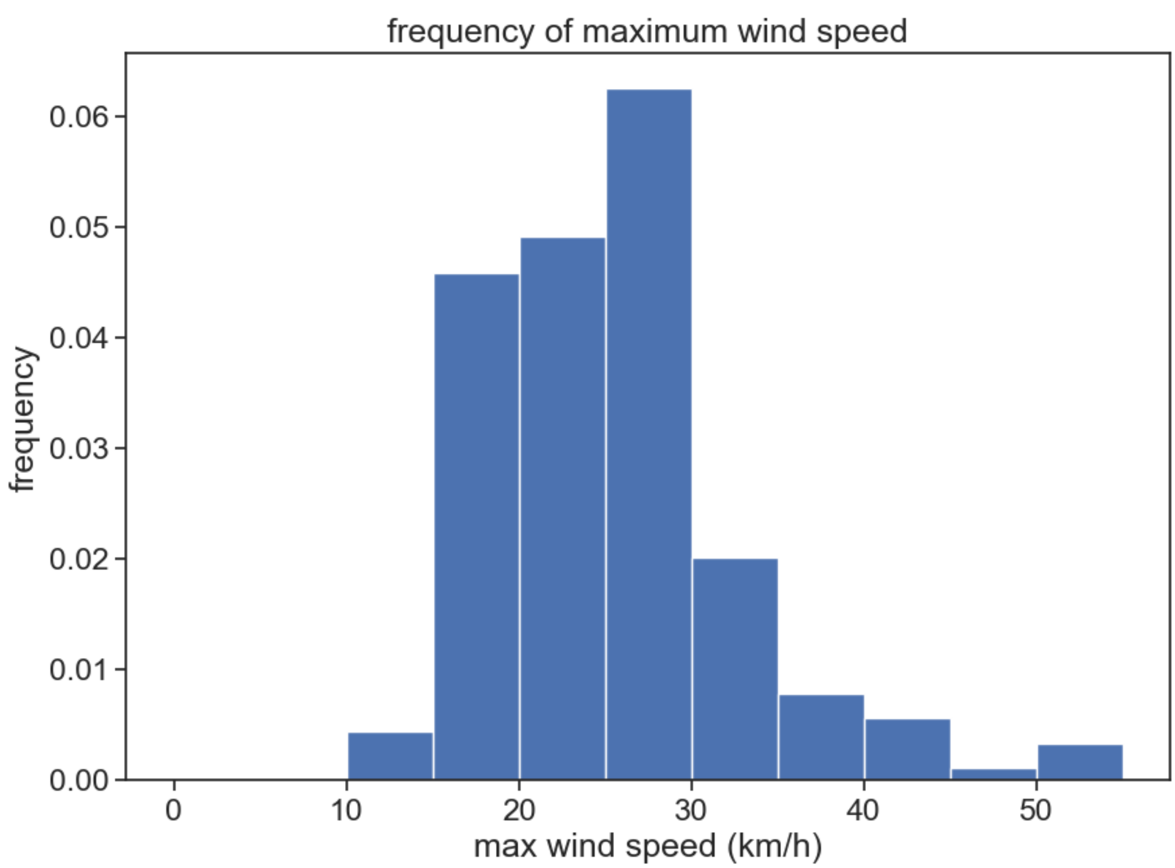
2.11 one last figure for today

Use the following code to create histograms with user-defined bins:

```
b = np.arange(0, 56, 5)  # bins from 0 to 55, width = 5
ax.hist(df['wind'], bins=b, density=True)
```

Play with the bins, see what happens. What does `density=True` do?

```
# creates figure (the canvas) and the axis (rectangle where the plot sits)
fig, ax = plt.subplots(1, figsize=(10,7))
# histogram
b = np.arange(0, 56, 5)  # bins from 0 to 55, width = 5
ax.hist(df['wind'], bins=b, density=True)
# axes labels and figure title
ax.set(xlabel='max wind speed (km/h)',
       ylabel='frequency',
       title='frequency of maximum wind speed'
      )
# save png figure
plt.savefig("wind-histogram.png")
```



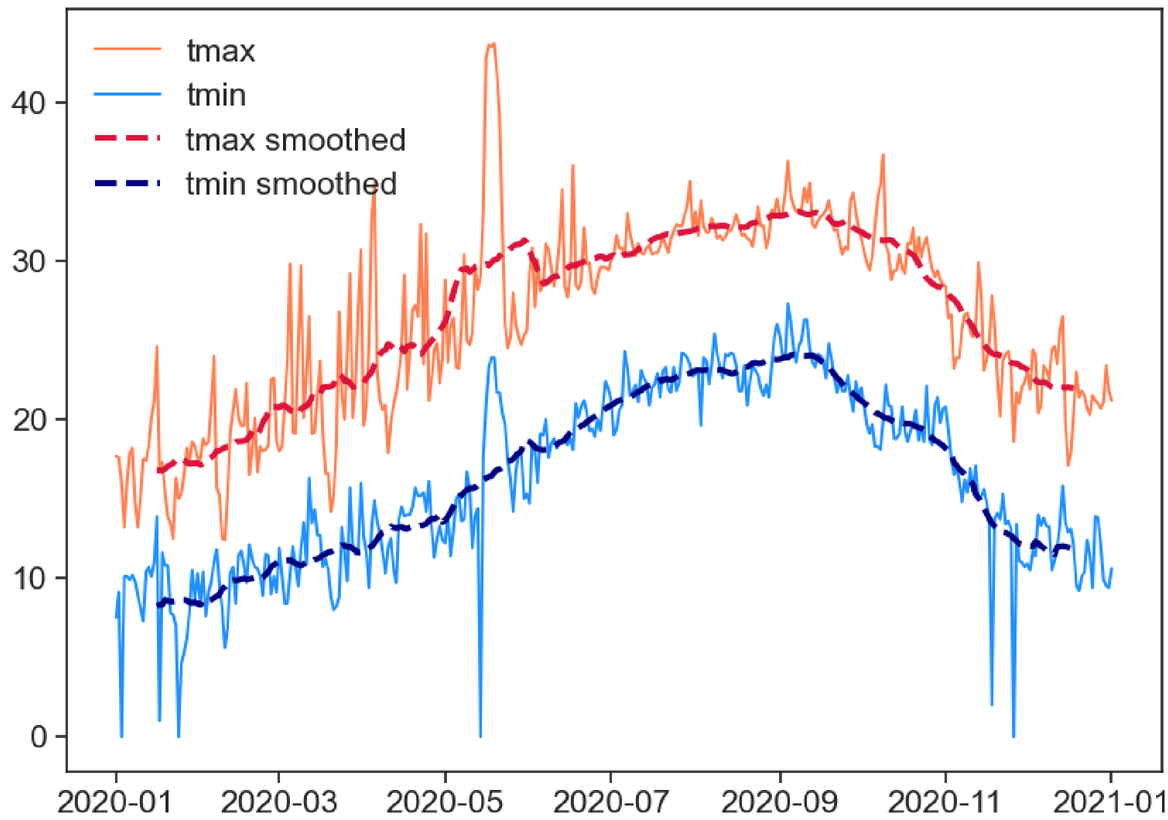
3 homework

Go back to the weather station website, download one year of data from 01.01.2020 to 31.12.2020 (24h data). If you can't download the data, just click here. Make the following graph: - daily tmax and tmin - smoothed data for tmax and tmin

In order to smooth the data with a 30 day window, use the following function:
`df['tmin'].rolling(30, center=True).mean()`
This means that you will take the mean of 30 days, and put the result in the center of this 30-day window.

Play with this function, see what you can do with it. What happens when you change the size of the window? Why is the smoothed data shorter than the original data? See the documentation for `rolling` to find more options.

```
fig, ax = plt.subplots(figsize=(10,7))
col_names = ['date', 'tmax', 'tmin', 'wind', 'rain24h', 'rain_cumulative']
df2 = pd.read_csv("1year.csv",
                  skiprows=5,
                  encoding='latin1',
                  names=col_names,
                  parse_dates=['date'],
                  dayfirst=True,
                  index_col='date'
                 )
plt.plot(df2['tmax'], label='tmax', color="coral")
plt.plot(df2['tmin'], label='tmin', color="dodgerblue")
tmin_smooth = df2['tmin'].rolling(30, center=True).mean()
tmax_smooth = df2['tmax'].rolling(30, center=True).mean()
plt.plot(tmax_smooth, label='tmax smoothed', color="crimson", linestyle="--", linewidth=3)
plt.plot(tmin_smooth, label='tmin smoothed', color="navy", linestyle="--", linewidth=3)
plt.legend(frameon=False)
plt.savefig("t_smoothed.png")
```


tmax
tmin
tmax smoothed
tmin smoothed
40
30
20
10
0
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
2021-01

Part II

Precipitation

Here are some of the files we'll use in this module, in case you can't download them from their original repositories.

- BEN_GURION_monthly.csv
- BEER_SHEVA_monthly.csv
- Eilat_daily.csv

4 Interannual variability of precipitation

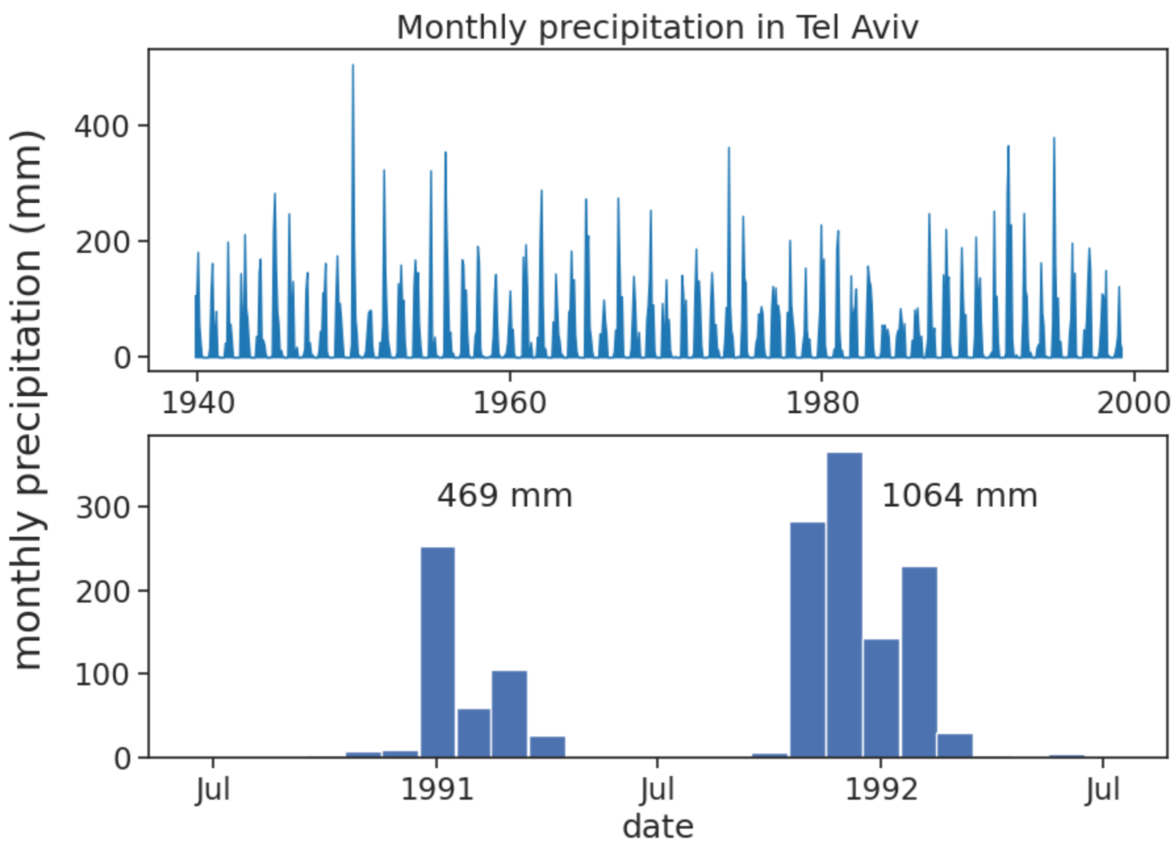
```python
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()  # datetime converter for a matplotlib
from calendar import month_abbr
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
import urllib.request
import matplotlib.dates as mdates
```

```python
df = pd.read_csv("TEL_AVIV_READING_monthly.csv",
                 sep=",",
                 parse_dates=['DATE'],
                 index_col='DATE'
                 )
```

```python
def concise(ax):
    locator = mdates.AutoDateLocator(minticks=3, maxticks=7)
    formatter = mdates.ConciseDateFormatter(locator)
    ax.xaxis.set_major_locator(locator)
    ax.xaxis.set_major_formatter(formatter)
```

```python
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,7))

# plot precipitation
ax1.fill_between(df.index, df['PRCP'], 0, color='tab:blue')
df_1990_1992 = df.loc['1990-07-01':'1992-07-01']
ax2.bar(df_1990_1992.index, df_1990_1992['PRCP'], width=30)

# adjust labels, ticks, title, etc
ax1.set_title("Monthly precipitation in Tel Aviv")
```

```
# ax2.tick_params(axis='x', rotation=45)
ax2.set_xlabel("date")
concise(ax1)
concise(ax2)

# common y label between the two panels:
fig.supylabel('monthly precipitation (mm)')

# write yearly rainfall
rain_1990_1991 = df.loc['1990-07-01':'1991-07-01','PRCP'].sum()
rain_1991_1992 = df.loc['1991-07-01':'1992-07-01','PRCP'].sum()
ax2.text('1991-01-01', 300, "{:.0f} mm".format(rain_1990_1991))
ax2.text('1992-01-01', 300, "{:.0f} mm".format(rain_1991_1992))
pass
# save figure
# plt.savefig("monthly_tel_aviv_1940-1999.png")
```

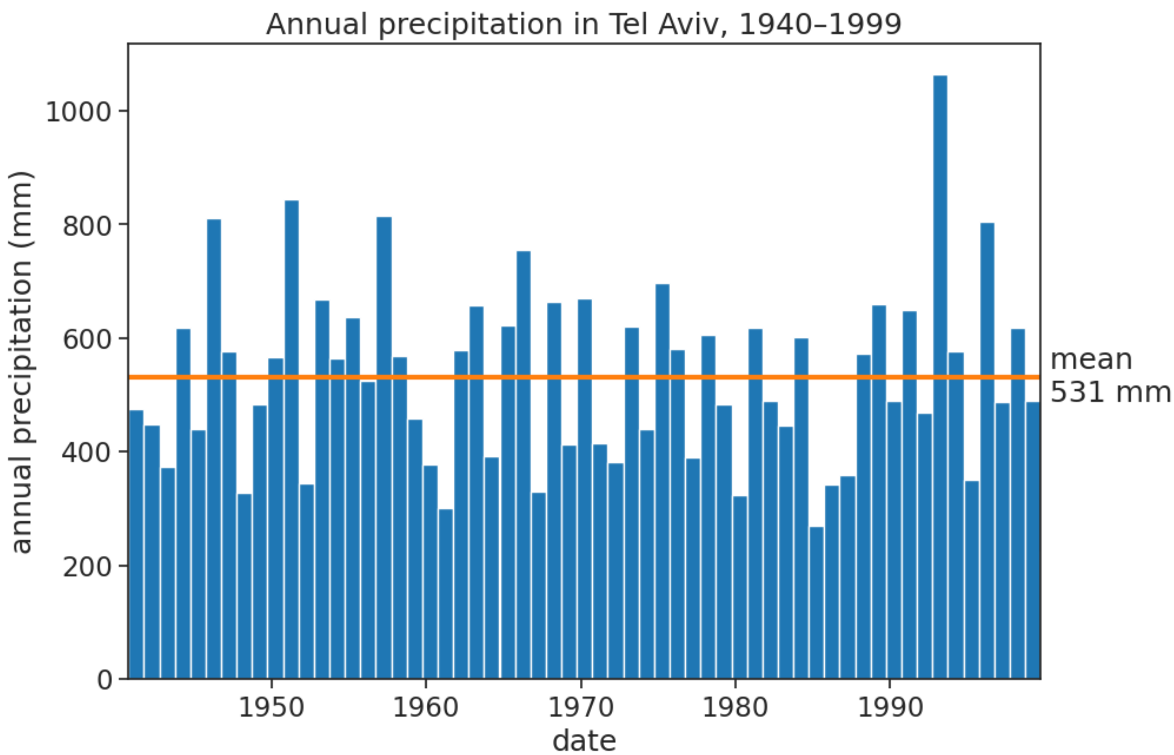


4.1 hydrological year

A time period of 12 months for which precipitation totals are measured. The hydrological year is designated by the calendar year in which it **ends**.
Let's define the hydrological year for Tel Aviv from 1 October to 30 September.

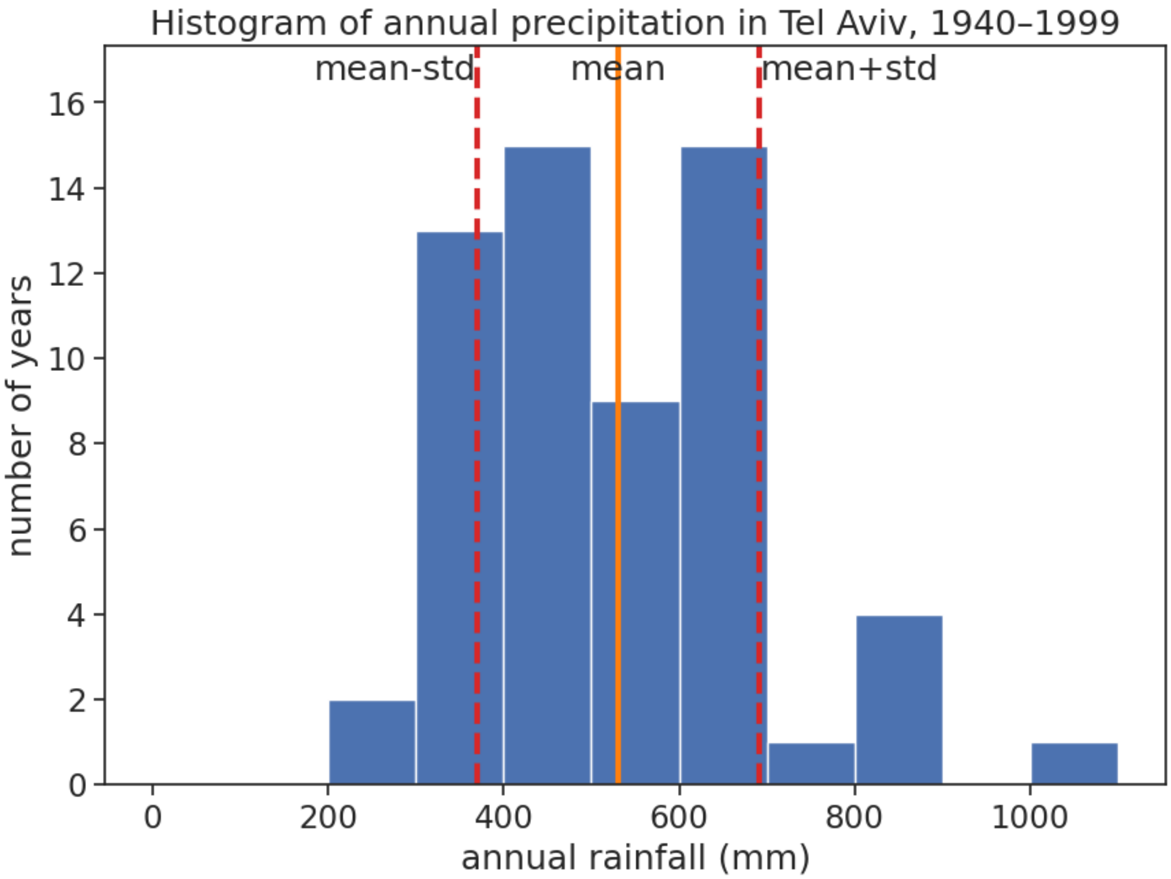
```
# read more about resampling options
# https://pandas.pydata.org/pandas-docs/version/0.12.0/timeseries.html#offset-aliases
# also, annual resampling can be anchored to the end of specific months:
# https://pandas.pydata.org/pandas-docs/version/0.12.0/timeseries.html#anchored-offsets
df_year = df['PRCP'].resample('YE-SEP').sum().to_frame()  # yearly frequency, anchored end of
df_year.columns = ['rain (mm)'] # rename 'PRCP' column to 'rain (mm)'
# the last year is the sum of only on month (November), let's take it out
df_year = df_year.iloc[:-1]  # exclude last row
```

```
fig, ax = plt.subplots(figsize=(10,7))

# plot YEARLY precipitation
ax.bar(df_year.index, df_year['rain (mm)'],
       width=365, align='edge', color="tab:blue")

# plot mean
rain_mean = df_year['rain (mm)'].mean()
ax.plot(df_year*0 + rain_mean, linewidth=3, color="tab:orange")

# adjust labels, ticks, title, etc
ax.set(title="Annual precipitation in Tel Aviv, 1940-1999",
       xlabel="date",
       ylabel="annual precipitation (mm)",
       xlim=[df_year.index[0], df_year.index[-1]]
      )

# write mean on the right
ax.text(df_year.index[-1], rain_mean, " mean\n {:.0f} mm".format(rain_mean),
        horizontalalignment="left", verticalalignment="center");

# save figure
# plt.savefig("annual_tel_aviv_with_mean.png")
```




```
fig, ax = plt.subplots(figsize=(10,7))

# calculate mean and standard deviation
rain_mean = df_year['rain (mm)'].mean()
rain_std = df_year['rain (mm)'].std()

# plot histogram
b = np.arange(0, 1101, 100)  # bins from 0 to 55, width = 5
ax.hist(df_year, bins=b)

# plot vertical lines with mean, std, etc
ylim = np.array(ax.get_ylim())
ylim[1] = ylim[1]*1.1
ax.plot([rain_mean]*2, ylim, linewidth=3, color="tab:orange")
ax.plot([rain_mean+rain_std]*2, ylim, linewidth=3, linestyle="--", color="tab:red")
ax.plot([rain_mean-rain_std]*2, ylim, linewidth=3, linestyle="--", color="tab:red")
ax.set_ylim(ylim)

# write mean, std, etc
ax.text(rain_mean, ylim[1]*0.99, "mean",
```

```
        horizontalalignment="center",
        verticalalignment="top",
        )
ax.text(rain_mean+rain_std, ylim[1]*0.99, "mean+std",
        horizontalalignment="left",
        verticalalignment="top",
        )
ax.text(rain_mean-rain_std, ylim[1]*0.99, "mean-std",
        horizontalalignment="right",
        verticalalignment="top",
        )

# adjust labels, ticks, title, limits, etc
ax.set(title="Histogram of annual precipitation in Tel Aviv, 1940-1999",
       xlabel="annual rainfall (mm)",
       ylabel="number of years"
       );

# save figure
# plt.savefig("histogram_tel_aviv_with_mean_and_std.png")
```



4.2 coefficient of variation

$\langle P \rangle$ = average precipitation
σ = standard deviation

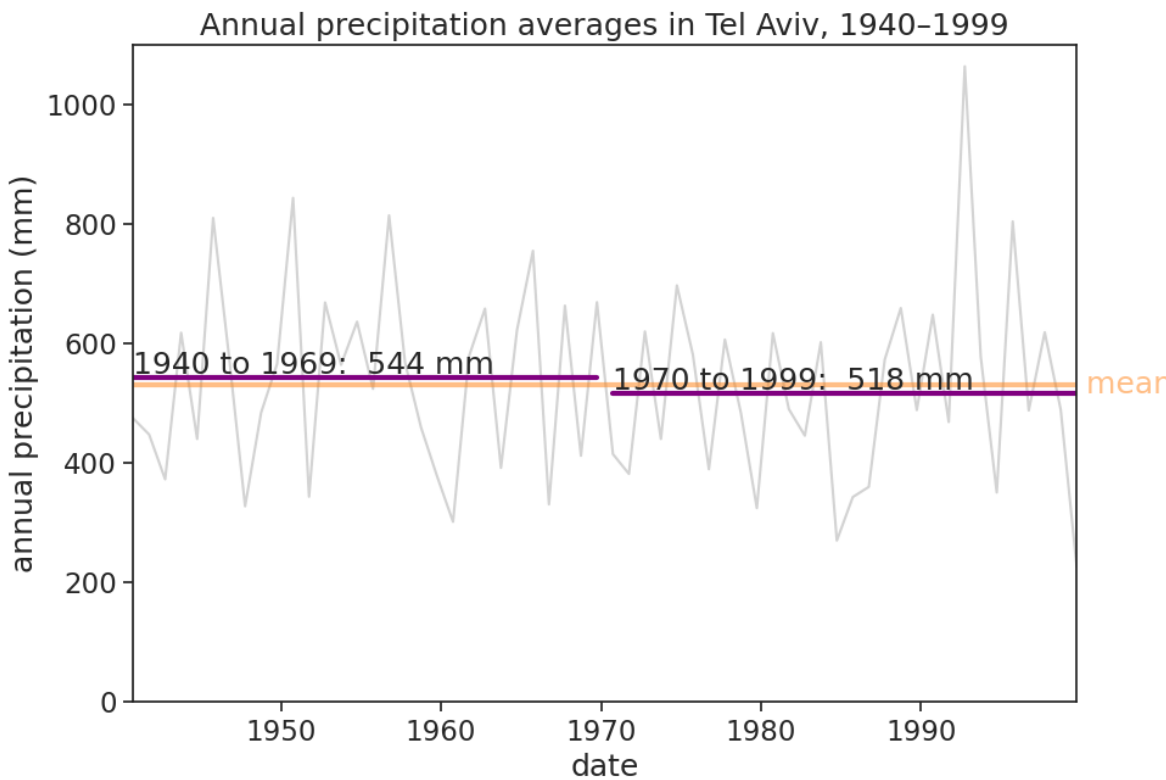
$$CV = \frac{\sigma}{\langle P \rangle}$$

Assuming that the inter-annual distribution is a gaussian: 67% of the time, rainfall will vary +/- 30% from its long term average in Tel Aviv.

Precipitation averages are usually calculated for time intervals of 30 years.

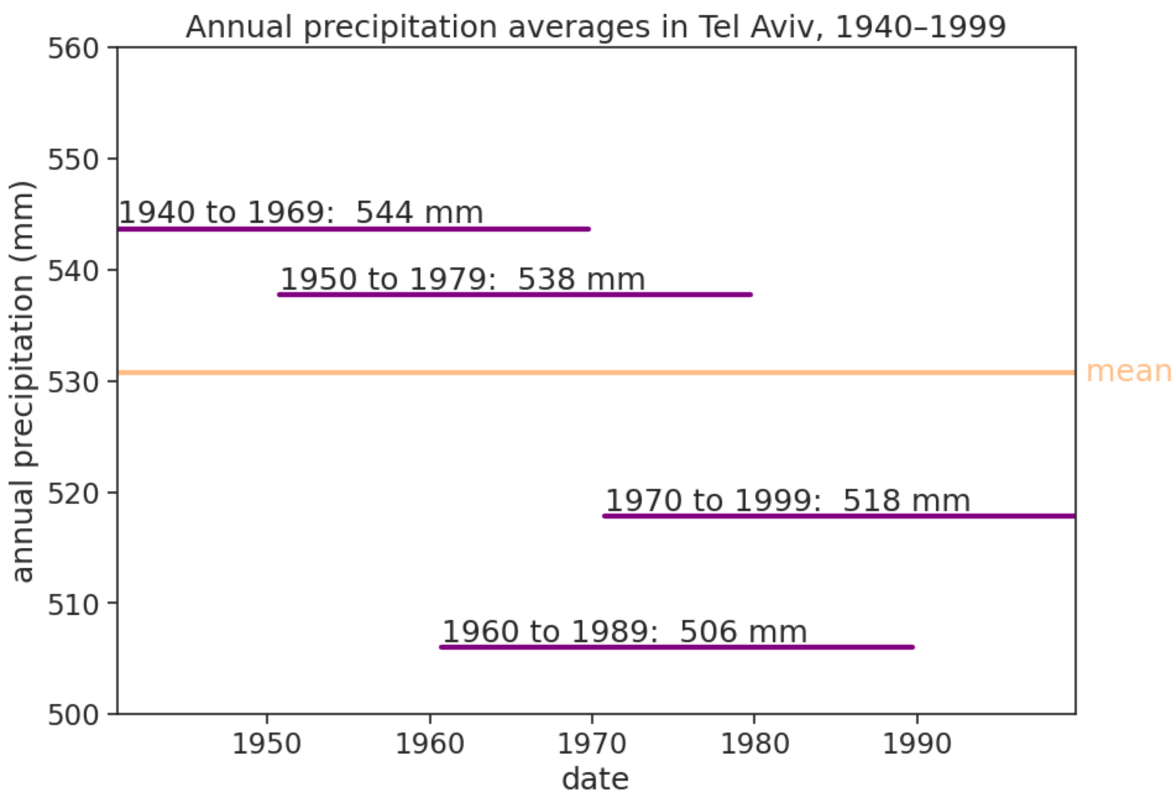
```
fig, ax = plt.subplots(figsize=(10,7))

ax.plot(df_year['rain (mm)'], color="lightgray")
```

```
# windows of length 30 years
windows = [[1940,1969], [1970,1999]]
for window in windows:
    start_date = f"{window[0]:d}-09-30"
    end_date = f"{window[1]:d}-09-30"
    window_mean = df_year['rain (mm)'][start_date:end_date].mean()
    ax.plot(df_year[start_date:end_date]*0+window_mean, color="purple", linewidth=3)
    ax.text(start_date, window_mean+5, f"{window[0]} to {window[1]}:  {window_mean:.0f} mm",

# plot mean
rain_mean = df_year['rain (mm)'].mean()
ax.plot(df_year*0 + rain_mean, linewidth=3, color="tab:orange", alpha=0.5)
ax.text(df_year.index[-1], rain_mean, " mean".format(rain_mean),
        horizontalalignment="left", verticalalignment="center",
        color="tab:orange", alpha=0.5)

# adjust labels, ticks, title, limits, etc
ax.set(title="Annual precipitation averages in Tel Aviv, 1940-1999",
       xlabel="date",
       ylabel="annual precipitation (mm)",
       xlim=[df_year.index[0], df_year.index[-1]],
       ylim=[0, 1100],
      );

# save figure
# plt.savefig("mean_tel_aviv_2_windows.png")
```




```
fig, ax = plt.subplots(figsize=(10,7))

# windows of length 30 years
windows = [[x,x+29] for x in [1940,1950,1960,1970]]
for window in windows:
    start_date = f"{window[0]:d}-09-30"
    end_date = f"{window[1]:d}-09-30"
    window_mean = df_year['rain (mm)'][start_date:end_date].mean()
    ax.plot(df_year[start_date:end_date]*0+window_mean, color="purple", linewidth=3)
    ax.text(start_date, window_mean+0.5, f"{window[0]} to {window[1]}:  {window_mean:.0f} mm'

# plot mean
rain_mean = df_year['rain (mm)'].mean()
ax.plot(df_year*0 + rain_mean, linewidth=3, color="tab:orange", alpha=0.5)
ax.text(df_year.index[-1], rain_mean, " mean".format(rain_mean),
        horizontalalignment="left", verticalalignment="center",
        color="tab:orange", alpha=0.5)

# adjust labels, ticks, title, limits, etc
```

```
ax.set(title="Annual precipitation averages in Tel Aviv, 1940-1999",
       xlabel="date",
       ylabel="annual precipitation (mm)",
       xlim=[df_year.index[0], df_year.index[-1]],
       ylim=[500, 560],
      );

# save figure
# plt.savefig("mean_tel_aviv_2_windows.png")
```



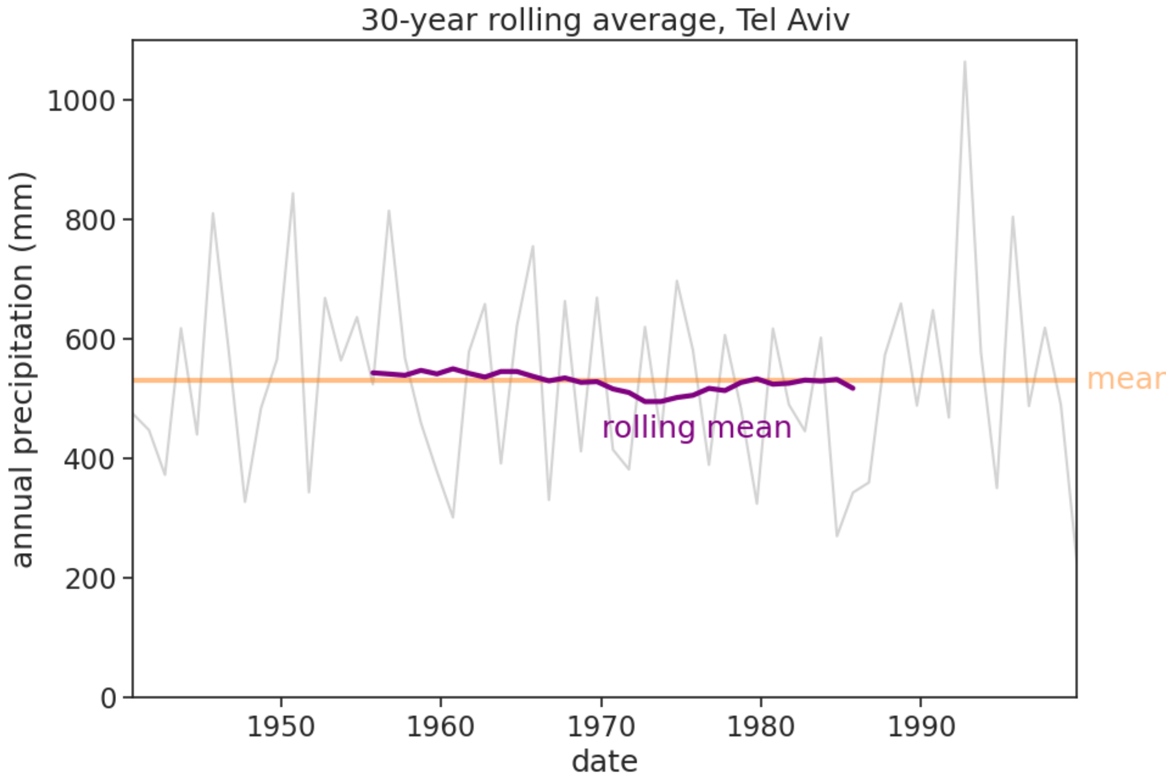

```
import altair as alt

# Custom theme for readability
def readable():
    return {
        "config" : {
             "title": {'fontSize': 16},
             "axis": {
```

```
                    "labelFontSize": 16,
                    "titleFontSize": 16,
                },
                "header": {
                    "labelFontSize": 14,
                    "titleFontSize": 14,
                },
                "legend": {
                    "labelFontSize": 14,
                    "titleFontSize": 14,
                },
                "mark": {
                    'fontSize': 14,
                    "tooltip": {"content": "encoding"},  # enable tooltips
                },
            }
        }

alt.themes.register('readable', readable)
alt.themes.enable('readable')

# Altair only recognizes column data; it ignores index values. You can plot the index data by
source = df_year.reset_index()
brush = alt.selection_interval(encodings=['x'])

# T: temporal, a time or date value
# Q: quantitative, a continuous real-valued quantity
# https://altair-viz.github.io/user_guide/encoding.html#encoding-data-types
bars = alt.Chart().mark_bar().encode(
    x=alt.X('DATE:T', axis=alt.Axis(title='date')),
    y=alt.Y('rain (mm):Q',  axis=alt.Axis(title='annual precipitation (mm) and average')),
    opacity=alt.condition(brush, alt.OpacityValue(1), alt.OpacityValue(0.2)),
).add_params(
    brush
).properties(
    title='Select year range and drag for rolling average of annual precipitation in Tel Aviv
).properties(
    width=600,
    height=400
)

line = alt.Chart().mark_rule(color='orange').encode(
```

```
    y='mean(rain (mm)):Q',
    size=alt.SizeValue(3)
).transform_filter(
    brush
)

alt.layer(bars, line, data=source)
```

```
alt.LayerChart(...)
```

```
fig, ax = plt.subplots(figsize=(10,7))

ax.plot(df_year['rain (mm)'], color="lightgray")

# plot rolling mean
rolling_mean = df_year.rolling(window=30, center=True).mean()
ax.plot(rolling_mean, linewidth=3, color="purple", zorder=5)
ax.text(pd.to_datetime("1970"), 450, "rolling mean".format(rain_mean),
        horizontalalignment="left", verticalalignment="center",
        color="purple",)

# plot mean
ax.plot(df_year*0 + rain_mean, linewidth=3, color="tab:orange", alpha=0.5)
ax.text(df_year.index[-1], rain_mean, " mean".format(rain_mean),
        horizontalalignment="left", verticalalignment="center",
        color="tab:orange", alpha=0.5);

ax.set(title="30-year rolling average, Tel Aviv",
       xlabel="date",
       ylabel="annual precipitation (mm)",
       ylim=[0, 1100],
       xlim=[df_year.index[0], df_year.index[-1]]
      );
# save figure
# plt.savefig("rolling_average_tel_aviv.png")
```



30-year rolling average, Tel Aviv
1000
800
600
400
200
0
annual precipitation (mm)
1950
1960
1970
1980
1990
date
mean
rolling mean

5 Exercises

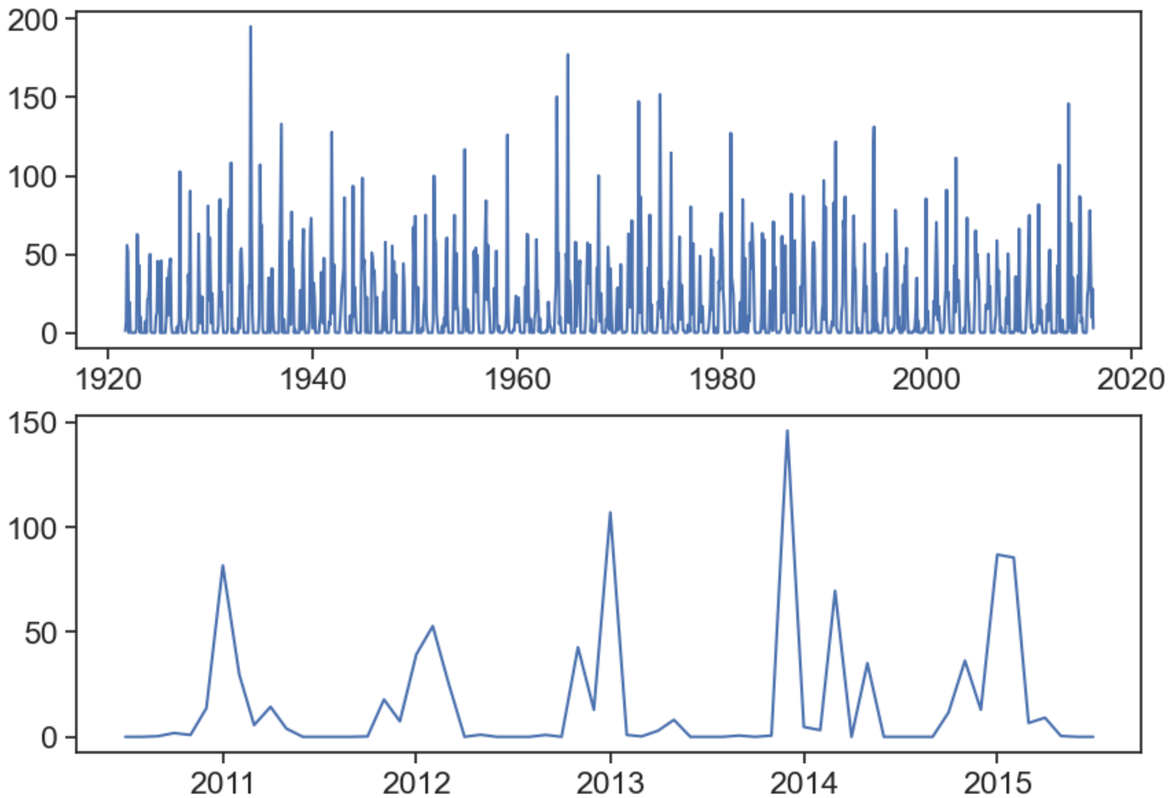
Import relevant packages

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from calendar import month_abbr
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
import urllib.request
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

Plot monthly rainfall for your station.

Load the data into a datafram, and before you continue with the analysis, plot the rainfall data, to see how it looks like.

```
df = pd.read_csv('BEN_GURION_monthly.csv', sep=",")
# make 'DATE' the dataframe index
df['DATE'] = pd.to_datetime(df['DATE'])
df = df.set_index('DATE')

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,7))
ax1.plot(df['PRCP'])
ax2.plot(df['PRCP']['2010-07-01':'2015-07-01'])
```



How to aggregate rainfall accoding to the hydrological year? We use the function `resample`.

read more about resampling options:
https://pandas.pydata.org/pandas-docs/version/0.12.0/timeseries.html#offset-aliases

also, annual resampling can be anchored to the end of specific months: https://pandas.pydata.org/pandas-docs/version/0.12.0/timeseries.html#anchored-offsets

```
# annual frequency, anchored 31 December
df_year_all = df['PRCP'].resample('A').sum().to_frame()
# annual frequency, anchored 01 January
df_year_all = df['PRCP'].resample('AS').sum().to_frame()
# annual frequency, anchored end of September
df_year_all = df['PRCP'].resample('A-SEP').sum().to_frame()
# rename 'PRCP' column to 'rain (mm)'
df_year_all.columns = ['rain (mm)']
df_year_all
```

DATE	rain (mm)
1922-09-30	136.6
1923-09-30	144.5
1924-09-30	130.4
1925-09-30	165.3
1926-09-30	188.7
...	...
2012-09-30	145.7
2013-09-30	175.3
2014-09-30	259.2
2015-09-30	249.3
2016-09-30	257.6

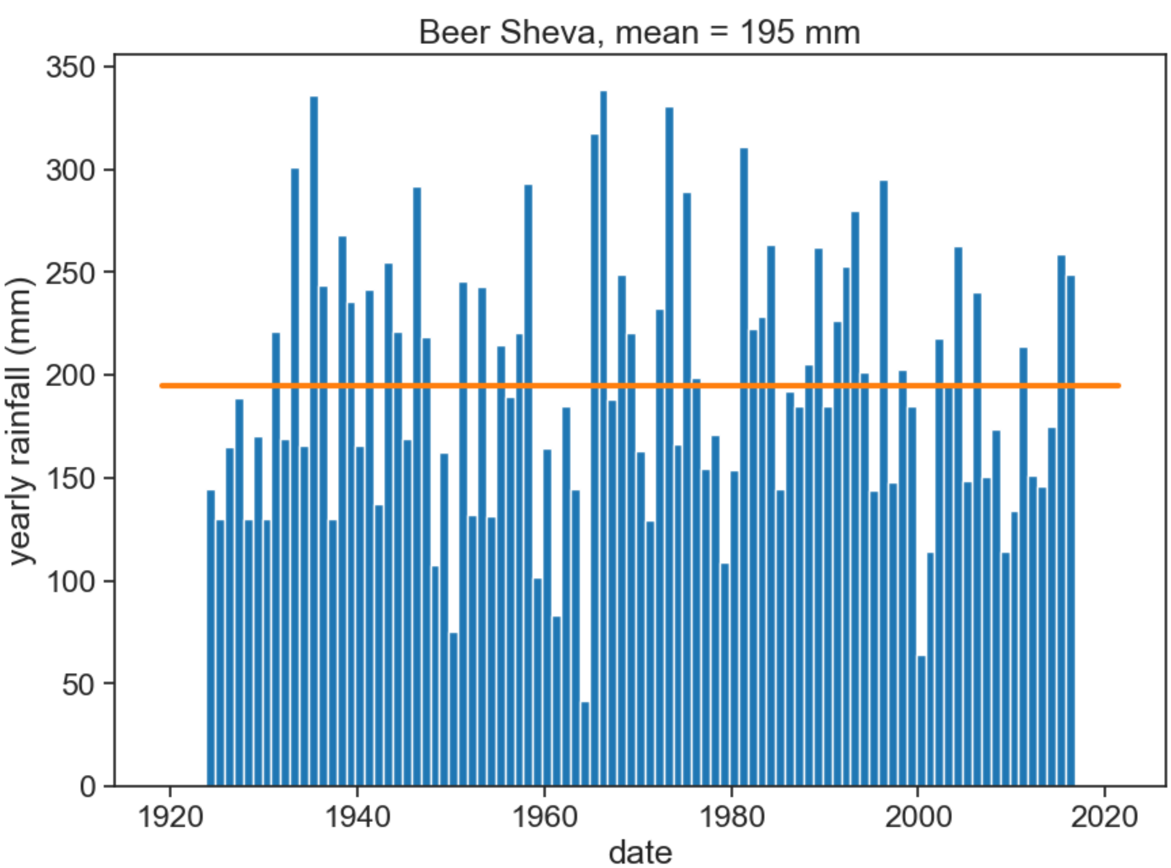
You might need to exclude the first or the last line, since their data might have less that 12 months. For example:

```
# exclude 1st row
df_year = df_year_all.iloc[1:]
# exclude last row
df_year = df_year_all.iloc[:-1]
# exclude both 1st and last rows
df_year = df_year_all.iloc[1:-1]
df_year
```

DATE	rain (mm)
1923-09-30	144.5
1924-09-30	130.4
1925-09-30	165.3
1926-09-30	188.7
1927-09-30	130.2
...	...
2011-09-30	151.6
2012-09-30	145.7
2013-09-30	175.3
2014-09-30	259.2
2015-09-30	249.3

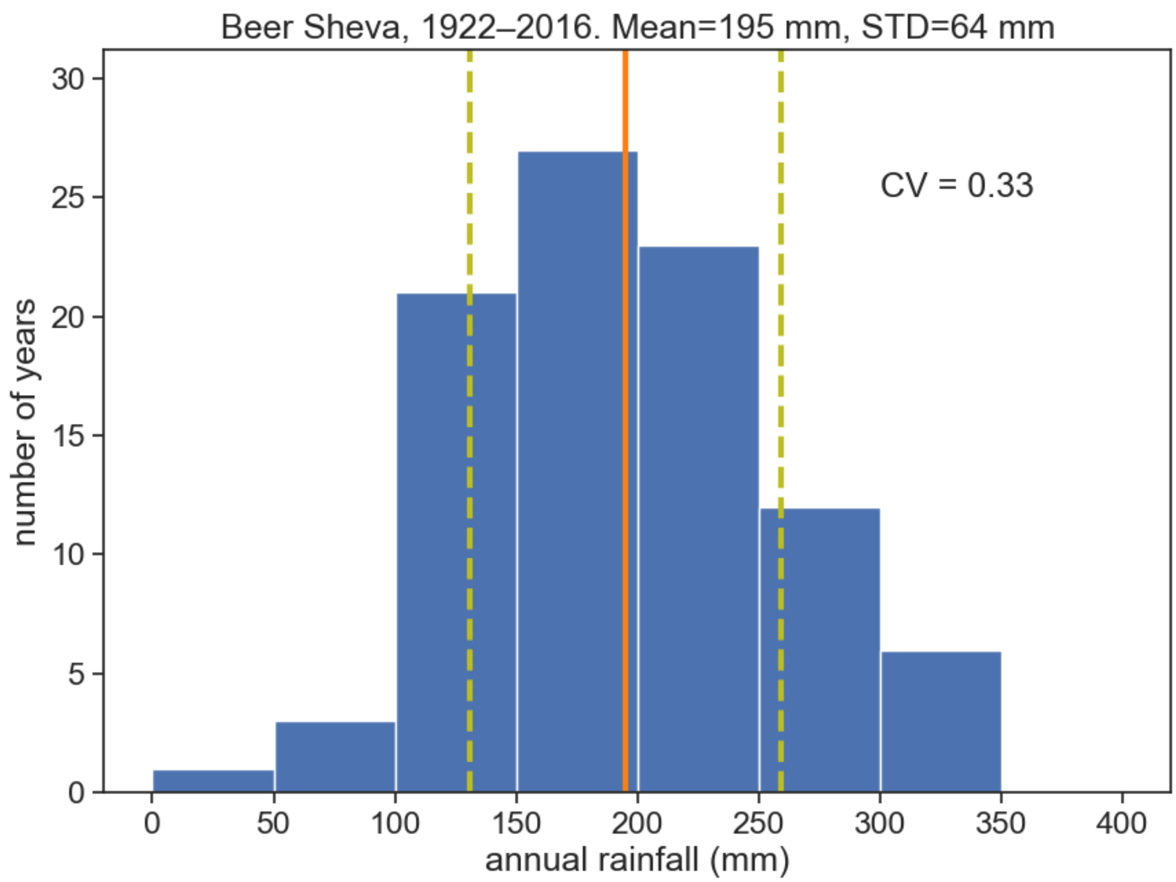
Calculate the average annual rainfall. Plot annual rainfall for the whole range, together with the average. You should get something like this:

```
fig, ax = plt.subplots(figsize=(10,7))

# plot YEARLY precipitation
ax.bar(df_year.index, df_year['rain (mm)'],
       width=365, align='edge', color="tab:blue")

# plot mean
rain_mean = df_year['rain (mm)'].mean()
ax.plot(ax.get_xlim(), [rain_mean]*2, linewidth=3, color="tab:orange")
ax.set(xlabel="date",
       ylabel="yearly rainfall (mm)",
       title=f"Beer Sheva, mean = {rain_mean:.0f} mm");
# save figure
# plt.savefig("hydrology_figures/beersheva_yearly_rainfall_1923_2016.png")
```



Plot a histogram of annual rainfall, with the mean and standard deviation. Calculate the coefficient of variation. Try to plot something like this:

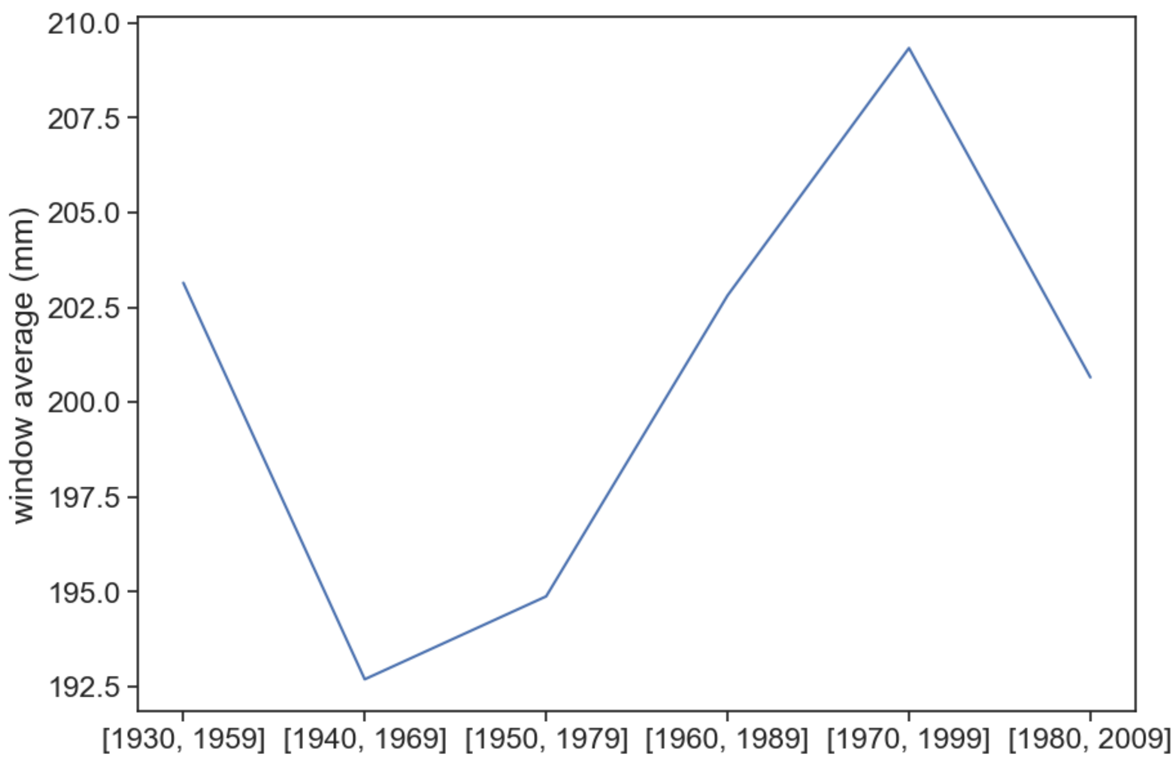
```
fig, ax = plt.subplots(figsize=(10,7))

# calculate mean and standard deviation
rain_mean = df_year['rain (mm)'].mean()
rain_std = df_year['rain (mm)'].std()

# plot histogram
b = np.arange(0, 401, 50)  # bins from 0 to 400, width = 50
ax.hist(df_year['rain (mm)'], bins=b)

# plot vertical lines with mean, std, etc
ylim = np.array(ax.get_ylim())
ylim[1] = ylim[1]*1.1
ax.plot([rain_mean]*2, ylim, linewidth=3, color="tab:orange")
ax.plot([rain_mean+rain_std]*2, ylim, linewidth=3, linestyle="--", color="tab:olive")
ax.plot([rain_mean-rain_std]*2, ylim, linewidth=3, linestyle="--", color="tab:olive")
ax.set(ylim=ylim,
       xlabel="annual rainfall (mm)",
       ylabel="number of years",
       title=f"Beer Sheva, 1922-2016. Mean={rain_mean:.0f} mm, STD={rain_std:.0f} mm")
ax.text(300, 25, f"CV = {rain_std/rain_mean:.2f}")
plt.savefig("histogram_beersheva.png")
```



Calculate the mean annual rainfall for various 30-year intervals

```
####### the hard way #######
# fig, ax = plt.subplots(figsize=(10,7))

# mean_30_59 = df_year.loc['1930-09-30':'1959-09-01','rain (mm)'].mean()
# mean_40_69 = df_year.loc['1940-09-30':'1969-09-01','rain (mm)'].mean()
# mean_50_79 = df_year.loc['1950-09-30':'1979-09-01','rain (mm)'].mean()
# mean_60_89 = df_year.loc['1960-09-30':'1989-09-01','rain (mm)'].mean()
# mean_70_99 = df_year.loc['1970-09-30':'1999-09-01','rain (mm)'].mean()
# mean_80_09 = df_year.loc['1980-09-30':'2009-09-01','rain (mm)'].mean()

# ax.plot([mean_30_59,
#          mean_40_69,
#          mean_50_79,
#          mean_60_89,
#          mean_70_99,
```

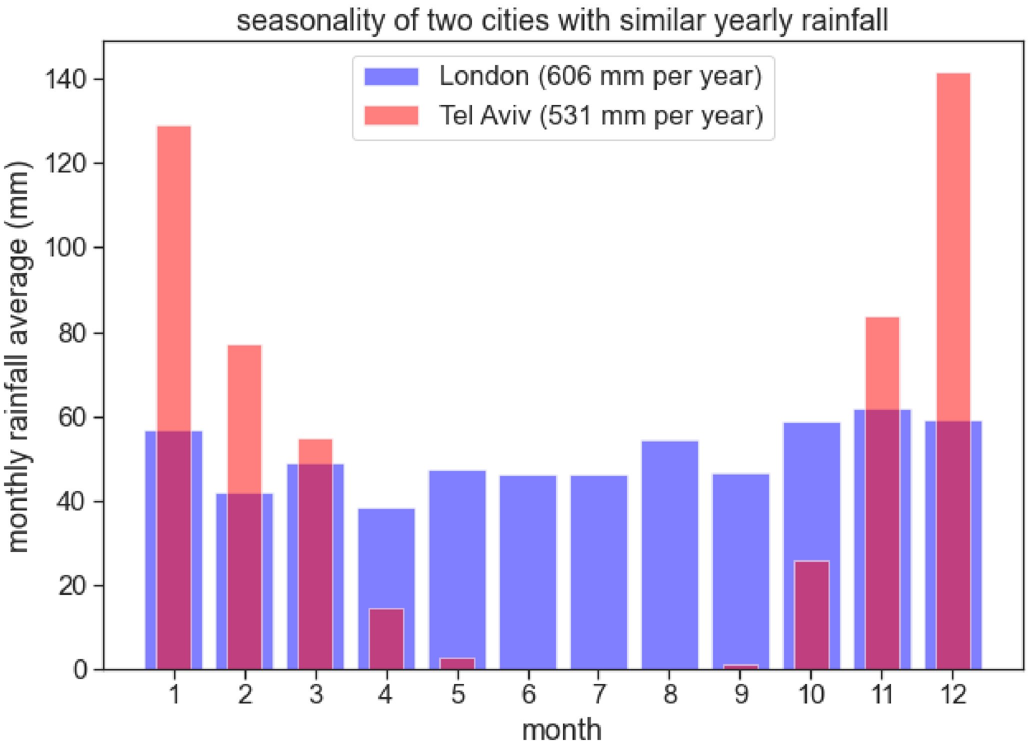
```
#          mean_80_09])


####### the easy way #######

fig, ax = plt.subplots(figsize=(10,7))

# use list comprehension
windows = [[x, x+29] for x in [1930,1940,1950,1960,1970,1980]]
mean = [df_year.loc[f'{w[0]:d}-09-30':f'{w[1]:d}-09-01','rain (mm)'].mean() for w in windows]
ax.plot(mean)
ax.set(xticks=np.arange(len(mean)),
       xticklabels=[str(w) for w in windows],
       ylabel="window average (mm)"
      );
```



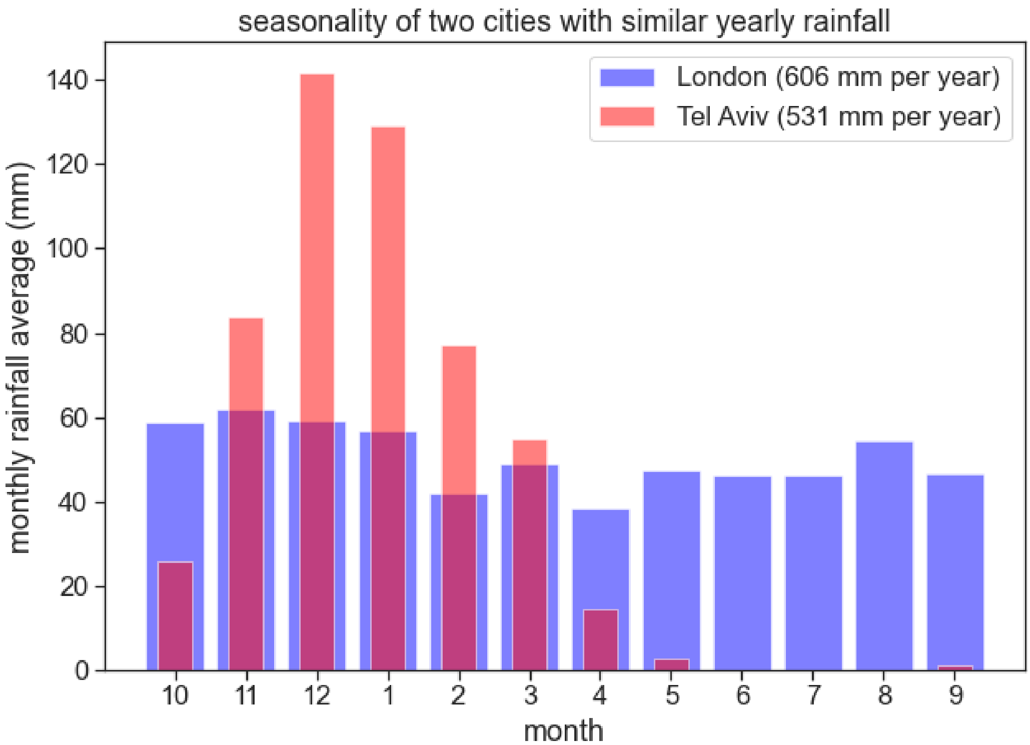
5.1 homework

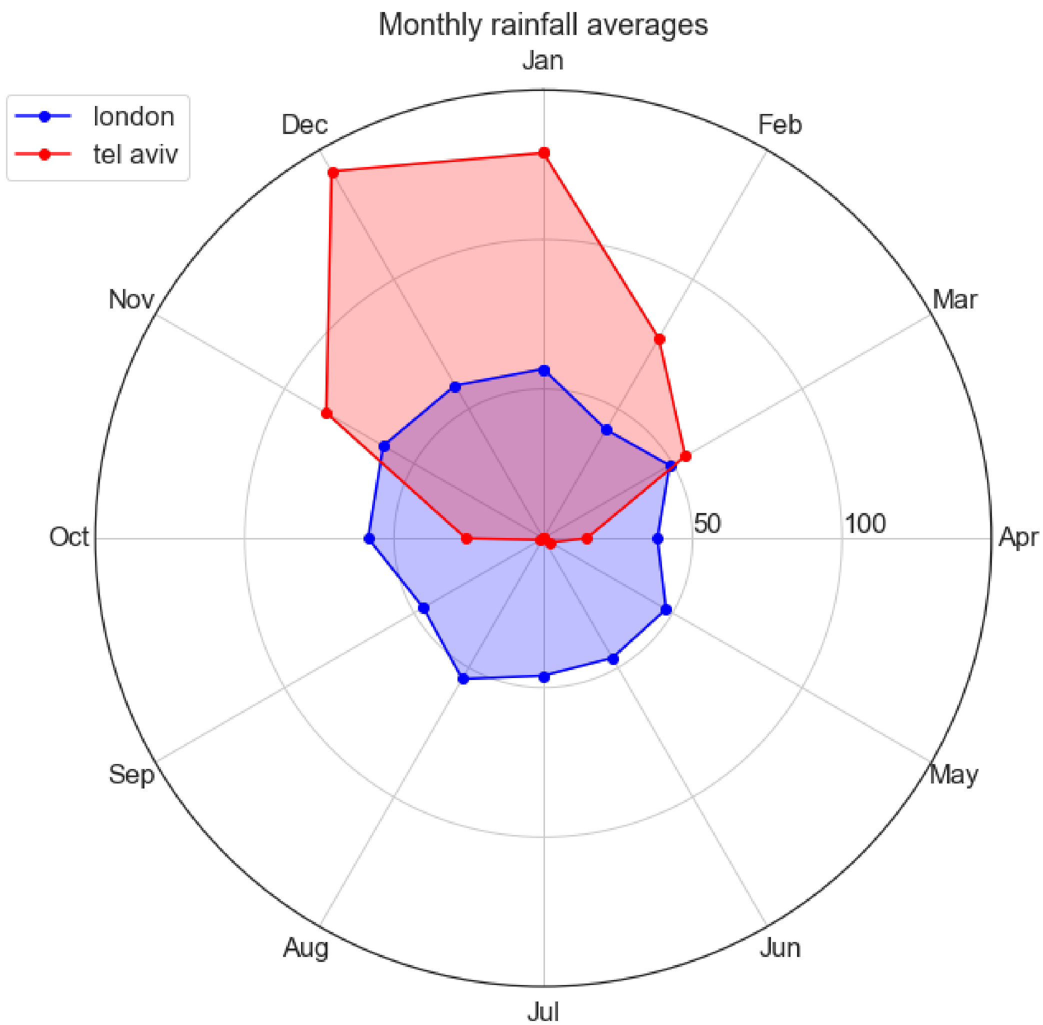
1. Download both daily and monthly data for London (LONDON HEATHROW, ID: UKM00003772). You should be aware that 'PRCP' for monthly data is in millimeters, while 'PRCP' for daily data is in **tens of millimiters**.
2. Aggregate daily data into monthly intervals using resample('MS').sum(). 'MS' means that the sum of all days in the month will be stored in the first day of the month. Supposedly both datasets are equal now.
3. Calculate the average annual rainfall, using each of these datasets.
4. Why is there such a big difference?

6 Intra-annual variability of precipitation



Let's shift the months according the the hydrological year:


seasonality of two cities with similar yearly rainfall
London (606 mm per year)
Tel Aviv (531 mm per year)
monthly rainfall average (mm)
0
20
40
60
80
100
120
140
10
11
12
1
2
3
4
5
6
7
8
9
month



6.1 Seasonality Index

Sources: leddris (2010), Walsh and Lawler (1981)

R = mean annual precipitation
m_i = precipitation mean for month i

$$SI = \frac{1}{R}\sum_{n=1}^{n=12}\left|m_i - \frac{R}{12}\right|$$

SI	Precipitation Regime
<0.19	Precipitation spread throughout the year
0.20-0.39	Precipitation spread throughout the year, but with a definite wetter season
0.40-0.59	Rather seasonal with a short dry season
0.60-0.79	Seasonal
0.80-0.99	Marked seasonal with a long dry season
1.00-1.19	Most precipitation in <3 months

Let's write some code to calculate the SI for Tel Aviv and London.

```
# import packages
import numpy as np
import pandas as pd
from calendar import month_abbr

# load data
month_numbers = np.arange(1,13)
month_names = [month_abbr[i] for i in month_numbers]
def monthly_mean(station_name, freq):
    # import daily data
    df = pd.read_csv(station_name + '_' + freq + '.csv', sep=",")
    # make 'DATE' the dataframe index
    df['DATE'] = pd.to_datetime(df['DATE'])
    df = df.set_index('DATE')
    # print(df.index[0], df.index[-1])
    if freq == 'daily':
        # resample data by month
        df_month = df['PRCP'].resample('M').sum()  # sum is labeled at the last day of the mo
        df_month = df_month/10                     # PRCP is given in tens of mm (see readme)
    if freq == 'monthly':
        df_month = df['PRCP']
    # calculate monthly mean
    monthly_mean = np.array([])  # empty array
    for m in month_numbers:      # cycle over months (1, 2, 3, etc)
        this_month_all_indices = (df_month.index.month == m)       # indices in df_month belo
        this_month_mean = df_month[this_month_all_indices].mean()  # this is the monthly mean
        monthly_mean = np.append(monthly_mean, this_month_mean)    # append
```

```python
    # make new df and return it
    df_return = pd.DataFrame({'monthly rainfall (mm)':monthly_mean,
                              'month names':month_names,
                              'month number':month_numbers
                            })
    return df_return

# load monthly mean
df_london = monthly_mean("LONDON_HEATHROW", 'monthly')
df_telaviv = monthly_mean("TEL_AVIV_READING", 'monthly')
```

```python
def walsh_index(df):
    m = df["monthly rainfall (mm)"]
    R = df["monthly rainfall (mm)"].sum()
    SI = np.sum(np.abs(m-R/12)) / R
    return SI

london_index = walsh_index(df_london)
telaviv_index = walsh_index(df_telaviv)
print("Seasonality index (Walsh and Lawler, 1981)")
print(f"London: {london_index:.2f}")
print(f"Tel Aviv: {telaviv_index:.2f}")
```

```
Seasonality index (Walsh and Lawler, 1981)
London: 0.13
Tel Aviv: 1.00
```

7 Exercises

Import relevant packages

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from calendar import month_abbr
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
import urllib.request
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

7.1 intra-annual variability

Go to NOAA's National Centers for Environmental Information (NCEI)
Climate Data Online: Dataset Discovery

Find station codes in this map. On the left, click on the little wrench ( ) next to "Global Summary of the Month", then click on "identify" on the panel that just opened, and click on a station (purple circle). You will see the station's name, it's ID, and the period of record. For example, for Ben-Gurion's Airport in Israel:
BEN GURION, IS
STATION ID: ISM00040180
Period of Record: 1951-01-01 to 2020-03-01

You can download **daily** or **monthly** data for each station. Use the function below to download this data to your computer.

```
def download_data(station_name, station_code):
    url_daily = 'https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/a
    url_monthly = 'https://www.ncei.noaa.gov/data/gsom/access/'
    # download daily data - uncomment the next 2 lines to make this work
    # urllib.request.urlretrieve(url_daily + station_code + '.csv',
    #                            station_name + '_daily.csv')
```

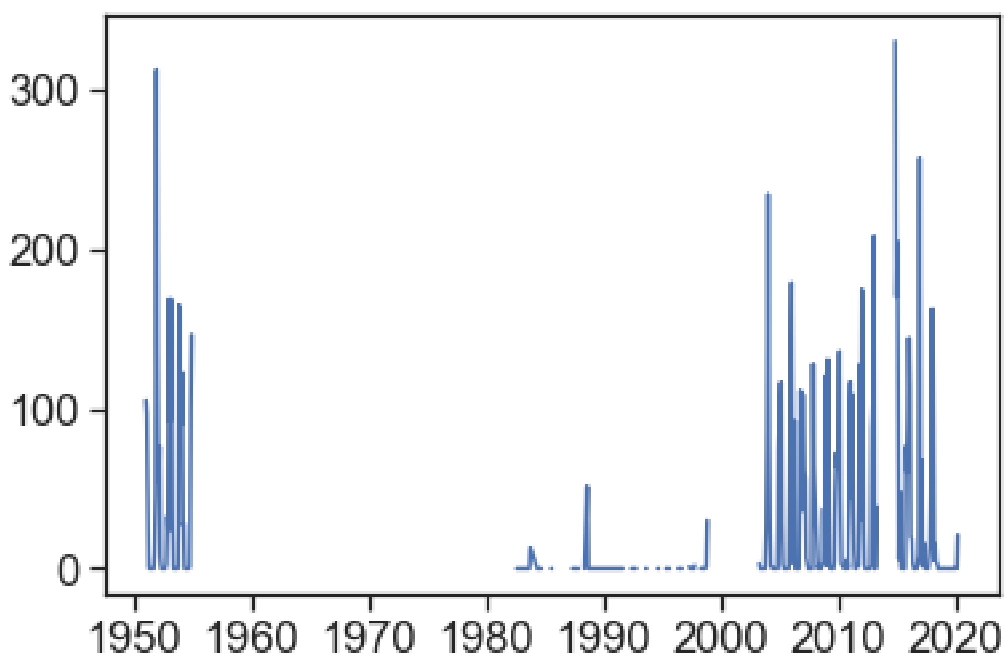
```
    # download monthly data
    urllib.request.urlretrieve(url_monthly + station_code + '.csv',
                               station_name + '_monthly.csv')
```

Now, choose any station with a period of record longer than 30 years, and download its data:

```
download_data('BEN_GURION', 'ISM00040180')
```

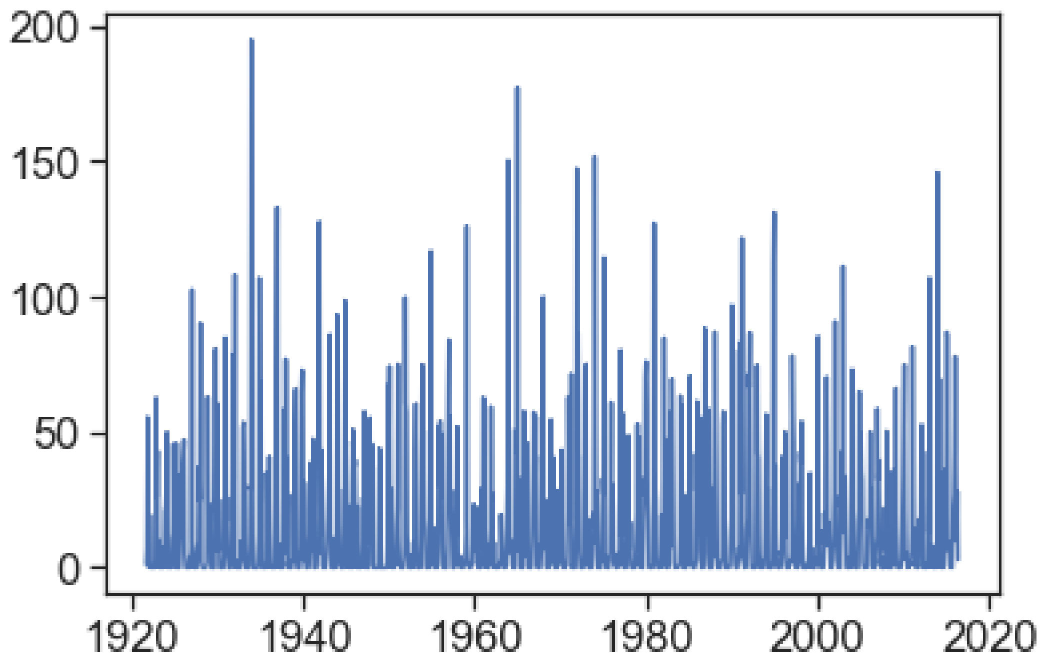
Load the data into a datafram, and before you continue with the analysis, plot the rainfall data, to see how it looks like.

```
download_data('BEN_GURION', 'ISM00040180')
df = pd.read_csv('BEN_GURION_monthly.csv', sep=",")
# make 'DATE' the dataframe index
df['DATE'] = pd.to_datetime(df['DATE'])
df = df.set_index('DATE')
plt.plot(df['PRCP'])
```



It doesn't look great for Ben-Gurion airport, lots of missing data! You might need to choose another station… Download data for Beer Sheva, ID `IS000051690`.

```
download_data('BEER_SHEVA', 'IS000051690')
df = pd.read_csv('BEER_SHEVA_monthly.csv', sep=",")
# make 'DATE' the dataframe index
df['DATE'] = pd.to_datetime(df['DATE'])
df = df.set_index('DATE')
plt.plot(df['PRCP'])
```



That's much better! We need to aggregate all data from each month, so we can calculate monthly averages. How to do that?

```
# choose only the precipitation column
df_month = df['PRCP']
# calculate monthly mean
monthly_mean = np.array([])  # empty array
month_numbers = np.arange(1,13)
month_names = [month_abbr[i] for i in month_numbers]

for m in month_numbers:      # cycle over months (1, 2, 3, etc)
    this_month_all_indices = (df_month.index.month == m)       # indices in df_month belongi
    this_month_mean = df_month[this_month_all_indices].mean()  # this is the monthly mean
    monthly_mean = np.append(monthly_mean, this_month_mean)    # append
```

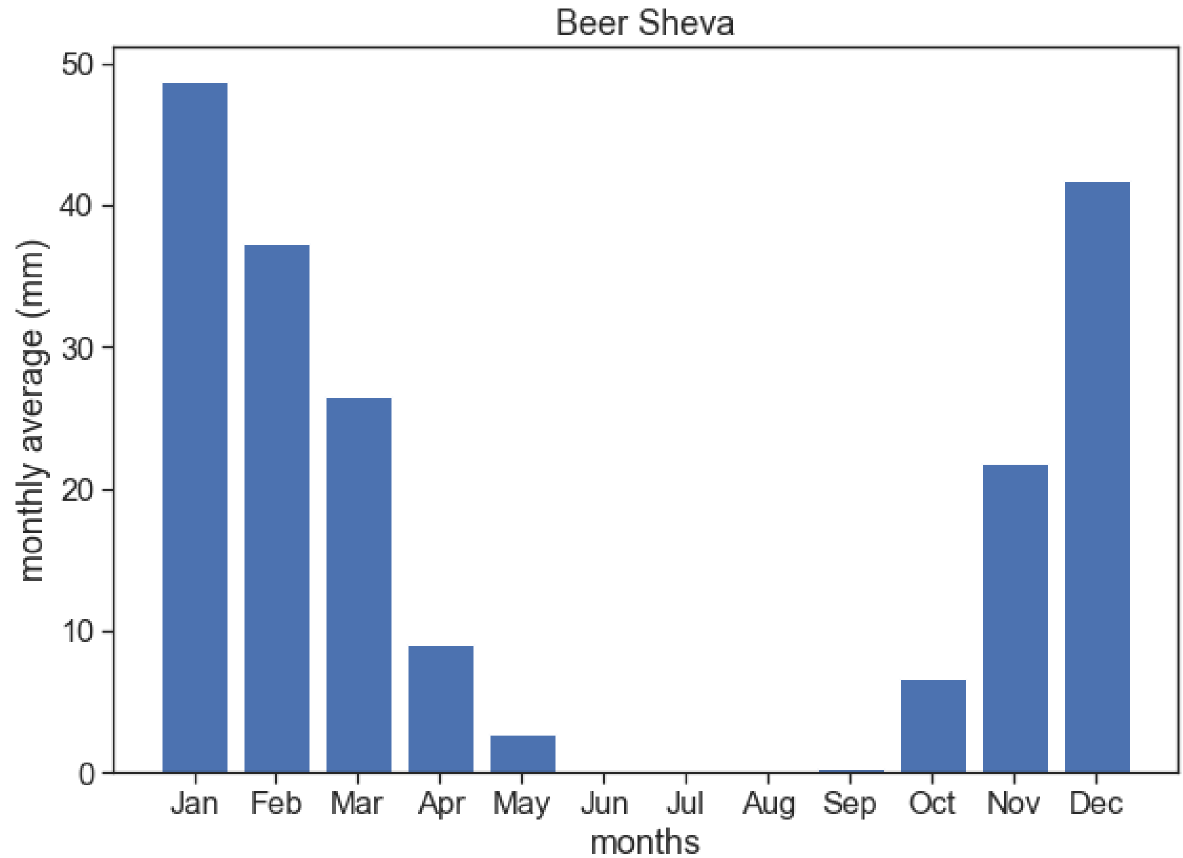
Now it is time to create a new dataframe with the monthly means.

```python
df_beersheva = pd.DataFrame({'monthly rainfall (mm)':monthly_mean,
                             'month names':month_names,
                             'month number':month_numbers
                            })
df_beersheva
```

	monthly rainfall (mm)	month names	month number
0	48.743158	Jan	1
1	37.347368	Feb	2
2	26.551579	Mar	3
3	9.038947	Apr	4
4	2.735789	May	5
5	0.013830	Jun	6
6	0.000000	Jul	7
7	0.002128	Aug	8
8	0.271277	Sep	9
9	6.669474	Oct	10
10	21.850526	Nov	11
11	41.786316	Dec	12

Plot the data and see if it makes sense. Try to get a figure like this one.

```python
fig, ax = plt.subplots(figsize=(10,7))
ax.bar(df_beersheva['month number'], df_beersheva['monthly rainfall (mm)'])
ax.set(xlabel="months",
       ylabel="monthly average (mm)",
       title="Beer Sheva",
       xticks=df_beersheva['month number'],
       xticklabels=df_beersheva['month names']);
plt.savefig("hydrology_figures/beersheva_monthly_average.png")
```



Let's calculate now the Walsh and Lawler Seasonality Index (leddris (2010), Walsh and Lawler (1981)) **Write a function** that receives a dataframe like the one we have just created, and returns the seasonality index.

R = mean annual precipitation
m_i precipitation mean for month i

$$SI = \frac{1}{R}\sum_{n=1}^{n=12}\left|m_i - \frac{R}{12}\right|$$

SI	Precipitation Regime
<0.19	Precipitation spread throughout the year
0.20-0.39	Precipitation spread throughout the year, but with a definite wetter season
0.40-0.59	Rather seasonal with a short dry season
0.60-0.79	Seasonal

SI	Precipitation Regime
0.80-0.99	Marked seasonal with a long dry season
1.00-1.19	Most precipitation in $<$ 3 months

```
def walsh_index(df):
    mi = df["monthly rainfall (mm)"]
    R = df["monthly rainfall (mm)"].sum()
    SI = np.sum(np.abs(mi - R/12)) / R
    return SI
beersheva_SI = walsh_index(df_beersheva)
print(f"Beer Sheva, SI = {beersheva_SI:.2f}")
```

```
Beer Sheva, SI = 0.97
```

8 Return Period

8.1 Bilbao, Spain

8.2 Today

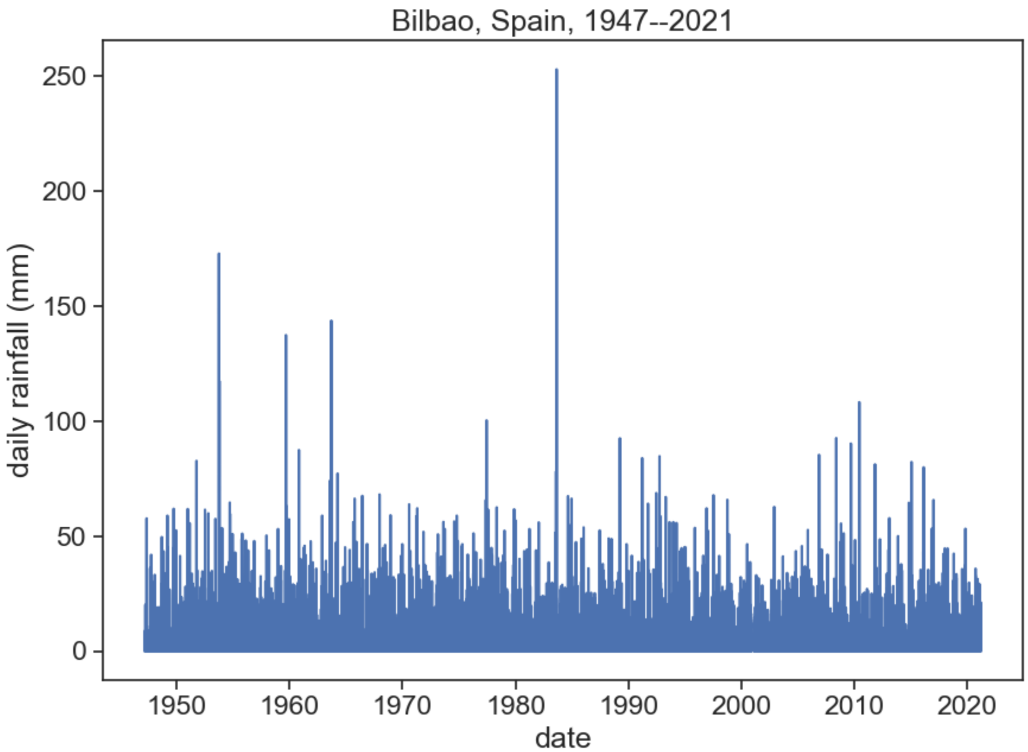
8.3 August 1983

On Friday, August 26, 1983, Bilbao was celebrating its Aste Nagusia or Great Week, the main annual festivity in the city, when it and other municipalities of the Basque Country, Burgos, and Cantabria suffered devastating flooding due to heavy rains. In 24 hours, the volume of water registered 600 liters per square meter. Across all the affected areas, the weather service recorded 1.5 billion tons of water. In areas of Bilbao, the water reached a height of 5 meters (15 feet). Transportation, electricity and gas services, drinking water, food, telephone, and many other basic services were severely affected. 32 people died in Biscay, 4 people died in Cantabria, 2 people died in Alava, and 2 people died Burgos. 5 more people went missing.

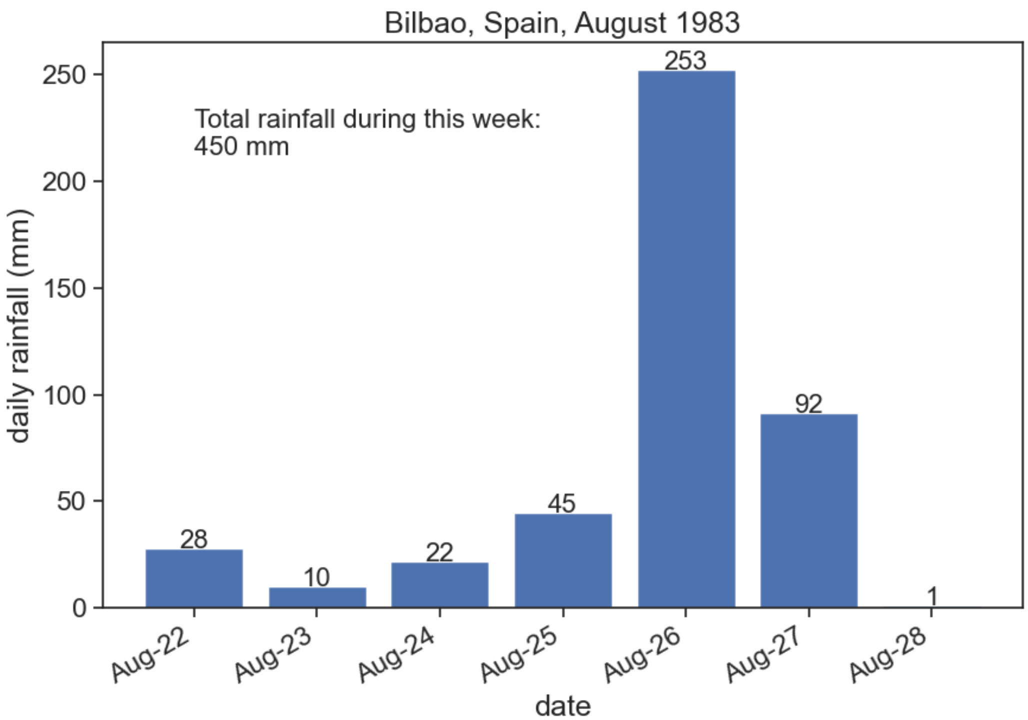
8.4 How often will such rainfall happen?

How often does it rain 50 mm in 1 day? What about 100 mm in 1 day? How big is a "once-in-a-century event"?

Let's examine Bilbao's daily rainfall (mm), between 1947 to 2021



On the week of 22-28 August 1983, Bilbao's weather station measured 45 cm of rainfall!



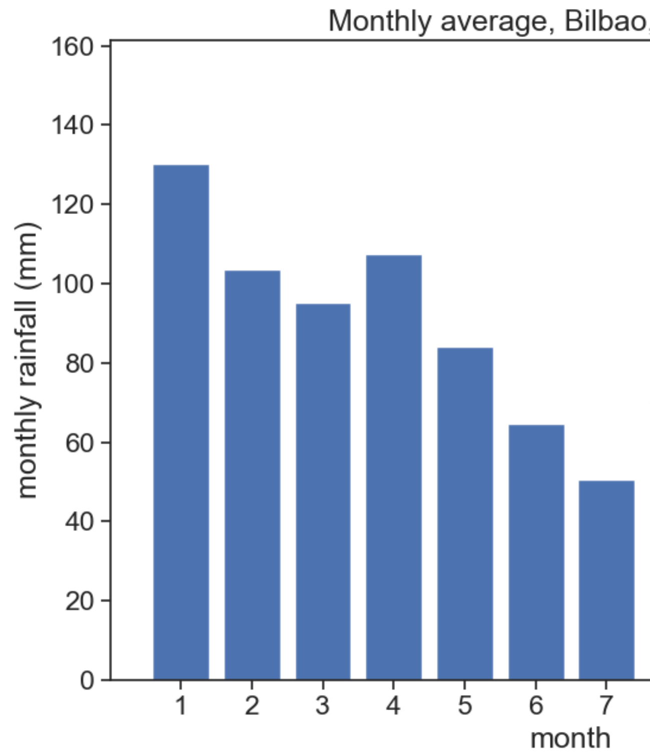
Let's analyze this data and find out how rare such events are. First we need to find the annual maximum for each hydrological year.

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

df = pd.read_csv('BILBAO_daily.csv', sep=",")
df['DATE'] = pd.to_datetime(df['DATE'])
df = df.set_index('DATE')
# IMPORTANT!! daily precipitation data is in tenths of mm, divide by 10 to get it in mm.
df['PRCP'] = df['PRCP'] / 10

import altair as alt
alt.data_transformers.disable_max_rows()

# Altair only recognizes column data; it ignores index values.
# You can plot the index data by first resetting the index
# I know that I've just made 'DATE' the index, but I want to have this here nonetheless so I
```

```python
df_new = df.reset_index()#.replace({0.0:np.nan})
source = df_new[['DATE', 'PRCP']]

brush = alt.selection(type='interval', encodings=['x'])

base = alt.Chart(source).mark_line().encode(
    x = 'DATE:T',
    y = 'PRCP:Q'
).properties(
    width=600,
    height=200
)

upper = base.encode(
    alt.X('DATE:T', scale=alt.Scale(domain=brush)),
    alt.Y('PRCP:Q', scale=alt.Scale(domain=(0,100)))
)

lower = base.properties(
    height=60
).add_selection(brush)

alt.vconcat(upper, lower)
```

```
/home/yairmau/miniforge3/lib/python3.10/site-packages/altair/utils/deprecation.py:65: Altair

'selection' is deprecated.
   Use 'selection_point()' or 'selection_interval()' instead; these functions also include mo

/home/yairmau/miniforge3/lib/python3.10/site-packages/altair/utils/deprecation.py:65: Altair

'add_selection' is deprecated. Use 'add_params' instead.
```

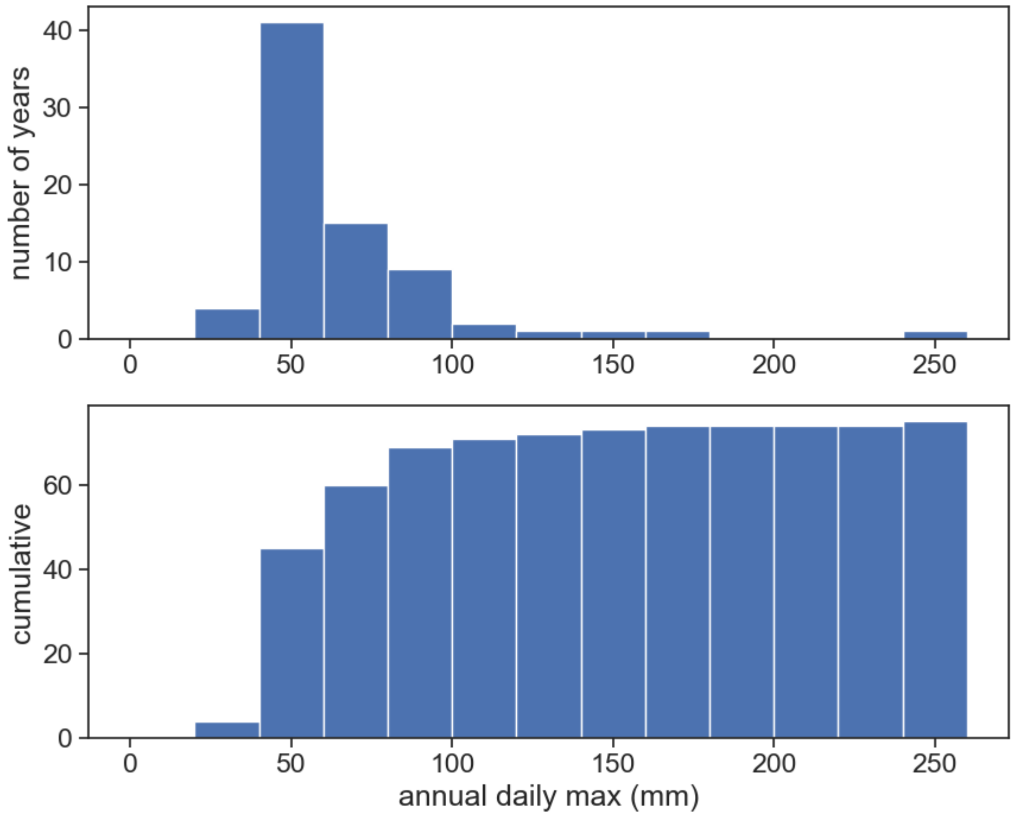
```
alt.VConcatChart(...)
```



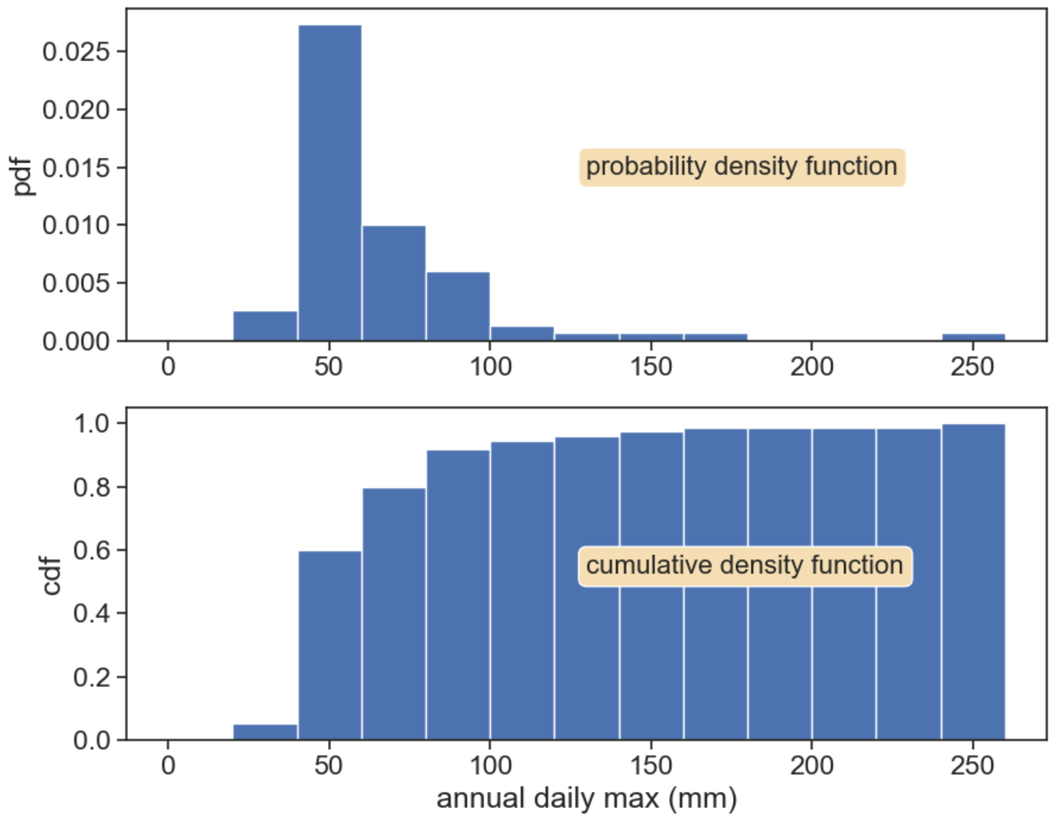
We will consider a hydrological year starting on 1 August.

- Top: Histogram of annual daily precipitation maximum events.

- Bottom: The cumulative answers the question: “How many events can be found **below** a given threshold?”



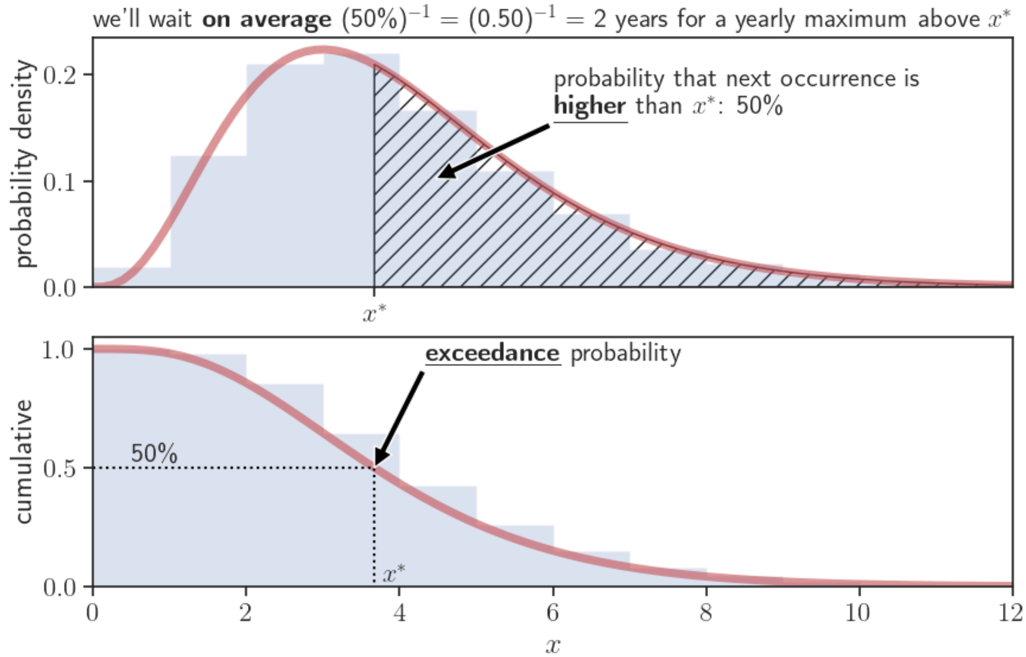
- Top: We can normalize the histogram such that the total area is 1. Now the histogram is called `probability density function (pdf)`. The probability is NOT the pdf, but the area between two thresholds.
- Bottom: The cumulative now becomes a probability between 0 and 1. It is now called `cumulative density function (cdf)`. The cdf answers the question: "What is the probability to choose an event **below** a given threshold?"

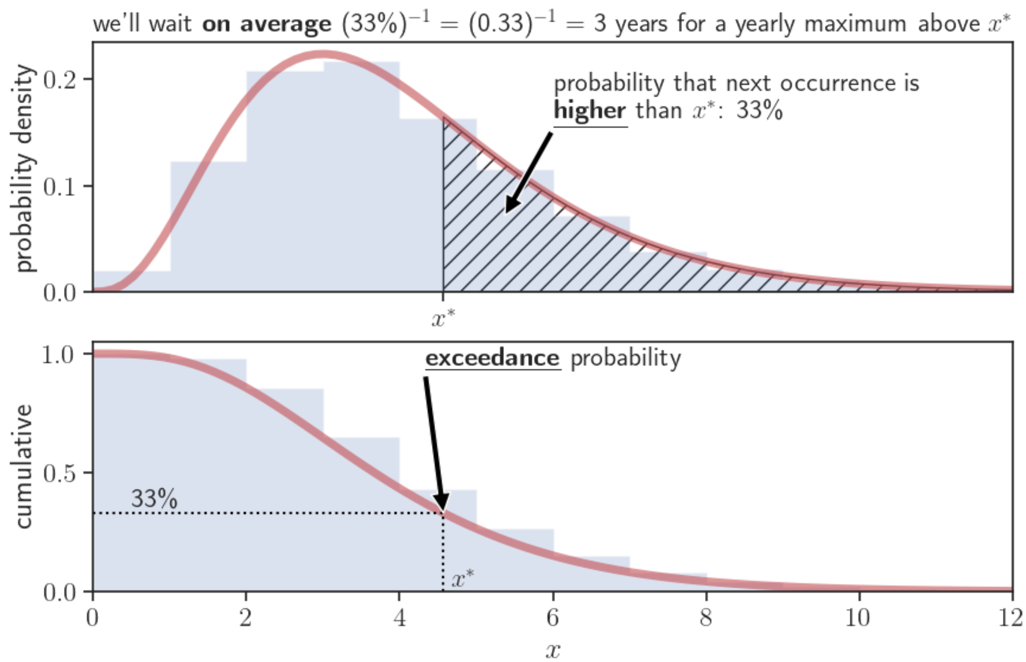


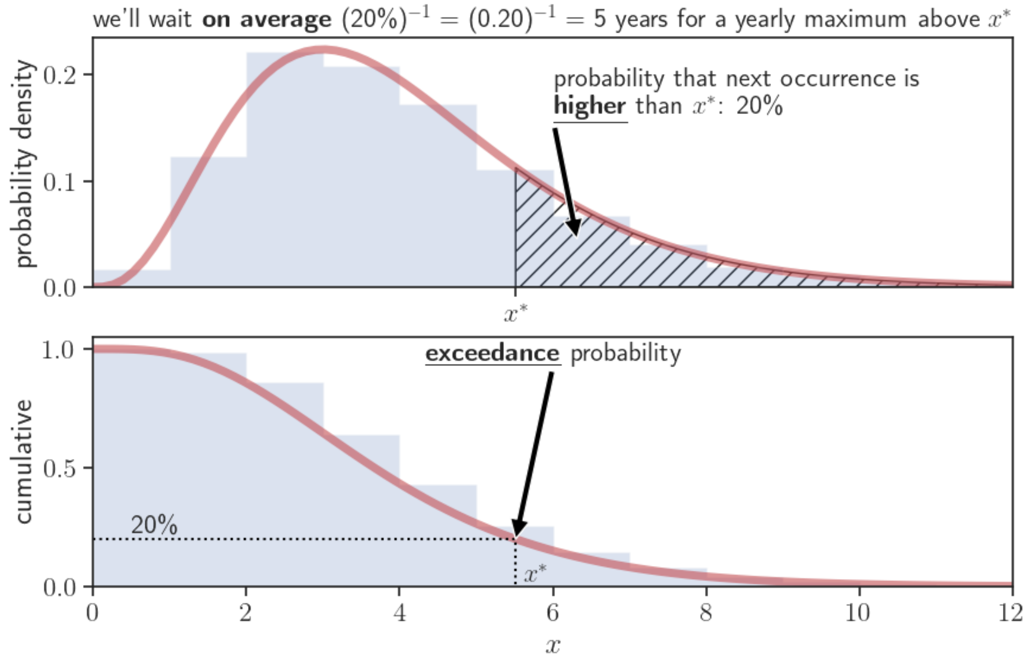
We are interested in extreme events, and we want to estimate how many years, **on average**, do we have to wait to get an annual maximum **above** a given threshold?

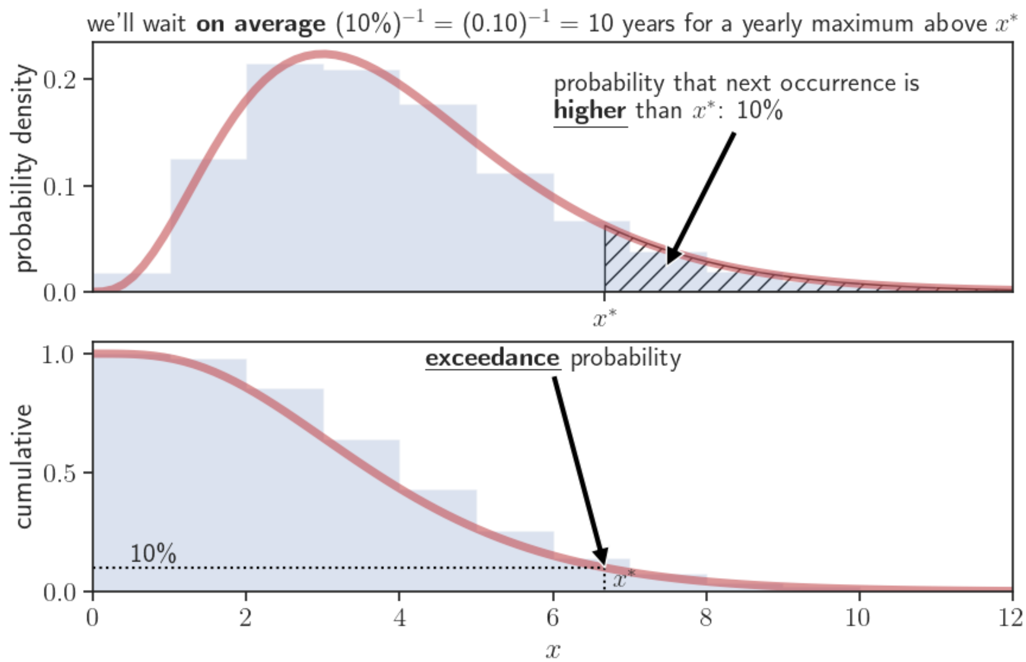
This question is very similar to what we asked regarding the cdf.
We switched the word "below" for "above". The complementary of the cumulative is called the exceedance (or survival) probability:

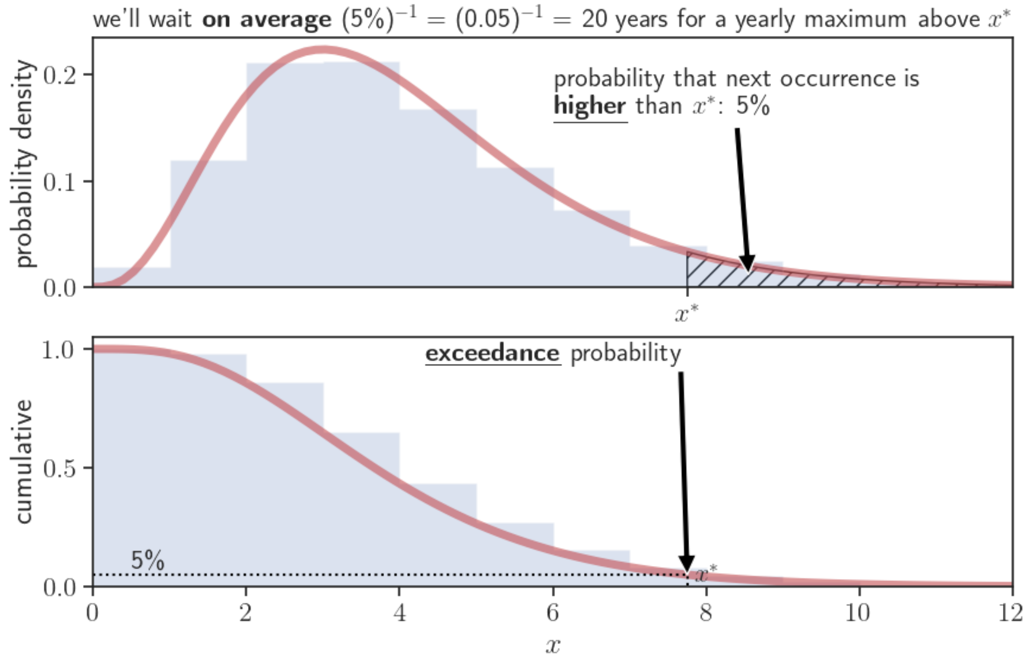
$$\text{exceedance, survival} = 1 - \text{cdf}$$


we'll wait **on average** $(50\%)^{-1} = (0.50)^{-1} = 2$ years for a yearly maximum above x^*
probability density
probability that next occurrence is **higher** than x^*: 50%
x^*
cumulative
exceedance probability
50%
x^*
x


we'll wait **on average** $(33\%)^{-1} = (0.33)^{-1} = 3$ years for a yearly maximum above x^*
probability density
probability that next occurrence is **higher** than x^*: 33%
x^*
cumulative
exceedance probability
33%
x^*
x


we'll wait **on average** $(20\%)^{-1} = (0.20)^{-1} = 5$ years for a yearly maximum above x^*
probability density
probability that next occurrence is
higher than x^*: 20%
x^*
cumulative
exceedance probability
20%
x^*
x

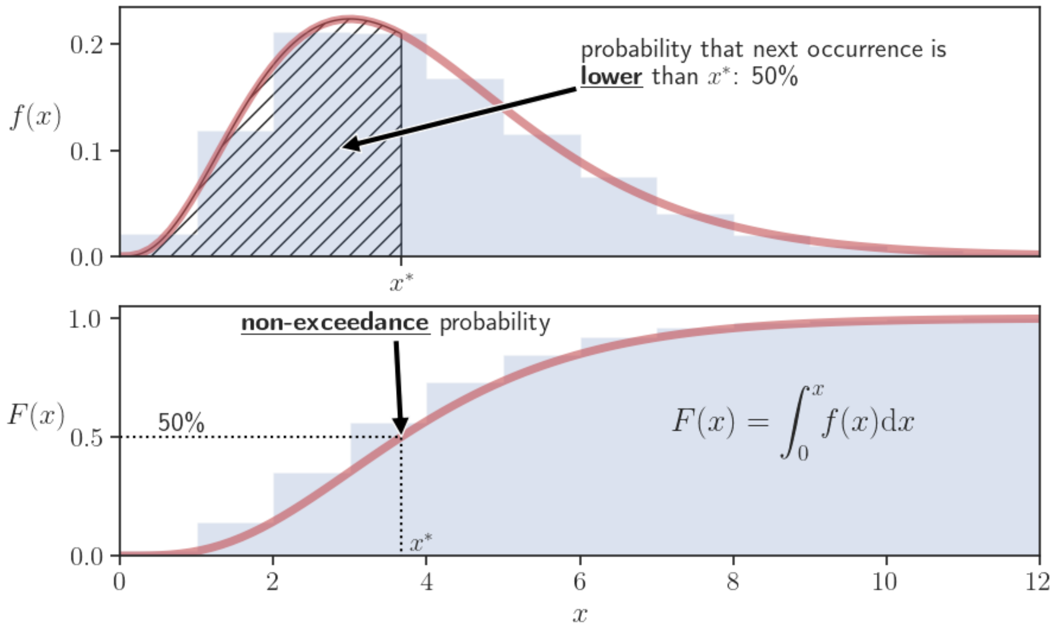

we'll wait **on average** $(10\%)^{-1} = (0.10)^{-1} = 10$ years for a yearly maximum above x^*
probability density
probability that next occurrence is
higher than x^*: 10%
x^*
cumulative
exceedance probability
10%
x^*
x



8.5 Return Period

We will follow Brutsaert (2005), page 513. He defines quantities a little different from what we did above.

$F(x)$ is the CDF of the PDF $f(x)$. $F(x)$ indicates the non-exceedance probability, i.e., the probability that a certain event above x has **not** occurred (or that an event below x has occurred, same thing). Modifying the graph shown above, we have



$1 - F(x)$ is the probability that a certain event above x *has* occurred. It's reciprocal is the return period:

$$T_r(x) = \frac{1}{1 - F(x)}$$

This return period is the expected number of observations required until x is exceeded once. In our case, we can ask the question: how many years will pass (on average) until we see a rainfall event greater that that of 26 August 1983?

Let's call $p = F(x)$ the probability that we measured once and that an event greater than x has *not* occurred. What is the probability that a rainfall above x will occur only on year number k?

- it hasn't occurred on year 1 (probability p)
- it hasn't occurred on year 2 (probability p)
- it hasn't occurred on year 3 (probability p)
- …
- it has occurred on year k (probability 1-p)

$P\{k \text{ trials until } X > x\} = p^{k-1}(1 - p)$

Every time the number k will be different. What will be k *on average*?

$$\bar{k} = \sum_{k=1}^{\infty} kP(k) = \sum_{k=1}^{\infty} kp^{k-1}(1-p)$$

Let's open that up:

$$\begin{aligned}
\bar{k} &= 1 - p + 2p(1-p) + 3p^2(1-p) + 4p^3(1-p) + \cdots \\
\bar{k} &= 1 - p + 2p - 2p^2 + 3p^2 - 3p^4 + 4p^3 - 4p^4 + \cdots \\
\bar{k} &= 1 + p + p^2 + p^3 + p^4 + \cdots
\end{aligned}$$

For $p < 1$, the series converges to

$$1 + p + p^2 + p^3 + p^4 + \cdots = \frac{1}{1-p},$$

therefore

$$\bar{k} = \frac{1}{1-p}.$$

We conclude that if we know the exceedance probability, we immediately can say what the return times are. We now need a way of estimating this exceedance probability.

8.6 Generalized extreme value distribution

This part of the lecture was heavily inspired by Alexandre Martinez's excellent blog post.

The Generalized Extreme Value (GEV) distribution is the limit distribution of properly normalized maxima of a sequence of independent and identically distributed random variables (from Wikipedia).

It has three parameters: shape, location and scale. We can fit Bilbao's annual daily maxima with a GEV distribution, yielding:

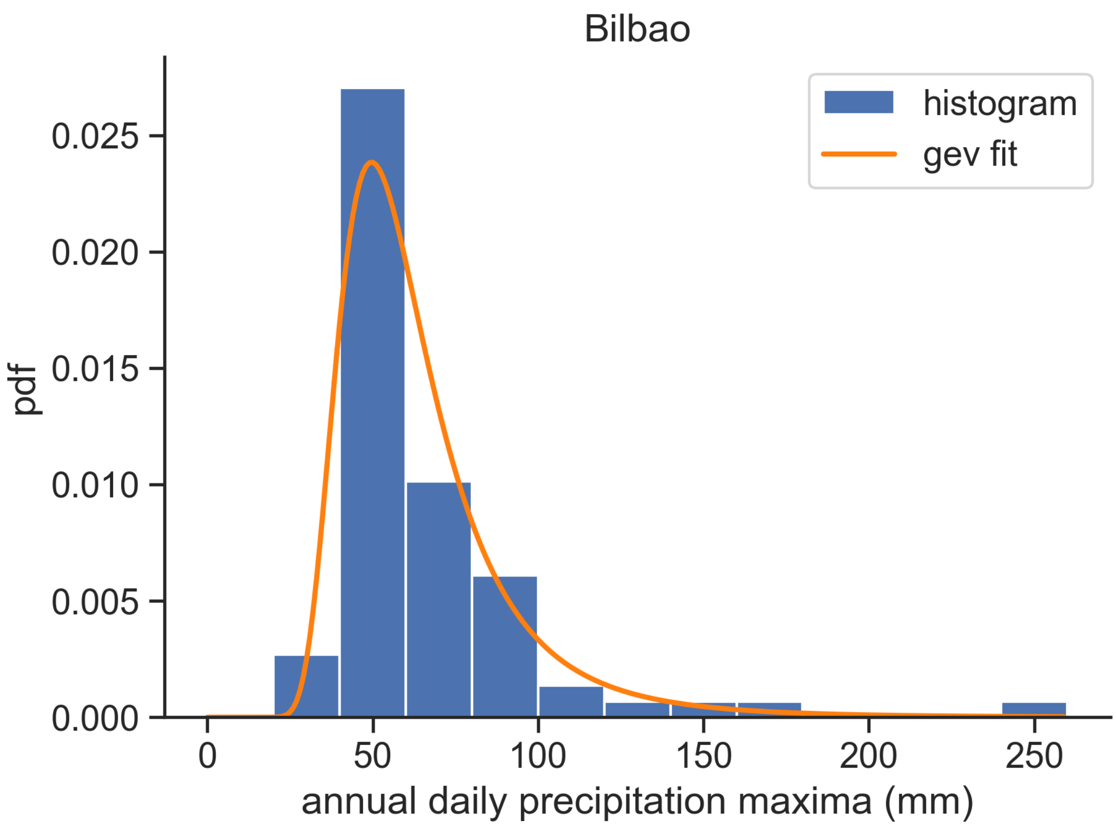
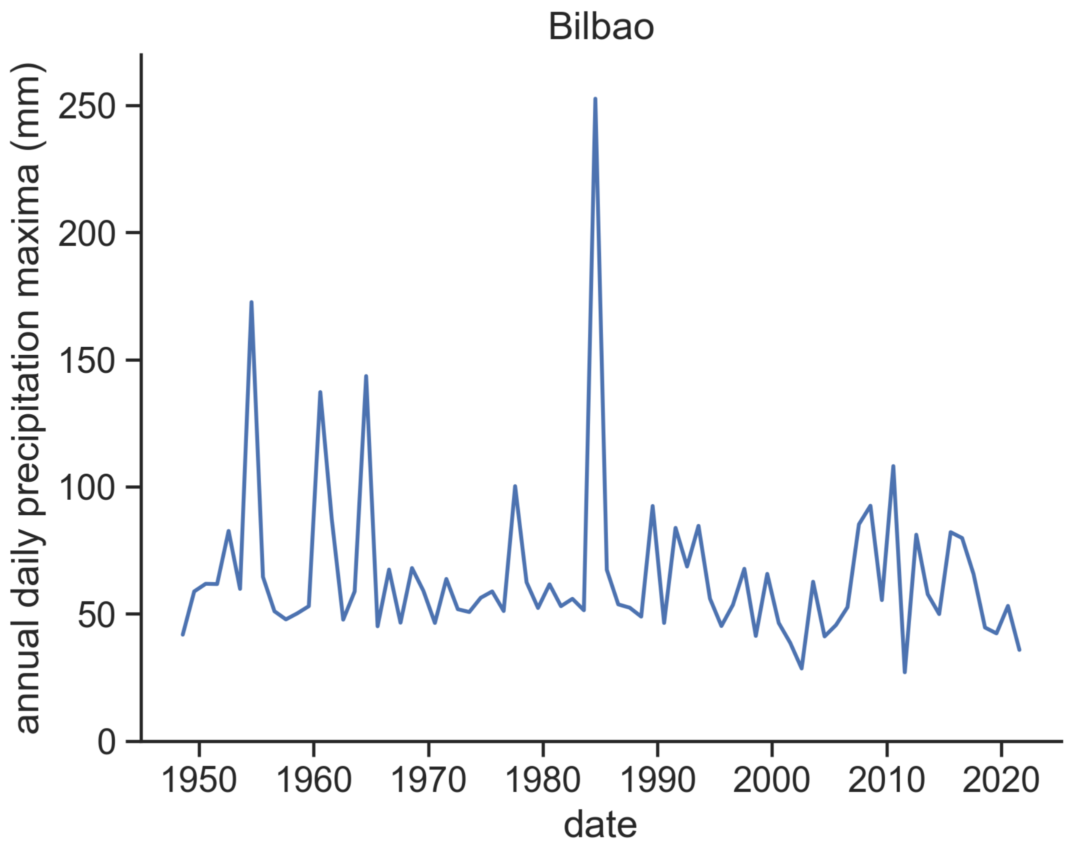


Figure 8.1: (shape=-0.18, location=52, shape=16)

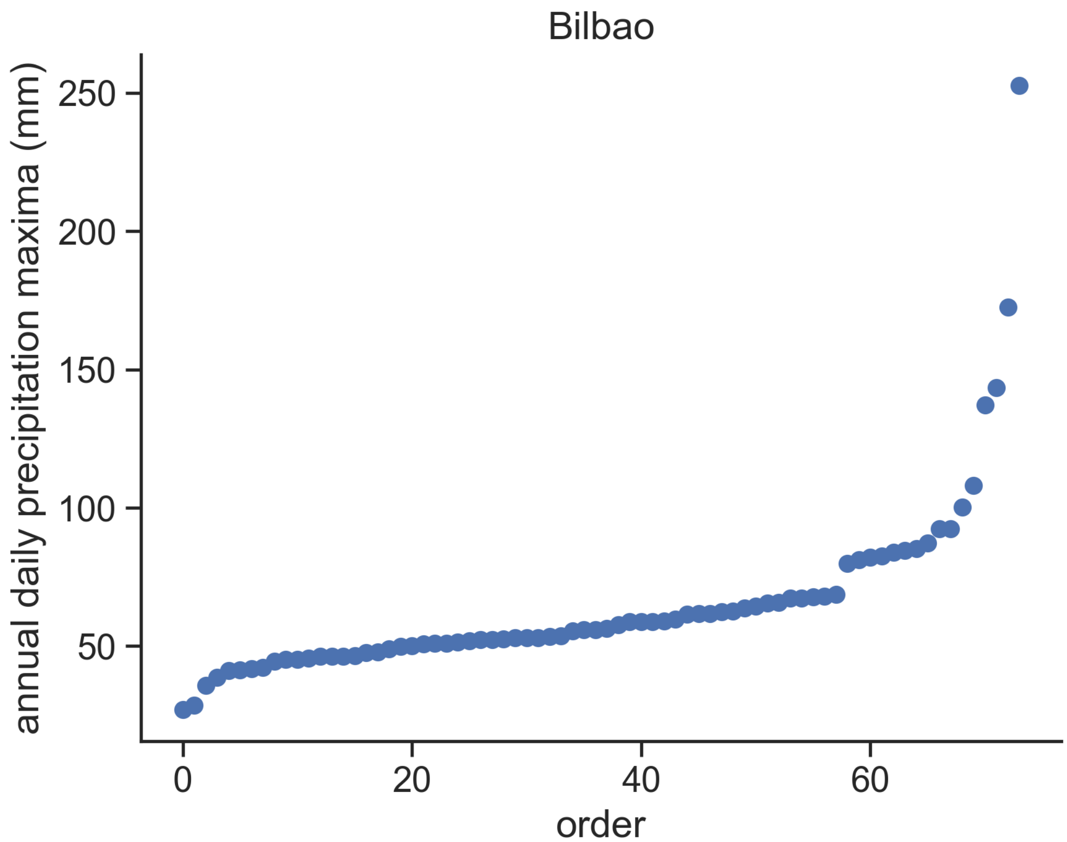
We will use the fitted parameters to plot the cdf, and then we will find the return periods. How can we calculate the cdf numerically form the data?

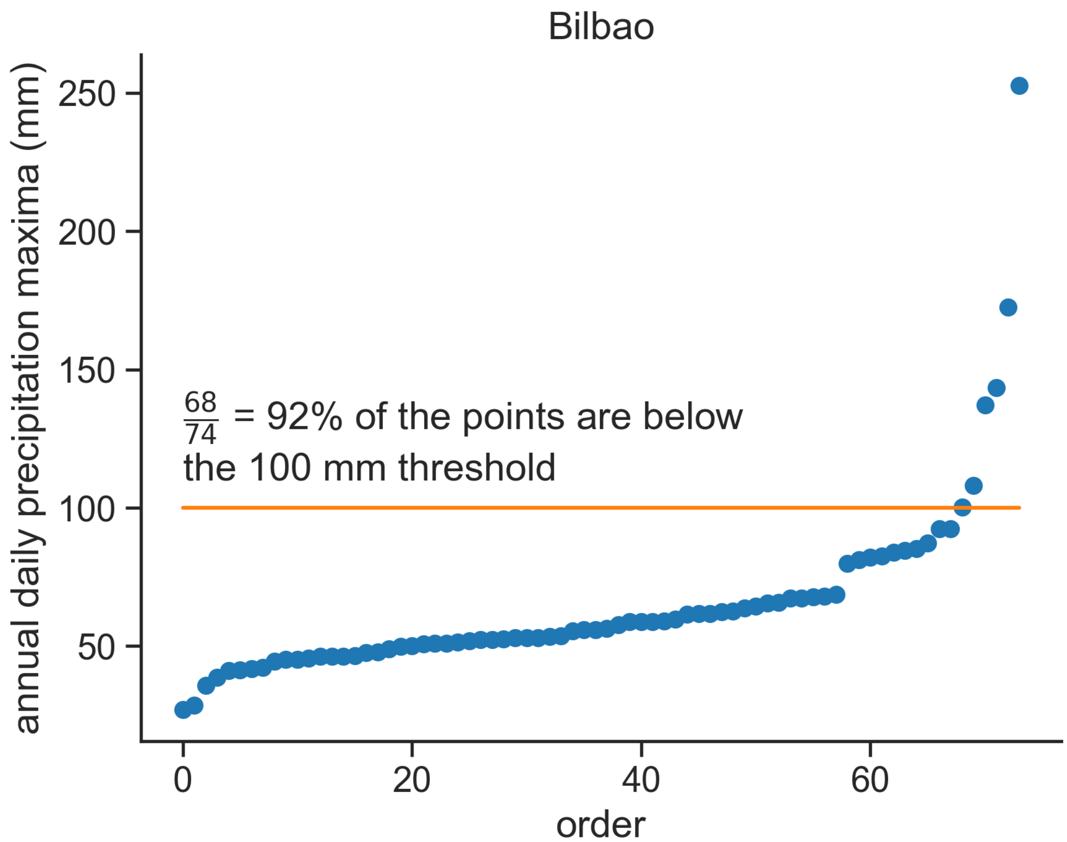
8.6.1 cdf from data

Here is a plot of annual daily precipitation maxima for Bilbao.

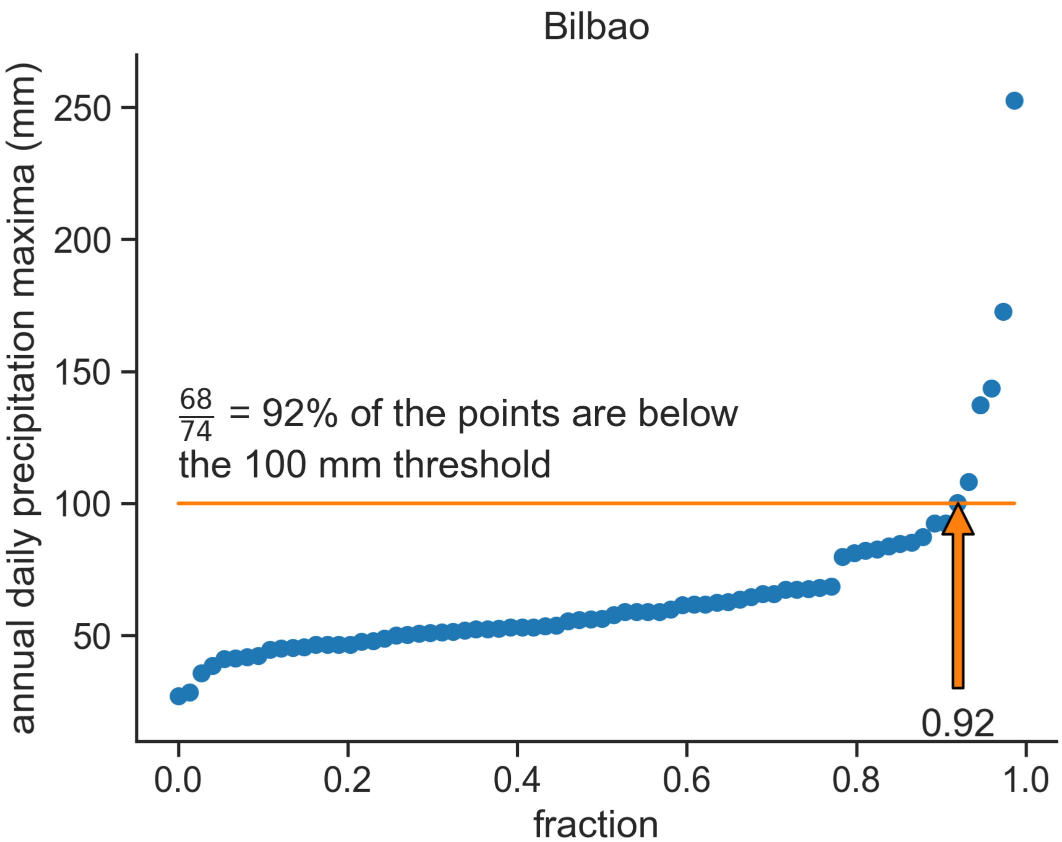


There are 74 points here. Let's order them from small to large:

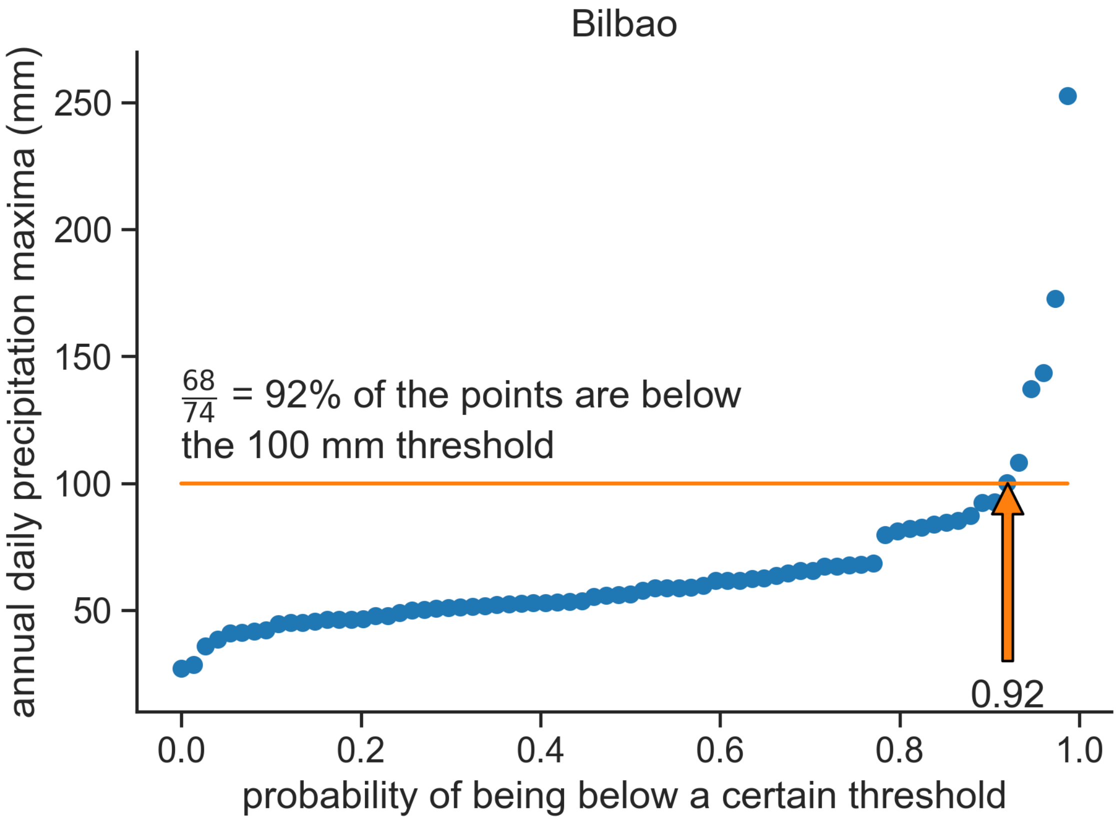

Bilbao
annual daily precipitation maxima (mm)
250
200
150
100
50
0
20
40
60
order



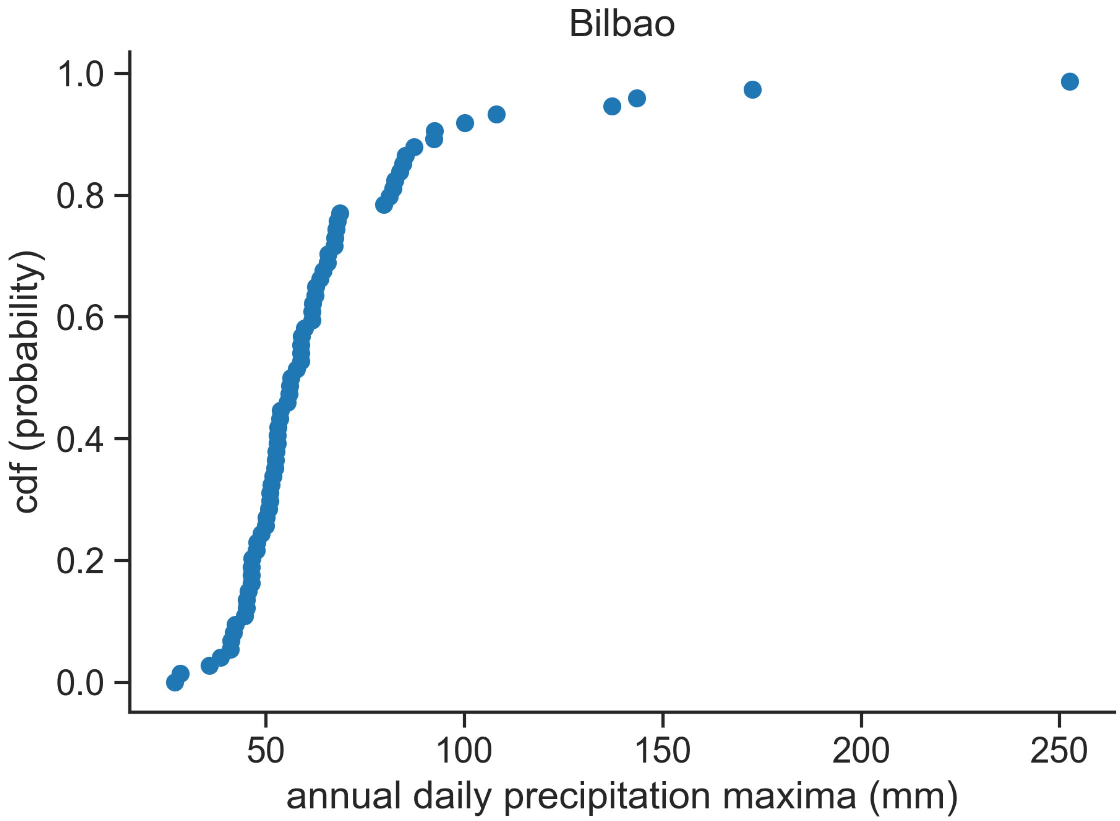
Now, instead of having the order of the event on the horizontal axis, let's make it a fraction from 0 to 1.



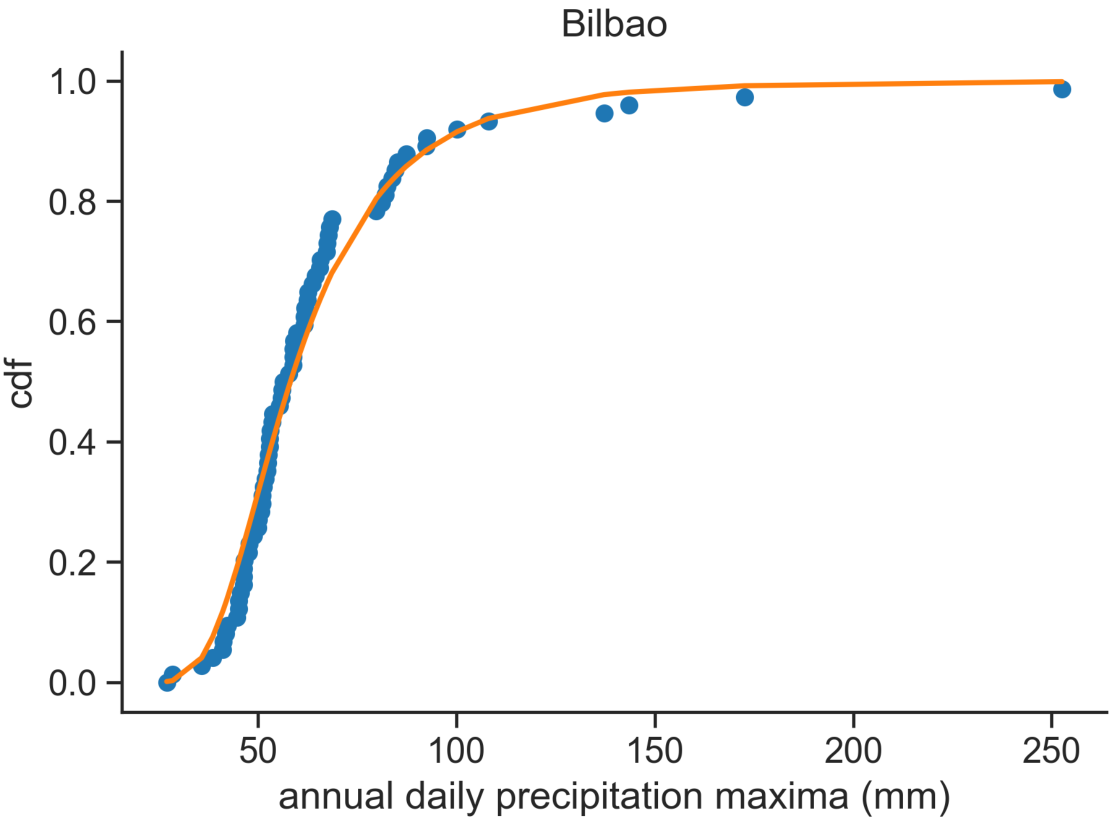
We are getting there! We can interpret this fraction as the probability of randomly choosing a point in the graph below the threshold.



Now we just need to flip the vertical and horizontal axes, and we're done! We have our cdf!



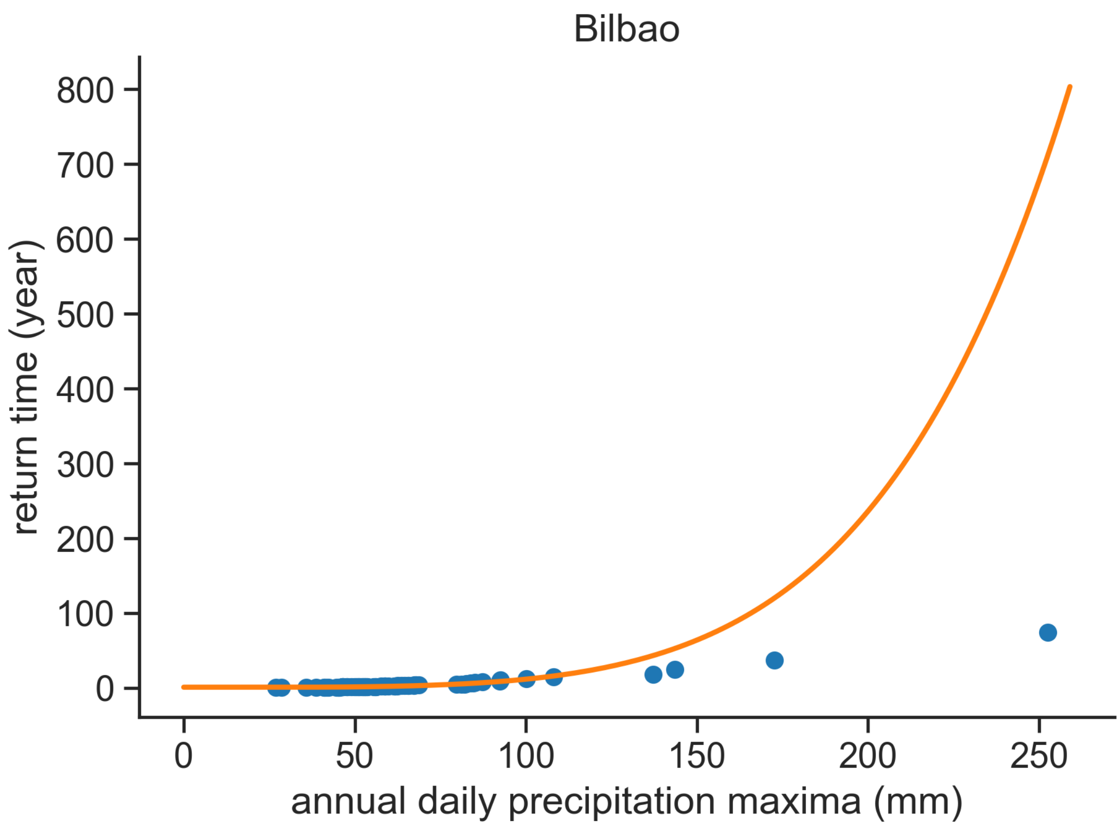
Now that we have the cdf from data, we can plot on top of it the GEV cdf with the parameters we found before.



The highest data point in this graph goes only to 252 mm, corresponding to the highest event recorded in 74 years. We can use the GEV cdf to calculate return times for any desired levels, simply by converting the vertical axis (cdf) to return period, using the equation we found earlier.

$$T_r(x) = \frac{1}{1 - F(x)},$$

where T_r is the return period (in years), and F is the cdf.



The information contained in the last two graphs is exactly the same, but somehow this last graph looks much worse! Why is this so? Acording to the GEV distribution we got, the return time for a 252 mm event is about 700 years!

8.7 Extra: K-S test

to be completed...

8.8 Extra: confidence interval

to be completed...

	PRCP	m	Pm	Tr
DATE				
2011-07-31	27.0	1	0.013158	1.013333
2002-07-31	28.5	2	0.026316	1.027027
2021-07-31	35.8	3	0.039474	1.041096
2001-07-31	38.6	4	0.052632	1.055556
2004-07-31	41.1	5	0.065789	1.070423
...	...	...	...	...
2010-07-31	108.1	71	0.934211	15.200000
1960-07-31	137.2	72	0.947368	19.000000
1964-07-31	143.5	73	0.960526	25.333333
1954-07-31	172.6	74	0.973684	38.000000
1984-07-31	252.6	75	0.986842	76.000000

9 Exercises

Import relevant packages

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# from functools import reduce
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
import urllib.request
from scipy.stats import genextreme
from scipy.optimize import curve_fit
```

Go to NOAA's National Centers for Environmental Information (NCEI)
Climate Data Online: Dataset Discovery

Find station codes in this map. On the left, click on the little wrench next to "Global Summary of the Month", then click on "identify" on the panel that just opened, and click on a station (purple circle). You will see the station's name, it's ID, and the period of record. For example, for Ben-Gurion's Airport in Israel:
BEN GURION, IS
STATION ID: ISM00040180
Period of Record: 1951-01-01 to 2020-03-01

You can download **daily** or **monthly** data for each station. Use the function below to download this data to your computer. `station_name` can be whatever you want, `station_code` is the station ID.

```
def download_data(station_name, station_code):
    url_daily = 'https://www.ncei.noaa.gov/data/global-historical-climatology-network-daily/a
    url_monthly = 'https://www.ncei.noaa.gov/data/gsom/access/'
    # download daily data - uncomment the following 2 lines to make this work
    urllib.request.urlretrieve(url_daily + station_code + '.csv',
                               station_name + '_daily.csv')
```

```
    # download monthly data
    urllib.request.urlretrieve(url_monthly + station_code + '.csv',
                               station_name + '_monthly.csv')
```

Download daily rainfall data for Eilat, Israel. ID: IS000009972

```
download_data('Eilat', 'IS000009972')
```

Then load the data into a dataframe.
IMPORTANT!! daily precipitation data is in tenths of mm, divide by 10 to get it in mm.

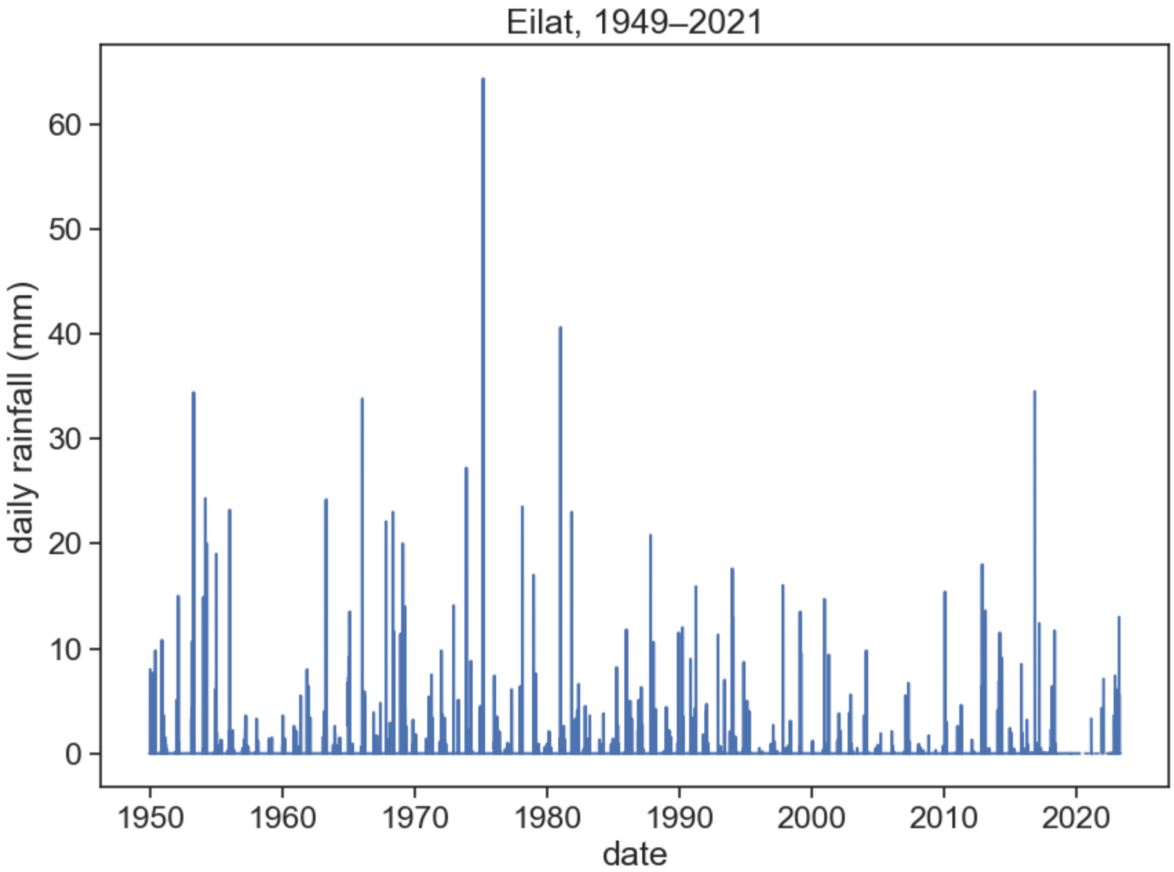
```
df = pd.read_csv('Eilat_daily.csv', sep=",")
# make 'DATE' the dataframe index
df['DATE'] = pd.to_datetime(df['DATE'])
df = df.set_index('DATE')
# IMPORTANT!! daily precipitation data is in tenths of mm, divide by 10 to get it in mm.
df['PRCP'] = df['PRCP'] / 10
df
```

	STATION	LATITUDE	LONGITUDE	ELEVATION	NAME	PRCP	PRCP_ATTR
DATE							
1949-11-30	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1949-12-01	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1949-12-02	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1949-12-03	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1949-12-04	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
...	...	...	...	...	...	...	...
2023-04-22	IS000009972	29.55	34.95	12.0	ELAT, IS	NaN	NaN
2023-04-23	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„S
2023-04-24	IS000009972	29.55	34.95	12.0	ELAT, IS	NaN	NaN
2023-04-25	IS000009972	29.55	34.95	12.0	ELAT, IS	NaN	NaN
2023-04-26	IS000009972	29.55	34.95	12.0	ELAT, IS	NaN	NaN

Plot precipitation data ('PRCP' column) and see if everything is all right.

```
fig, ax = plt.subplots(figsize=(10,7))
ax.plot(df['PRCP'])
ax.set_xlabel("date")
ax.set_ylabel("daily rainfall (mm)")
ax.set_title("Eilat, 1949-2021")
```

```
Text(0.5, 1.0, 'Eilat, 1949-2021')
```



Based on what you see, you might want to exclude certain periods, e.g.:

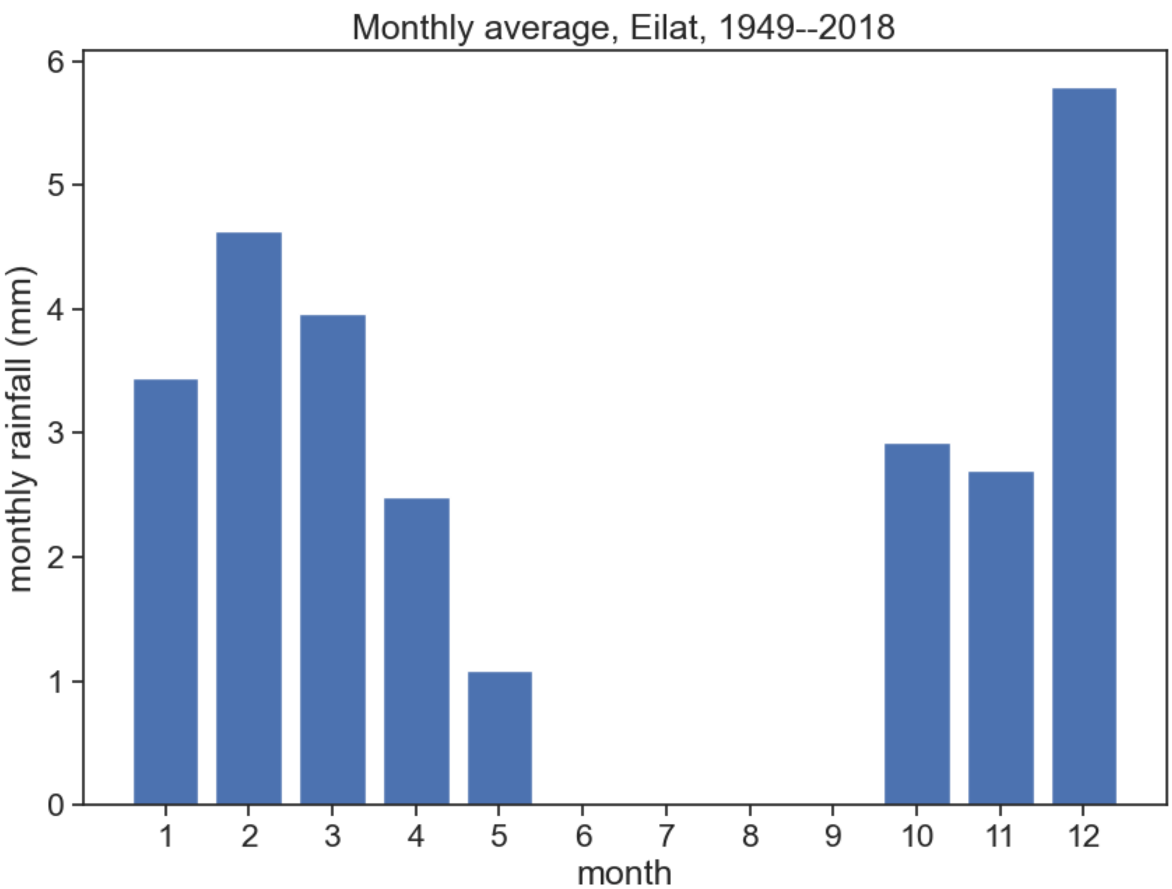
```
last_date = '2018-08-01'
first_date = '1950-08-01'
df = df[((df.index < last_date) & (df.index > first_date))]
df
```

	STATION	LATITUDE	LONGITUDE	ELEVATION	NAME	PRCP	PRCP_ATTR
DATE							
1950-08-02	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1950-08-03	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1950-08-04	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1950-08-05	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E
1950-08-06	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„E

DATE	STATION	LATITUDE	LONGITUDE	ELEVATION	NAME	PRCP	PRCP_ATTR
...	...	...	...	...	...	...	...
2018-07-27	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„S
2018-07-28	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„S
2018-07-29	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„S
2018-07-30	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„S
2018-07-31	IS000009972	29.55	34.95	12.0	ELAT, IS	0.0	„S

The rainfall data for Eilat is VERY seasonal, it's easy to see that there is no rainfall at all during the summer. We can assume a hydrological year starting on 1 August. If you're not sure, you can plot the monthly means (see last week's lecture) and find what date makes sense best.

```
df_month = df['PRCP'].resample('M').sum().to_frame()
month_numbers = np.arange(1,13)
monthly_mean = np.array([])  # empty array
for m in month_numbers:      # cycle over months (1, 2, 3, etc)
    this_month_mean = df_month[df_month.index.month == m].mean()  # this is the monthly mean
    monthly_mean = np.append(monthly_mean, this_month_mean)    # append
    # make new df and return it
df_month = pd.DataFrame({'monthly rainfall (mm)':monthly_mean,
                          'month number':month_numbers
                         })
fig, ax = plt.subplots(figsize=(10,7))
ax.bar(df_month['month number'], df_month['monthly rainfall (mm)'])
ax.set(xlabel="month",
       ylabel="monthly rainfall (mm)",
       title="Monthly average, Eilat, 1949--2018",
       xticks=np.arange(1,13));
```



Let's resample the data according to the hydrological year (1 August), and we'll keep the maximum value:

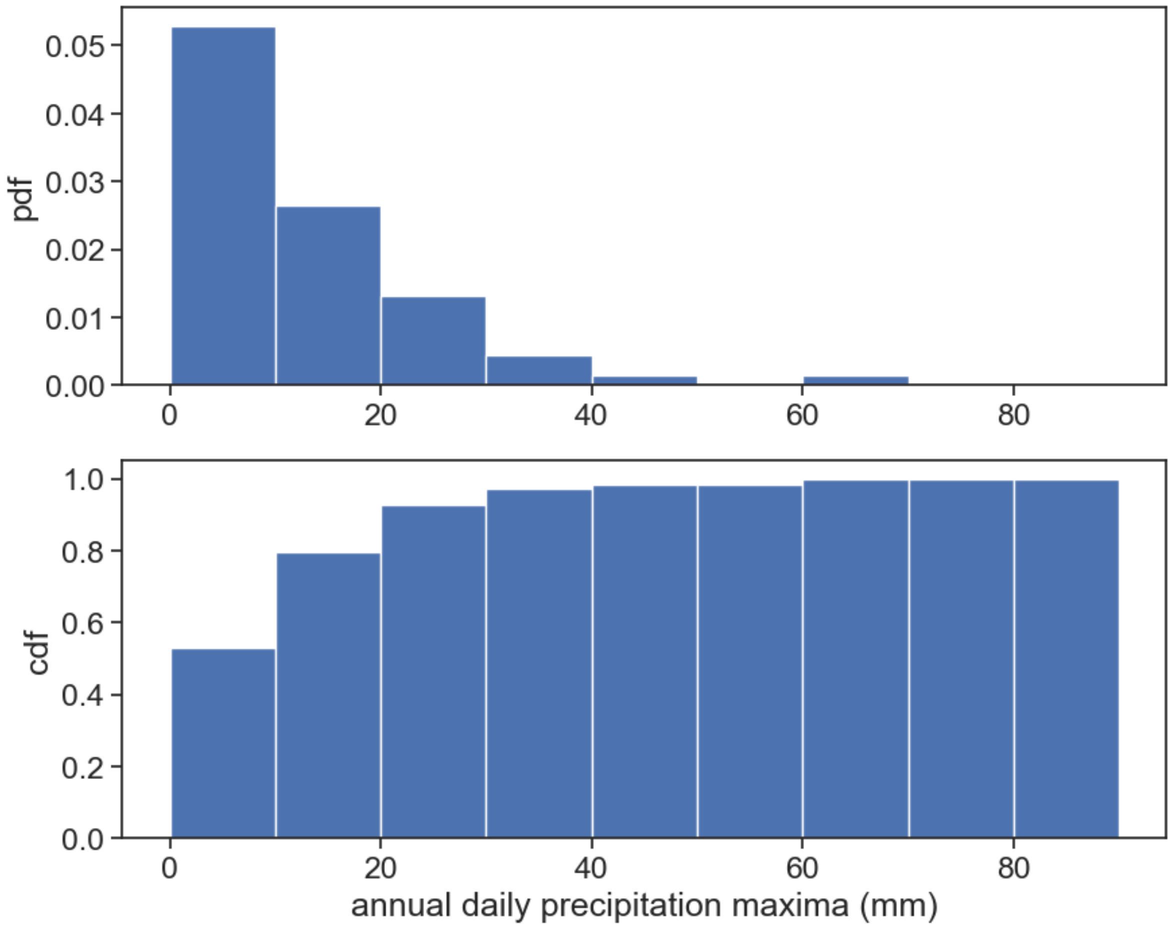
```
max_annual = (df['PRCP'].resample('A-JUL')
                        .max()
                        .to_frame()
             )
max_annual
```

DATE	PRCP
1951-07-31	10.8
1952-07-31	15.0
1953-07-31	34.4
1954-07-31	24.3

	PRCP
DATE	
1955-07-31	19.0
...	...
2014-07-31	11.5
2015-07-31	2.4
2016-07-31	8.5
2017-07-31	34.5
2018-07-31	11.7

Make two graphs: a) the histogram for the annual maximum (pdf) b) the cumulative probability (cdf)

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,8))

h=max_annual['PRCP'].values
ax1.hist(h, bins=np.arange(0,100,10), density=True)
ax2.hist(h, bins=np.arange(0,100,10), cumulative=1, density=True)

ax1.set(ylabel="pdf")
ax2.set(xlabel="annual daily precipitation maxima (mm)",
        ylabel="cdf",
        );
```

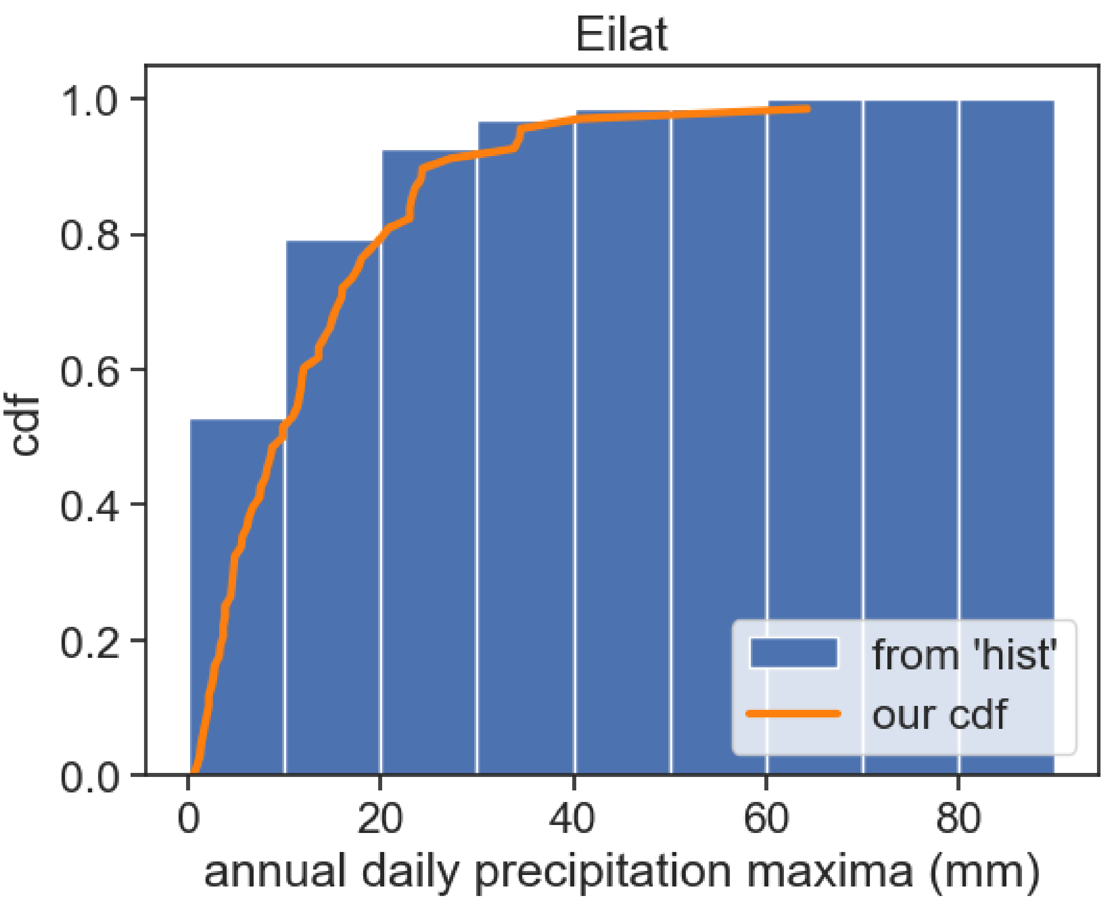


How to make a cdf by yourself?

```
# sort the annual daily precipitation maxima, from lowest to highest
max_annual['max_sorted'] = np.sort(max_annual['PRCP'])
# let's give it a name, h
h = max_annual['max_sorted'].values
# make an array "order" of size N=len(h), from 0 to N-1
N = len(h)
order = np.arange(N)
# make a new array, "fraction"
fraction = order / N
```

Plot it next to the cdf that pandas' `hist` makes for you. What do you see?

```python
fig, ax = plt.subplots(1, 1)
ax.hist(h, bins=np.arange(0,100,10), cumulative=1, density=True, label="from 'hist'")
ax.plot(h, fraction, color="tab:orange", linewidth=3, label="our cdf")
ax.set_ylabel("cdf")
ax.set_xlabel("annual daily precipitation maxima (mm)")
ax.set_title("Eilat")
ax.legend()
```



The generalized extreme value distribution has 3 parameters: shape, location, scale.

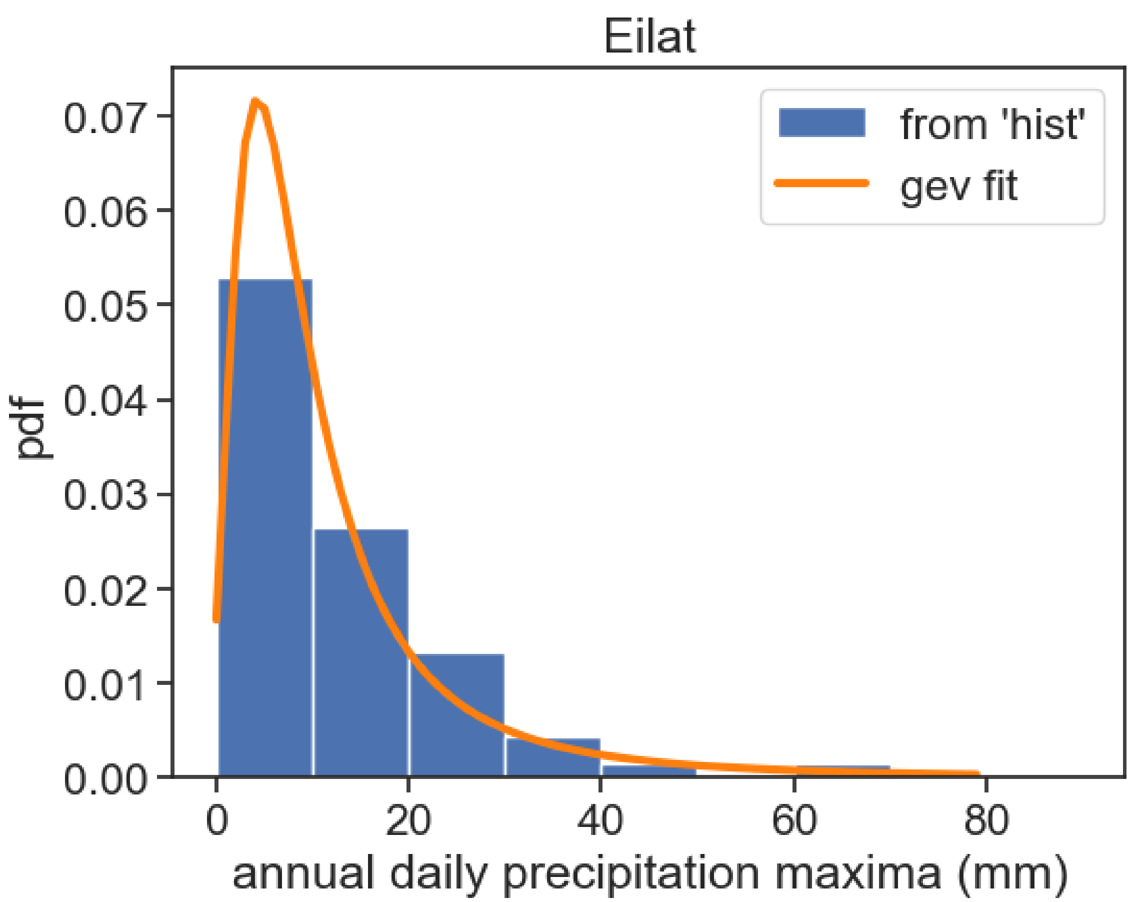
Let's get a "best fit" estimate of these parameters for Eilat's rainfall statistics.

```python
params = genextreme.fit(h)
print("Best fit:")
print(f"shape = {params[0]:.2f}\nlocation = {params[1]:.2f}\nscale = {params[2]:.2f}")
```

```
Best fit:
shape = -0.44
location = 6.17
scale = 5.58
```

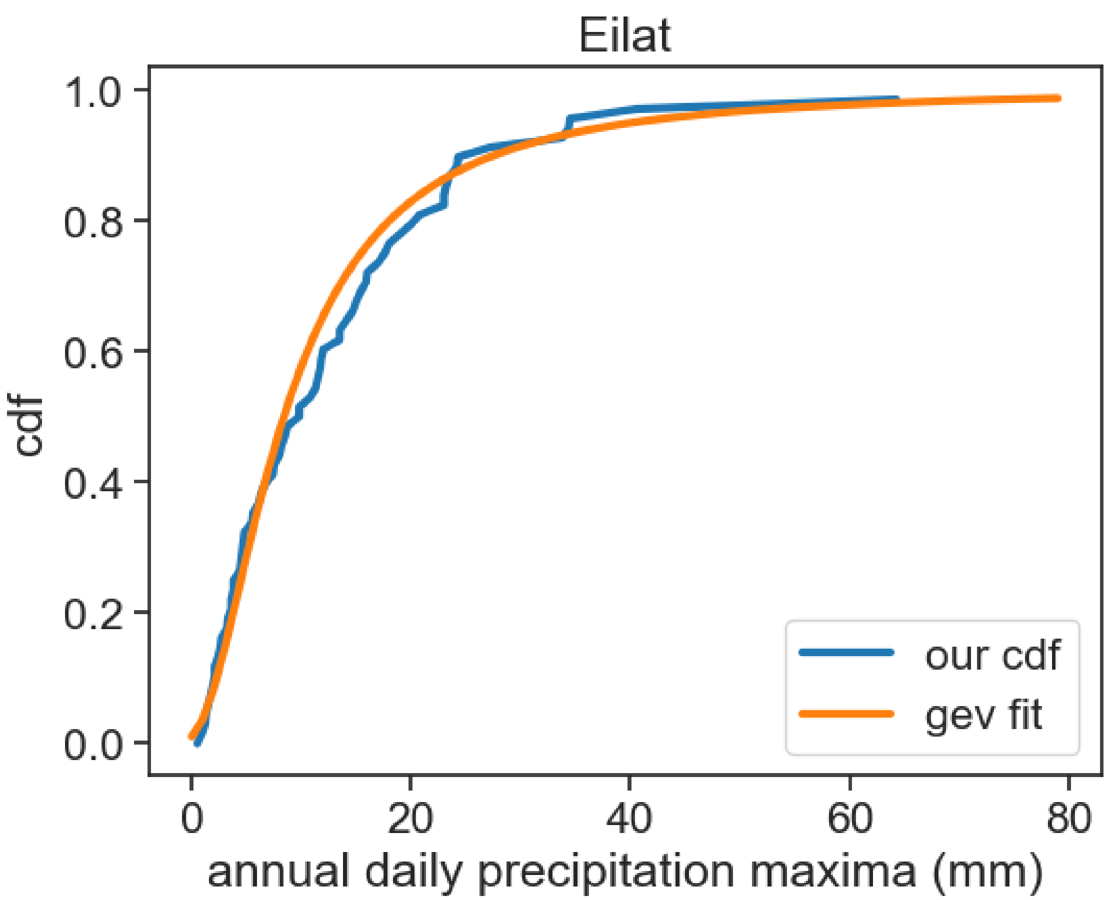
Let's see the GEV distribution for these parameters

```
fig, ax = plt.subplots(1, 1)
ax.hist(h, bins=np.arange(0,100,10), density=True, label="from 'hist'")
rain = np.arange(0,80)
pdf_rain = genextreme(c=params[0], loc=params[1], scale=params[2]).pdf(rain)
ax.plot(rain, pdf_rain, color="tab:orange", lw=3, label="gev fit")
ax.set_ylabel("pdf")
ax.set_xlabel("annual daily precipitation maxima (mm)")
ax.set_title("Eilat")
ax.legend()
```



We can do the same for the cdf...

```
fig, ax = plt.subplots(1, 1)
ax.plot(h, fraction, color="tab:blue", linewidth=3, label="our cdf")
rain = np.arange(0,80)
cdf_rain = genextreme(c=params[0], loc=params[1], scale=params[2]).cdf(rain)
ax.plot(rain, cdf_rain, color="tab:orange", lw=3, label="gev fit")
ax.set_ylabel("cdf")
ax.set_xlabel("annual daily precipitation maxima (mm)")
ax.set_title("Eilat")
ax.legend()
```



We are almost there! Remember that the return time are given by:

$$T_r(x) = \frac{1}{1 - F(x)},$$

where F is the cdf.

Survival $= 1 - F$

The package that we are using, scipy.stats.genextreme has a method called `isf`, or inverse survival function, which is exactly what we want! In order to use it, you have to feed it a "quantile" `q` or probability. Suppose you want to know how strong is a 1 in a 100 year event, then your return period is 100 (years), and the probability is simply its inverse, 1/100.

```
# Compute the return levels for several return periods.
return_periods = np.array([5, 10, 20, 50, 100, 500])
return_levels = genextreme.isf(1/return_periods, *params)

print("Return levels:")
print()
print("Period    Level")
print("(years)   (mm)")

for period, level in zip(return_periods, return_levels):
    print(f'{period:4.0f}  {level:9.2f}')
```

```
Return levels:

Period    Level
(years)   (mm)
   5       18.03
  10       27.64
  20       40.38
  50       64.17
 100       89.63
 500      189.22
```

You might want to do the opposite: given a list of critical daily max levels, what are the return periods for them? In this case you can use the `sf` method, "survival function".

```
levels_mm = np.array([20, 50, 100, 200, 300])
return_per = 1/genextreme.sf(levels_mm, *params)

print("Return levels:")
print()
print("Level       Period")
print("(mm)       (years)")
```

```
for level,period in zip(levels_mm, return_per):
    print(f'{level:9.2f}  {period:4.0f}')
```

```
Return levels:

Level       Period
(mm)       (years)
    20.00     6
    50.00    30
   100.00   126
   200.00   565
   300.00  1383
```

Part III

Evapotranspiration

10 Evapotranspiration

Dingman (2015), chapter 6.

Globally, about 62% of the precipitation that falls on the continents is evapotranspirated, amounting to 73 thousand km^3/yr. Of this, about 42% (29 thousand km^3/yr) is transpiration, and about 3% is open-water evaporation. Most of the remainder is interception loss; soil evaporation is a minor component of the total.

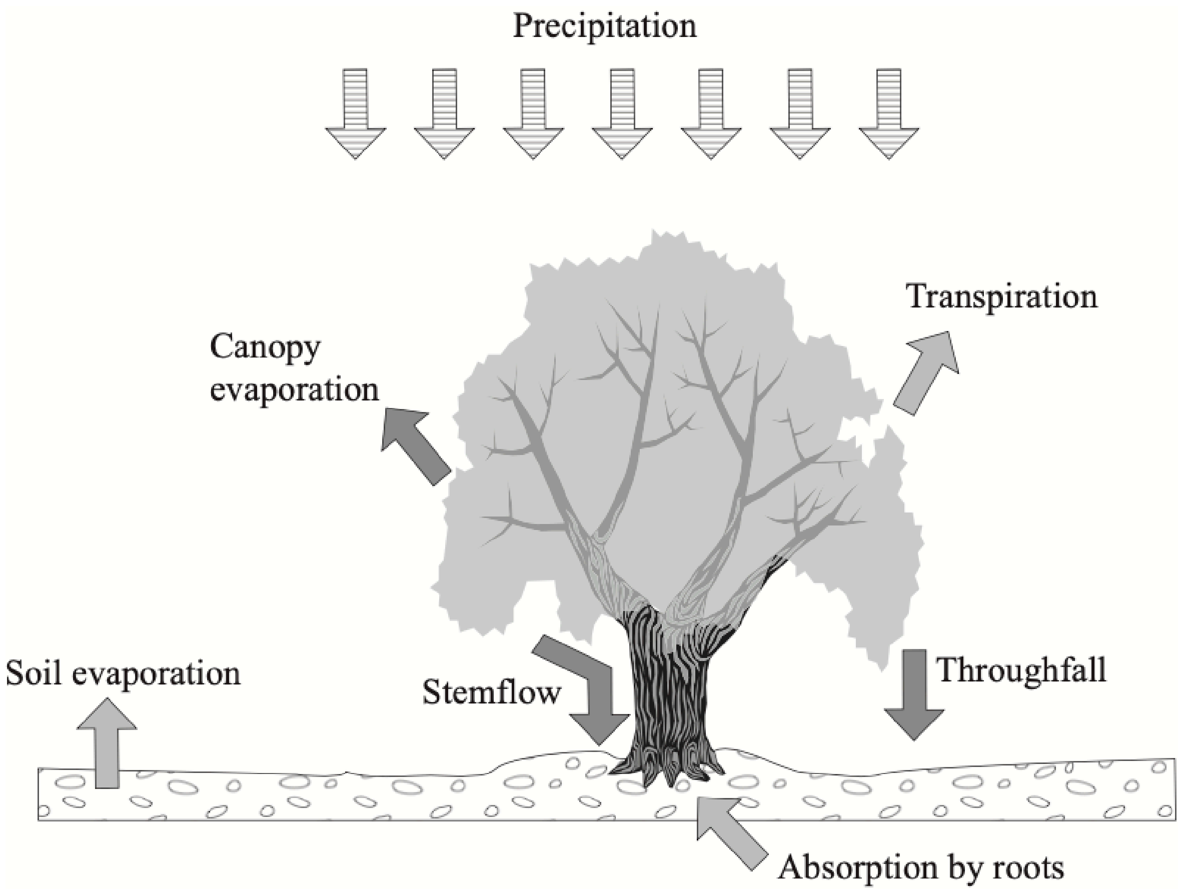


Fig. 5.1 : Principal elements of the interception and evaporation processes.

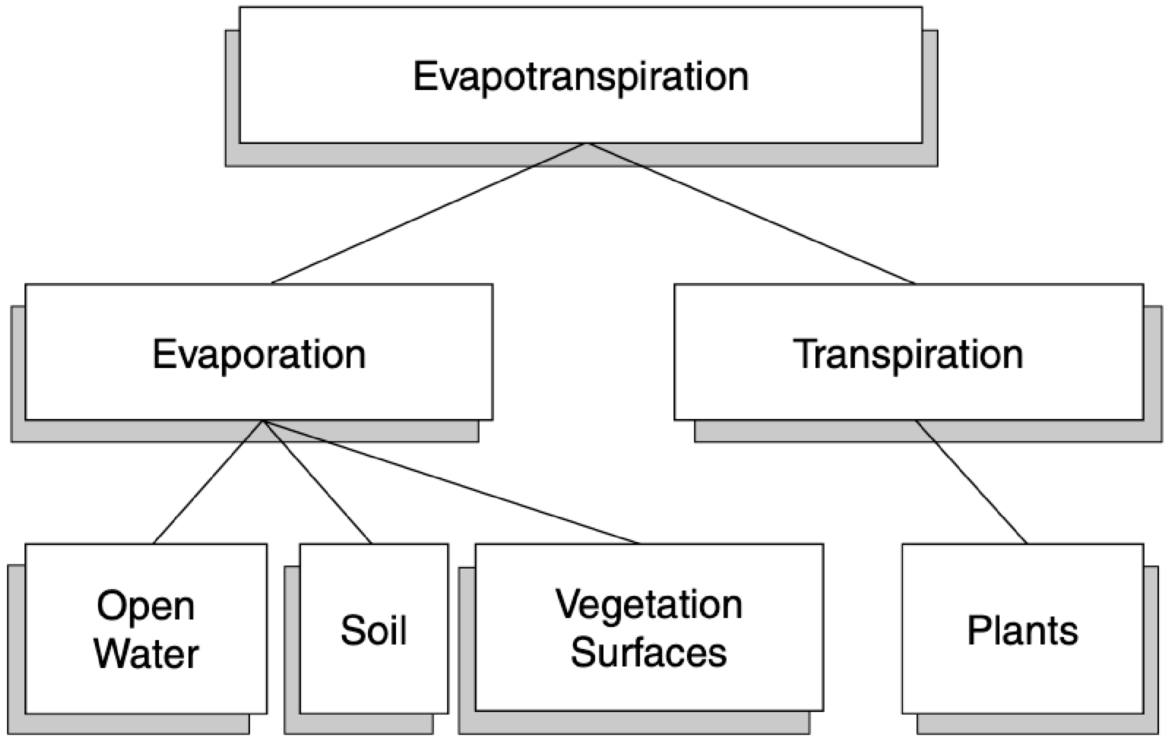


FIGURE 4.1 Evapotranspiration divided into subprocesses.

10.0.1 Potential Evapotranspiration

Potential Evapotranspiration (PET) is the rate at which evapotranspiration would occur from a large area completely and uniformly covered with growing vegetation with access to an unlimited supply of soil water and without advection or heat-storage effects.

Several characteristics of a vegetative surface have a strong influence on ET rate.

a) the albedo of the surface, which determines the net radiation;

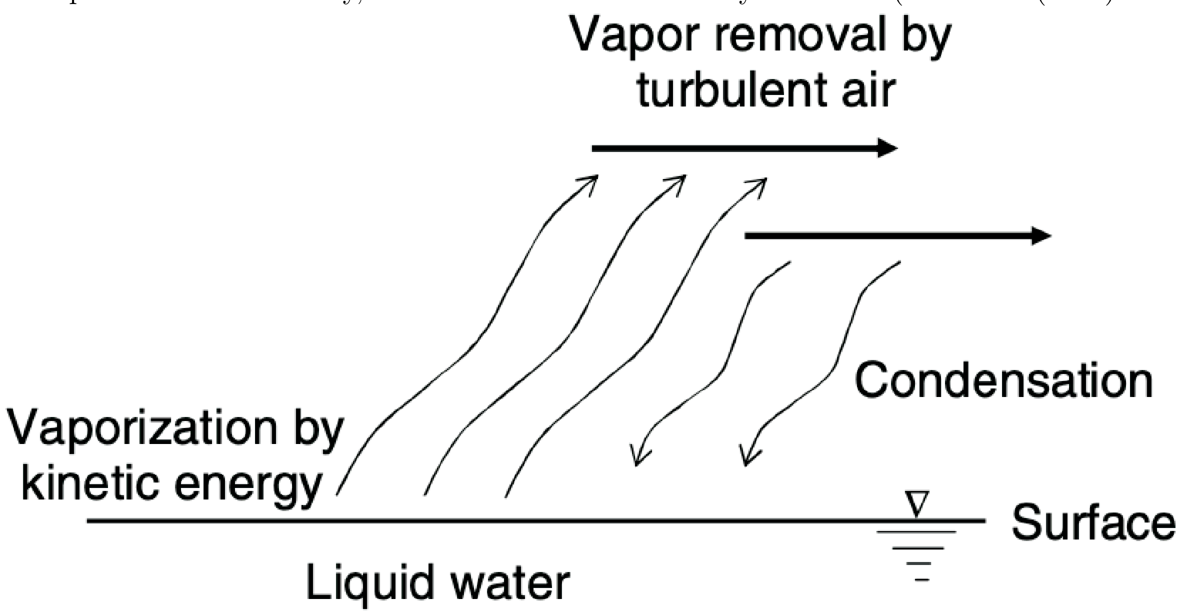
b) the maximum leaf conductance;

c) the atmospheric conductance, largely determined by vegetation height; d) presence or absence of intercepted water.

10.0.2 Reference-Crop Evapotranspiration

Reference-crop evapotranspiration (RET) is the amount of water transpired by a short green crop, completely shading the ground, of uniform height, and never short of water.

The magnitude of PET is often calculated from meteorological data collected under conditions in which the actual ET rate **is less than the potential rate**. If ET had been occurring at the potential rate, the latent- and sensible-heat exchanges between air and the surface, and hence the air temperature and humidity, would have been considerably different. (Brutsaert (2005) 1982)



10.1 Review of methods

There are a variety of ways to estimate evaporative flux in nature. The following table categorizes each method based on the data that must be acquired to apply it:

TABLE 4.3
Minimum Climatic Information Needs of ET Estimation Methods

Method	T^a	RH^b or e_d^c	Latitude	Elevation	R_s^d	u^e
Penman	x	x		x	x	x
Jensen–Haise	x			x	x	
SCS Blaney–Criddle	x		x			
Thornthwaite	x					

[a] Air temperature.
[b] Relative humidity.
[c] Actual vapor pressure of the air.
[d] Solar radiation.
[e] Wind speed.

Figure 10.1: Source: Ward and Trimble (2003)

These methods also vary in the timescales in which they are relevant, typically in correlation with the variety of data needed:

- Thornthwaite and SCS Blaney-Criddle: monthly or seasonal estimations (minimal data)
- Jensen-Haise: 5-day estimates (good enough timescale and data for irrigation scheduling)
- Penman: daily estimates
- Penman-Monteith: hourly estimates (requires a lot of data)

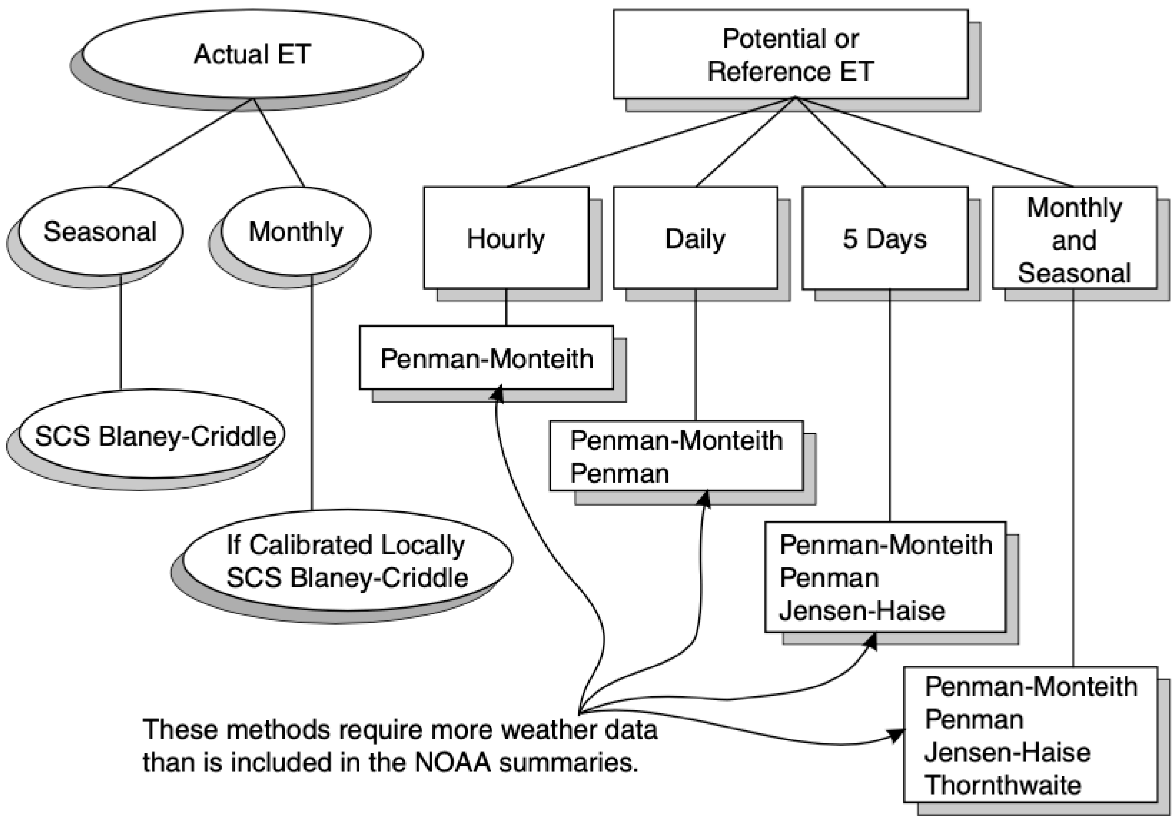


Figure 10.2: Source: Ward and Trimble (2003)

10.2 Thornthwaite

Source: Ward and Trimble (2003), pages 107-108.

Thornthwaite (1948) developed an equation to predict monthly evapotranspiration from mean monthly temperature and latitude data (Equation 4.27). The small amount of data needed is attractive because often ET needs to be predicted for sites where few weather data are available. Based on what we know about ET, we should be skeptical about the general applicability of such a simple equation. Thornthwaite (1948) was not satisfied with the proposed approach: "The mathematical development is far from satisfactory. It is empirical. … The chief obstacle at present to the development of a rational equation is the lack of understanding of why potential ET corresponding to a given temperature is not the same everywhere."

Taylor and Ashcroft (1972), as cited in Skaggs (1980), provided insight into the answer to Thornthwaite's ques- tion. They said:

> This equation, being based entirely upon a temperature relationship, has the disadvantage of a rather flimsy phys- ical basis and has only weak theoretical justification. Since temperature and vapor pressure gradients are mod- ified by the movement of air and by the heating of the soil and surroundings, the formula is not generally valid, but must be tested empirically whenever the climate is appreciably different from areas in which it has been tested. … In spite of these shortcomings, the method has been widely used. Because it is based entirely on temper- ature data that are available in a large number of localities, it can be applied in situations where the basic data of the Penman method are not available.

M.E. Jensen et al. (1990) warn that Thornthwaite's method is generally only applicable to areas that have climates similar to that of the east central U.S., and it is not applicable to arid and semiarid regions.

Thornthwaite (1948) found that evapotranspiration could be predicted from an equation of the form

$$E = 16\left[\frac{10\,T^{\text{monthly mean}}}{I}\right]^a,$$

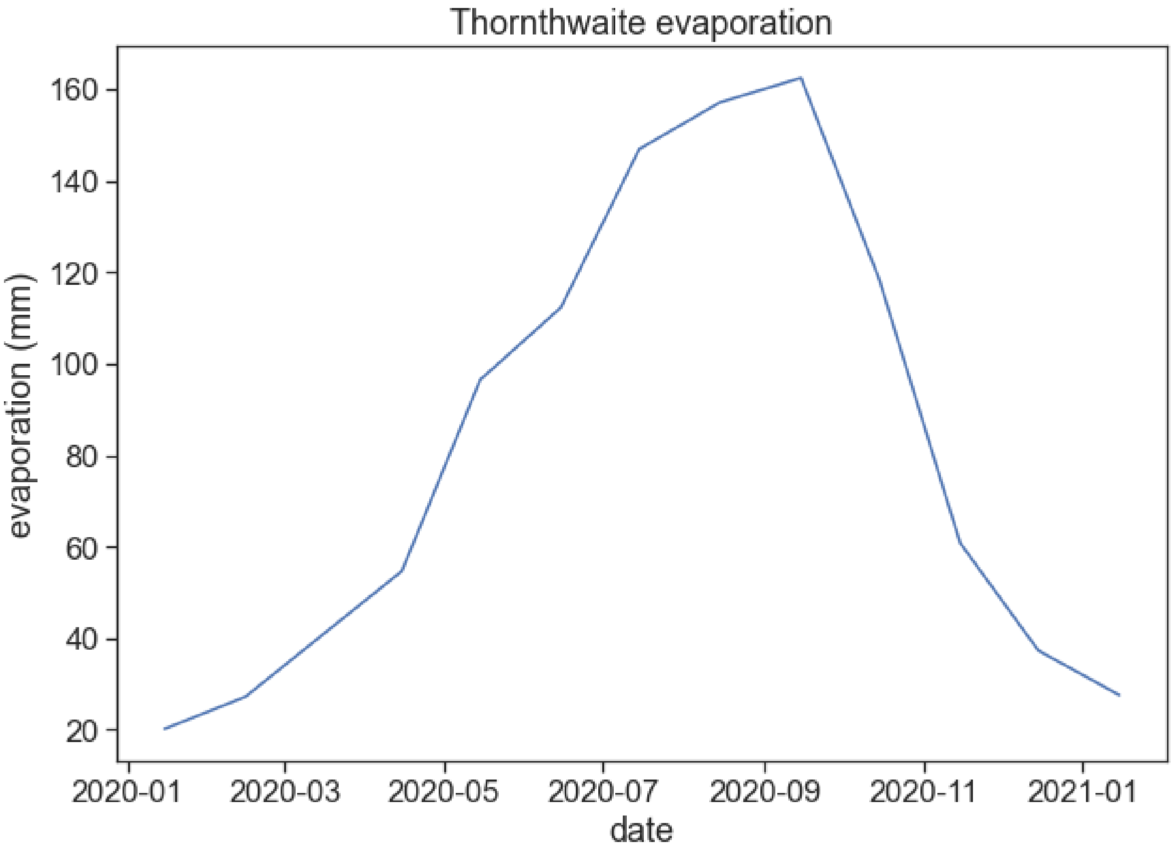
where

$$I = \sum_{i=1}^{12}\left[\frac{T_i^{\text{monthly mean}}}{5}\right]^{1.514},$$

and

$$\begin{aligned} a = &+ 6.75\times10^{-7}I^3 \\ &- 7.71\times10^{-5}I^2 \\ &+ 1.792\times10^{-2}I \\ &+ 0.49239 \end{aligned}$$

- E is the monthly potential ET (mm)
- $T_{\text{monthly mean}}$ is the mean monthly temperature in °C
- I is a heat index
- a is a location-dependent coefficient



10.3 Penman

Sources:
Brutsaert (2005), pages 123-127.
Ward and Trimble (2003), subsections 4.5.2, 4.5.3, 4.5.5, 4.6.6.
Allen et al. (1998), "Crop evapotranspiration - Guidelines for computing crop water requirements - FAO Irrigation and drainage paper 56"

The Penman model is almost entirely a theory-based formula for predicting evaporative flux. It can run on a much finer timescale, and requires a much wider variety of data than most models. In addition to temperature, the Penman functions on measurements of radiation, wind speed, elevation above sea level, vapor-pressure deficit, and heat flux density to the ground. The potential ET (in mm d^{-1}) is given by:

$$E = \frac{1}{\lambda}\left[\frac{\Delta}{\Delta+\gamma}Q_{ne} + \frac{\gamma}{\Delta+\gamma}E_A\right],$$

where Q_n is the available energy flux density

$$Q_n = R_n - G,$$

and E_A is the drying power of the air

$$E_A = 6.43 \cdot f(u) \cdot \text{VPD}.$$

The constituents of the equations above are

- E: potential evapotranspiration (mm d^{-1})
- Δ: slope of the saturation water vapor pressure curve (kPa °C^{-1})
- γ: psychrometric constant (kPA °C^{-1})
- λ: latent heat of vaporization (MJ kg^{-1})
- R_n: net radiation (MJ m$^{-2}d^{-1}$)
- G: heat flux density to the ground (MJ m$^{-2}d^{-1}$)
- $f(u)$: wind function (dimensionless)
- VPD: vapor pressure deficit (kPa)

and the number 6.43 adjusts the units of E_A so it is in MJ m$^{-2}d^{-1}$. In what follows, we will further discuss these constituents.

10.3.1 Psychrometric Constant

The psychrometric constant γ (kPA °C^{-1}) relates the partial pressure of water in air to the air temperature:

$$\begin{aligned}
\gamma &= \frac{c_p\, P}{\lambda \cdot MW_{\text{ratio}}} \\
P &= 101.3 - 0.01055H \\
\lambda &= 2.501 - 2.361 \times 10^{-3}\, T
\end{aligned}$$

- $MW_{\text{ratio}} = 0.622$: ratio molecular weight of water vapor/dry air
- P: atmospheric pressure (kPa). Can be either measured or inferred from station height above sea level (m).
- λ: latent heat of water vaporization (MJ kg^{-1})
- $c_p = 0.001013$: specific heat capacity of moist air (MJ kg^{-1} °C^{-1})

10.3.2 Net Radiation

Source: Ward and Trimble (2003), page 99.

R_n (MJ m$^{-2}d^{-1}$) is net radiation, the balance between net short wave R_s and the long wave R_b components of the radiation:

$$R_n = (1 - \alpha) R_s \downarrow - R_b \uparrow,$$

where α (dimensionless) is the albedo. The net outgoing thermal radiation R_b is given by

$$R_b = \left(a \frac{R_s}{R_{so} + b} \right) R_{bo},$$

where R_{so} is the solar radiation on a cloudless day, and it depends on latitude and day of the year. R_{bo} is given by

$$R_{bo} = \epsilon \, \sigma \, T^4_{Kelvin},$$

where $\sigma = 4.903 \times 10^{-9}$ MJ m^{-2} d^{-1} K^{-4}, and ϵ is net net emissivity:

$$\epsilon = -0.02 + 0.261 \exp\left(-7.77 \times 10^{-4} T^2_{Celcius}\right).$$

The parameters a and b are determined for the climate of the area:

- $a = 1.0$, $b = 0.0$ for humid areas,
- $a = 1.2$, $b = -0.2$ for arid areas,
- $a = 1.1$, $b = -0.1$ for semihumid areas.

We can find below a table for R_{so}, from Ward and Trimble (2003), page 100.

TABLE 4.5
Mean Solar Radiation R_{so} for Cloud

	Mean Solar R			
Latitude	**Jan**	**Feb**	**Mar**	**April**
60° N	2.51	5.99	13.82	22.32
55° N	4.31	8.67	16.33	24.16
50° N	6.70	11.43	18.55	25.83
45° N	9.34	14.36	20.64	27.21
40° N	12.27	17.04	22.90	28.34
35° N	14.95	19.55	24.58	29.31
30° N	17.46	21.65	25.96	29.85
25° N	19.68	23.45	27.21	30.14
20° N	21.65	25.00	28.18	30.14
15° N	23.57	26.50	29.01	29.85
10° N	25.25	27.63	29.43	29.60
5° N	26.92	28.47	29.85	29.31
0° E	28.18	29.18	30.02	28.47
5° S	28.05	29.85	29.85	27.76
10° S	30.69	30.44	29.43	26.80
15° S	31.53	30.69	28.76	25.54
20° S	32.36	30.69	27.93	23.99
25° S	32.95	30.69	27.09	22.32
30° S	33.37	30.44	25.96	20.81
35° S	33.49	29.73	24.58	18.97
40° S	33.49	28.76	22.90	17.04
45° S	33.49	27.76	21.23	14.95
50° S	32.95	26.38	19.26	12.85
55° S	32.36	24.83	17.17	10.47
60° S	31.53	23.15	15.07	7.83

Note: August values in the Southern Hemisphere
months were assumed because when actual days
March did not occur.

Source: Jensen, M.E., R.D. Burman, and R.G.
Society of Civil Engineers, New York, 1990. Co
permission of the American Society of Civil En

10.3.3 Heat Flux Density to the Ground

The heat flux density to the ground G (MJ m$^{-2}d^{-1}$) can be calculated using

$$G = 4.2\frac{T_{i+1} - T_{i-1}}{\Delta t},$$

where Δt is the time *in days* between midpoints of time periods $i+1$ and $i-1$, and T is the

air temperature (°C).
This expression is really a finite differences implementation of a time derivative:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{T(t+\Delta t) - T(t-\Delta t)}{2\Delta t}.$$

10.3.4 Vapor Pressure

Source: Ward and Trimble (2003), page 95.

The Vapor Pressure Deficit (VPD, in kPa) is the difference between saturation vapor pressure e_s and actual vapor pressure e_d:

$$\mathrm{VPD} = e_s - e_d.$$

For temperatures ranging from 0 to 50 °C, the saturation vapor pressure can be calculated with

$$e_s = \exp\left[\frac{16.78\,T - 116.9}{T + 237.3}\right],$$

and the actual vapor pressure is given by

$$e_d = e_s \frac{RH}{100},$$

where RH is the relative humidity (%), and the temperature T in the equations above is in degrees Celcius.

We can see below a graph of $e_s(T)$

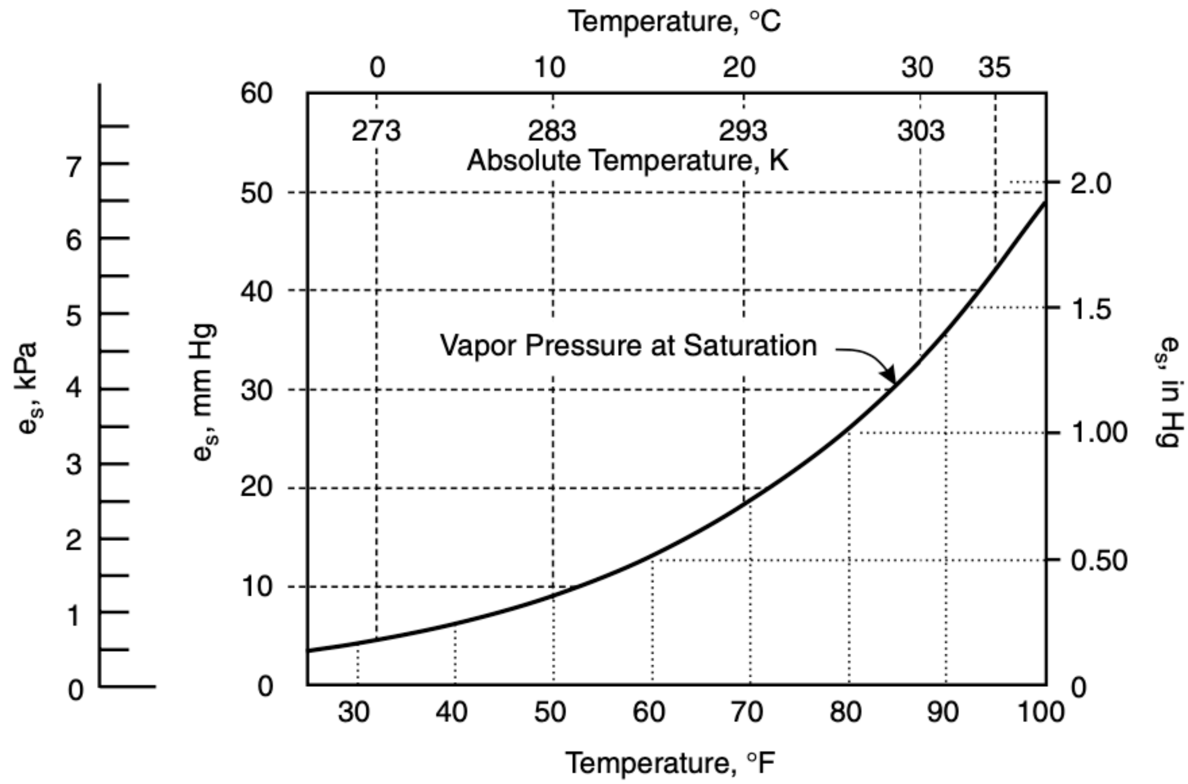


Figure 10.3: Source: Ward and Trimble (2003), page 96

The factor Δ is the slope of $e_s(T)$. See the figure below from Brutsaert, where the saturation vapor pressure is called e^*):

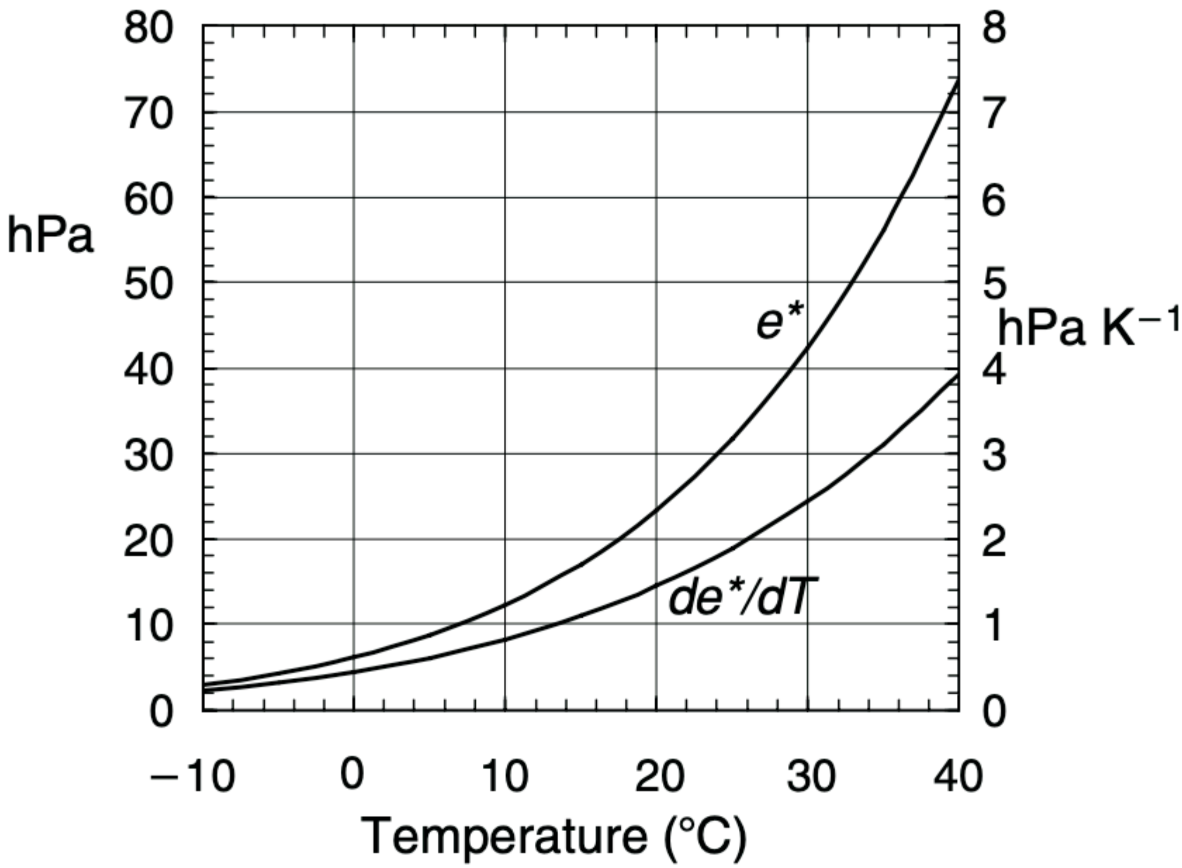


Figure 10.4: Source: Brutsaert (2005), page 28

There are a few ways of defining the function for $\Delta(T)$ (kPa °C^{-1}). Ward and Trimble (2003) give the following:

$$\Delta = 0.200 \cdot (0.00738\,T + 0.8072)^7 - 0.000116,$$

while differentiating the exponential expression given before yields:

$$\Delta = \frac{\mathrm{d}e_s}{\mathrm{d}T} = e_s(T) \cdot \frac{4098.79}{(T + 237.3)^2}.$$

10.3.5 Wind Function

Source: Ward and Trimble (2003), page 108

$$f(u) = 0.26(1.0 + 0.54\,u_2)$$

11 Meaning of "potential" evapotranspiration

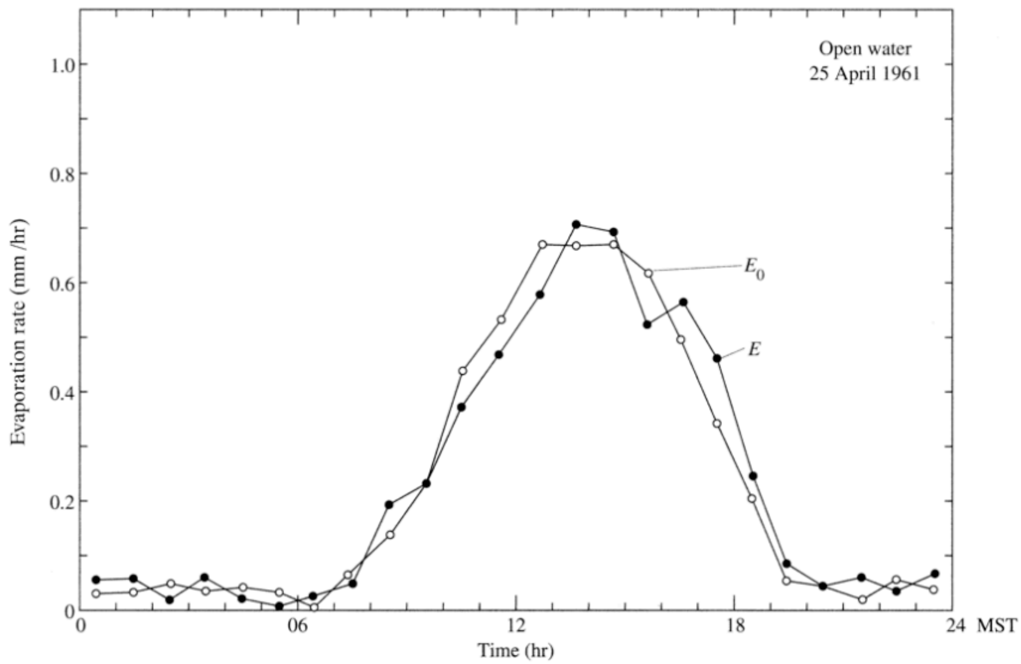


Figure 6.3 Comparison of hourly evaporation rates determined from measurements in a shallow pan (E) with those computed via the combination approach [E_0, equation (6.36)] at Tempe, Arizona [Van Bavel (1966). Potential evaporation: The combination concept and its experimental verification. *Water Resources Research* 2:455–467, with permission of the American Geophysical Union].

11.0.1 Crop Coefficient

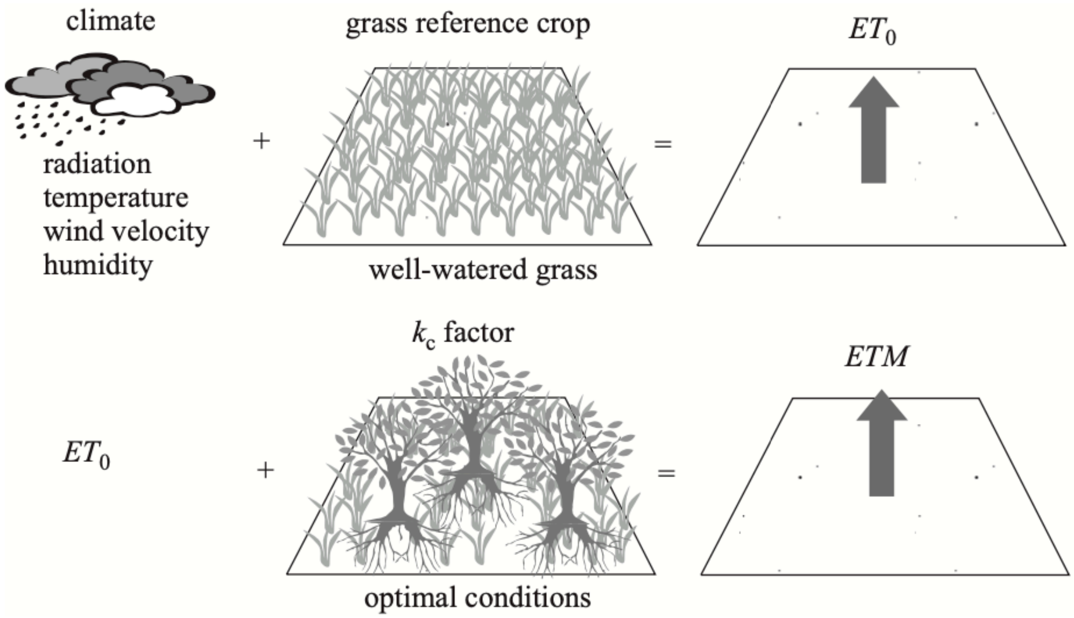


Fig. 5.9 : Water requirements of crops (ETM) and reference evapotranspiration (ET_0) (modified from FAO 1998).

$$E_t = k_c E_{tr,\,tp}$$

E_t = actual ET
k_c = crop coefficient
E_{tr} = reference crop ET
E_{tp} = potential ET

TABLE 4.7
Seasonal Consumptive Use Coefficients *K* for Irrigated Crops in the Western U.S.[a]

Crop	Length of Normal Growing Season[b]	Coefficient K[c]
Alfalfa	Between frosts	0.80 to 0.90
Bananas	Full year	0.80 to 1.00
Beans	3 months	0.60 to 0.70
Cocoa	Full year	0.70 to 0.80
Coffee	Full year	0.70 to 0.80
Corn (maize)	4 months	0.75 to 0.85
Cotton	7 months	0.60 to 0.70
Dates	Full year	0.65 to 0.85
Flax	7 to 8 months	0.70 to 0.80
Grains, small	3 months	0.75 to 0.85
Grains, sorghum	4 to 5 months	0.70 to 0.80
Oil seeds	3 to 5 months	0.65 to 0.75
Orchard crops		
Avocado	Full year	0.50 to 0.55
Grapefruit	Full year	0.55 to 0.65
Orange and lemon	Full year	0.45 to 0.55
Walnuts	Between frosts	0.60 to 0.70
Deciduous	Between frosts	0.60 to 0.70
Pasture crops		
Grass	Between frosts	0.75 to 0.85
Ladino white clover	Between frosts	0.80 to 0.85
Potatoes	3 to 5 months	0.65 to 0.75
Rice	3 to 5 months	1.00 to 1.10
Soybeans	140 days	0.65 to 0.70
Sugar beets	6 months	0.65 to 0.75
Sugarcane	Full year	0.80 to 0.90
Tobacco	4 months	0.70 to 0.80
Tomatoes	4 months	0.65 to 0.70
Vineyard	5 to 7 months	0.50 to 0.60

[a] From USDA (1970).

[b] Length of season depends largely on variety and time of year when the crop is grown. Annual crops grown during the winter period may take much longer than if grown in the summer.

[c] The lower values of K for use in the Blaney–Criddle formula are for the more humid areas; the higher values are for the more arid climates.

Source: Jensen, M.E. et al., *Evapotranspiration and Irrigation Water Requirements,* ASCE, New York, 1990. ©1990 by ASCE. Reproduced with permission.

11.1 Pitfalls

Different books and papers will present slightly different versions of the Penman equation. Basically, they differ in the units they use for the various components, and one should be vary aware of what inputs any given equation is expecting to get.

12 Exercises

We will calculate evapotranspiration using two methods: Thornthwaite and Penman. After that, we will compare these estimates with measurements of pan evaporation.

12.1 Download data from the IMS

Please follow the instructions below **exactly as they are written**. Go to the Israel Meteorological Service website, and download the following data:

1. 10-min data

- On the navigation bar on the left, choose "10 Minutes Observations"
- Clock: Local time winter (UTC +2)
- Choose the following date range: 01/01/2020 00:00 to 01/01/2021 00:00
- Choose station Bet Dagan
- Select units: Celcius, m/s, KJ/m^2
- Under "Select parameters", choose "Check All"
- Choose option "by station", then "Submit"
- "Download Result as" CSV, call it `bet-dagan-10min.csv`

2. radiation data

- On the navigation bar on the left, choose "Hourly Radiation"
- Clock: Local time winter (UTC +2)
- Choose the following date range: 01/01/2020 00:00 to 01/01/2021 00:00
- Select hours: Check all hours
- Choose station Bet Dagan
- Select units: KJ/m^2
- Under "Select parameters", choose "Check All"
- "Download Result as" CSV, call it `bet-dagan-radiation.csv`

3. pan evaporation data

- On the navigation bar on the left, choose "Daily Observations"
- Choose the following date range: 01/01/2020 00:00 to 01/01/2021 00:00
- Choose station Bet Dagan Man

- Select units: Celcius
- Under "Select parameters", choose "Check All"
- "Download Result as" CSV, call it `bet-dagan-pan.csv`

12.2 Install and import relevant packages

We will need to use two new packages:

- pyet: Estimation of Potential Evapotranspiration
- noaa_ftp: Download data from NOAA

If you don't have them installed yet, run this:

```
!pip install pyet
!pip install noaa_ftp
```

Once they are installed, import all the necessary packages for today's exercises.

```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import pandas as pd
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()  # datetime converter for a matplotlib
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
import pyet
from noaa_ftp import NOAA
```

12.3 import 10-minute data

```
df = pd.read_csv('bet-dagan-10min.csv',
                #  encoding = "ISO-8859-8",  # this shows hebrew characters properly
                 na_values=["-"]           # substitute "-" for NaN
                 )
df['timestamp'] = pd.to_datetime(df['Date & Time (Winter)'], dayfirst=True)
df = df.set_index('timestamp')
# resample to daily data according to "mean"
df = df.resample('D').mean()
```

```
# convert hecto pascals (hPa) to kilo pascals (kPa)
df["Pressure (kPa)"] = df["Pressure at station level (hPa)"] / 10.0
df
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_16976/111106941.py:8: FutureWarnin
  df = df.resample('D').mean()
```

	Pressure at station level (hPa)	Relative humidity (%)	Temperature (°C)	Maximum temperat
timestamp				
2020-01-01	1013.263889	80.590278	12.375000	12.486806
2020-01-02	1011.922917	85.631944	12.020833	12.104861
2020-01-03	1013.757639	60.756944	12.962500	13.086111
2020-01-04	1011.581250	76.909722	10.849306	10.938194
2020-01-05	1012.361806	79.583333	12.956250	13.053472
...	...	...	...	...
2020-12-28	1014.429861	58.729167	14.797917	14.915972
2020-12-29	1015.031944	71.215278	14.146528	14.315278
2020-12-30	1013.234028	68.923611	14.186806	14.336111
2020-12-31	1011.840972	75.465278	14.915278	15.068056
2021-01-01	1012.748760	85.115702	14.980992	15.133884

12.4 import radiation data

```
df_rad = pd.read_csv('bet-dagan-radiation.csv',
                     na_values=["-"]
                     )
df_rad['Date'] = pd.to_datetime(df_rad['Date'], dayfirst=True)
df_rad = df_rad.set_index('Date')
df_rad
```

	Station	Radiation type	Hourly radiation 05-06 (KJ/m^2)	Hourly r
Date				
2020-01-01	Bet Dagan Rad 01/1991-04/2023	Global	0.0	10.8
2020-01-01	Bet Dagan Rad 01/1991-04/2023	Direct	0.0	3.6
2020-01-01	Bet Dagan Rad 01/1991-04/2023	Diffused	0.0	10.8
2020-01-02	Bet Dagan Rad 01/1991-04/2023	Global	0.0	10.8

Date	Station	Radiation type	Hourly radiation 05-06 (KJ/m^2)	Hourly r
2020-01-02	Bet Dagan Rad 01/1991-04/2023	Direct	0.0	3.6
...	...	...	...	...
2020-12-31	Bet Dagan Rad 01/1991-04/2023	Direct	0.0	0.0
2020-12-31	Bet Dagan Rad 01/1991-04/2023	Diffused	0.0	14.4
2021-01-01	Bet Dagan Rad 01/1991-04/2023	Global	0.0	14.4
2021-01-01	Bet Dagan Rad 01/1991-04/2023	Direct	0.0	0.0
2021-01-01	Bet Dagan Rad 01/1991-04/2023	Diffused	0.0	14.4

Choose only "Global" radiation. Then sum all hours of the day, and convert from kJ to MJ.

```
df_rad = df_rad[df_rad["Radiation type"] == "Global "]
df_rad['daily_radiation_MJ_per_m2_per_day'] = (df_rad.iloc[:, 3:]    # take all rows, columns
                                                      .sum(axis=1) /  # sum all columns
                                                      1000            # divide by 1000 t
                                                )
df_rad
```

```
/var/folders/c3/7hp0d36n6vv8jc9hm2440__00000gn/T/ipykernel_16976/3934990453.py:2: SettingWit
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
  df_rad['daily_radiation_MJ_per_m2_per_day'] = (df_rad.iloc[:, 3:]    # take all rows, colu
```

Date	Station	Radiation type	Hourly radiation 05-06 (KJ/m^2)	Hourly r
2020-01-01	Bet Dagan Rad 01/1991-04/2023	Global	0.0	10.8
2020-01-02	Bet Dagan Rad 01/1991-04/2023	Global	0.0	10.8
2020-01-03	Bet Dagan Rad 01/1991-04/2023	Global	0.0	10.8
2020-01-04	Bet Dagan Rad 01/1991-04/2023	Global	0.0	3.6
2020-01-05	Bet Dagan Rad 01/1991-04/2023	Global	0.0	7.2
...	...	...	...	...
2020-12-28	Bet Dagan Rad 01/1991-04/2023	Global	0.0	14.4
2020-12-29	Bet Dagan Rad 01/1991-04/2023	Global	0.0	14.4
2020-12-30	Bet Dagan Rad 01/1991-04/2023	Global	0.0	21.6
2020-12-31	Bet Dagan Rad 01/1991-04/2023	Global	0.0	14.4
2021-01-01	Bet Dagan Rad 01/1991-04/2023	Global	0.0	14.4

12.5 import pan evaporation data

```
df_pan = pd.read_csv('bet-dagan-pan.csv',
                      na_values=["-"]           # substitute "-" for NaN
                     )
df_pan['Date'] = pd.to_datetime(df_pan['Date'], dayfirst=True)
df_pan = df_pan.set_index('Date')
df_pan
```

	Station	Maximum Temperature (°C)	Minimum Temperature (°C)
Date			
2020-01-01	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-01-02	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-01-03	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-01-04	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-01-05	Bet Dagan Man 01/1964-03/2023	NaN	NaN
...	...	...	...
2020-12-28	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-12-29	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-12-30	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2020-12-31	Bet Dagan Man 01/1964-03/2023	NaN	NaN
2021-01-01	Bet Dagan Man 01/1964-03/2023	NaN	NaN

12.6 calculate penman

We need some data about the Bet Dagan Station. See here.

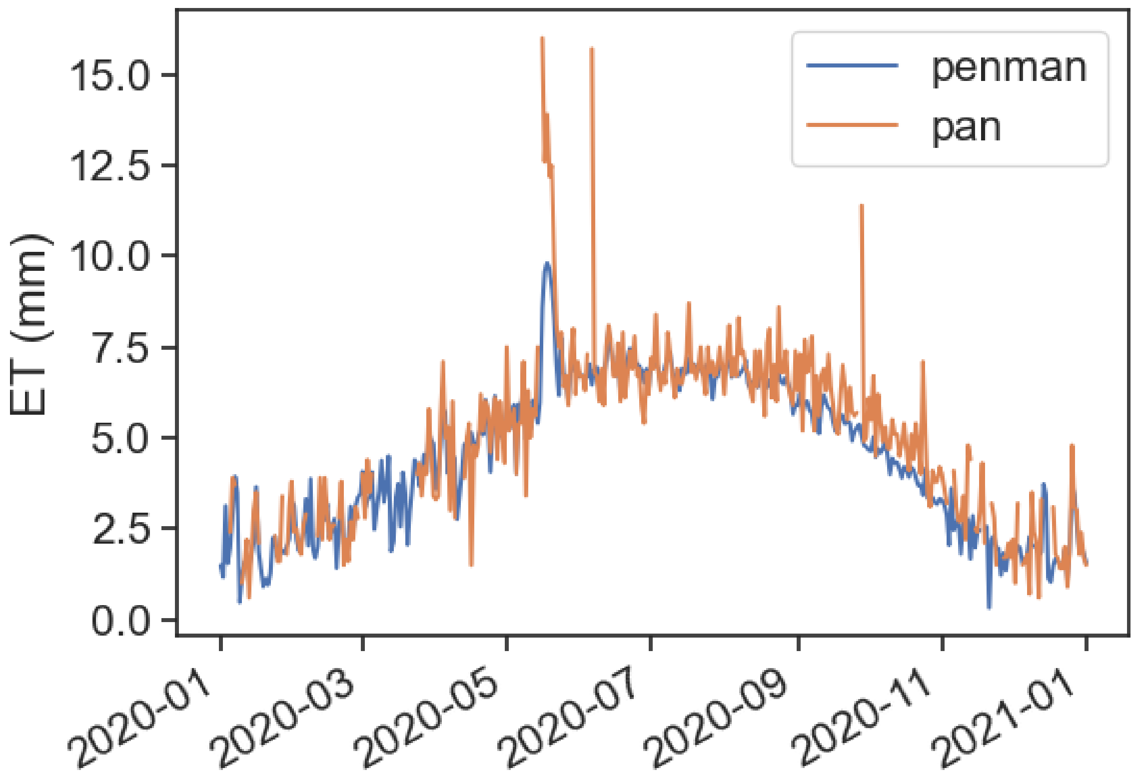
- Latitude: 32.0073°
- Elevation: 31 m

```
# average day temperature [°C]
tmean = df["Temperature (°C)"]
# mean day wind speed [m/s]
wind = df["Wind speed (m/s)"]
# mean daily relative humidity [%]
rh = df["Relative humidity (%)"]
# incoming solar radiation [MJ m-2 d-1]
rs = df_rad["daily_radiation_MJ_per_m2_per_day"]
```

```python
# atmospheric pressure [kPa]
pressure = df["Pressure (kPa)"]
# the site elevation [m]
elevation = 31.0
# the site latitude [rad]
latitude = pyet.deg_to_rad(32.0073)
penm = pyet.combination.penman(tmean=tmean,
                               wind=wind,
                               pressure=pressure,
                               elevation=elevation,
                               rh=rh,
                               rs=rs,
                               lat=latitude,
                              )
```

```python
fig, ax = plt.subplots(1)
ax.plot(penm, label="penman")
ax.plot(df_pan["Daily evaporation type A (mm)"], label="pan")
ax.legend()
plt.gcf().autofmt_xdate()  # makes slated dates
ax.set_ylabel("ET (mm)")
```

```
Text(0, 0.5, 'ET (mm)')
```



12.7 Thornthwaite

$$E = 16 \left[\frac{10\, T^{\text{monthly mean}}}{I} \right]^a,$$

where

$$I = \sum_{i=1}^{12} \left[\frac{T_i^{\text{monthly mean}}}{5} \right]^{1.514},$$

and

$$\begin{aligned} a = {} & 6.75 \times 10^{-7} I^3 \\ & - 7.71 \times 10^{-5} I^2 \\ & + 1.792 \times 10^{-2} I \\ & + 0.49239 \end{aligned}$$

- E is the monthly potential ET (mm)
- $T_{\text{monthly mean}}$ is the mean monthly temperature in °C
- I is a heat index
- a is a location-dependent coefficient

From df, make a new dataframe, df_th, that stores monthly temperatures means. Use `resample` function.

```
# monthly data
df_th = (df['Temperature (°C)'].resample('MS')  # MS assigns mean to first day in the month
                               .mean()
                               .to_frame()
        )

# we now add 14 days to the index, so that all monthly data is in the middle of the month
# not really necessary, makes plot look better
df_th.index = df_th.index + pd.DateOffset(days=14)
df_th
```

	Temperature (°C)
timestamp	
2020-01-15	12.484812
2020-02-15	14.044349
2020-03-15	16.371381
2020-04-15	18.476339
2020-05-15	23.177769
2020-06-15	24.666423
2020-07-15	27.380466
2020-08-15	28.099328
2020-09-15	28.421644
2020-10-15	25.058944
2020-11-15	19.266082
2020-12-15	15.915031
2021-01-15	14.980992

Calculate I, then a, and finally E_p. Add E_p as a new column in df_th.

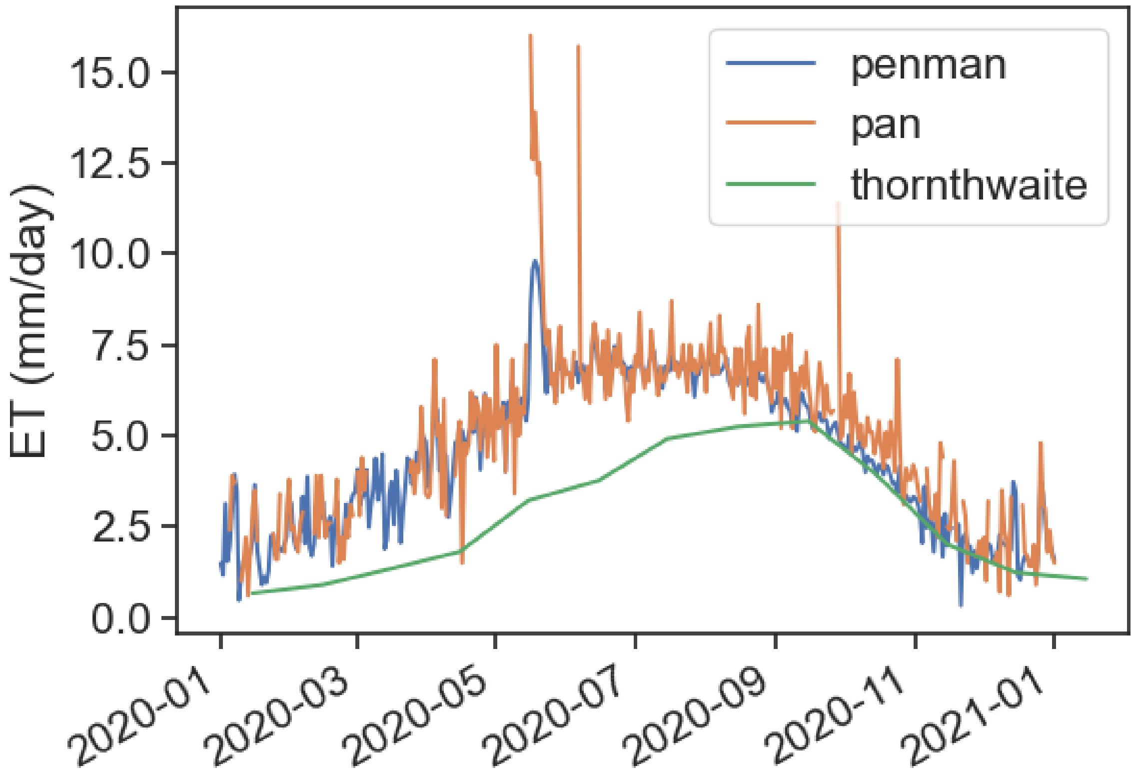
```
# Preparing "I" for the Thornthwaite equation
I = np.sum(
            (
              df_th['Temperature (°C)'] / 5
```

```
             ) ** (1.514)
          )

# Preparing "a" for the Thornthwaite equation
a = (+6.75e-7 * I**3
     -7.71e-5 * I**2
     +1.792e-2 * I
     + 0.49239)

# The final Thornthwaite model for monthly potential ET (mm)
df_th['Ep (mm/month)'] = 16*(
                               (
                                  10 * df_th['Temperature (°C)'] / I
                               ) ** a
                            )
df_th
```

timestamp	Temperature (°C)	Ep (mm/month)
2020-01-15	12.484812	20.018337
2020-02-15	14.044349	26.999656
2020-03-15	16.371381	39.865158
2020-04-15	18.476339	54.213825
2020-05-15	23.177769	96.461505
2020-06-15	24.666423	112.997152
2020-07-15	27.380466	147.331024
2020-08-15	28.099328	157.362480
2020-09-15	28.421644	161.990981
2020-10-15	25.058944	117.623700
2020-11-15	19.266082	60.299205
2020-12-15	15.915031	37.101123
2021-01-15	14.980992	31.814464

```
fig, ax = plt.subplots(1)
ax.plot(penm, label="penman")
ax.plot(df_pan["Daily evaporation type A (mm)"], label="pan")
ax.plot(df_th['Ep (mm/month)']/30, label="thornthwaite")

ax.legend()
plt.gcf().autofmt_xdate()  # makes slated dates
ax.set_ylabel("ET (mm/day)")
```

```
Text(0, 0.5, 'ET (mm/day)')
```



12.8 Data from NOAA

Let's download data from a different repository. More specifically, we will retrieve sub-hourly data from the U.S. Climate Reference Network. We can see all the sites in this map. Besides the sub-hourly data, we can find other datasets (monthly, daily, etc).

As an example, we will choose the statin in Austin, Texas. In order to download, we will access the data from the FTP client using the python package `noaa_ftp`.

The `dir` command list everything in the folder:

```
noaa_dir = NOAA("ftp.ncdc.noaa.gov", 'pub/data/uscrn/products/subhourly01').dir()
```

```
drwxrwsr-x   2 ftp      ftp          8192 Oct  7  2020 2006
drwxrwsr-x   2 ftp      ftp          8192 Nov 10  2021 2007
drwxrwsr-x   2 ftp      ftp          8192 Dec  1  2020 2008
```

```
drwxrwsr-x   2 ftp      ftp         12288 May 25  2021 2009
drwxrwsr-x   2 ftp      ftp         12288 Nov 10  2021 2010
drwxrwsr-x   2 ftp      ftp         12288 Nov 12  2021 2011
drwxrwsr-x   2 ftp      ftp         12288 Nov 12  2021 2012
drwxrwsr-x   2 ftp      ftp         12288 Nov 15  2021 2013
drwxrwsr-x   2 ftp      ftp         12288 Nov 15  2021 2014
drwxrwsr-x   2 ftp      ftp         12288 Nov 12  2021 2015
drwxrwsr-x   2 ftp      ftp         12288 Nov 12  2021 2016
drwxrwsr-x   2 ftp      ftp         12288 Nov 15  2021 2017
drwxrwsr-x   2 ftp      ftp         12288 Nov 12  2021 2018
drwxrwsr-x   2 ftp      ftp         12288 Nov 24  2021 2019
drwxrwsr-x   2 ftp      ftp         12288 Nov 30  2021 2020
drwxrwsr-x   2 ftp      ftp         12288 Jan 29  2022 2021
drwxrwsr-x   2 ftp      ftp         12288 Aug 23  2022 2022
drwxrwsr-x   2 ftp      ftp         12288 Feb  2 16:55 2023
-rw-rw-r--   1 ftp      ftp          2157 Feb 18  2022 headers.txt
-rw-rw-r--   1 ftp      ftp           456 Oct  7  2020 HEADERS.txt
-rw-rw-r--   1 ftp      ftp         14892 Feb 18  2022 readme.txt
-rw-rw-r--   1 ftp      ftp         14936 Sep 21  2021 README.txt
drwxrwsr-x   2 ftp      ftp          4096 Apr 24 01:50 snapshots
```

Let's download two files: * sub-hourly data for the year 2022 for Austin, TX. * the `HEADERS.txt` contains the names of the columns in the csv.

```
noaa = NOAA("ftp.ncdc.noaa.gov", 'pub/data/uscrn/products/subhourly01').download('HEADERS.txt
noaa = NOAA("ftp.ncdc.noaa.gov", 'pub/data/uscrn/products/subhourly01/2022').download('CRNS0
```

```
Downloading:   0% [                                  ] ETA:  --:--:--   0.0 s/B
Downloading: 100% [##################################] ETA:  00:00:00 505.2 B/s
Downloading:   0% [                                  ] ETA:  --:--:--   0.0 s/B
Downloading:   0% [                                ] ETA:   0:32:21   7.1 KiB/s
Downloading:   0% [                                ] ETA:   0:22:44  10.1 KiB/s
Downloading:   0% [                                ] ETA:   0:10:06  22.8 KiB/s
Downloading:   0% [                                ] ETA:   0:08:11  28.1 KiB/s
Downloading:   0% [                                ] ETA:   0:07:16  31.6 KiB/s
Downloading:   0% [                                ] ETA:   0:06:41  34.2 KiB/s
Downloading:   0% [                                ] ETA:   0:06:27  35.5 KiB/s
Downloading:   1% [                                ] ETA:   0:06:10  37.1 KiB/s
Downloading:   1% [                                ] ETA:   0:06:17  36.3 KiB/s
Downloading:   1% [                                ] ETA:   0:06:14  36.5 KiB/s
Downloading:   1% [                                ] ETA:   0:06:05  37.5 KiB/s
Downloading:   1% [                                ] ETA:   0:06:03  37.6 KiB/s
```

```
Downloading:   1% [                                                    ] ETA:   0:05:45  39.5 KiB/s
Downloading:   1% [                                                    ] ETA:   0:05:34  40.8 KiB/s
Downloading:   1% [                                                    ] ETA:   0:05:24  42.0 KiB/s
Downloading:   1% [                                                    ] ETA:   0:05:17  42.9 KiB/s
Downloading:   1% [                                                    ] ETA:   0:05:13  43.3 KiB/s
Downloading:   2% [                                                    ] ETA:   0:05:00  45.2 KiB/s
Downloading:   2% [                                                    ] ETA:   0:05:02  44.9 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:52  46.3 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:53  46.2 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:44  47.6 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:43  47.8 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:35  49.1 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:28  50.2 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:29  50.2 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:20  51.7 KiB/s
Downloading:   2% [                                                    ] ETA:   0:04:13  53.0 KiB/s
Downloading:   3% [                                                    ] ETA:   0:04:07  54.2 KiB/s
Downloading:   3% [#                                                   ] ETA:   0:03:57  56.4 KiB/s
Downloading:   3% [#                                                   ] ETA:   0:03:52  57.5 KiB/s
Downloading:   3% [#                                                   ] ETA:   0:03:43  59.6 KiB/s
Downloading:   3% [#                                                   ] ETA:   0:03:36  61.6 KiB/s
Downloading:   3% [#                                                   ] ETA:   0:03:36  61.6 KiB/s
Downloading:   4% [#                                                   ] ETA:   0:03:26  64.3 KiB/s
Downloading:   4% [#                                                   ] ETA:   0:03:23  65.2 KiB/s
Downloading:   4% [#                                                   ] ETA:   0:03:17  67.1 KiB/s
Downloading:   4% [#                                                   ] ETA:   0:03:16  67.2 KiB/s
Downloading:   4% [#                                                   ] ETA:   0:03:08  70.0 KiB/s
Downloading:   5% [#                                                   ] ETA:   0:02:59  73.4 KiB/s
Downloading:   5% [#                                                   ] ETA:   0:02:59  73.3 KiB/s
Downloading:   5% [#                                                   ] ETA:   0:02:49  77.2 KiB/s
Downloading:   5% [#                                                   ] ETA:   0:02:49  77.2 KiB/s
Downloading:   5% [#                                                   ] ETA:   0:02:39  81.5 KiB/s
Downloading:   6% [#                                                   ] ETA:   0:02:39  81.6 KiB/s
Downloading:   6% [##                                                  ] ETA:   0:02:31  85.8 KiB/s
Downloading:   6% [##                                                  ] ETA:   0:02:30  86.2 KiB/s
Downloading:   6% [##                                                  ] ETA:   0:02:22  90.5 KiB/s
Downloading:   7% [##                                                  ] ETA:   0:02:21  91.0 KiB/s
Downloading:   7% [##                                                  ] ETA:   0:02:14  95.7 KiB/s
Downloading:   7% [##                                                  ] ETA:   0:02:13  96.2 KiB/s
Downloading:   7% [##                                                  ] ETA:   0:02:06 100.8 KiB/s
Downloading:   8% [##                                                  ] ETA:   0:02:05 101.4 KiB/s
Downloading:   8% [##                                                  ] ETA:   0:01:59 106.1 KiB/s
Downloading:   8% [##                                                  ] ETA:   0:01:58 106.8 KiB/s
```

```
Downloading:   9% [##                              ] ETA:   0:01:53 111.3 KiB/s
Downloading:   9% [##                              ] ETA:   0:01:52 111.4 KiB/s
Downloading:   9% [###                             ] ETA:   0:01:46 117.3 KiB/s
Downloading:  10% [###                             ] ETA:   0:01:45 117.8 KiB/s
Downloading:  10% [###                             ] ETA:   0:01:39 124.0 KiB/s
Downloading:  10% [###                             ] ETA:   0:01:38 124.9 KiB/s
Downloading:  11% [###                             ] ETA:   0:01:33 131.3 KiB/s
Downloading:  11% [###                             ] ETA:   0:01:32 131.9 KiB/s
Downloading:  12% [###                             ] ETA:   0:01:27 139.0 KiB/s
Downloading:  12% [####                            ] ETA:   0:01:26 139.7 KiB/s
Downloading:  13% [####                            ] ETA:   0:01:21 146.7 KiB/s
Downloading:  13% [####                            ] ETA:   0:01:18 152.7 KiB/s
Downloading:  14% [####                            ] ETA:   0:01:16 154.5 KiB/s
Downloading:  15% [####                            ] ETA:   0:01:11 163.5 KiB/s
Downloading:  15% [####                            ] ETA:   0:01:12 162.6 KiB/s
Downloading:  16% [#####                           ] ETA:   0:01:07 170.8 KiB/s
Downloading:  17% [#####                           ] ETA:   0:01:03 180.1 KiB/s
Downloading:  18% [######                          ] ETA:   0:00:59 189.5 KiB/s
Downloading:  19% [######                          ] ETA:   0:00:58 192.1 KiB/s
Downloading:  20% [######                          ] ETA:   0:00:55 198.7 KiB/s
Downloading:  20% [######                          ] ETA:   0:00:55 199.5 KiB/s
Downloading:  21% [######                          ] ETA:   0:00:52 207.9 KiB/s
Downloading:  21% [######                          ] ETA:   0:00:52 208.0 KiB/s
Downloading:  22% [#######                         ] ETA:   0:00:48 218.6 KiB/s
Downloading:  22% [#######                         ] ETA:   0:00:49 215.6 KiB/s
Downloading:  24% [#######                         ] ETA:   0:00:46 224.9 KiB/s
Downloading:  25% [########                        ] ETA:   0:00:44 233.9 KiB/s
Downloading:  26% [########                        ] ETA:   0:00:41 242.9 KiB/s
Downloading:  28% [#########                       ] ETA:   0:00:39 250.9 KiB/s
Downloading:  29% [#########                       ] ETA:   0:00:37 259.9 KiB/s
Downloading:  31% [#########                       ] ETA:   0:00:35 268.5 KiB/s
Downloading:  32% [##########                      ] ETA:   0:00:33 277.3 KiB/s
Downloading:  34% [##########                      ] ETA:   0:00:31 286.4 KiB/s
Downloading:  35% [###########                     ] ETA:   0:00:30 295.8 KiB/s
Downloading:  35% [###########                     ] ETA:   0:00:30 295.4 KiB/s
Downloading:  36% [###########                     ] ETA:   0:00:29 298.8 KiB/s
Downloading:  37% [############                    ] ETA:   0:00:28 305.8 KiB/s
Downloading:  38% [############                    ] ETA:   0:00:27 308.5 KiB/s
Downloading:  39% [############                    ] ETA:   0:00:26 315.9 KiB/s
Downloading:  39% [############                    ] ETA:   0:00:26 318.1 KiB/s
Downloading:  41% [#############                   ] ETA:   0:00:25 325.2 KiB/s
Downloading:  41% [#############                   ] ETA:   0:00:24 328.1 KiB/s
Downloading:  42% [#############                   ] ETA:   0:00:23 333.2 KiB/s
```

```
Downloading:  43% [#############                   ] ETA:   0:00:23 336.6 KiB/s
Downloading:  45% [##############                  ] ETA:   0:00:21 347.1 KiB/s
Downloading:  46% [##############                  ] ETA:   0:00:20 355.8 KiB/s
Downloading:  47% [###############                 ] ETA:   0:00:20 357.9 KiB/s
Downloading:  49% [###############                 ] ETA:   0:00:18 370.3 KiB/s
Downloading:  49% [###############                 ] ETA:   0:00:18 369.1 KiB/s
Downloading:  51% [################                ] ETA:   0:00:17 380.5 KiB/s
Downloading:  53% [#################               ] ETA:   0:00:16 389.4 KiB/s
Downloading:  53% [#################               ] ETA:   0:00:16 392.2 KiB/s
Downloading:  55% [#################               ] ETA:   0:00:15 403.1 KiB/s
Downloading:  56% [#################               ] ETA:   0:00:15 403.9 KiB/s
Downloading:  56% [##################              ] ETA:   0:00:14 408.3 KiB/s
Downloading:  58% [##################              ] ETA:   0:00:13 416.2 KiB/s
Downloading:  59% [###################             ] ETA:   0:00:13 421.4 KiB/s
Downloading:  61% [###################             ] ETA:   0:00:12 429.2 KiB/s
Downloading:  62% [###################             ] ETA:   0:00:12 434.6 KiB/s
Downloading:  63% [####################            ] ETA:   0:00:11 442.5 KiB/s
Downloading:  64% [####################            ] ETA:   0:00:10 447.5 KiB/s
Downloading:  66% [#####################           ] ETA:   0:00:10 456.3 KiB/s
Downloading:  67% [#####################           ] ETA:   0:00:09 459.6 KiB/s
Downloading:  69% [######################          ] ETA:   0:00:09 469.8 KiB/s
Downloading:  69% [######################          ] ETA:   0:00:08 471.9 KiB/s
Downloading:  71% [#######################         ] ETA:   0:00:08 484.1 KiB/s
Downloading:  72% [#######################         ] ETA:   0:00:07 484.3 KiB/s
Downloading:  74% [#######################         ] ETA:   0:00:07 499.3 KiB/s
Downloading:  75% [########################        ] ETA:   0:00:06 498.8 KiB/s
Downloading:  75% [########################        ] ETA:   0:00:06 503.6 KiB/s
Downloading:  78% [#########################       ] ETA:   0:00:05 514.8 KiB/s
Downloading:  78% [#########################       ] ETA:   0:00:05 515.7 KiB/s
Downloading:  79% [#########################       ] ETA:   0:00:05 522.2 KiB/s
Downloading:  81% [##########################      ] ETA:   0:00:04 529.8 KiB/s
Downloading:  83% [##########################      ] ETA:   0:00:04 541.1 KiB/s
Downloading:  84% [###########################     ] ETA:   0:00:03 547.3 KiB/s
Downloading:  86% [###########################     ] ETA:   0:00:03 552.2 KiB/s
Downloading:  87% [###########################     ] ETA:   0:00:03 559.2 KiB/s
Downloading:  88% [############################    ] ETA:   0:00:02 563.3 KiB/s
Downloading:  89% [############################    ] ETA:   0:00:02 570.2 KiB/s
Downloading:  92% [#############################   ] ETA:   0:00:01 580.3 KiB/s
Downloading:  93% [#############################   ] ETA:   0:00:01 588.3 KiB/s
Downloading:  95% [##############################  ] ETA:   0:00:00 598.3 KiB/s
Downloading:  97% [############################### ] ETA:   0:00:00 604.1 KiB/s
Downloading:  99% [############################### ] ETA:   0:00:00 616.4 KiB/s
```

```
# Read column names from another file
column_names = pd.read_csv('HEADERS.txt',
                           header=None,
                           delim_whitespace=True,
                           )
# Read CSV file using column names from another file
df = pd.read_csv("CRNS0101-05-2022-TX_Austin_33_NW.txt",  # file to read
                 delim_whitespace=True,  # use (any number of) white spaces as delimiter betw
                 names=column_names.iloc[1],  # column names from row i=1 of "column_names"
                 na_values=[-99, -9999, -99999],  # substitute these values by NaN
                 )
# make integer column LST_DATE as string
df['LST_DATE'] = df['LST_DATE'].astype(str)#.apply(lambda x: f'{x:0>4}')
# make integer column LST_DATE as string
# pad numbers with 0 from the left, such that 15 becomes 0015
df['LST_TIME'] = df['LST_TIME'].apply(lambda x: f'{x:0>4}')
# combine both DATE and TIME
df['datetime'] = pd.to_datetime(df['LST_DATE'] + df['LST_TIME'], format='%Y%m%d%H%M')
df = df.set_index('datetime')
df
```

	WBANNO	UTC_DATE	UTC_TIME	LST_DATE	LST_TIME	CRX_VN	LO
datetime							
2021-12-31 18:05:00	23907	20220101	5	20211231	1805	2.623	-9
2021-12-31 18:10:00	23907	20220101	10	20211231	1810	2.623	-9
2021-12-31 18:15:00	23907	20220101	15	20211231	1815	2.623	-9
2021-12-31 18:20:00	23907	20220101	20	20211231	1820	2.623	-9
2021-12-31 18:25:00	23907	20220101	25	20211231	1825	2.623	-9
...	...	...	...	...	...	...	...
2022-12-31 17:40:00	23907	20221231	2340	20221231	1740	2.623	-9
2022-12-31 17:45:00	23907	20221231	2345	20221231	1745	2.623	-9
2022-12-31 17:50:00	23907	20221231	2350	20221231	1750	2.623	-9
2022-12-31 17:55:00	23907	20221231	2355	20221231	1755	2.623	-9
2022-12-31 18:00:00	23907	20230101	0	20221231	1800	2.623	-9

The provided data is not the same as what is provided by the IMS.

- Now we don't have air pressure values, so we need to provide elevation.
- We do have R_n (net radiation), so there is no need to provide latitude.

Attention! According to the headers file, net radiation provided by NOAA is in W m^{-2}, but pyet requires it to be MJ m^{-2} d^{-1}. We need to agreggate the downloaded data into daily radiation.

Data comes in 5-minute intervals, so if we have a value x W m^{-2} over a 5-minute interval, the total amount of energy is:

$$x\frac{W}{m^{-2}} \times 5\,min = x\frac{J}{m^{-2}\cdot s} \times 5 \cdot 60\,s = x \cdot 5 \cdot 60\frac{J}{m^{-2}}$$

Let's call $X = \sum_{day} x$. Then, summing all energy in 1 day:

$$\sum_{day} x \cdot 5 \cdot 60\frac{J}{m^{-2}\cdot day} = \sum_{day} x \cdot 5 \cdot 60 \cdot 10^{-6}\frac{MJ}{m^{-2}\cdot day}$$

Make a new dataframe with daily means for temperature, relative humidity and wind speed.

```
df_TX = df[['AIR_TEMPERATURE',
            'RELATIVE_HUMIDITY',
            'WIND_1_5']].resample('D').mean()
df_TX
```

	AIR_TEMPERATURE	RELATIVE_HUMIDITY	WIND_1_5
datetime			
2021-12-31	21.721127	79.985915	2.113662
2022-01-01	18.336806	65.833333	2.586840
2022-01-02	-0.222222	46.187500	3.849757
2022-01-03	4.592708	36.927083	0.892778
2022-01-04	9.964583	41.996528	2.428924
...	...	...	...
2022-12-27	6.534375	50.871528	1.881597
2022-12-28	14.418056	65.090278	3.289167
2022-12-29	16.878125	72.934028	2.068750
2022-12-30	13.047917	51.006944	1.180729
2022-12-31	15.010138	58.387097	2.590276

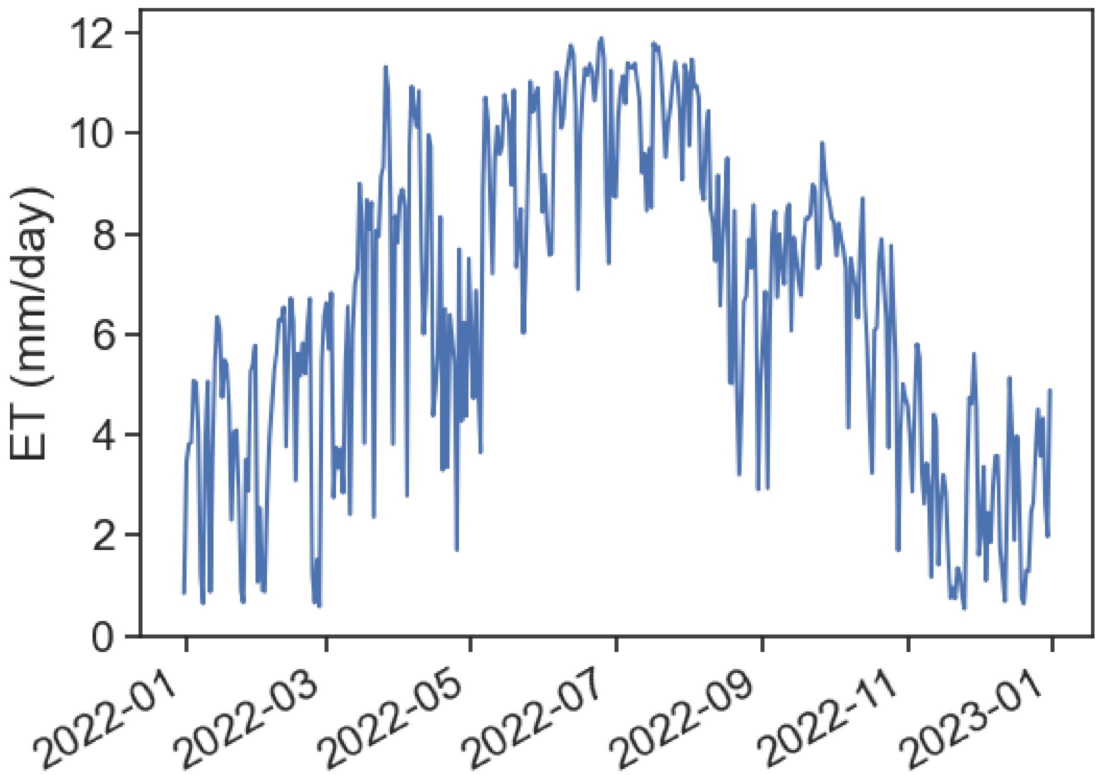
```
X = df['SOLAR_RADIATION'].resample('D').sum()
df_TX['SOLAR_RADIATION'] = X * 5 * 60 * 1e-6
df_TX
```

	AIR_TEMPERATURE	RELATIVE_HUMIDITY	WIND_1_5	SOLAR_RADIATION
datetime				
2021-12-31	21.721127	79.985915	2.113662	0.0000
2022-01-01	18.336806	65.833333	2.586840	7.2870
2022-01-02	-0.222222	46.187500	3.849757	14.2536
2022-01-03	4.592708	36.927083	0.892778	14.4309
2022-01-04	9.964583	41.996528	2.428924	14.2056
...	...	...	...	...
2022-12-27	6.534375	50.871528	1.881597	11.5200
2022-12-28	14.418056	65.090278	3.289167	10.9917
2022-12-29	16.878125	72.934028	2.068750	5.9829
2022-12-30	13.047917	51.006944	1.180729	2.9967
2022-12-31	15.010138	58.387097	2.590276	12.7248

```python
tmean = df_TX["AIR_TEMPERATURE"]
wind = df_TX["WIND_1_5"]
rh = df_TX["RELATIVE_HUMIDITY"]
rn = df_TX["SOLAR_RADIATION"]
elevation = 358
penm2 = pyet.combination.penman(tmean=tmean,
                                wind=wind,
                                elevation=elevation,
                                rh=rh,
                                rn=rn,
                                )
```

```python
fig, ax = plt.subplots(1)
ax.plot(penm2, label="penman")
# ax.legend()
plt.gcf().autofmt_xdate()  # makes slated dates
ax.set_ylabel("ET (mm/day)")
```

```
Text(0, 0.5, 'ET (mm/day)')
```



Does this make sense? How can we know?

Part IV

Infiltration

Here are some of the files we'll use in this module, in case you can't download them from their original repositories.

- input rate 78 mm/h, 16% slope
- input rate 156 mm/h, 16% slope
- input rate 234 mm/h, 16% slope
- input rate 312 mm/h, 16% slope

13 Infiltration

13.1 Definitions

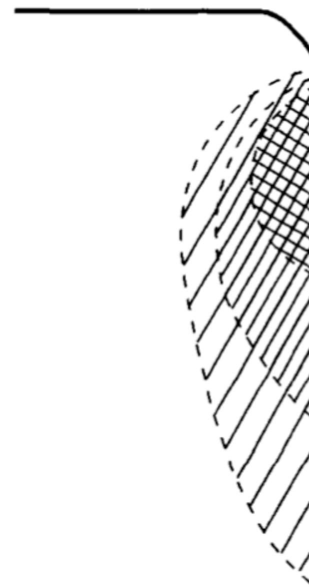
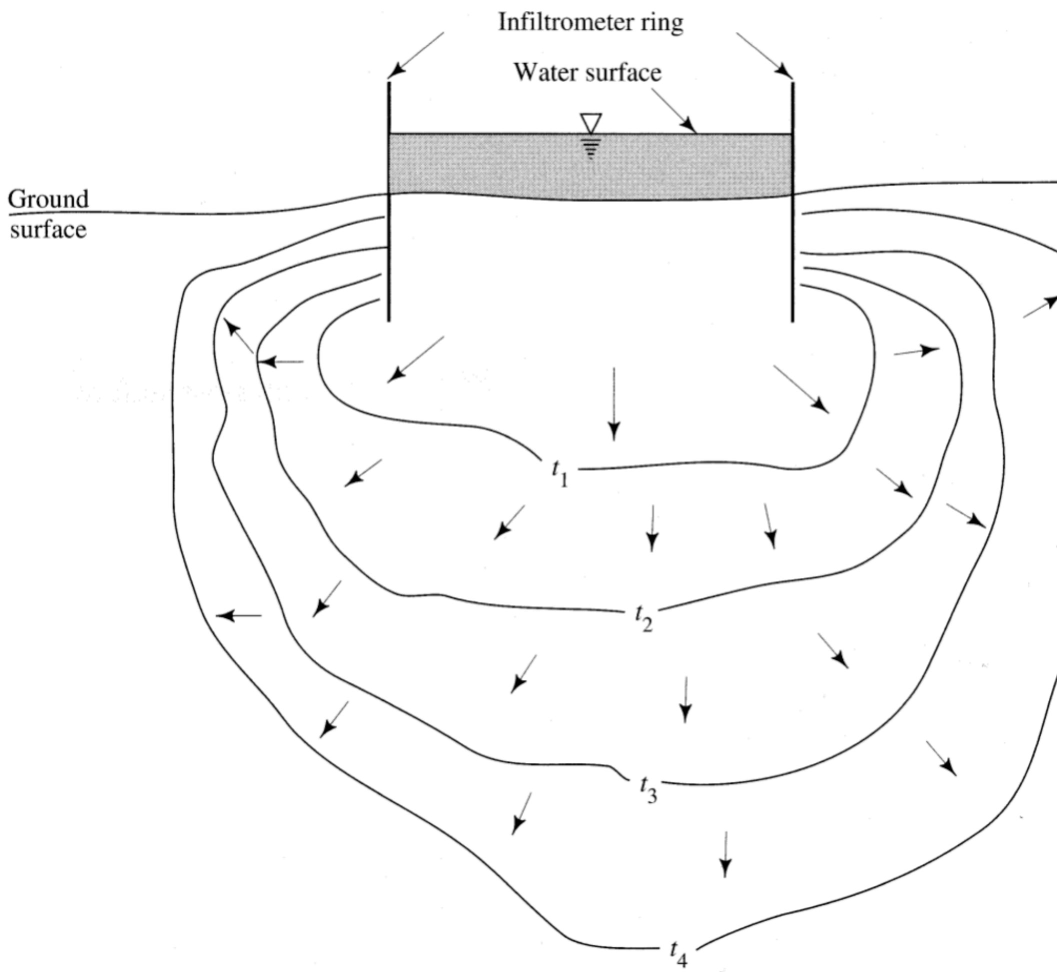
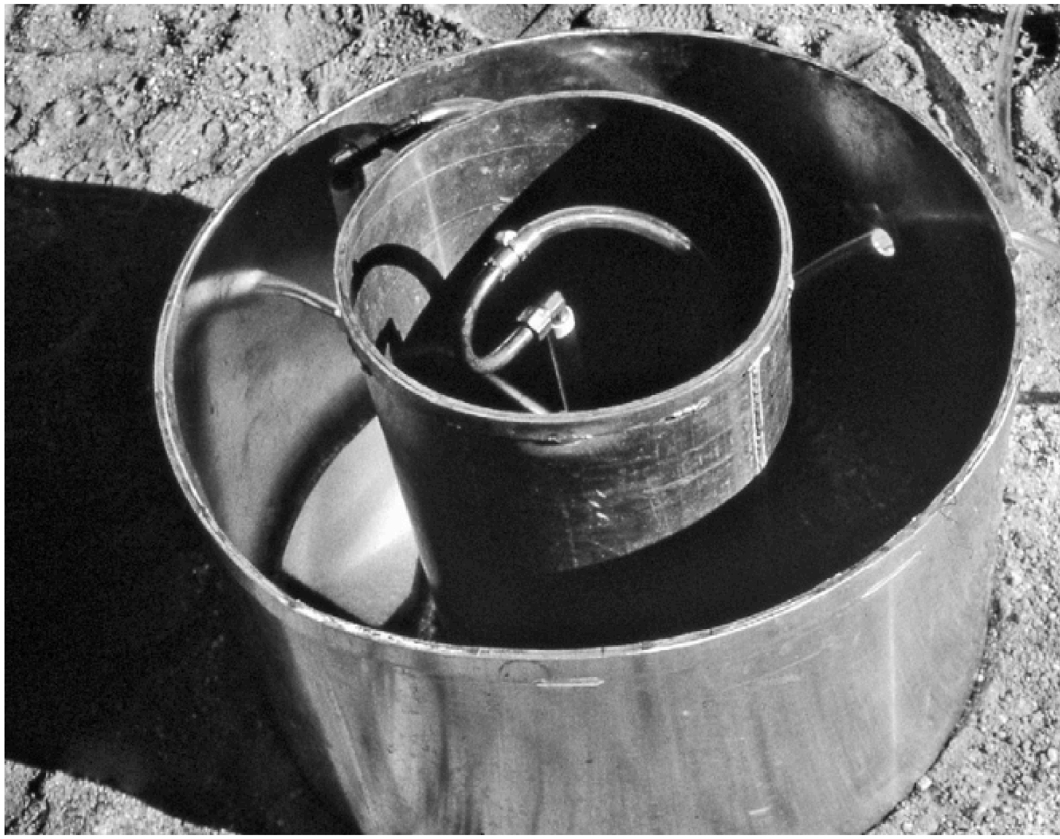


Fig. 14.6. Infiltration from an irrigation
shown after different periods of time (t_1<
infiltration to be nearly uniform in all directio
gradients diminish and the gravitational gra

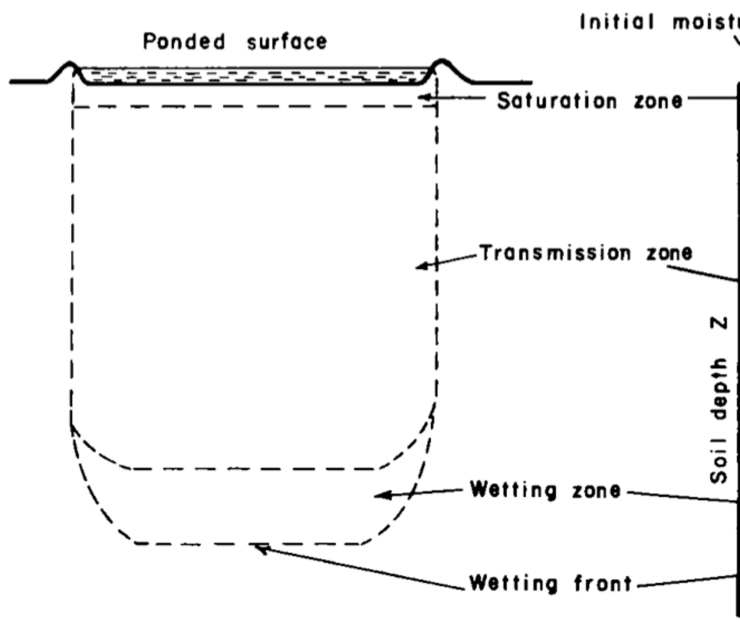
Hillel (2003), Introduction to Environmental Soil Physics, figure 14.6



Dingman (2015), figure 8.13



Dingman (2015), figure 8.14

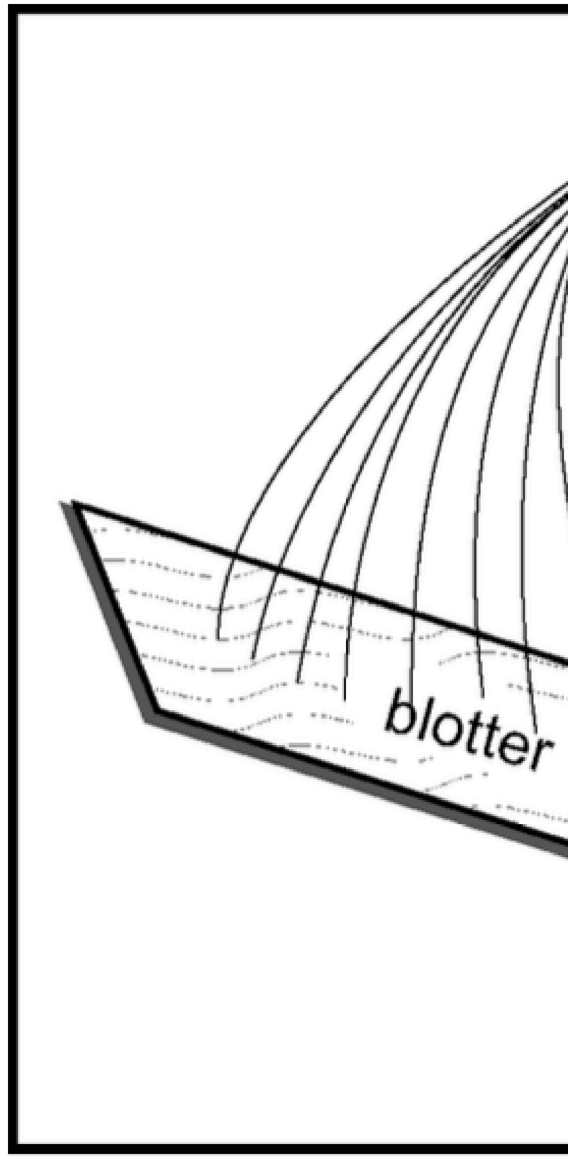


Hillel (2003), Introduction to Soil Physics, figure 12.3

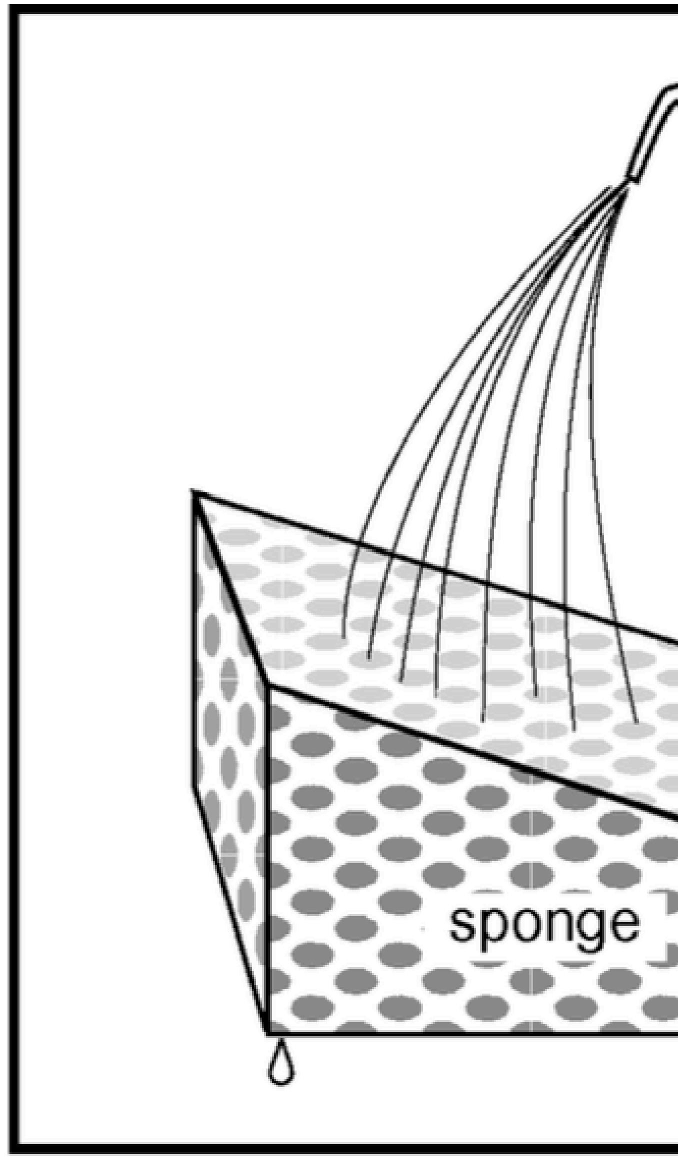
Source: Dingman (2015), page 355

- The **water-input rate**, $w(t)$ [L T^{-1}], is the rate at which water arrives at the surface due to rain, snowmelt, or irrigation. A water-input event begins at time $t = 0$ and ends at $t = T_w$.
- The **infiltration rate**, $f(t)$ [L T^{-1}], is the rate at which water enters the soil from the surface.
- The **infiltrability**, also called **infiltration capacity**, $f^*(t)$ [L T^{-1}], is the maximum rate at which infiltration could occur at any time; note that this changes during the infiltration event.
- The **depth of ponding**, $H(t)$ [L], is the depth of water standing on the surface.

Source: Ward and Trimble (2003), page 63, 64

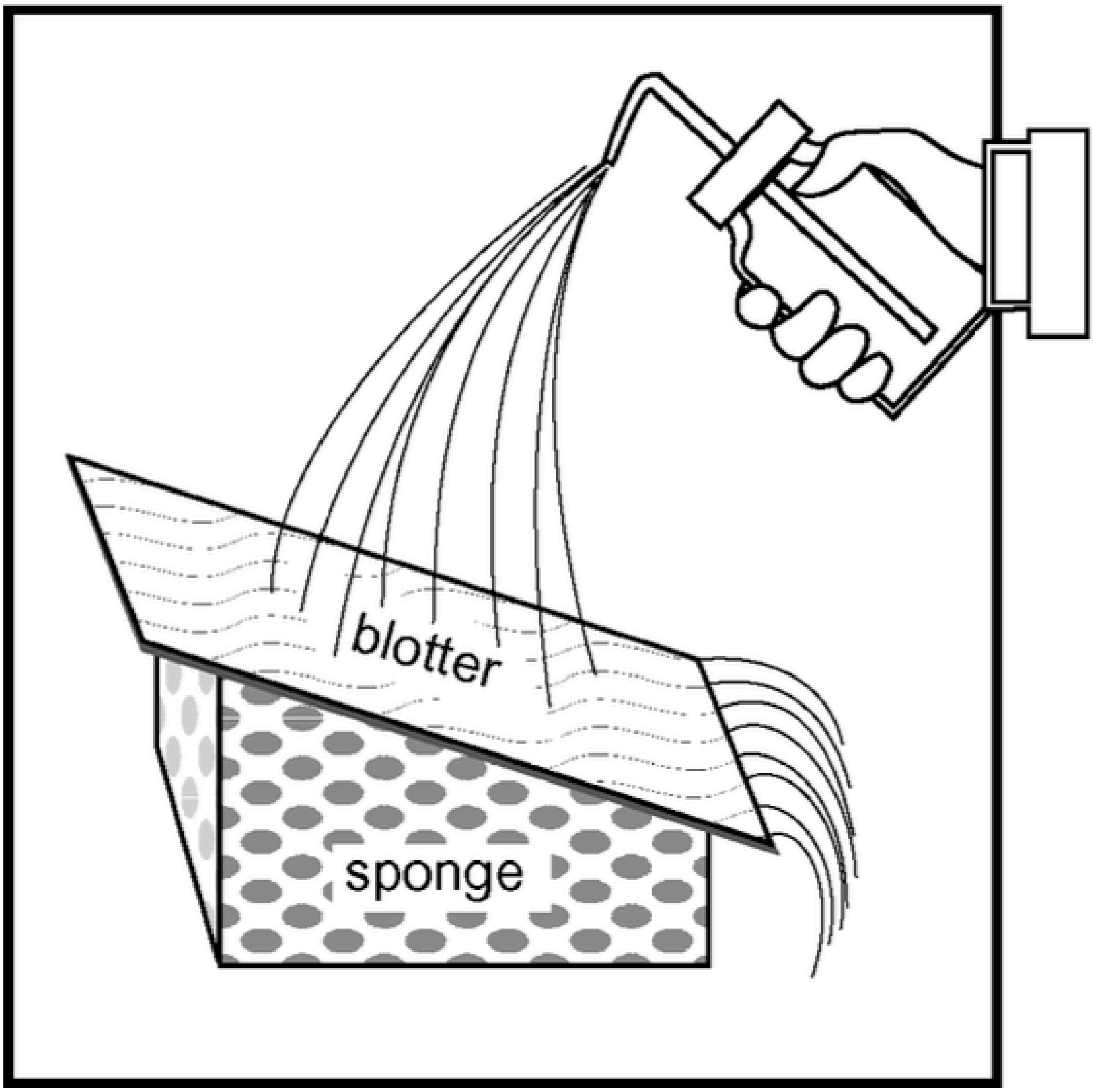


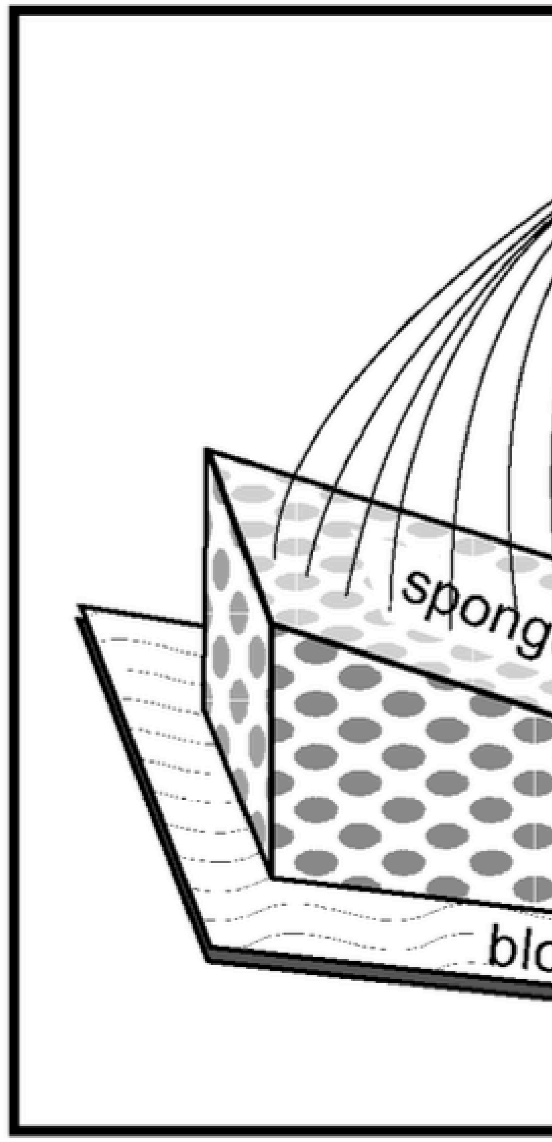
Infiltration capacity of absorbent paper is low, there is much runoff



Infiltration capacity of sponge is high, there is little runoff

Infiltration capacity of the sponge is limited by the overlying layer with low permeability


blotter
sponge



Infiltration capacity of the sponge is limited by the underlying layer

https://youtu.be/ego2FkuQwxc

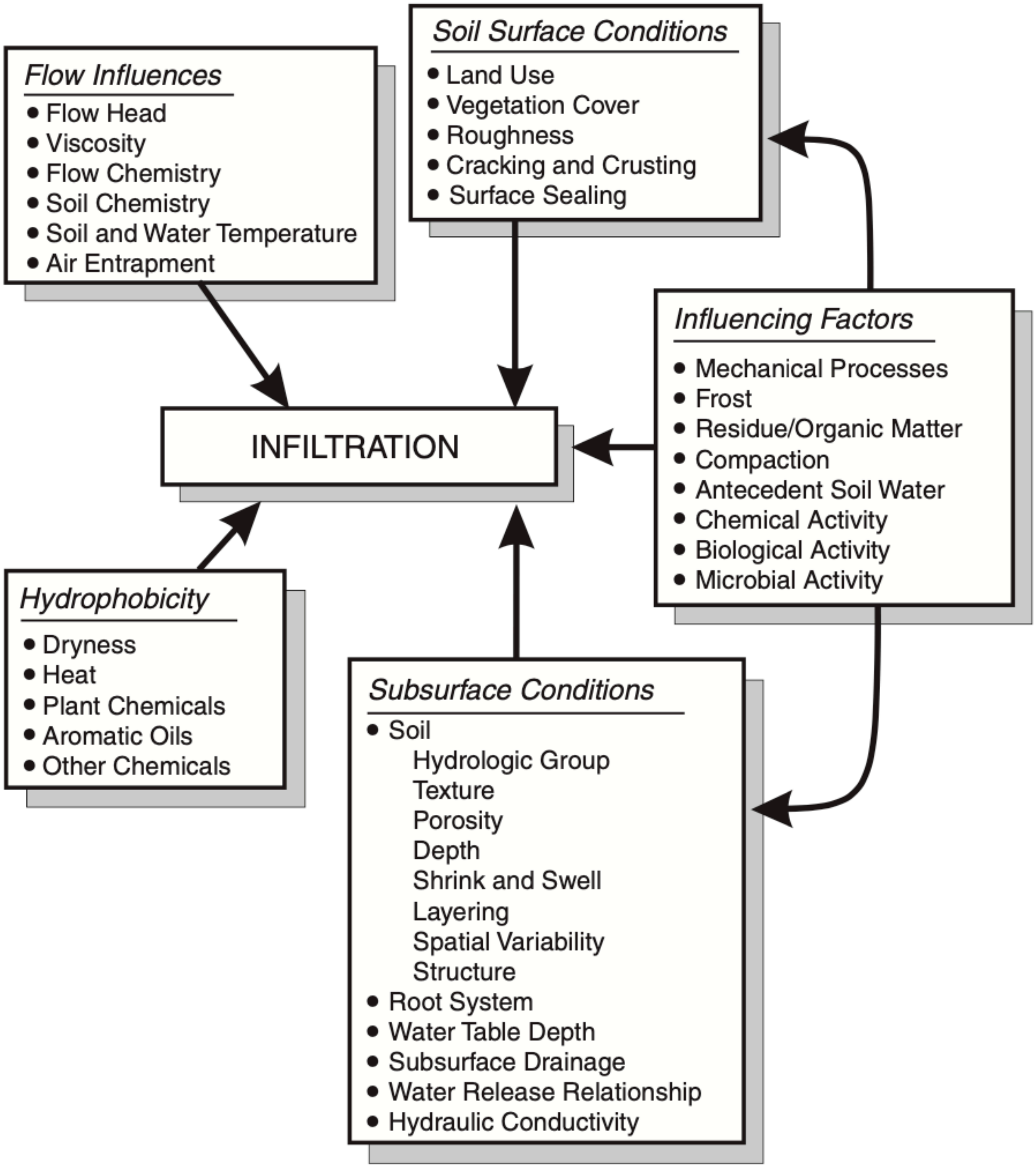


Figure 13.1: Ward and Trimble (2003), page 65

13.2 Darcy

Darcy's equation for vertical flow

$$q = -K\frac{\partial H_{\text{total}}}{\partial z}$$

where the total head $H_{\text{total}} = -H_{\text{suction}} - z_{\text{depth}}$, and

- H_{suction} is the suction head (negative pressure head)
- z_{depth} is the depth, points *downward*.

Substituting:

$$q = K\frac{\partial H_{\text{suction}}}{\partial z} + K$$

Substituting the above into the continuity equation

$$\frac{\partial \theta}{\partial t} = \frac{\partial q}{\partial z}$$

yields the Richards equation.

13.3 Richards

Richards equation:

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial z}\left[K(\theta)\frac{\partial H_{\text{total}}}{\partial z}\right]$$

Substituting $H_{\text{total}} = -H_{\text{suction}} - z_{\text{depth}}$ yields:

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial z}\left[K(\theta)\left(\frac{\partial(-H_{\text{suction}} - z)}{\partial z}\right)\right]$$

$$\frac{\partial \theta}{\partial t} = -\underbrace{\frac{\partial}{\partial z}\left(K(\theta)\frac{\partial H_{\text{suction}}}{\partial z}\right)}_{\text{matric}} - \underbrace{\frac{\partial K(\theta)}{\partial z}}_{\text{gravitational}}$$

13.3.1 short times

As the water starts to enter the relatively dry soil, the pressure differences in the water at the surface and in the soil are quite large and, as a result, the second term on the right is practically negligible compared to the first one.

$$\frac{\partial \theta}{\partial t} = -\frac{\partial}{\partial z}\left(K(\theta)\frac{\partial H}{\partial z}\right)$$

13.3.2 long times

As illustrated in the figure below (Davidson et al., 1963), after longer times of infiltration, the water content profile near the surface gradually becomes more uniform and it eventually assumes the satiation value, or $\theta \to \theta_0$; similarly, the pressure in the upper layers of the soil becomes gradually atmospheric, or $H \to 0$. Hence, their vertical gradients

$$\frac{\partial \theta}{\partial z} \text{ and } \frac{\partial H_{\text{suction}}}{\partial z} \longrightarrow 0$$

From Darcy's equation we have that

$$q = K(\theta_0) = K_{\text{sat}}$$

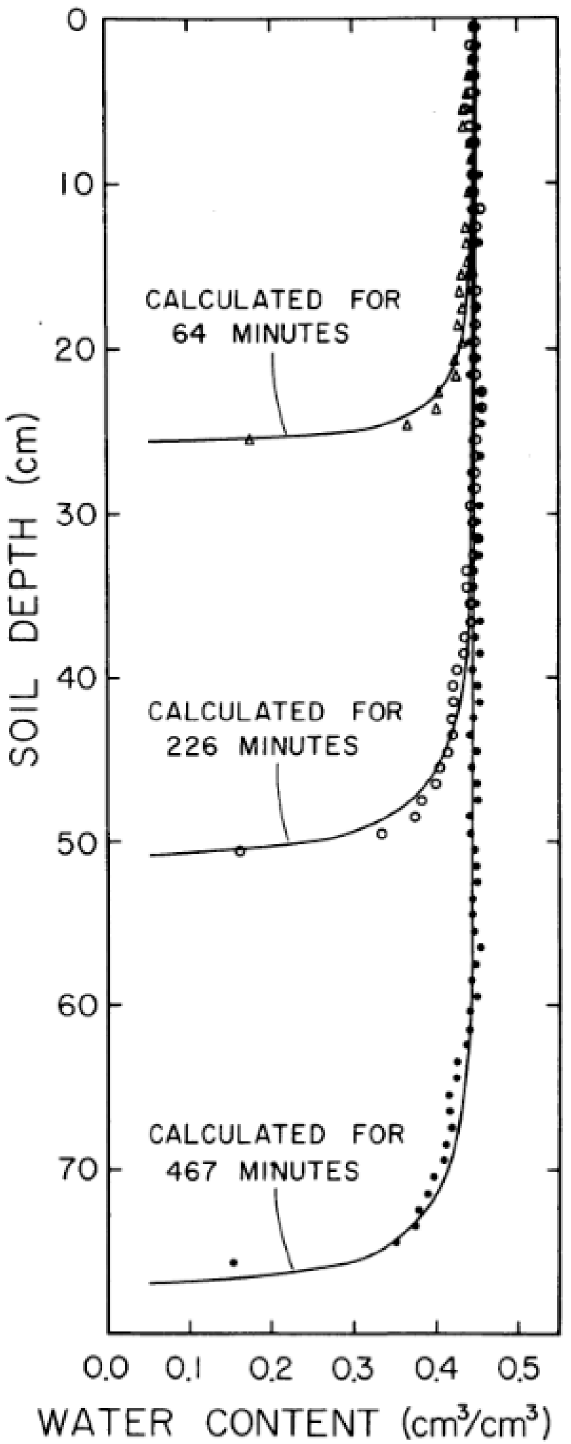
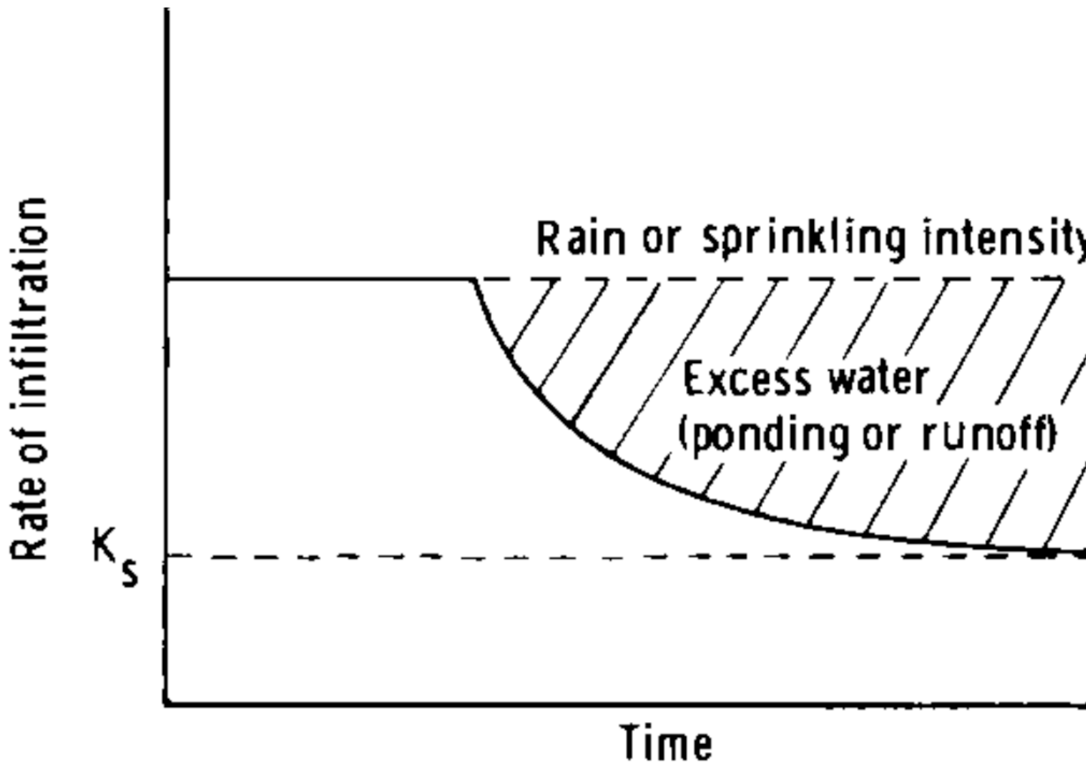


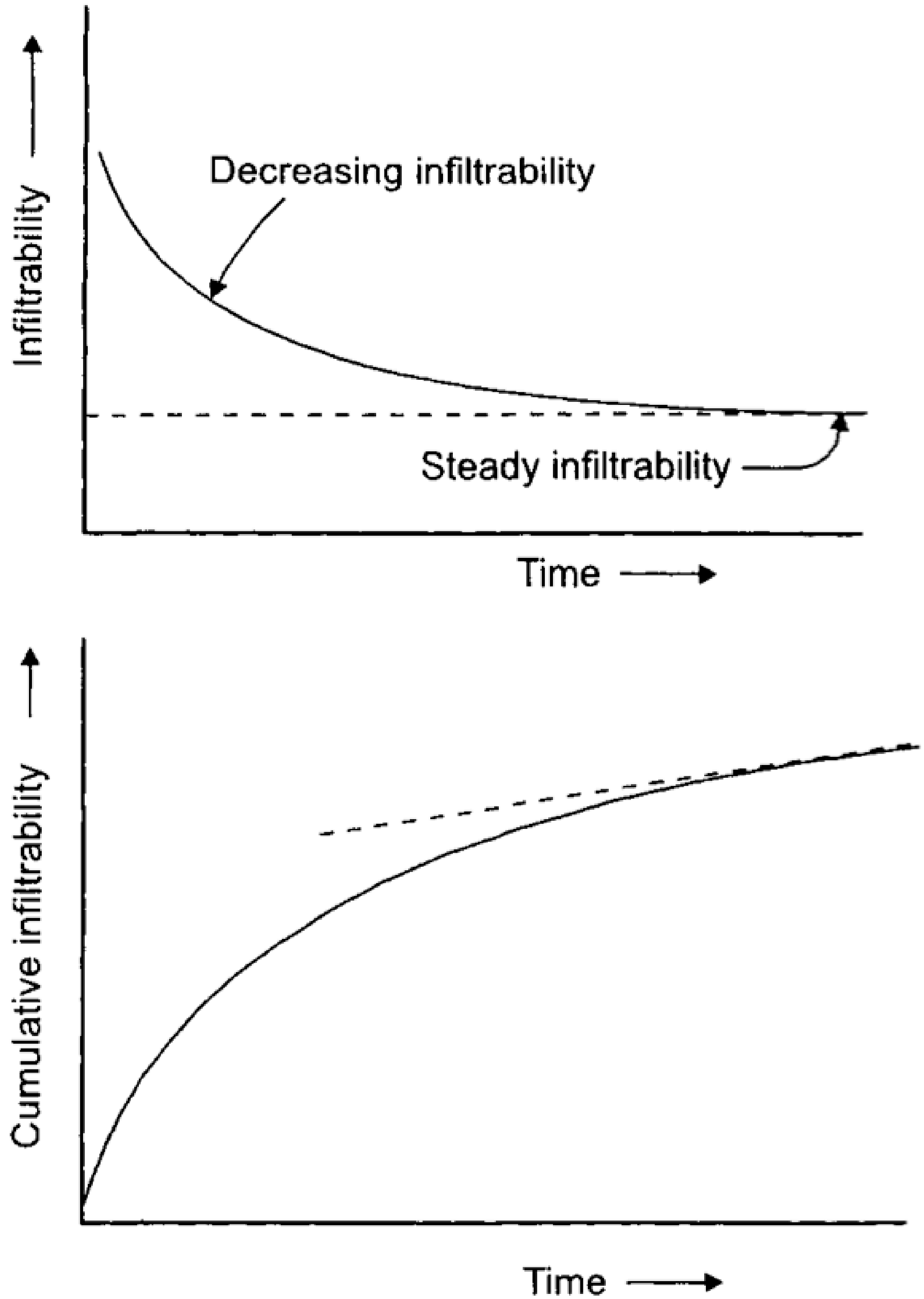
Fig. 19. Calculated and measured soil water profiles for air-dry Columbia soil allowed to wet at $\theta_o = 0.45\ cm^3/cm^3$.

13.3.3 Rainfall infiltration

Infiltration rate is equal to rainfal rate, at least at first. If rainfall rate w is lower than K_{sat}, than everything enters the soil, i.e., $f = K_{\text{sat}}$. However, if $w > K_{\text{sat}}$, water content θ will increase at the surface, until it reaches θ_0, and at that moment, called ponding time t_p, water will begin to accumulate at the surface.



Hillel (2003), figure 12.1



Hillel (2003), figure 12.2

13.4 Horton equation

One of the most widely used models, developed by R.E. Horton (1939), considered to be the father of modern hydrology.

$$f = f_c + (f_0 - f_c)e^{-\beta t}$$

- f: infiltration rate
- f_c: infiltration capacity at large t
- f_0: initial infiltration capacity
- β: best fit empirical parameter

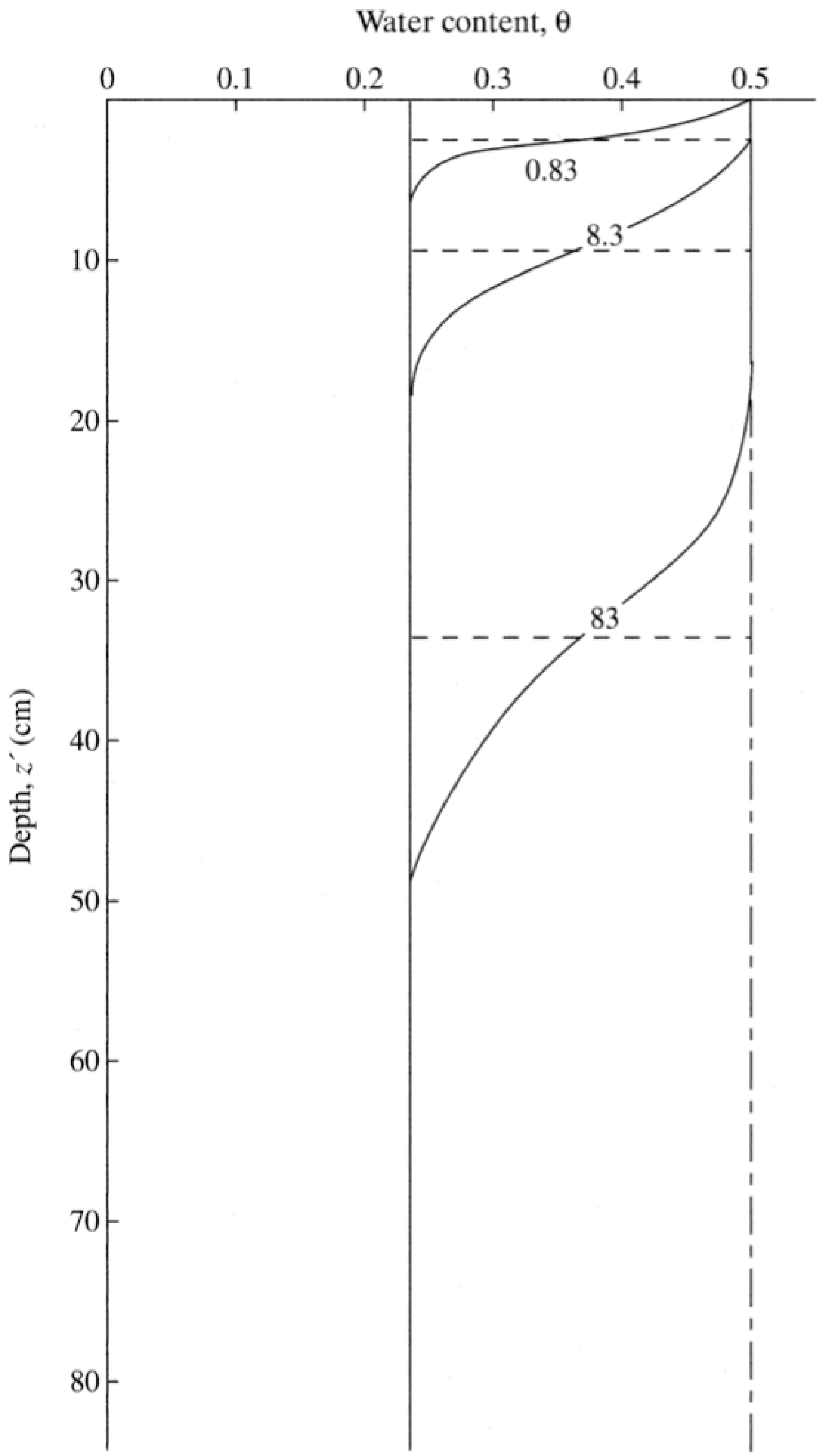
Advantages

- Simple equation
- Usually gives good fit to measured data because it is dependent on three parameters

Disadvantages

- This method has no physical significance, it is not based on any water transport mechanism
- Does not describe infiltration prior to ponding

13.5 Green & Ampt



Dingman (2015), figure 8.11

Assumptions:

- homogeneous soil, infinite depth (no water table)
- horizontal surface
- constant water head equal to zero is maintained at the surface
- uniform water content prior to wetting, $\theta(t=0,z)=\theta_0$
- moving front is characterized by a constant matric suction, ψ_f

Source: Dingman (2015), page 370

This equation was developed under the scenario of constant rainfall or irrigation on an initially dry soil as a sharp wetting front (such as piston flow). Water penetrates a dry soil with a certain initial moisture content, and wets the layer to a saturated moisture content as it traverses deeper. The connection between soil moisture and infiltration rate is modeled in the Green-Ampt equation:

$$f(t) = K_\text{sat}\left[1 + \frac{|\psi_f|\cdot(\phi-\theta_0)}{F(t)}\right]$$

- $f(t)$: infiltration rate
- $F(t)$: cumulative infiltration rate, $F=\int f\ \text{d}t$
- ψ_f: effective wetting-front suction
- ϕ: soil porosity
- θ_0: initial soil water content

The same equation can be simply be rewritten as

$$f = \frac{A}{F} + B$$

where

- $A = K_\text{sat}\cdot|\psi_f|\cdot(\phi-\theta_0)$
- $B = K_\text{sat}$

The porosity ϕ and the saturated hydraulic conductivity K_sat can be estimated from the soil texture. The wetting-front suction ψ_f can be estimated using the Brooks-Corey parameters:

$$|\psi_f| = \frac{2b+3}{2b+6}\cdot|\psi_{ae}|,$$

where ψ_{ae} is the air-entry pressure head. Values for the parameters above can be found in this table:

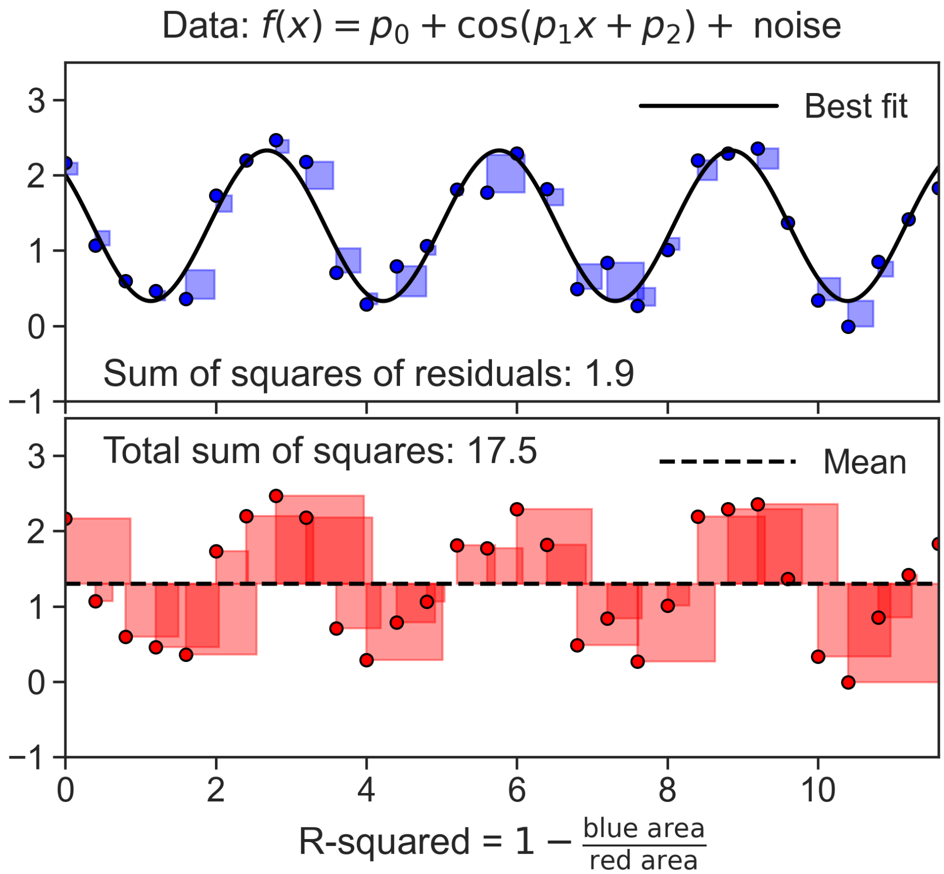
Table 7.4 Brooks–Corey and Campbell Parameters (Table 7.2) for Various Soil Textures Based on Analysis of 1845 Soils.[a]

Soil Texture	ϕ	K_h **(cm/s)**	$\|\psi_{ae}\|$ **(cm)**	b
Sand	0.395 (0.056)	1.76×10^{-2}	12.1 (14.3)	4.05 (1.78)
Loamy sand	0.410 (0.068)	1.56×10^{-2}	9.0 (12.4)	4.38 (1.47)
Sandy loam	0.435 (0.086)	3.47×10^{-3}	21.8 (31.0)	4.90 (1.75)
Silt loam	0.485 (0.059)	7.20×10^{-4}	78.6 (51.2)	5.30 (1.96)
Loam	0.451 (0.078)	6.95×10^{-4}	47.8 (51.2)	5.39 (1.87)
Sandy clay loam	0.420 (0.059)	6.30×10^{-4}	29.9 (37.8)	7.12 (2.43)
Silty clay loam	0.477 (0.057)	1.70×10^{-4}	35.6 (37.8)	7.75 (2.77)
Clay loam	0.476 (0.053)	2.45×10^{-4}	63.0 (51.0)	8.52 (3.44)
Sandy clay	0.426 (0.057)	2.17×10^{-4}	15.3 (17.3)	10.4 (1.64)
Silty clay	0.492 (0.064)	1.03×10^{-4}	49.0 (62.1)	10.4 (4.45)
Clay	0.482 (0.050)	1.28×10^{-4}	40.5 (39.7)	11.4 (3.70)

[a]Values in parentheses are standard deviations.

Source: Data from Clapp and Hornberger (1978).

13.6 Best Fit, Least Squares Method



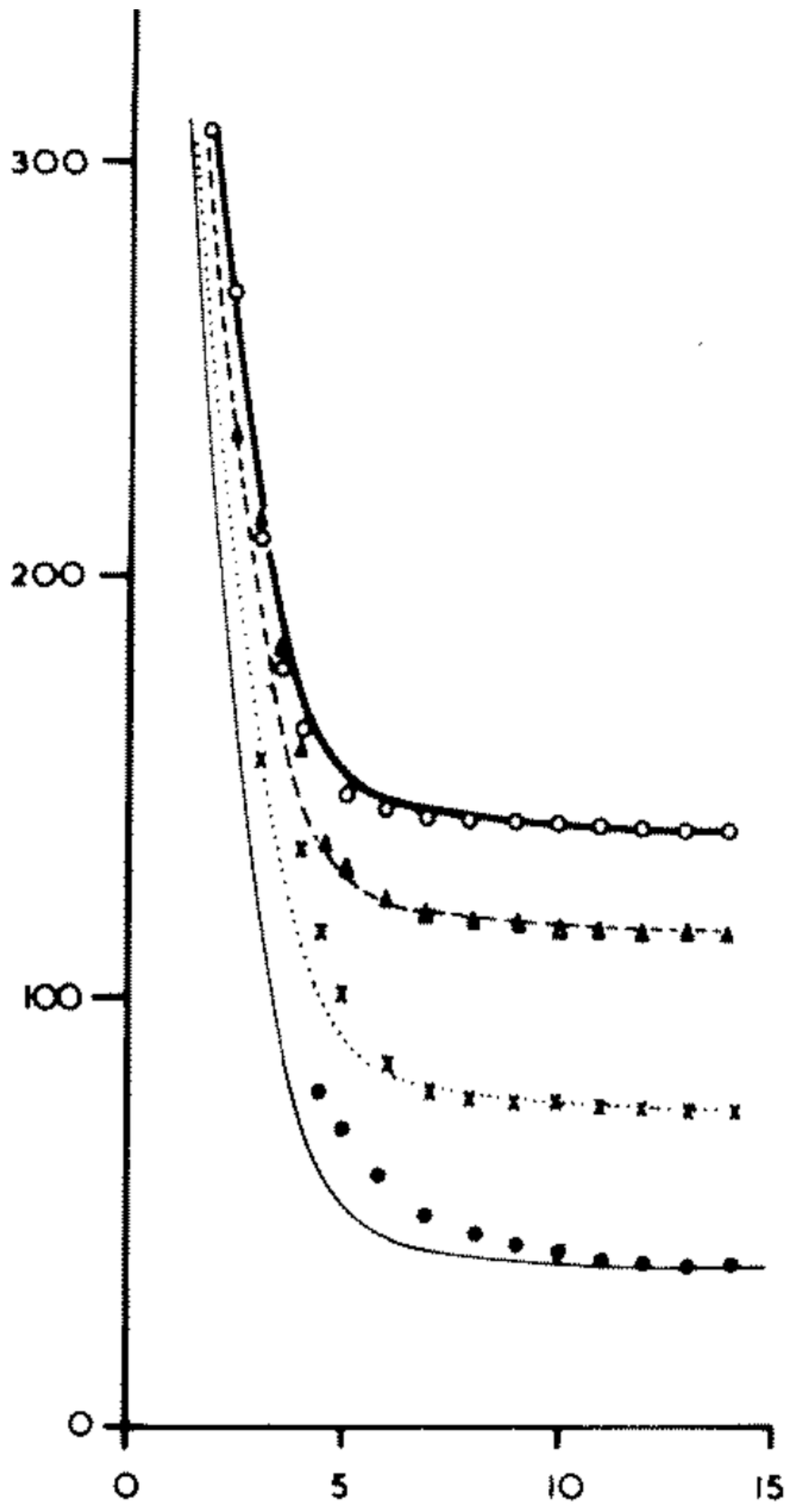
14 Exercises

14.1 Tasks

1. Google the following: web plot digitizer
2. Load image "nassif-16percent-slope.png" (see below)
3. Create four csv files, one for each data set. Call them whatever you want. Legend: white circle = 312 mm/h, triangle = 234 mm/h, x = 156 mm/h, black circle = 78 mm/h.

The image is the second panel of Fig. 8, from

> Nassif, S. H., and E. M. Wilson, 1975, "THE INFLUENCE OF SLOPE AND RAIN INTENSITY ON RUNOFF AND INFILTRATION", Hydrological Sciences Journal. download here


300
200
100
0
0
5
10
15

Import relevant packages

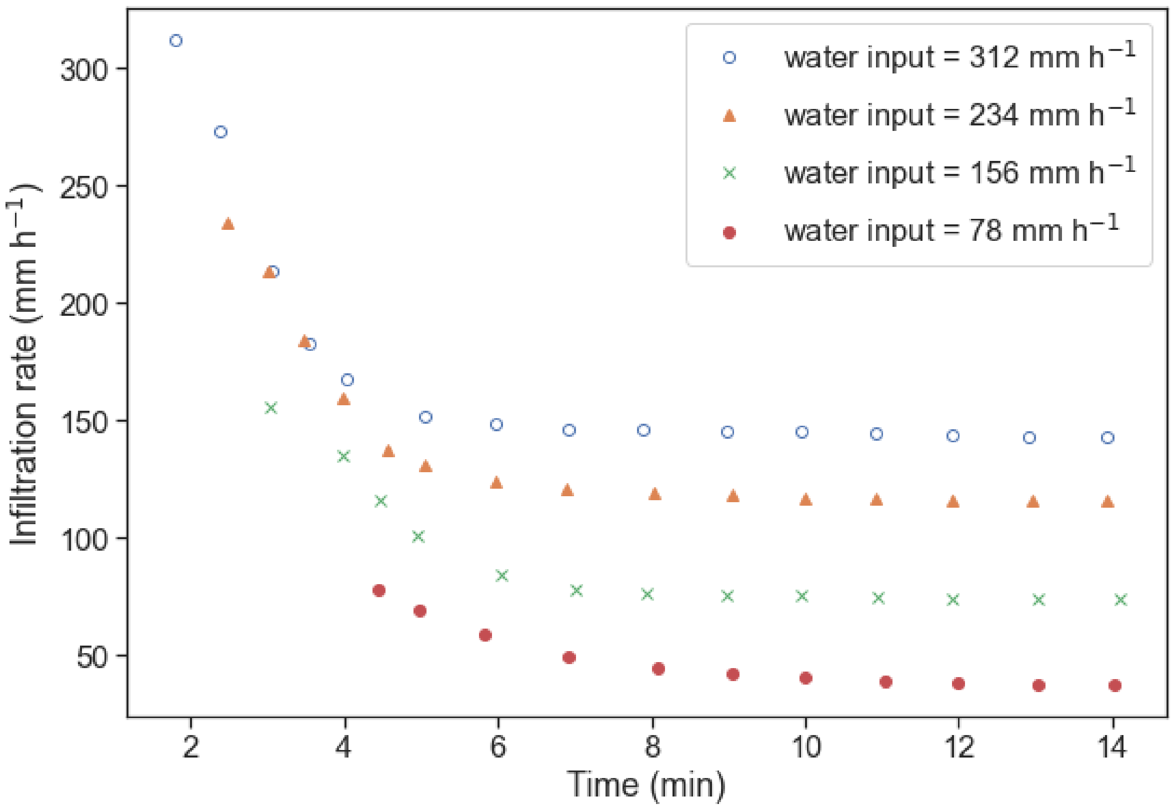
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
from scipy.optimize import curve_fit
import matplotlib.patches as patches
```

Load all four files you created. Use numpy's function `loadtxt`. Make sure that the first point in each table corresponds to the appropriate rainfall rate. You can normalize the data if it is not.

```
d1 = np.loadtxt("input_rate_078mm_per_h_16percent_slope.csv", delimiter=',')
d2 = np.loadtxt("input_rate_156mm_per_h_16percent_slope.csv", delimiter=',')
d3 = np.loadtxt("input_rate_234mm_per_h_16percent_slope.csv", delimiter=',')
d4 = np.loadtxt("input_rate_312mm_per_h_16percent_slope.csv", delimiter=',')
d1[:,1] = 78  * d1[:,1] / d1[:,1].max()
d2[:,1] = 156 * d2[:,1] / d2[:,1].max()
d3[:,1] = 234 * d3[:,1] / d3[:,1].max()
d4[:,1] = 312 * d4[:,1] / d4[:,1].max()
```

Reproduce the original figure, make it look good, something like this:

```
fig, ax = plt.subplots(figsize=(10,7))
ax.plot(d4[:,0], d4[:,1], 'o', markerfacecolor="None", label=r"water input = 312 mm h$^{-1}$"
ax.plot(d3[:,0], d3[:,1], '^', label=r"water input = 234 mm h$^{-1}$")
ax.plot(d2[:,0], d2[:,1], 'x', label=r"water input = 156 mm h$^{-1}$")
ax.plot(d1[:,0], d1[:,1], 'o', label=r"water input = 78 mm h$^{-1}$")
ax.set(xlabel="Time (min)",
       ylabel=r"Infiltration rate (mm h$^{-1}$)")
ax.legend(loc="upper right");
```



14.2 Horton's equation

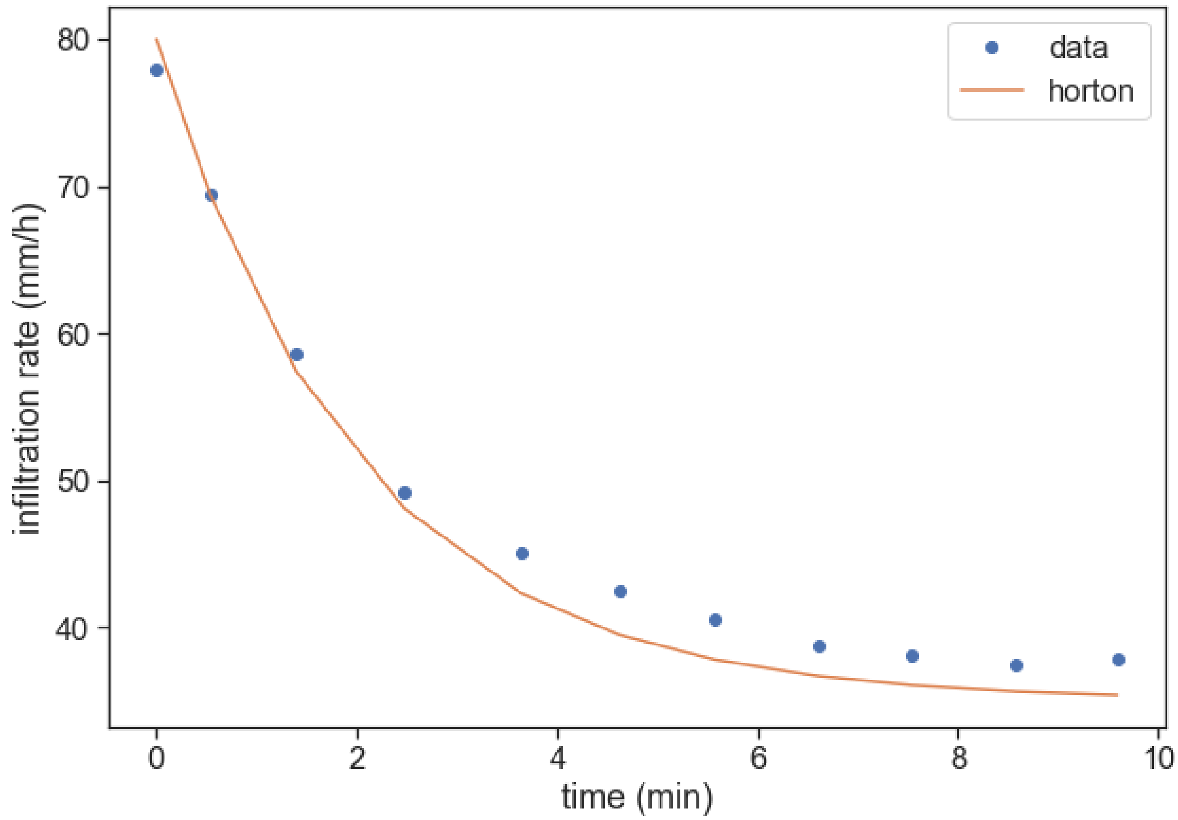
$$f = f_c + (f_0 - f_c)e^{-\beta t}$$

- f: infiltration rate
- f_c: infiltration capacity at large t
- f_0: initial infiltration capacity
- β: best fit empirical parameter

Write a function called `horton`, that receives time `t` and the three parameters, and returns the right-hand side of the equation above. Plot one of the data sets, together with a guess of the parameters that should **roughly** fit the data.

```
def horton(t, fc, f0, beta):
    return fc + (f0 - fc)*np.exp(-beta*t)

fig, ax = plt.subplots(figsize=(10,7))
t = d1[:,0]
```

```
t = t - t[0]
f = d1[:,1]
ax.plot(t, f, 'o', label="data")
ax.plot(t, horton(t, 35, 80, 0.5), '-', label="horton")
ax.set(xlabel="time (min)",
       ylabel="infiltration rate (mm/h)")
ax.legend(loc="upper right");
```



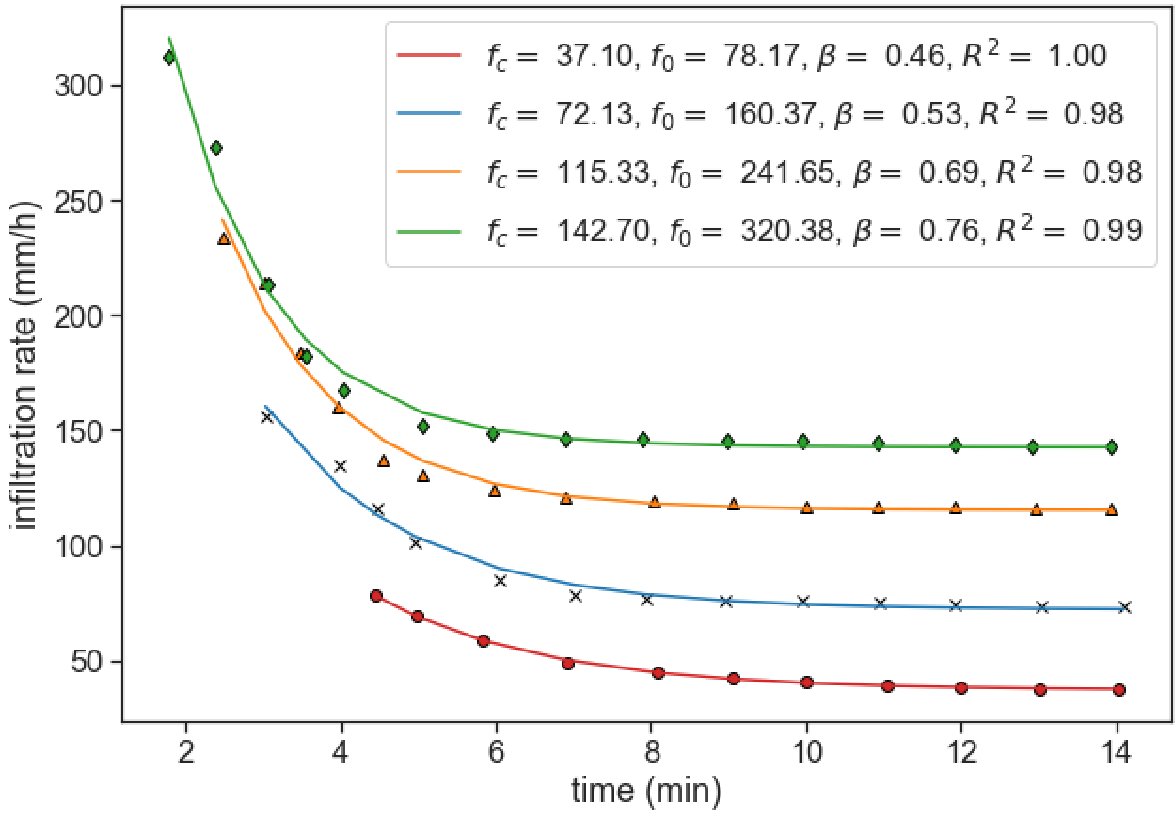
Find the best fit for the parameters f_c, f_0, β. Calculate the R^2 for each data set.

For the best fit, use scipy's `curve_fit`. Write a function to compute the R-squared of your fit.

```
def horton(t, fc, f0, beta):
    return fc + (f0 - fc)*np.exp(-beta*t)

def best_fit(data):
    t = data[:,0]
```

```
    t0 = t[0]
    t = t - t0
    f = data[:,1]
    # best fit
    popt, pcov = curve_fit(f=horton,             # model function
                           xdata=t,              # x data
                           ydata=f,              # y data
                           p0=(130, 800, 0.5),   # initial guess of the parameters
                          )
    return [popt, pcov]

def calculate_r_squared(data, popt):
    t = data[:,0]
    t = t - t[0]
    f = data[:,1]
    # Calculate residuals
    residuals = f - horton(t, *popt)
    # You can get the residual sum of squares (ss_res) with
    ss_res = np.sum(residuals**2)
    # You can get the total sum of squares (ss_tot) with
    ss_tot = np.sum((f - np.mean(f))**2)
    # And finally, the r_squared-value with,
    r_squared = 1 - (ss_res / ss_tot)
    return r_squared

def plot_best_fit(data, axis, marker, markercolor):
    # calculate best fit parameters
    popt, pcov = best_fit(data)
    t = data[:,0]
    f = data[:,1]
    # plot data points
    ax.plot(t, f, marker, markerfacecolor=markercolor, markeredgecolor="black")
    # plot best fit line
    r_squared = calculate_r_squared(data, popt)
    labeltext = r"$f_c=$ {:.2f}, $f_0=$ {:.2f}, $\beta=$ {:.2f}, $R^2=$ {:.2f}".format(popt[
    ax.plot(t, horton(t-t[0], *popt), color=markercolor, label=labeltext)

fig, ax = plt.subplots(figsize=(10,7))
plot_best_fit(d1, ax, 'o', "tab:red")
plot_best_fit(d2, ax, 'x', "tab:blue")
plot_best_fit(d3, ax, '^', "tab:orange")
plot_best_fit(d4, ax, 'd', "tab:green")
```

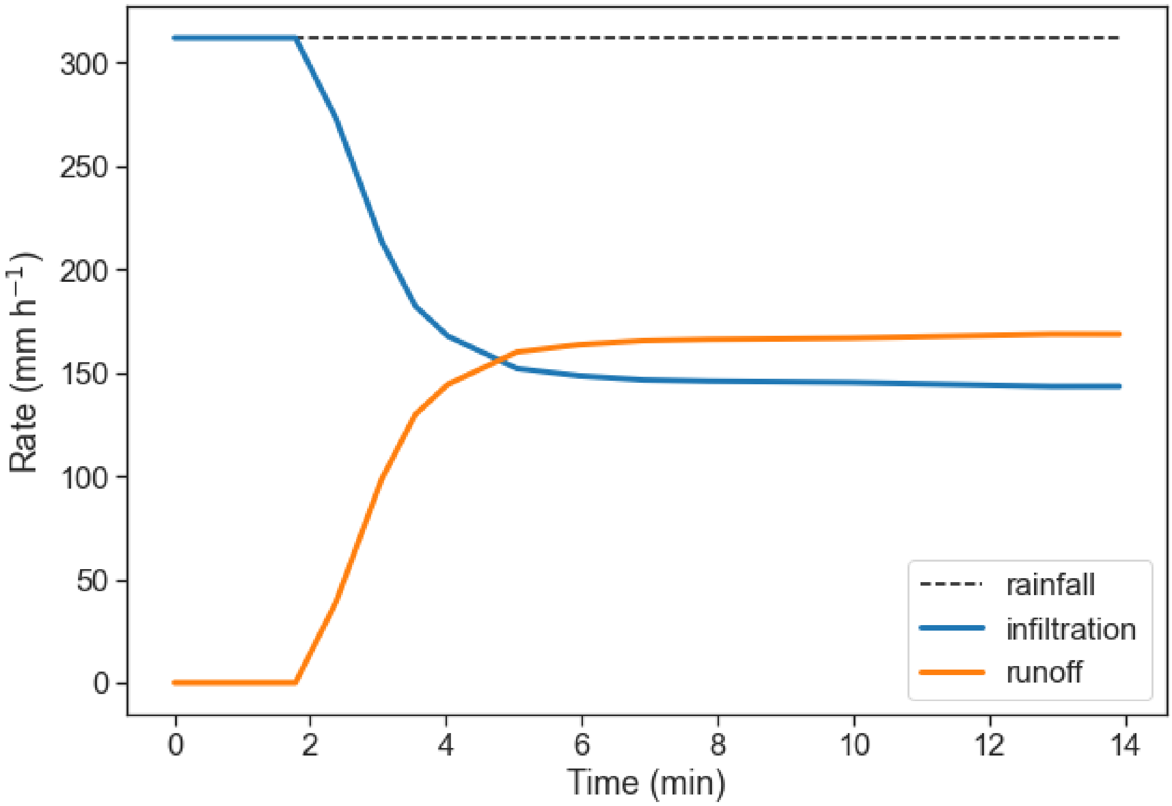
```
ax.set(xlabel="time (min)",
       ylabel="infiltration rate (mm/h)")
ax.legend();
```



Make a graph of the infiltration rate and of the runoff, as a function of time. Use any of the four data sets you have.

```
fig, ax = plt.subplots(figsize=(10,7))
data = d4
t = data[:, 0]
f = data[:, 1]
t = np.concatenate([ [0], t])
f = np.concatenate([ [f[0]], f])
runoff = f[0] - f
ax.plot(t, f*0 + f[0], ls="--", color="black", label="rainfall")
ax.plot(t, f, color="tab:blue", lw=3, label=r"infiltration")
ax.plot(t, runoff, color="tab:orange", lw=3, label=r"runoff")
ax.set(xlabel="Time (min)",
```

```
        ylabel=r"Rate (mm h$^{-1}$)")
ax.legend(loc="lower right");
```



14.3 Green & Ampt

$$f = \frac{A}{F} + B$$

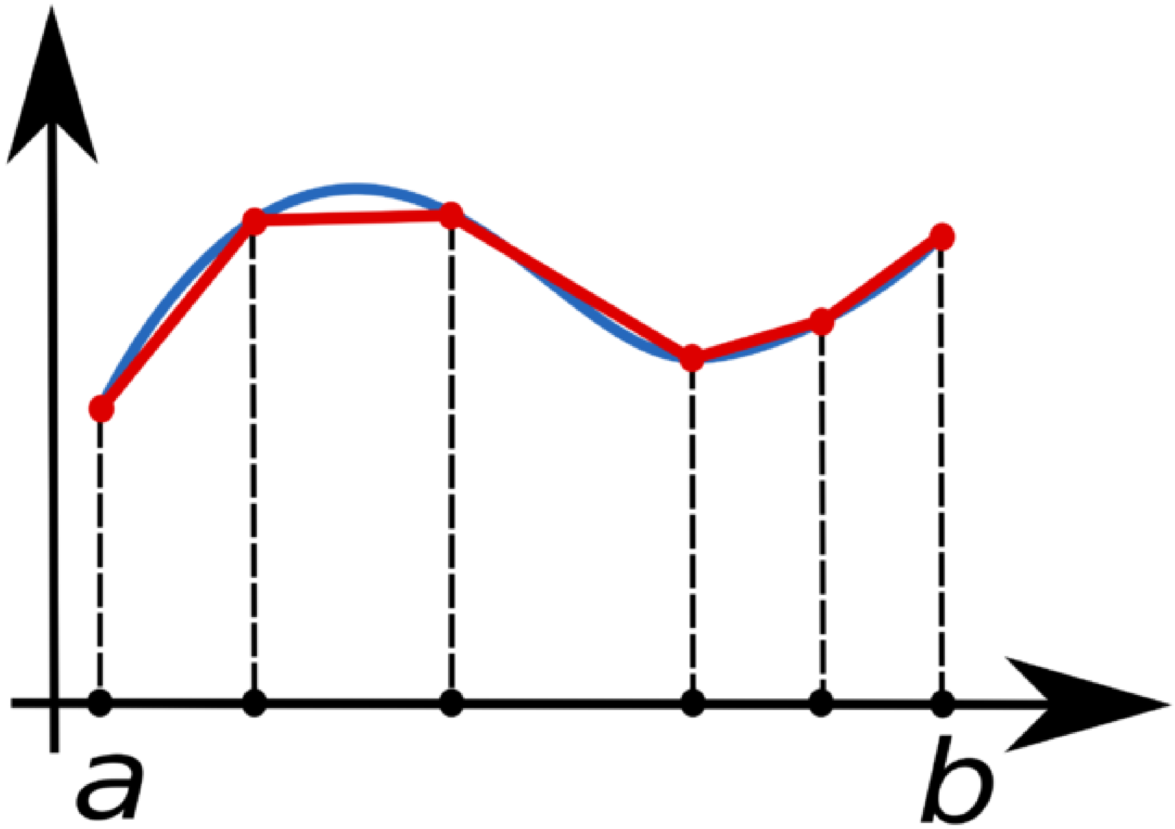
where

- $A = K_{\text{sat}} \cdot |\psi_f| \cdot (\phi - \theta_0)$
- $B = K_{\text{sat}}$

Write a function that calculates the cumulative of the infiltration rate.

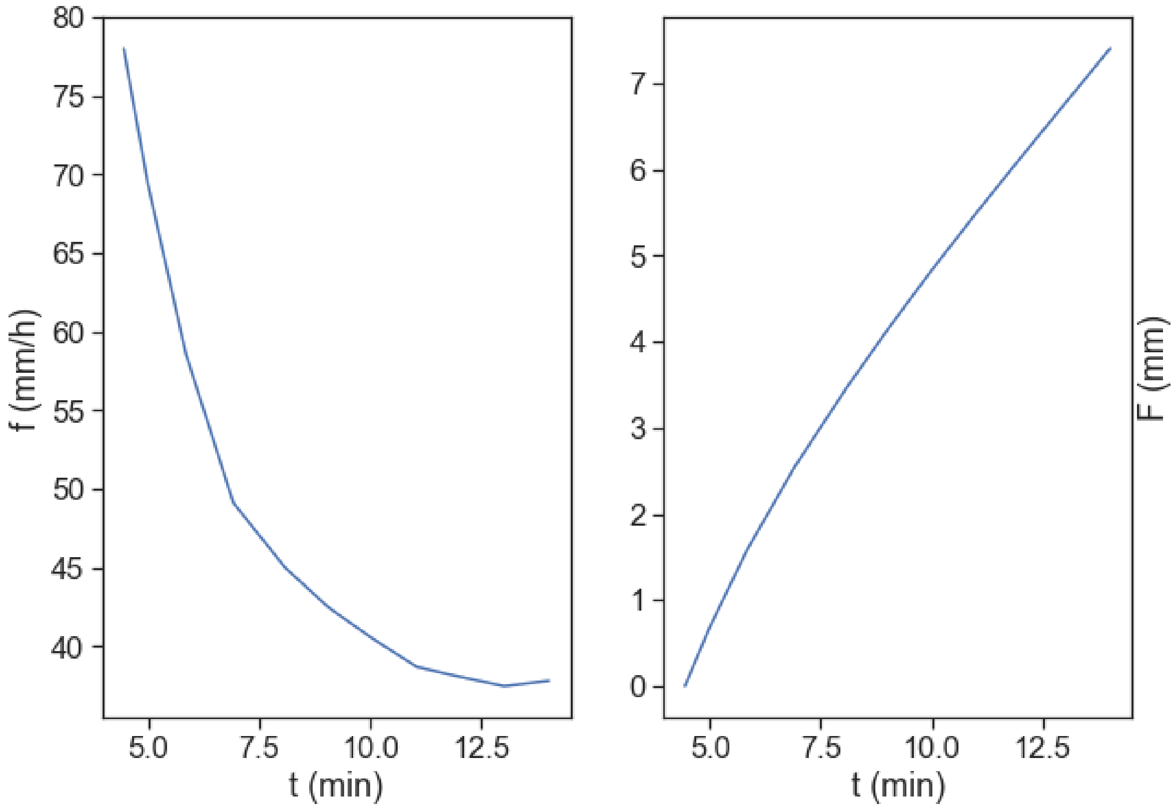
$$F(t) = \int_0^t f(t)\ \mathrm{d}t$$

Use numpy's `trapz` function, that implements the "trapezoidal rule"




```
def cumulative_F(t, f):
    F = np.array([0])
    t = t/60 # convert minute to hour
    for i in np.arange(2,len(t)+1):
        area = np.trapz(f[:i], t[:i])
        F = np.concatenate([F, [area]])
    return F

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7))
t, f = d1[:,0], d1[:,1]
F = cumulative_F(t, f)
ax1.plot(t, f, label="f, rate")
ax2.plot(t, F, label="F, cumulative")
ax1.set(xlabel="t (min)",
        ylabel="f (mm/h)")
ax2.set(xlabel="t (min)",
```

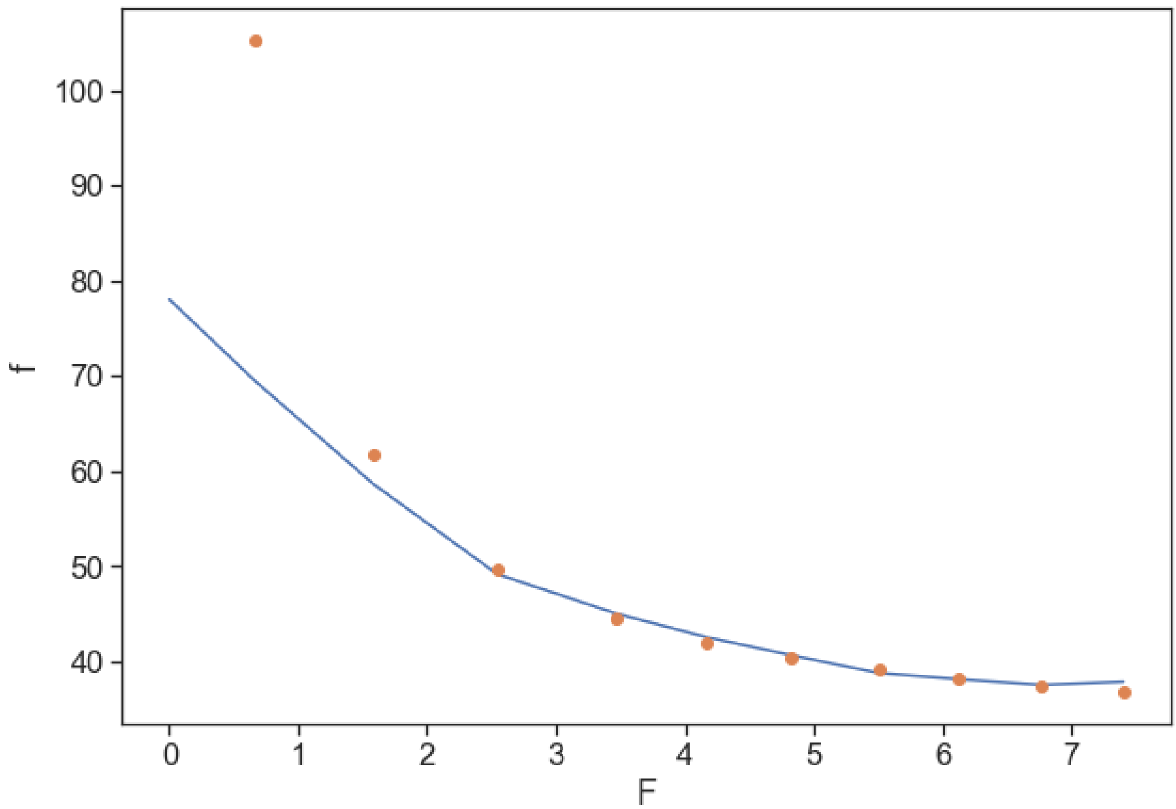
```
        ylabel="F (mm)")
ax2.yaxis.set_label_position("right")
```



Plot f as a function of F. Try to guess A and B that give reasonable results.

```
fig, ax = plt.subplots(figsize=(10,7))
t, f = d1[:,0], d1[:,1]
F = cumulative_F(t, f)
ax.plot(F, f)
A=50; B=30;
ax.plot(F, A/F + B, 'o')
ax.set(xlabel="F",
       ylabel="f")
```

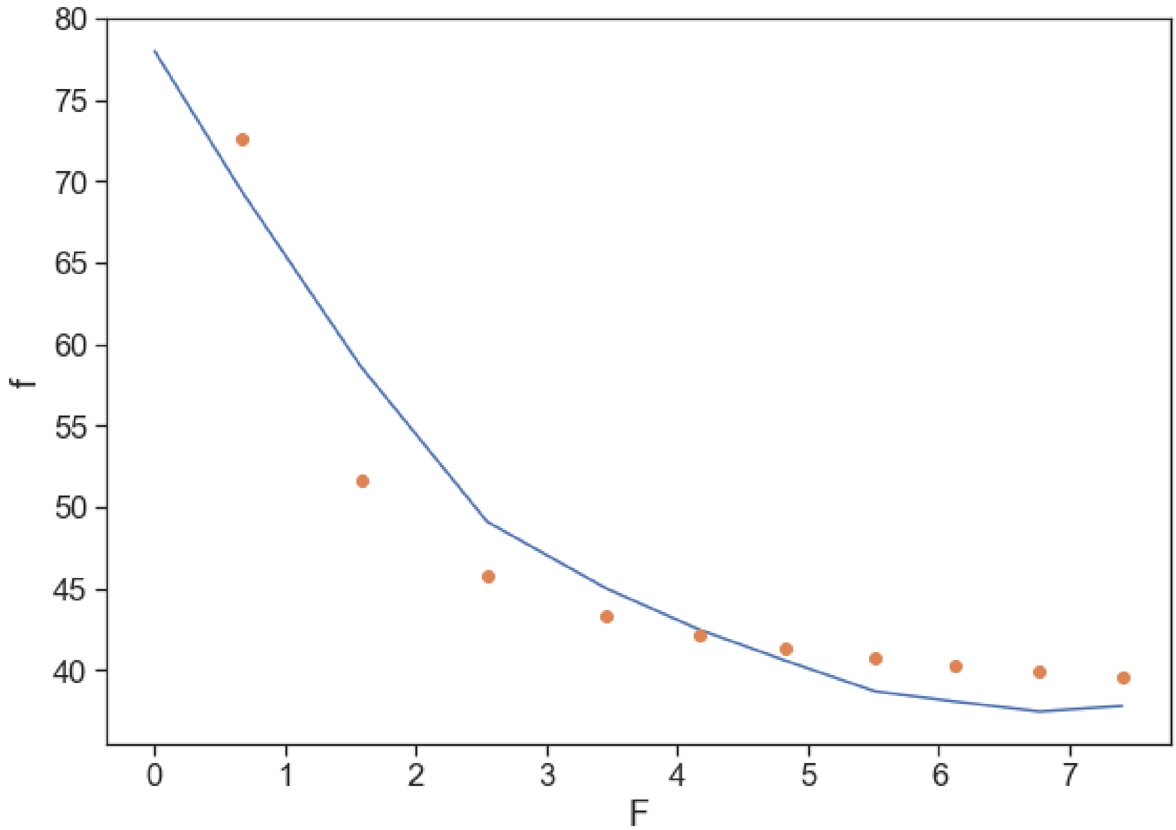
```
/Users/yairmau/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: RuntimeWarning
```



Use the `curve_fit` to find the optimal values for A and B.

```
def G_and_A(F, A, B):
    return A/F + B

popt, pcov = curve_fit(f=G_and_A,      # model function
                       xdata=F[1:],        # x data
                       ydata=f[1:],        # y data
                       p0=(50, 30),    # initial guess of the parameters
                      )

# popt, pcov = curve_fit(G_and_A, F[1:], f[1:], p0=(50, 30))  # p0 = initial guess
print(popt)

fig, ax = plt.subplots(figsize=(10,7))
ax.plot(F, f)
ax.plot(F[1:], popt[0]/F[1:] + popt[1], 'o')
ax.set(xlabel="F",
       ylabel="f")
```

```
[24.12368526 36.34242813]
```



14.4 Homework

Go to Soil Texture Calculator, estimate the texture of "standard soil" in Nassif & Wilson, 1975.

Part V

Streamflow

15 Streamflow

15.1 Watershed -

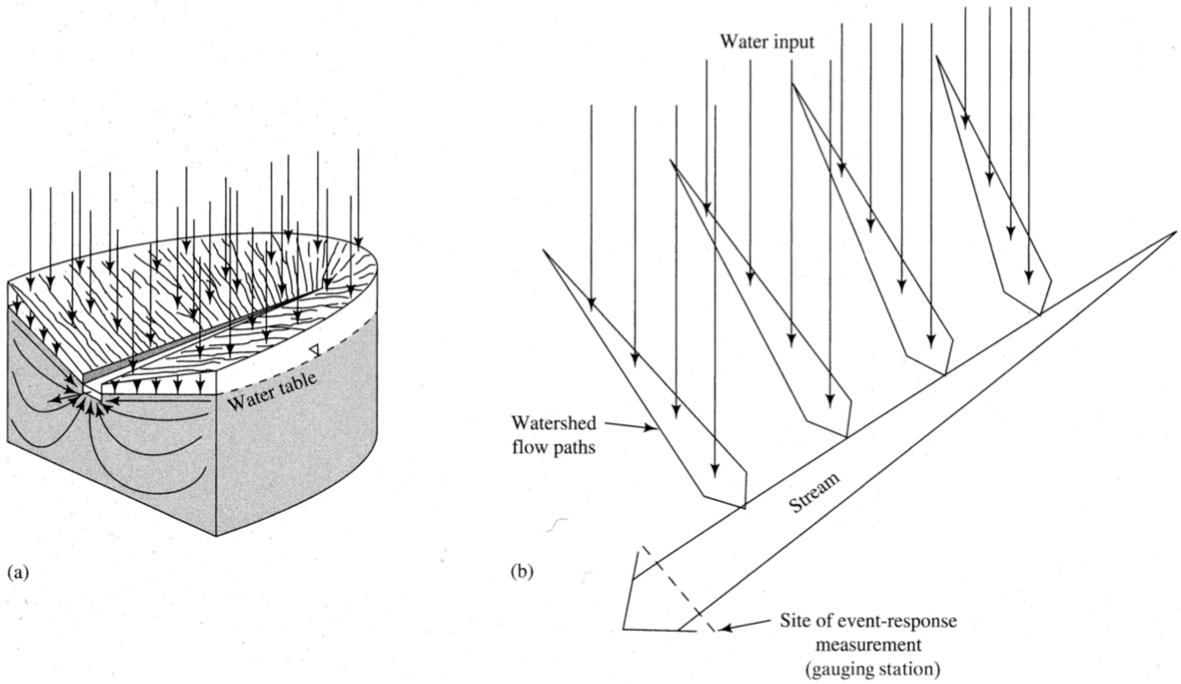
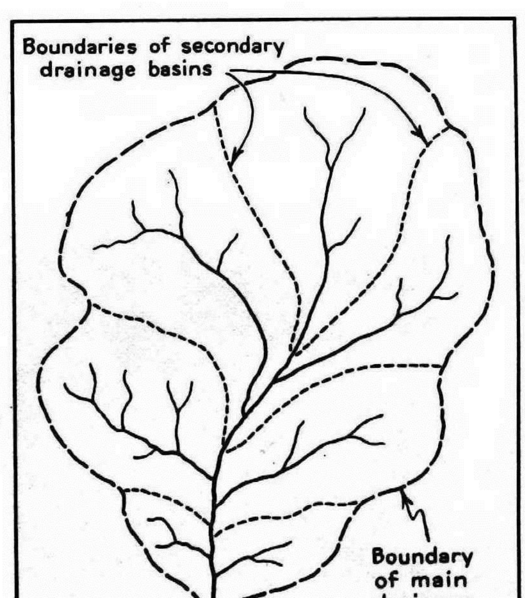


Figure 10.3 (a) Schematic flow paths in a small upland watershed receiving water input. (b) The essence of watershed response as the space- and time-integrated result of flow with lateral inflows.



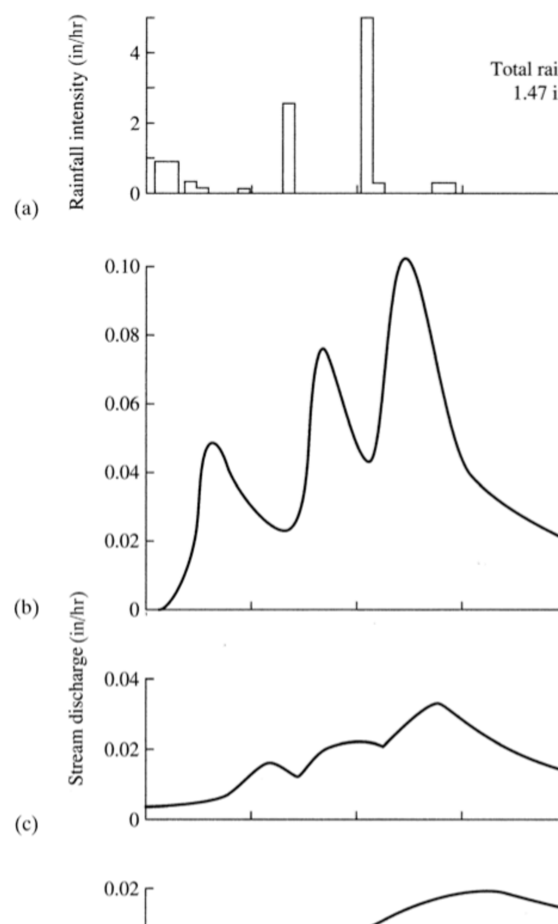
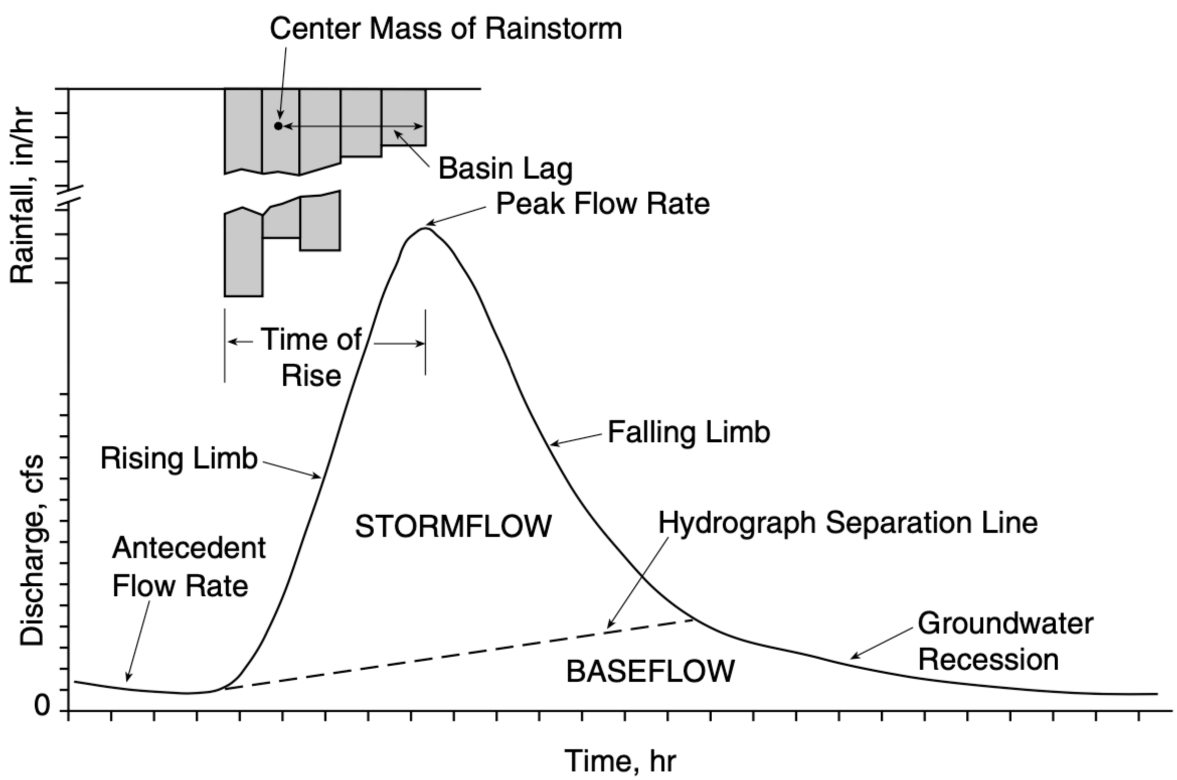


Figure 10.5 Changes in hydrograph shape at a series of gauging stations along the Sleepers River in Danville, Vermont, in response to an intense rainstorm [(a) is a hyetograph]. Note that the left-hand hydrograph ordi-

Watershed response:

- The volume of water appearing in the apparent response hydrograph for a given event is usually only a fraction (often a very small fraction) of the total input. The remainder of the water input ultimately leaves the watershed as:
    1. evapotranspiration;
    2. streamflow that occurs so long after the event that it cannot be associated with that event; or
    3. ground-water outflow from the watershed.
- The water identified as the response to a given event may originate on only a fraction of the watershed; this fraction is called the contributing area.
- The extent of the contributing area may vary from event to event and during an event.
- At least some of the water identified as the response to a given event may be "old water" that entered the watershed in a previous event.

15.2 base flow separation



Base flow

Base flow is the portion of streamflow that is presumed to have entered the watershed in previous events and to be derived from persistent, slowly varying sources. (Ground water is usually assumed to be the main, if not the only, such source.)

Event flow

Event flow (also called direct runoff, storm runoff, quick flow, or storm flow) is considered to be the direct response to a given water-input event.

Total flow

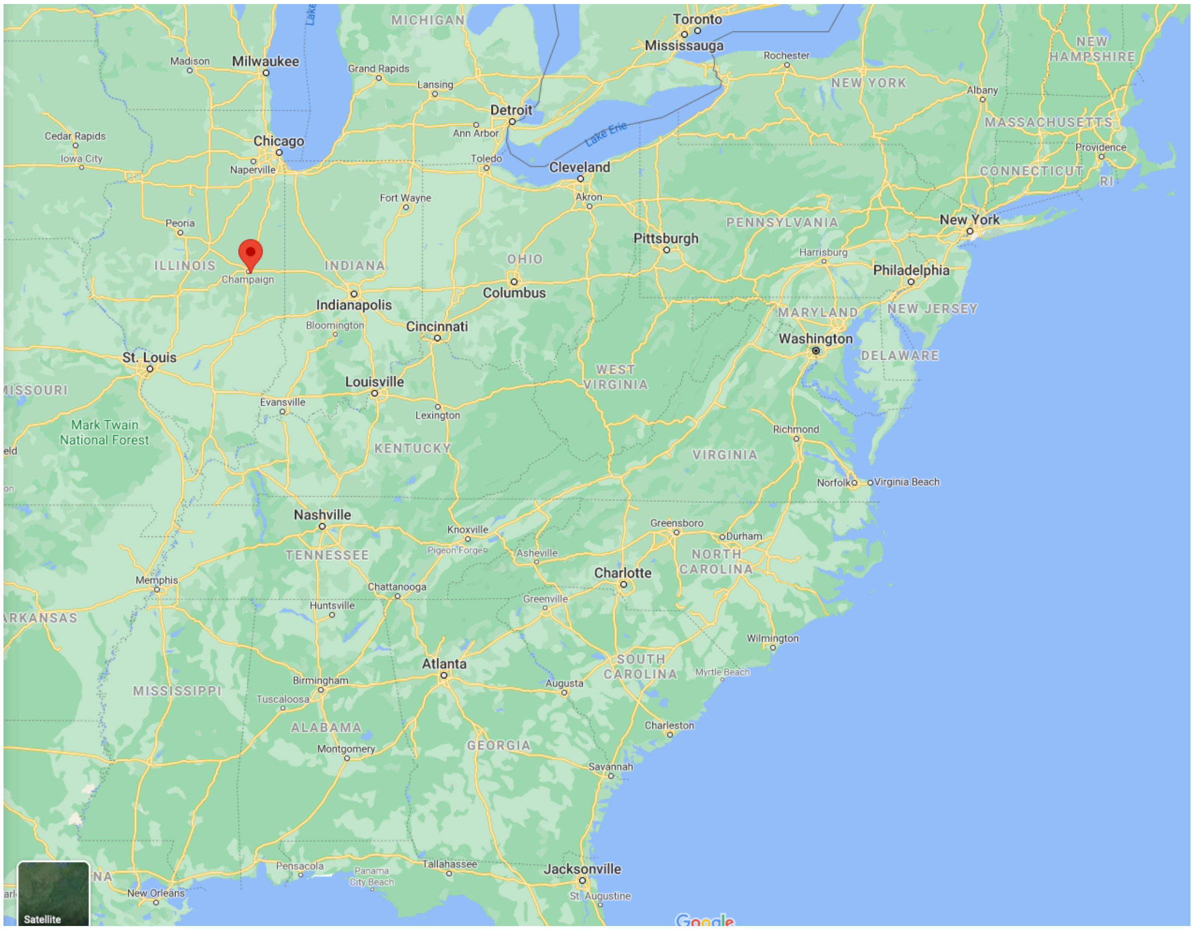
Total flow rate at any instant $q(t)$ is the sum of event-flow rate $q^*(t)$ and base-flow rate q_{BF}(t):

$$q(t) = q^*(t) + q_{BF}(t)$$

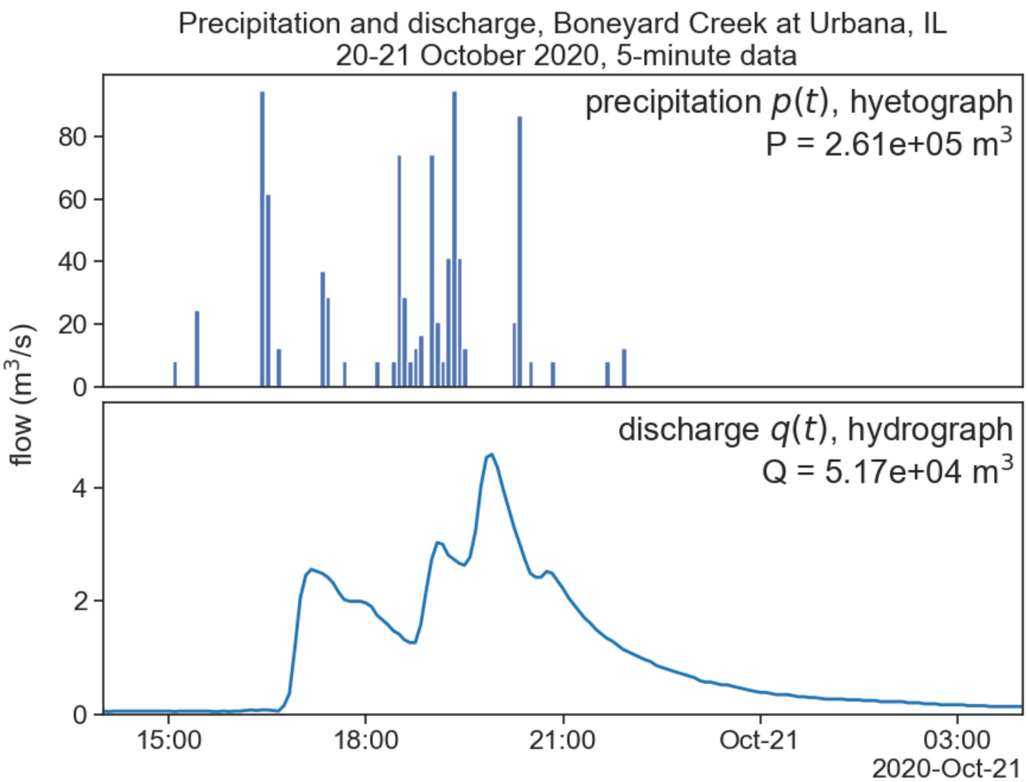
Attention!

Graphical flow separation techniques are heuristic and have no direct scientific basis.

15.3 Urbana, IL



15.3.1 hyetograph, hydrograph



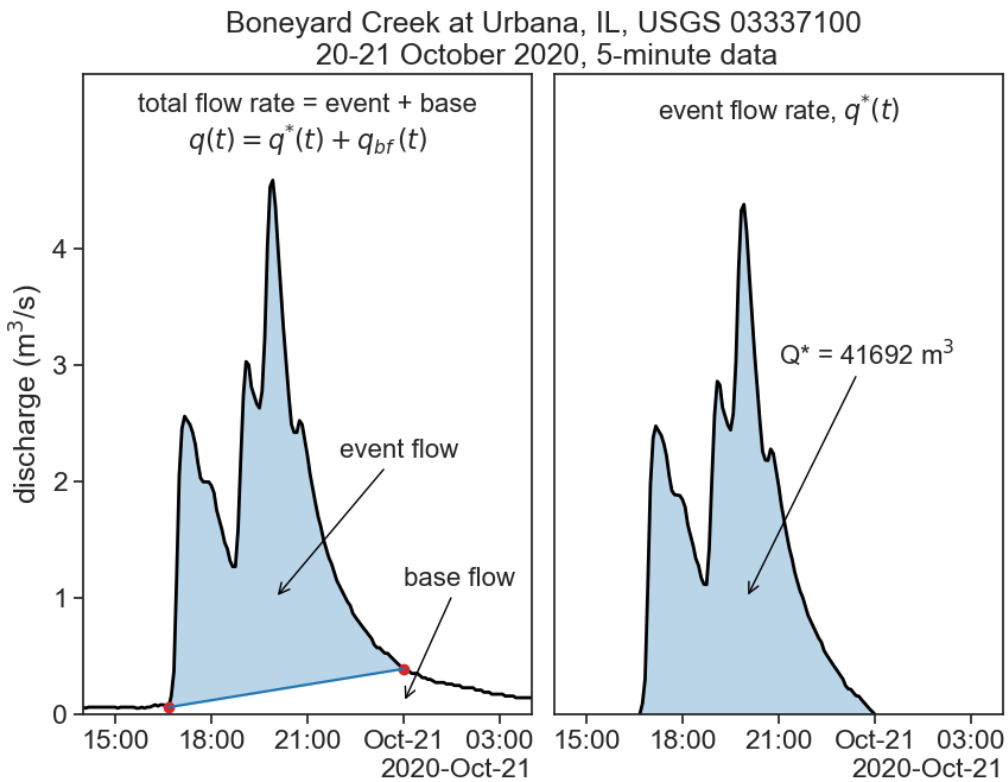
15.3.2 notation

Box 10.1 Notation

In this chapter, we use the following notation:

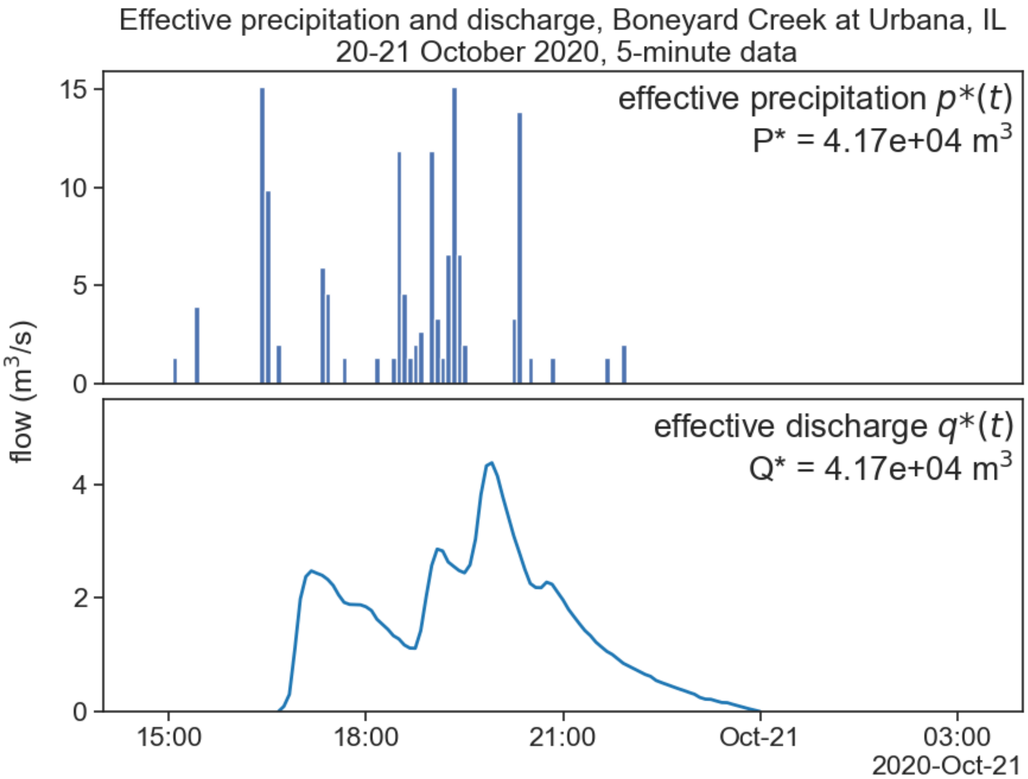
p	rate of water input (rain plus snowmelt) in a storm event [L T^{-1}]
p^*	rate of effective water input (≡ water that appears as streamflow in response to a water-input event) [L T^{-1}]
P	total volume of water input in a storm event [L] or [L^3]
P^*	total volume of effective water input in a storm event [L] or [L^3]
$q(t)$	total streamflow rate as a continuous function of time [L T^{-1}] or [L^3 T^{-1}]
$q^*(t)$	event-flow rate as a continuous function of time [L T^{-1}] or [L^3 T^{-1}]
$q_{BF}(t)$	base-flow rate as a continuous function of time [L T^{-1}] or [L^3 T^{-1}]
Q	total volume of streamflow in a storm event [L^3] or [L]
Q^*	total volume of event flow in a storm event [L^3] or [L]

15.3.3 base flow separation



15.3.4 effective precipitation = effective discharge

$$P^* = Q^*$$


Effective precipitation and discharge, Boneyard Creek at Urbana, IL
20-21 October 2020, 5-minute data
effective precipitation $p*(t)$
P* = 4.17e+04 m^3
effective discharge $q*(t)$
Q* = 4.17e+04 m^3
flow (m^3/s)
15:00
18:00
21:00
Oct-21
03:00
2020-Oct-21

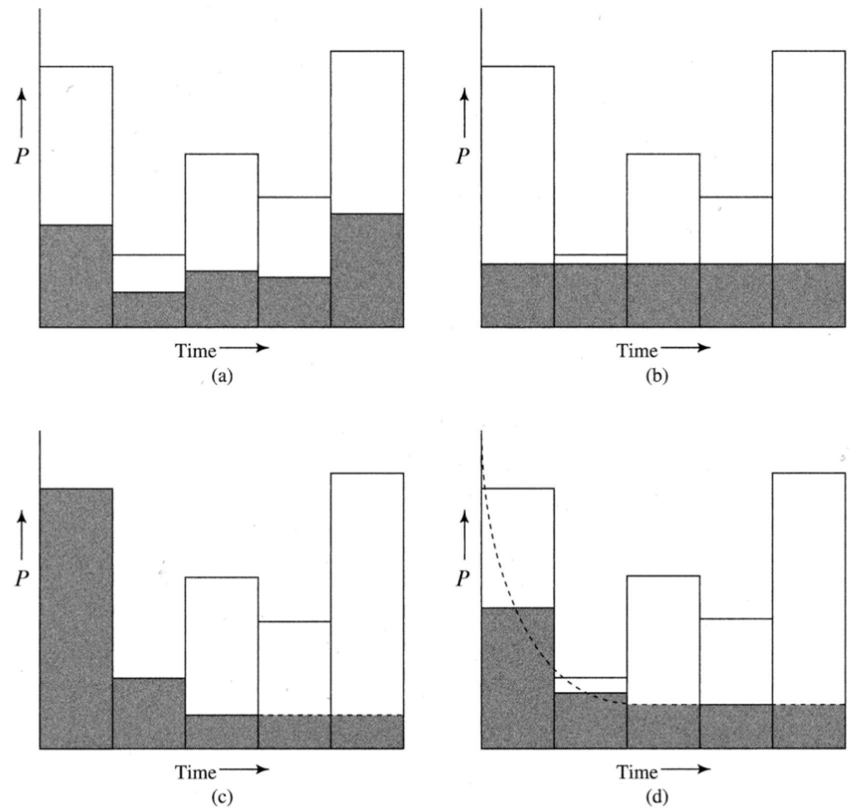


Figure 10.49 Conceptual models for estimating effective water input, P^*, from hyetograph of water input, P. Losses are shaded portions, P^* is unshaded. (a) Losses equal a constant fraction of the water input for each time period. (b) Losses equal a constant rate throughout the event. (c) Losses are given by an **initial abstraction** (which may be a specified amount or all input over an initial time period) followed by a constant rate (which may be zero). (d) Losses are given by an approximation to an infiltration-type curve (dotted line), such as given by the Green-and-Ampt or Philip approach (see chapter 8) [adapted from Pilgrim and Cordery (1992)].

15.3.5 time lags

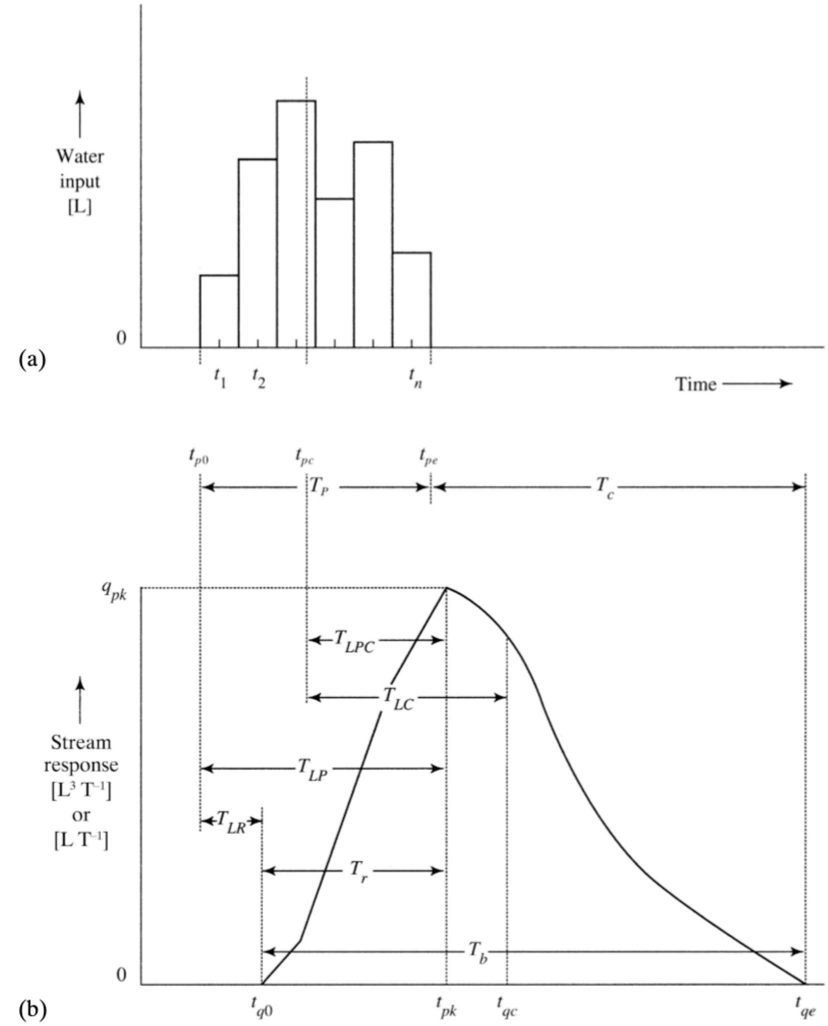
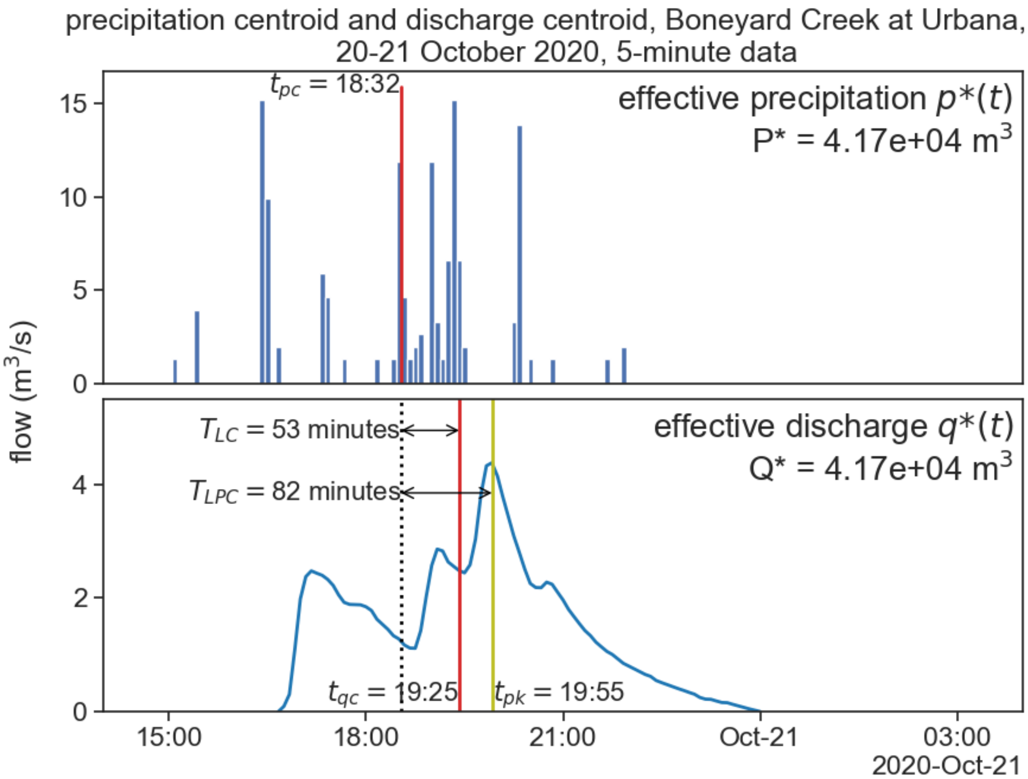


Figure 10.12 Definitions of terms used to describe (a) hyetographs and (b) response hydrographs. See table 10.2.

Table 10.2 Definitions of Terms Used to Describe Hyetographs and Response Hydrographs (see figure 10.12).

Time Instants		**Time Durations**	
t_{p0}	$\equiv$ beginning of effective water input	T_P	$\equiv$ duration of effective water input $= t_{p0} - t_{pe}$
t_{pc}	$\equiv$ centroid of effective water input	T_{LR}	$\equiv$ response lag $= t_{q0} - t_{p0}$
t_{pe}	$\equiv$ end of effective water input	T_r	$\equiv$ time of rise $= t_{pk} - t_{q0}$
t_{q0}	$\equiv$ beginning of hydrograph rise	T_{LP}	$\equiv$ lag-to-peak $= t_{pk} - t_{p0}$
t_{pk}	$\equiv$ time of peak discharge	T_{LPC}	$\equiv$ centroid lag-to-peak $= t_{pk} - t_{pc}$
t_{qc}	$\equiv$ centroid of response hydrograph	T_{LC}	$\equiv$ centroid lag $= t_{qc} - t_{pc}$
t_{qe}	$\equiv$ end of response	T_b	$\equiv$ time base $= t_{qe} - t_{q0}$
		T_c	$\equiv$ time of concentration $= t_{qe} - t_{pe}$
		T_{eq}	$\equiv$ time to equilibrium $\approx T_c$



It is commonly assumed that $T_{LPC} \simeq 0.60 \cdot T_c$, where T_c is the time of concentration, i.e., the time it takes water to travel from the hydraulically most distant part of the contributing area to the outlet.

The centroid is a weighted-average time, each time instant is multiplied by the amount of flow in that instant.

Time of precipitation centroid:

$$t_{pc} = \frac{\sum_{i=1}^{n} p_i^* \cdot t_i}{P^*}$$

Time of streamflow centroid:

$$t_{qc} = \frac{\sum_{i=1}^{n} q_i^* \cdot t_i}{Q^*}$$

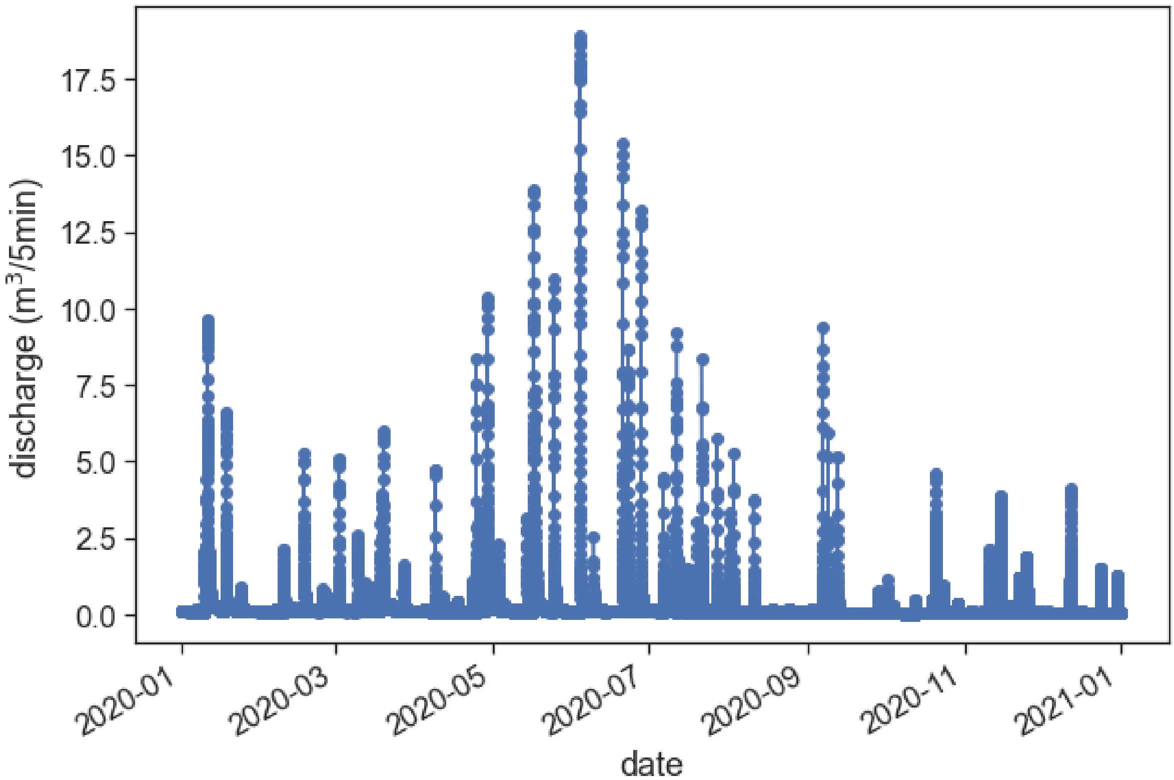
16 Exercises

Import relevant packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
from ipywidgets import *
```

Import streamflow data from USGS's National Water Information System. We will be using data from Urbana, IL.

```
# Drainage area: 4.78 square miles
data_file = "USGS 03337100 BONEYARD CREEK AT LINCOLN AVE AT URBANA, IL.dat"
df_q_2020 = pd.read_csv(data_file,
                        header=31,                      # no headers needed, we'll do that la
                        delim_whitespace=True,            # blank spaces separate between col
                        na_values=["Bkw"]  # substitute these values for missing (NaN) values
                )
df_q_2020.columns = ['agency_cd', 'site_no','datetime','tz_cd','EDT','discharge','code']
df_q_2020['date_and_time'] = df_q_2020['datetime'] + ' ' + df_q_2020['tz_cd'] # combine date
df_q_2020['date_and_time'] = pd.to_datetime(df_q_2020['date_and_time'])        # interpret da
df_q_2020 = df_q_2020.set_index('date_and_time')                              # make datetime the
df_q_2020['discharge'] = df_q_2020['discharge'].astype(float)
df_q_2020['discharge'] = df_q_2020['discharge'] * 0.0283168 # convert cubic feet to m3

fig, ax = plt.subplots(figsize=(10,7))
ax.plot(df_q_2020['discharge'], '-o')
plt.gcf().autofmt_xdate()
ax.set(xlabel="date",
       ylabel=r"discharge (m$^3$/5min)");
```

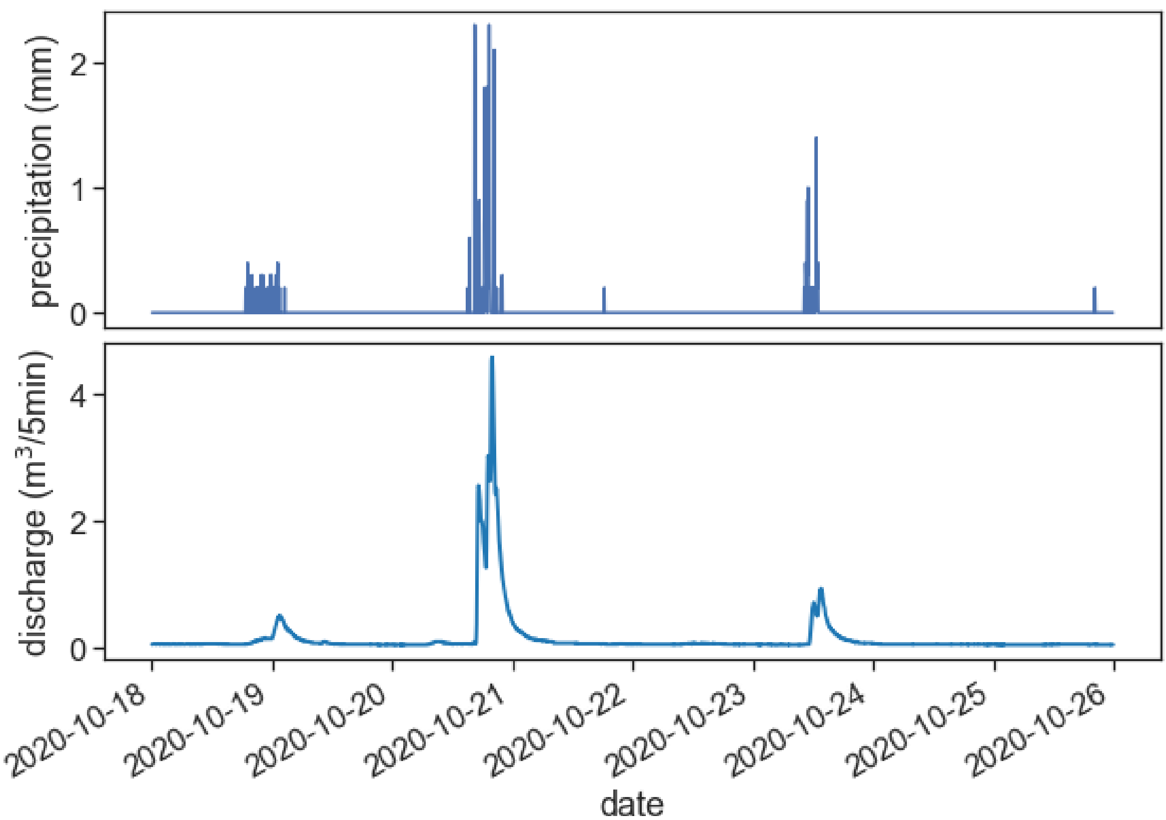


Import sub-hourly (5-min) rainfall data from NOAA's Climate Reference Network Data website

```
data_file = "Champaign - IL.txt"
df_p_2020 = pd.read_csv(data_file,
                        header=None,                      # no headers needed, we'll do that
                        delim_whitespace=True,            # blank spaces separate between col
                        na_values=["-99.000", "-9999.0"]  # substitute these values for missi
                       )
headers = pd.read_csv("HEADERS_sub_hourly.txt",    # load headers file
                      header=1,                    # skip the first [0] line
                      delim_whitespace=True
                     )
df_p_2020.columns = headers.columns                       # rename df columns with headers co
# LST = local standard time
df_p_2020["LST_TIME"] = [f"{x:04d}" for x in df_p_2020["LST_TIME"]]  # time needs padding of
df_p_2020['LST_DATE'] = df_p_2020['LST_DATE'].astype(str)           # convert date into stri
df_p_2020['datetime'] = df_p_2020['LST_DATE'] + ' ' + df_p_2020['LST_TIME'] # combine date+ti
df_p_2020['datetime'] = pd.to_datetime(df_p_2020['datetime'])        # interpret datetime
df_p_2020 = df_p_2020.set_index('datetime')                          # make datetime the inde
```

Plot rainfall and streamflow. Does this makes sense?

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,7))
fig.subplots_adjust(hspace=0.05)

start = "2020-10-18"
end = "2020-10-25"
ax1.plot(df_p_2020[start:end]['PRECIPITATION'])
ax2.plot(df_q_2020[start:end]['discharge'], color="tab:blue", lw=2)

ax1.set(xticks=[],
        ylabel=r"precipitation (mm)")
ax2.set(xlabel="date",
        ylabel=r"discharge (m$^3$/5min)")

plt.gcf().autofmt_xdate()  # makes slated dates
```



Define smaller dataframes for $p(t)$ and $q(t)$, between the dates:

```
start = "2020-10-20 14:00:00"
end = "2020-10-21 04:00:00"
```

Don't forget to convert the units to SI!

Calculate total rainfall P^* and total discharge Q^*, in m^3.

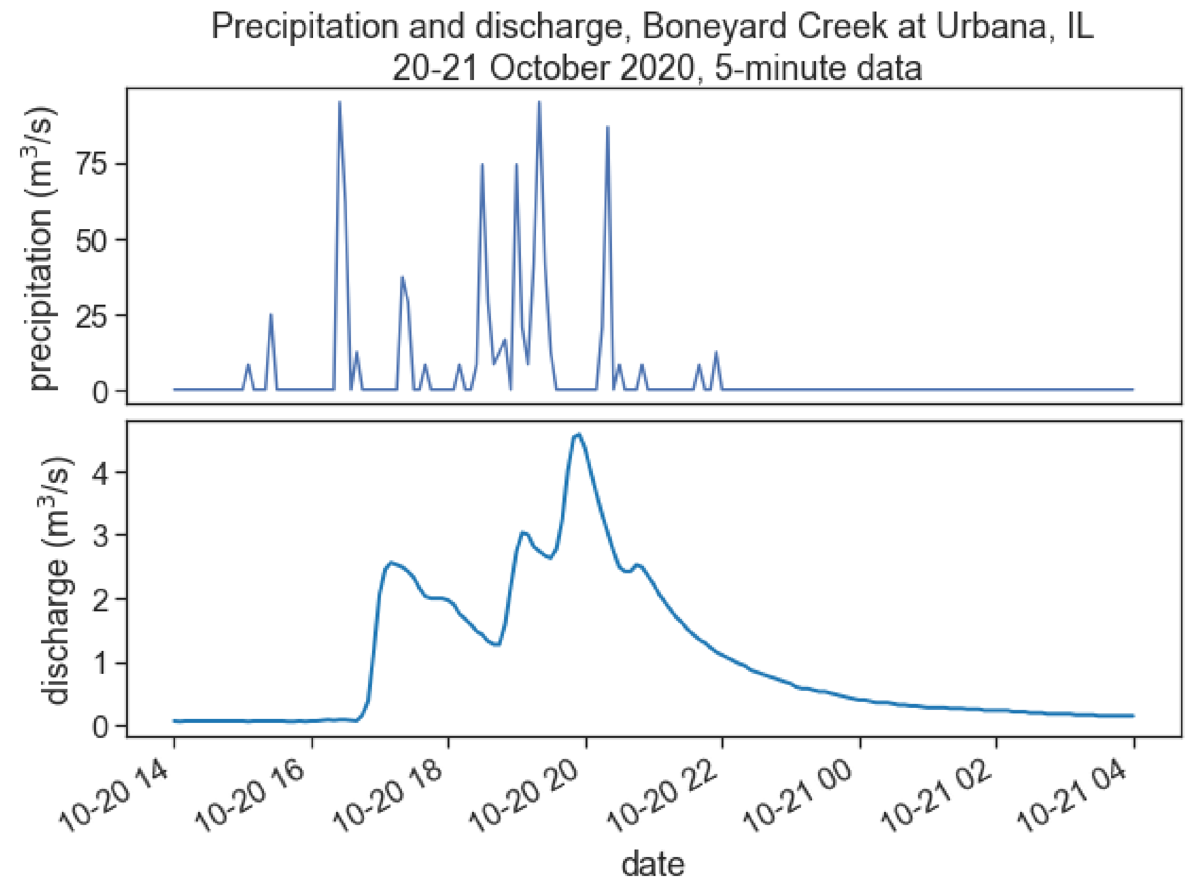
```
# Drainage area: 4.78 square miles
area = 4.78 / 0.00000038610  # squared miles to squared meters
start = "2020-10-20 14:00:00"
end = "2020-10-21 04:00:00"

df_p = df_p_2020.loc[start:end]['PRECIPITATION'].to_frame()
df_p_mm = df_p_2020.loc[start:end]['PRECIPITATION'].to_frame()
df_q = df_q_2020.loc[start:end]['discharge'].to_frame()

df_p['PRECIPITATION'] = df_p['PRECIPITATION'].values * area / 1000  # mm to m3 in the whole w
df_p['PRECIPITATION'] = df_p['PRECIPITATION'] / 60 / 5 # convert m3 per 5 min to m3/s

P = df_p['PRECIPITATION'].sum() * 60 * 5
Q = df_q['discharge'].sum() * 60 * 5

print("total precipitation during event: Pstar = {:.1e} m3".format(P.sum()))
print("total streamflow during event: Qstar = {:.1e} m3".format(Q.sum()))
```

```
total precipitation during event: Pstar = 2.6e+05 m3
total streamflow during event: Qstar = 5.2e+04 m3
```

Make another graph of $p(t)$ and $q(t)$, now with SI units.

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,7))
fig.subplots_adjust(hspace=0.05)

start = "2020-10-18"
end = "2020-10-25"
ax1.plot(df_p['PRECIPITATION'])
ax2.plot(df_q['discharge'], color="tab:blue", lw=2)

ax1.set(xticks=[],
        ylabel=r"precipitation (m$^3$/s)",
        title="Precipitation and discharge, Boneyard Creek at Urbana, IL\n 20-21 October 202
```

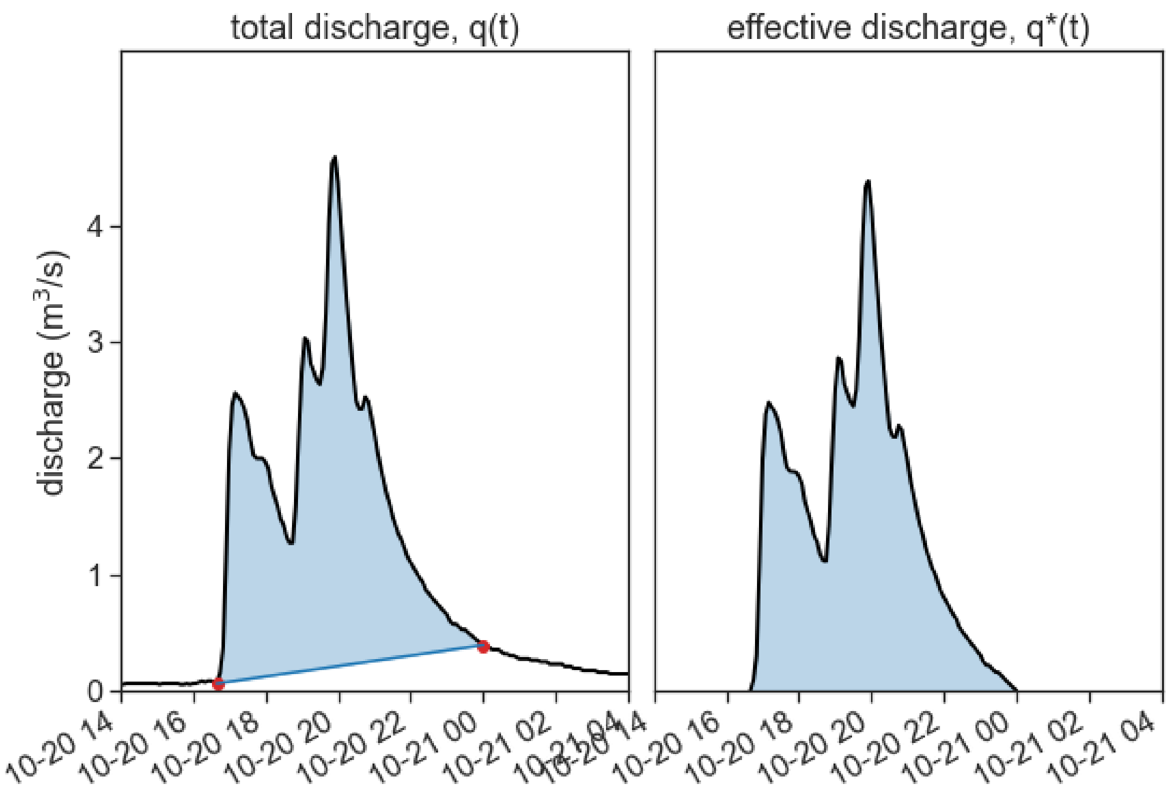
```python
ax2.set(xlabel="date",
        ylabel=r"discharge (m$^3$/s)")

plt.gcf().autofmt_xdate()  # makes slated dates
```



It's time for base flow separation! Convert $q(t)$ into $q^*(t)$

```python
from matplotlib.dates import HourLocator, DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,7))
fig.subplots_adjust(wspace=0.05)

ax1.plot(df_q['discharge'], color="black", lw=2)
point1 = pd.to_datetime("2020-10-20 16:40:00")
```

```python
point2 = pd.to_datetime("2020-10-21 00:00:00")
two_points = df_q.loc[[point1, point2]]['discharge']
ax1.plot(two_points, 'o', color="tab:red")

new = pd.DataFrame(data=two_points, index=two_points.index)

df_linear = (new.resample("5min") #resample
                .interpolate(method='time') #interpolate by time
            )

ax1.plot(df_linear, color="tab:blue")

df_between_2_points = df_q.loc[df_linear.index]
ax1.fill_between(df_between_2_points.index, df_between_2_points['discharge'],
                 y2=df_linear['discharge'],
                 color="tab:blue", alpha=0.3)

qstar = df_q.loc[df_linear.index]['discharge'] - df_linear['discharge']
Qstar = qstar.sum() * 60 * 5

ax2.plot(qstar, color="black", lw=2)
ax2.fill_between(qstar.index, qstar,
                 y2=0.0,
                 color="tab:blue", alpha=0.3)

ax1.set(xlim=[df_q.index[0],
              df_q.index[-1]],
        ylabel=r"discharge (m$^3$/s)",
        ylim=[0, 5.5],
        yticks=[0,1,2,3,4],
        title="total discharge, q(t)")
ax2.set(yticks=[],
        ylim=[0, 5.5],
        xlim=[df_q.index[0],
              df_q.index[-1]],
        title="effective discharge, q*(t)"
       )

plt.gcf().autofmt_xdate()  # makes slated dates
```



We can calculate p^* now, using

$$P^* = Q^*$$

One of the simplest methods is to multiply $p(t)$ by a fixed constant (<1) to obtain p^*, so that the equation above holds true.

```
ratio = Qstar/ P
pstar = df_p['PRECIPITATION'] * ratio
Pstar = pstar.sum() * 5 * 60
print(f"Qstar / P = {ratio:.2f}")
```

```
Qstar / P = 0.16
```

Calculate now the centroid (t_pc) for effective precipitation p^* and centroid (t_{qc}) of effective discharge q^*. Calculate also the time of peak discharge (t_{pk}). Then, calculate the centroid lag (T_{LC}), the centroid lag-to-peak (T_{LPC}), and the time of concentration (T_c). Use the equations below:

$T_{LPC} \simeq 0.60 \cdot T_c$

Time of precipitation centroid:

$$t_{pc} = \frac{\sum_{i=1}^{n} p_i^* \cdot t_i}{P^*}$$

Time of streamflow centroid:

$$t_{qc} = \frac{\sum_{i=1}^{n} q_i^* \cdot t_i}{Q^*}$$

Centroid lag:

$$T_{LC} = t_{qc} - t_{pc}$$

Centroid lag-to-peak:

$$T_{LPC} = t_{pk} - t_{pc}$$

Time of concentration:

$$T_{LPC} \simeq 0.60 \cdot T_c$$

```
# pstar centroid
# time of the first (nonzero) rainfall data point
t0 = pstar[pstar != 0.0].index[0]
# time of the last (nonzero) rainfall data point
tf = pstar[pstar != 0.0].index[-1]
# duration of the rainfall event, in minutes
td = (tf-t0) / pd.Timedelta('1 min')
# make time array, add 2.5 minutes (half of dt)
time = np.arange(0, td+1, 5) + 2.5
# create pi array, only with relevant data (during rainfall duration)
pi = pstar.loc[(pstar.index >= t0) & (pstar.index <= tf)]
# convert from m3/5min to m3/s
pi = pi.values * 60 * 5
# time of precipitation centroid
t_pc = (pi * time).sum() / pi.sum()
# add initial time
t_pc = t0 + pd.Timedelta(minutes=t_pc)
t_pc
```

```python
# qstar centroid
# time of the first (nonzero) discharge data point
t0 = qstar[qstar != 0.0].index[0]
# time of the last (nonzero) discharge data point
tf = qstar[pstar != 0.0].index[-1]
# duration of the discharge event, in minutes
td = (tf-t0) / pd.Timedelta('1 min')
# make time array, add 2.5 minutes (half of dt)
time = np.arange(0, td+1, 5) + 2.5
# create qi array, only with relevant data (during discharge duration)
qi = qstar.loc[(qstar.index >= t0) & (qstar.index <= tf)]
# convert from m3/5min to m3/s
qi = qi.values * 60 * 5
# time of discharge centroid
t_qc = (qi * time).sum() / qi.sum()
# add initial time
t_qc = t0 + pd.Timedelta(minutes=t_qc)
t_qc

# time of peak discharge
max_discharge = qstar.max()
t_pk = qstar[qstar == max_discharge].index[0]

# centroid lag
T_LC = t_qc - t_pc

# centroid lag-to-peak
T_LPC = t_pk - t_pc

# time of concentration
T_c = T_LPC / 0.60

print(f"T_LC = {T_LC}")
print(f"T_LPC = {T_LPC}")
print(f"T_c = {T_c}")
```

```
T_LC = 0 days 00:53:03.186594
T_LPC = 0 days 01:22:59.857820
T_c = 0 days 02:18:19.763033333
```

17 Unit Hydrograph

17.1 Linear reservoir model

Box 10.3 Linear-Reservoir Model of Watershed Response

We can develop a very simple conceptual model of the response of a watershed to effective precipitation based on: (1) the principle of conservation of mass,

$$p^* - q^* = \frac{dS^*}{dt}; \qquad \textbf{(10B3.1)}$$

(section 1.6.2) and (2) the linear-reservoir conceptual model of watershed behavior (section 1.10.2):

$$q^* = \frac{1}{T^*} \cdot S^*, \qquad \textbf{(10B3.2)}$$

where p^* is effective rainfall rate ($[L^3\ T^{-1}]$ or $[L\ T^{-1}]$), q^* is event-flow rate ($[L^3\ T^{-1}]$ or $[L\ T^{-1}]$), S^* is storage of event-flow water ($[L^3]$ or $[L]$), t is time, and T^* is a time constant that characterizes the watershed response [T]. Real watersheds are not strictly linear reservoirs, but equation (10B3.2) captures the most basic aspects of watershed response and is mathematically tractable.

Combining equations (10B3.1) and (10B3.2) yields

$$p^* - q^* = T^* \cdot \frac{dq^*}{dt}, \qquad \textbf{(10B3.3)}$$

which for constant p^* has the solution

$$q^* = p^* + \left(q_0^* - p^*\right) \cdot \exp\left(-t/T^*\right), \qquad \textbf{(10B3.4)}$$

where q_0^* is the outflow at $t = 0$.

To model an isolated hydrograph rise in response to a constant input beginning at $t = 0$ and lasting until time T_P, we can set $q_0^* = 0$ at $t = 0$ so that equation (10B3.4) becomes

$$q^* = p^* \cdot [1 - \exp(-t/T^*)],\ t \le T_P \qquad \textbf{(10B3.5)}$$

The peak discharge, q_{pk}^*, occurs when water input ceases, i.e., when $t = T_P$; q_{pk}^* is then given by

$$q_{pk}^* = p^* \cdot \left[1 - \exp\left(-T_P/T^*\right)\right]. \qquad \textbf{(10B3.6)}$$

When input ceases the hydrograph recession begins and we have new conditions: $p^* = 0$ and the "initial" discharge is q_{pk}^* at time $t = T_P$. Thus for the recession equation (10B3.4) becomes

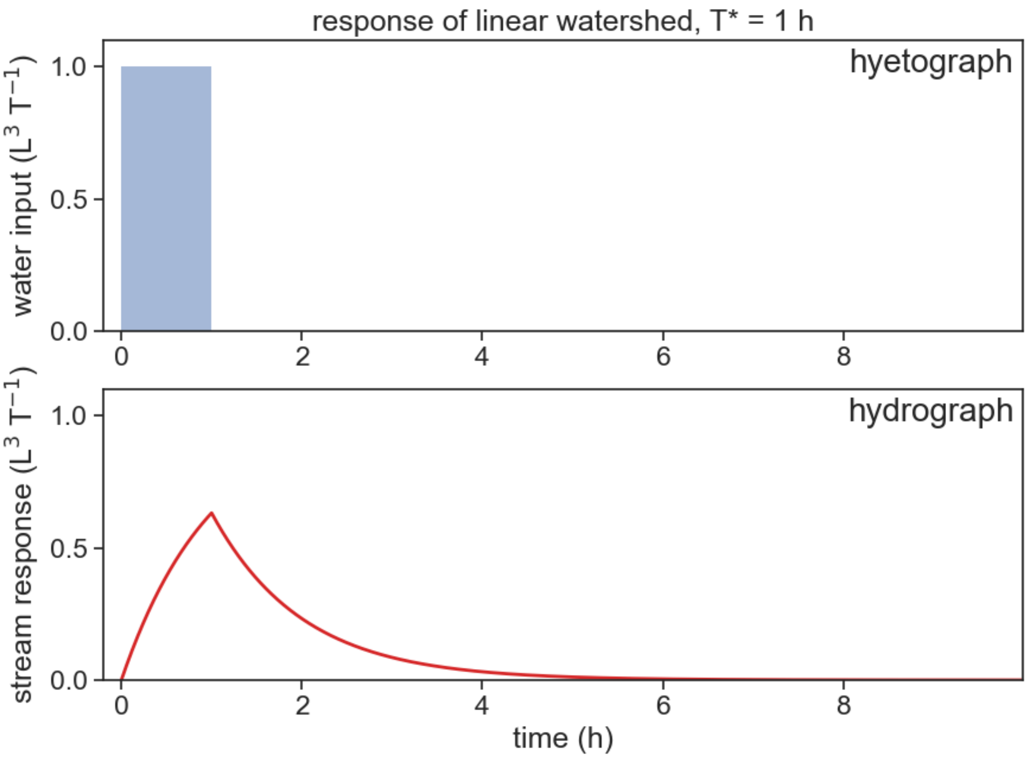
$$q^* = q_{pk}^* \cdot \exp\left[-\left(t - T_P\right)/T^*\right],\ t \ge T_P. \qquad \textbf{(10B3.7)}$$

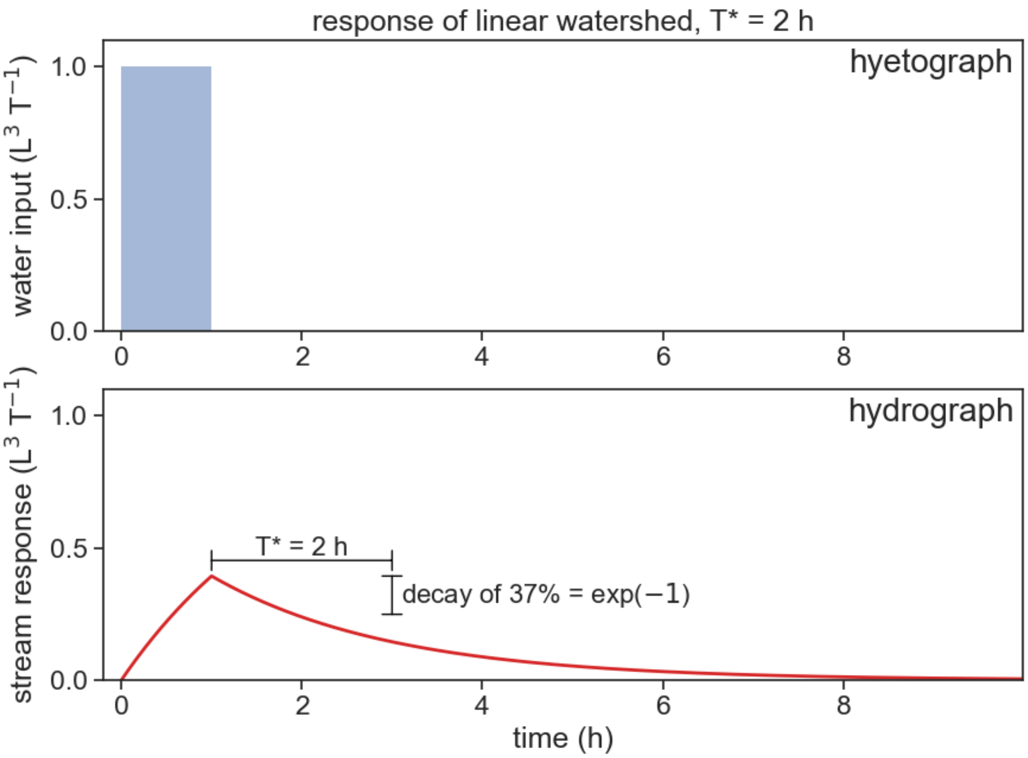
The properties of this model are:

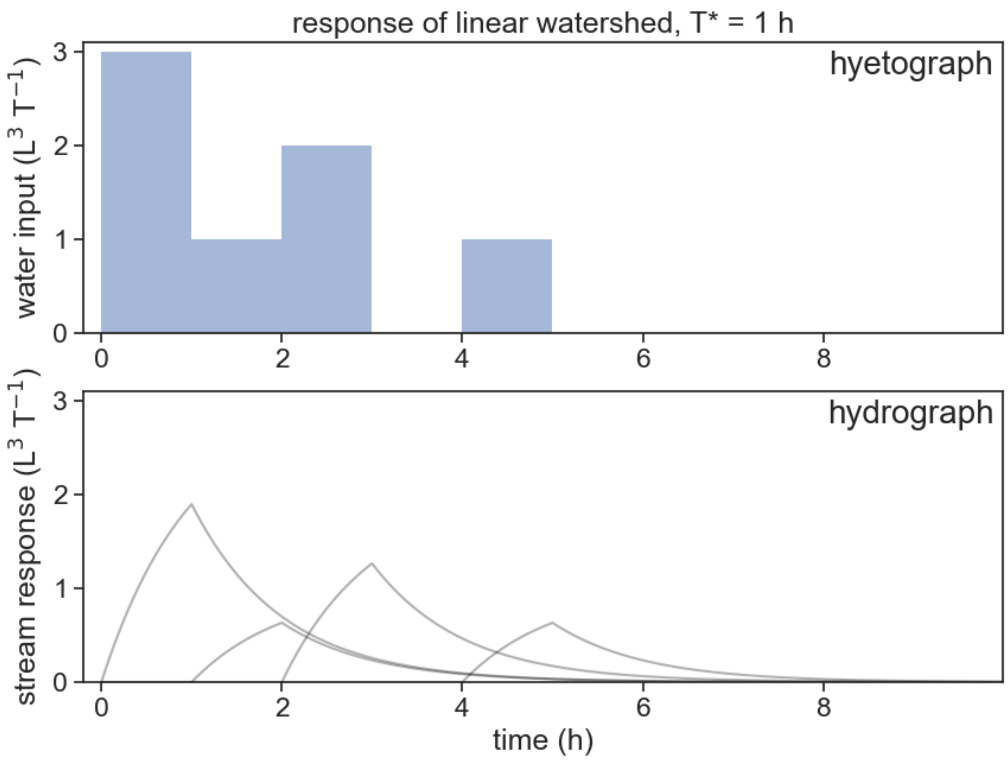
1. The model preserves continuity; i.e., the volume of event response Q^* equals the volume of effective water input, $P^* = p^* \cdot T_P$.
2. T^* can be shown to be equal to the centroid lag of the watershed (i.e., $t_{qc} - t_{pc} = T^*$).
3. The hydrograph rise begins as soon as input begins ($T_{LR} = 0$) and peaks exactly at $t_{pk} = T_P$.
4. Because outflow decreases exponentially after input ceases and approaches zero asymptotically, time of concentration is infinite. However, if we define the time to equilibrium, T_{eq}, as the time it takes for the outflow rate to reach 99% of a constant inflow rate, it can be shown that

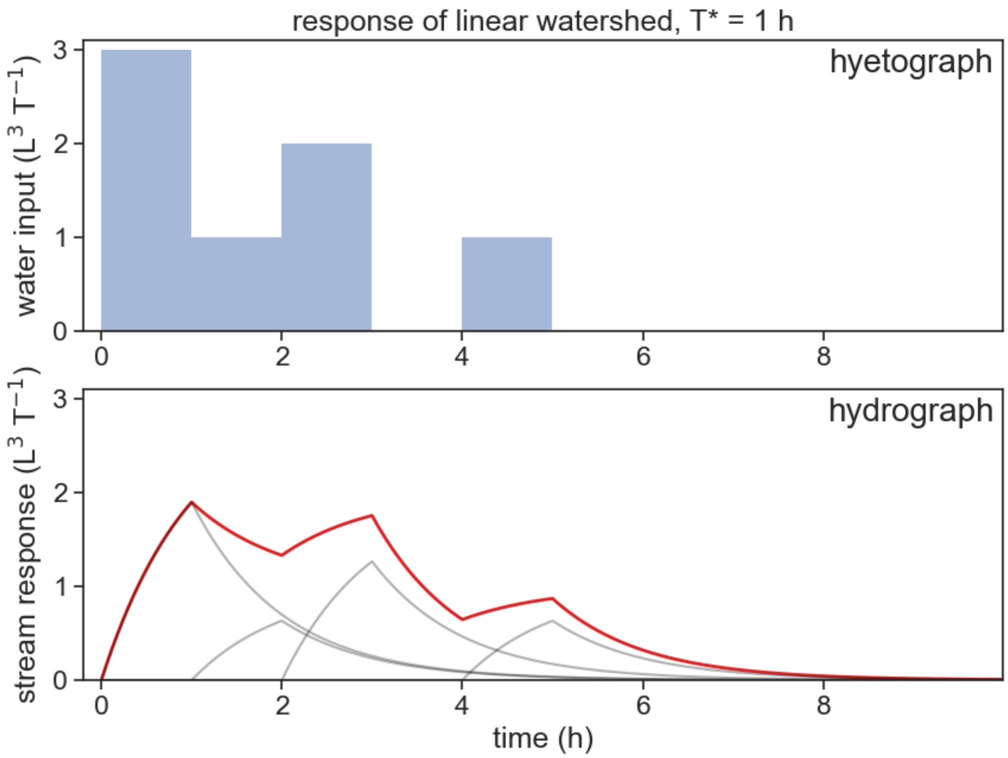
$$T_{eq} = -\ln(0.01) \cdot T^* = 4.605 \cdot T^*. \qquad \textbf{(10B3.8)}$$

Figure 17.1: Source: Dingman (2015), page 472


response of linear watershed, T* = 1 h
hyetograph
water input (L^3 T^{-1})
1.0
0.5
0.0
0
2
4
6
8
hydrograph
stream response (L^3 T^{-1})
1.0
0.5
0.0
0
2
4
6
8
time (h)


response of linear watershed, T* = 2 h
hyetograph
water input (L^3 T^{-1})
1.0
0.5
0.0
0
2
4
6
8
hydrograph
stream response (L^3 T^{-1})
T* = 2 h
decay of 37% = exp(−1)
time (h)


response of linear watershed, T* = 1 h
hyetograph
water input (L^3 T^{-1})
hydrograph
stream response (L^3 T^{-1})
time (h)
0
1
2
3
0
2
4
6
8



17.2 Rainfall-Runoff Models

17.2.1 The Rational Method

The rational method postulates a simple proportionality between peak discharge, q_{pk}, and rainfall intensity, p^*:

$$q_{pk} = \varepsilon_R \cdot C_R \cdot A_D \cdot p^*$$

- q_{pk}: peak discharge (m^3/s)
- $\varepsilon_R = 0.278$: unit-conversion factor
- C_R: dimensionless runoff coefficient
- A_D: drainage area (km^2)
- p^*: rainfall intensity (mm/h)

  Obviously the results obtained with the method are highly sensitive to the value chosen for CR; values range from 0.05 for gently sloping lawns up to 0.95 for highly

urbanized areas of roofs and pavement.
The rational method is widely used in urban drainage design, but Pilgrim and Cordery (1992) caution that there are typically few data available to guide the selection of CR, and that CR for a given watershed may vary widely from storm to storm due to differing antecedent conditions.

17.2.2 The Soil Conservation Service Curve-Number Method (SCS-CN)

Also called NRCS curve number procedure. NRCS = Natural Resources Conservation Service - USDA

$$Q^* = P^* = \frac{(P - S_I)^2}{P - S_I + S_{max}},$$

where

- Q^*: total volume of event flow in a storm event
- P^*: total volume of effective water input in a storm event
- P: total volume of water input in a storm event
- S_I: initial abstraction = amount of rainfall that is retained or absorbed by the land surface before runoff occurs
- S_{max}: potential maximum retention = maximum amount of water that the watershed can retain after runoff starts

The initial abstraction S_I is usually approximated as $0.2 \cdot S_{max}$, therefore:

$$Q^* = P^* = \frac{(P - 0.2 \cdot S_{max})^2}{P + 0.8 \cdot S_{max}}$$

$$S_{max} = 25.4 \left(\frac{1000}{CN} - 10 \right)$$

The number 25.4 is a conversion factor from inches to millimeters.

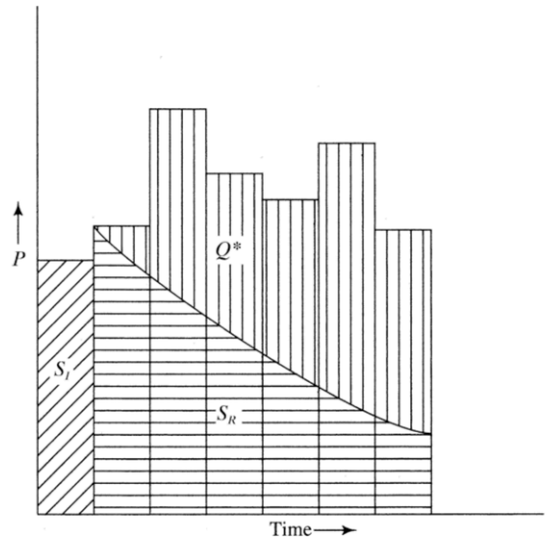


Figure 10.51 Definitions of initial abstraction, S_I, retention, S_R, and event flow, Q^*, in the SCS method.

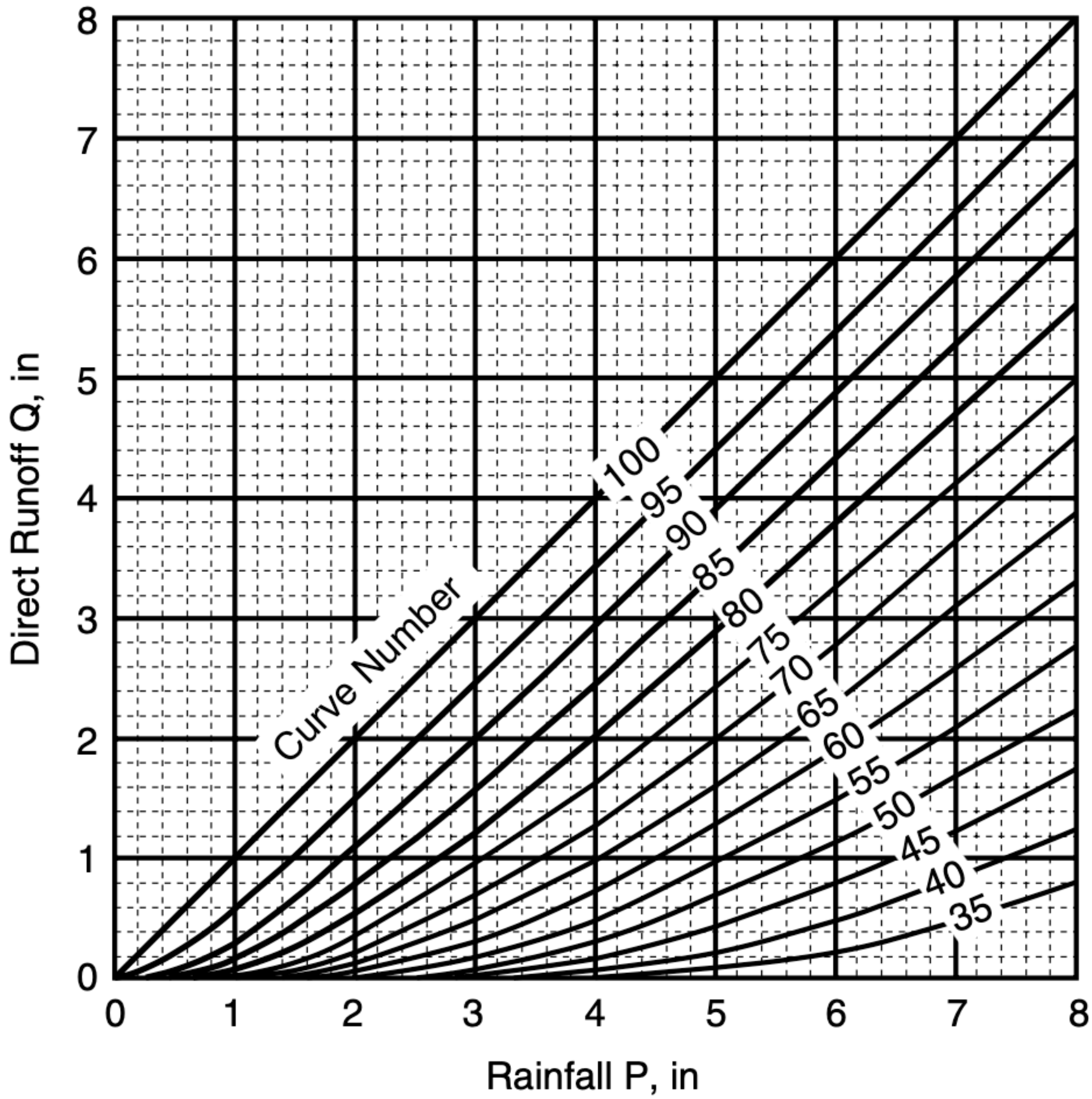


FIGURE 5.15 Relationships of runoff to rainfall based on NRCS curve number method.

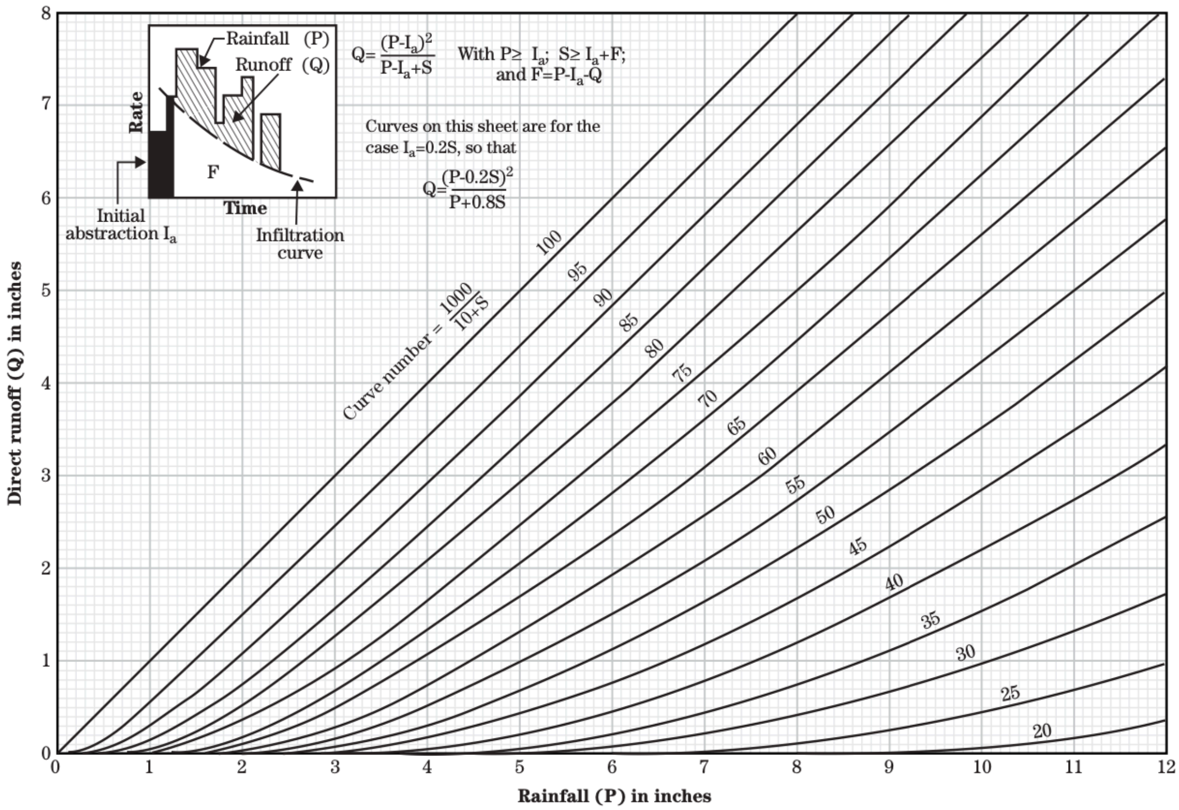


Figure 17.2: Source: United States Department of Agriculture (2004)

The curve number (CN) is a function of the ability of soils to infiltrate water, land use, and the soil water conditions at the start of a rainfall event (antecedent soil water condition). To account for the infiltration characteristics of soils, the NRCS has divided soils into four hydrologic soil groups, which are defined as follows (NRCS, 1984):

- **Group A** (low runoff potential): Soils with high infiltration rates even when thoroughly wetted. These consist chiefly of deep, well-drained sands and gravels. These soils have a high rate of water transmission (final infiltration rate greater than 0.3 in./h).
- **Group B**: Soils with moderate infiltration rates when thoroughly wetted. These consist chiefly of soils that are moderately deep to deep, moderately well drained to well drained with moderately fine to moderately coarse textures. These soils have a moderate rate of water transmission (final infil- tration rate 0.15 to 0.30 in./h).
- **Group C**: Soils with slow infiltration rates when thoroughly wetted. These consist chiefly of soils with a layer that impedes downward movement of water or soils with moderately fine to fine texture. These soils have a slow rate of water transmission (final infiltration rate 0.05 to 0.15 in./h).
- **Group D** (high runoff potential): Soils with very slow infiltration rates when thoroughly

wetted. These consist chiefly of clay soils with a high swelling potential, soils with a permanent high water table, soils with a claypan or clay layer at or near the surface, and shallow soils over nearly impervious materials. These soils have a very slow rate of water transmission (final infiltration rate less than 0.05 in./h).

There are also three categories for Antecedent Soil Moisture Condition (AMC):

- **AMC I**: Dormant season antecedent soil moisture less than 0.5 in. Growing season antecedent soil moisture less than 1.4 in.
- **AMC II**: Dormant season antecedent soil moisture between 0.5 and 1.1 in. Growing season anteced- ent soil moisture between 1.4 and 2.1 in.
- **AMC III**: Dormant season antecedent soil mois- ture greater than 1.1 in. Growing season anteced- ent soil moisture greater than 2.1 in.

See the table below to find curve numbers for AMC II:

TABLE 5.1
Curve Numbers for Antecedent Soil

Land Use Description
Commercial, row houses and townhouses
Fallow, poor condition
Cultivated with conventional tillage
Cultivated with conservation tillage
Lawns, poor condition
Lawns, good condition
Pasture or range, poor condition
Pasture or range, good condition
Meadow
Pavement and roofs
Woods or forest thin stand, poor cover
Woods or forest, good cover
Farmsteads
Residential quarter-acre lot, poor condition
Residential quarter-acre lot, good condition
Residential half-acre lot, poor condition
Residential half-acre lot, good condition
Residential 2-acre lot, poor condition
Residential 2-acre lot, good condition
Roads

Source: From NRCS, 1984.

```
P=21
ratio = 4.17e4/2.61e5
CN=86
Smax = 25.4 * (1000/CN - 10)
```

```
Pmin = 0.2 * Smax
Qstar = 0.0
if P > Pmin:
    Qstar = (P - 0.2*Smax)**2 / (P+0.8*Smax)
Qstar/P
```

```
0.14270006393832066
```

```
ratio
```

```
0.15977011494252874
```

```
Qstar / P
```

```
0.9148811393863234
```

```
%matplotlib notebook

import numpy as np
import matplotlib.pyplot as plt

def Qstar_f(pe, CN):
#     Smax = 25.4*(1000/CN - 10)
    Smax = (1000/CN - 10)
#     Smax = (1000/CN - 10) / 25.4
    Qstar = (pe - 0.2*Smax)**2 / (pe+0.8*Smax)
    return Qstar

pe = np.linspace(0,8,101)
# plt.plot(pe, Qstar_f(pe, 35))
plt.plot(pe, Qstar_f(pe, 50))
# plt.plot(pe, Qstar_f(pe, 85))
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Part VI

summing up

18 Budyko framework

Sources used:

Daly et al. (2019), Sposito (2017), Jones et al. (2012), Krajewski et al. (2021), Berghuijs, Gnann, and Woods (2020), Creed and Spargo (2012)

18.1 Water and surface energy balances

For long-term averages:

$$P = ET + Q$$

$$R_n = \lambda_w \cdot ET + H$$

- P: precipitation (L T^{-1}, e.g.: mm/day)
- ET: evapotranspiration (L T^{-1})
- Q: streamflow (L T^{-1})
- R_n: net energy available at soil surface (M T^{-3}, e.g.: W m^{-2})
- λ_w: latent heat of vaporization of water (M $\mathrm{L}^{-1}\mathrm{T}^{-2}$, as defined here, the units will be weird)
- H: sensible heat flux from the surface into the atmosphere (M T^{-3})
- $\lambda_w \cdot ET$: latent heat flux (M T^{-3})

18.2 Assumptions

1. because we are dealing with long-term averages, there are negligible changes of watershed stored water.
2. negligible energy is stored at the soil surface, and heat transfer from soil surface to deeper soil layers (G) averages zero.

18.3 Question

Given measurements of rainfall and meteorological conditions, can we predict the partitioning of P between ET and Q?

18.4 Limits

For very dry watersheds (deserts, for example), almost all precipitation (P) is lost via evapotranspiration (ET). These watersheds are called water limited.

In wet watersheds, at the annual scale, the sensible heat (H) is directed from the surface to the atmosphere in almost all climatic zones on Earth (meaning: soil heats air). Therefore, H cannot supply much energy to the soil surface, and it is assumed that R_n provides entirely the energy required for evapotranspiration. Dividing the second equation by λ_w, we get $R_n/\lambda_w = ET + H/\lambda_w$. It is clear that the maximum possible ET occurs when all incoming radiation energy R_n is consumed by evapotranspiration ET, and there is negligible sensible heat flux H. As a result, the upper limit of $\lambda_w E$ is R_n, in wet watersheds. In these watersheds, called energy limited, ET tends to the potential evapotranspiration (ET_0).

18.4.1 Summary:

For energy-limited watersheds

(1) As precipitation $P \to \infty$, evapotranspiration $ET \to ET_0$

For water-limited watersheds

(2) As potential evapotranspiration $ET_0 \to \infty$, actual evaporation $ET \to P$

In general, we can write

$$ET = f(P, ET_0)$$

The variables P and ET have the same dimenstions (L T^{-1}), and we can divide the equation above by P:

$$\frac{ET}{P} = f(D_I),$$

where

$$D_I = \frac{ET_0}{P}$$

is called the **dryness index**. A useful classification is

Dryness Index	Classification
$D_I < 1.54$	Humid
$1.54 < D_I < 2$	Dry Subhumid
$2 < D_I < 5$	Semi-arid
$5 < D_I < 20$	Arid
$20 < D_I$	Hyper-arid

ATTENTION. The dryness index can also be called the "Aridity Index" (AI), **however** sometimes the AI means the inverse of D_I:

$$AI = 1/D_I$$

Be careful to check the definitions.

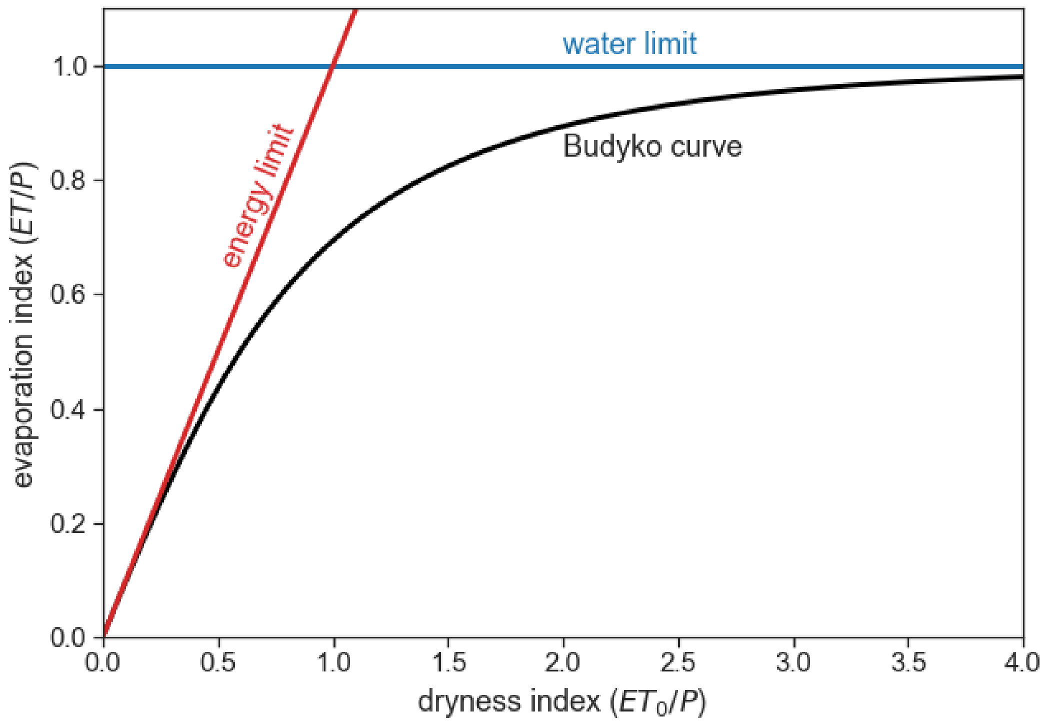
The summary (1) and (2) above can be now represented as:

(1) As $D_I \to 0$, $\dfrac{ET}{P} \to D_I$

(2) As $D_I \to \infty$, $\dfrac{ET}{P} \to 1$

Budyko (1974), proposed the following equation:

$$\frac{ET}{P} = \left[D_I \tanh\left(\frac{1}{D_I}\right)\left(1 - e^{-D_I}\right)\right]^{1/2}$$



water limit
energy limit
Budyko curve
evaporation index (ET/P)
dryness index (ET_0/P)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

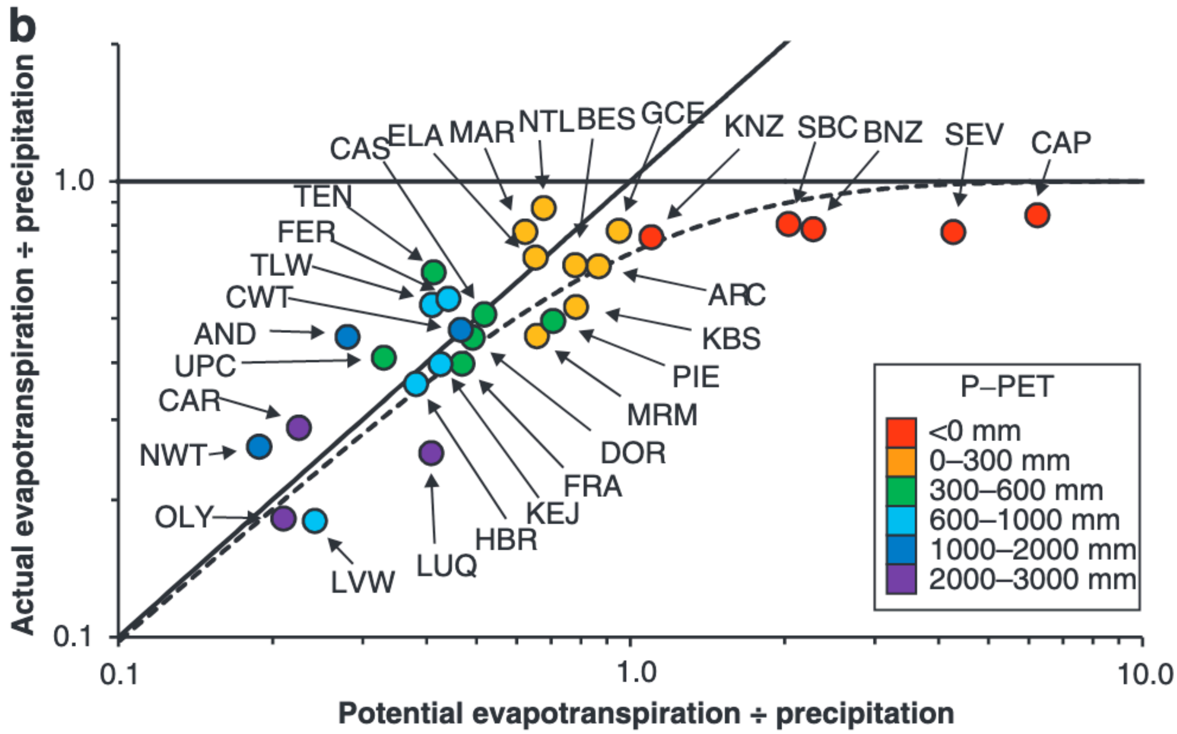


Figure 18.1: Source: Jones et al. (2012)

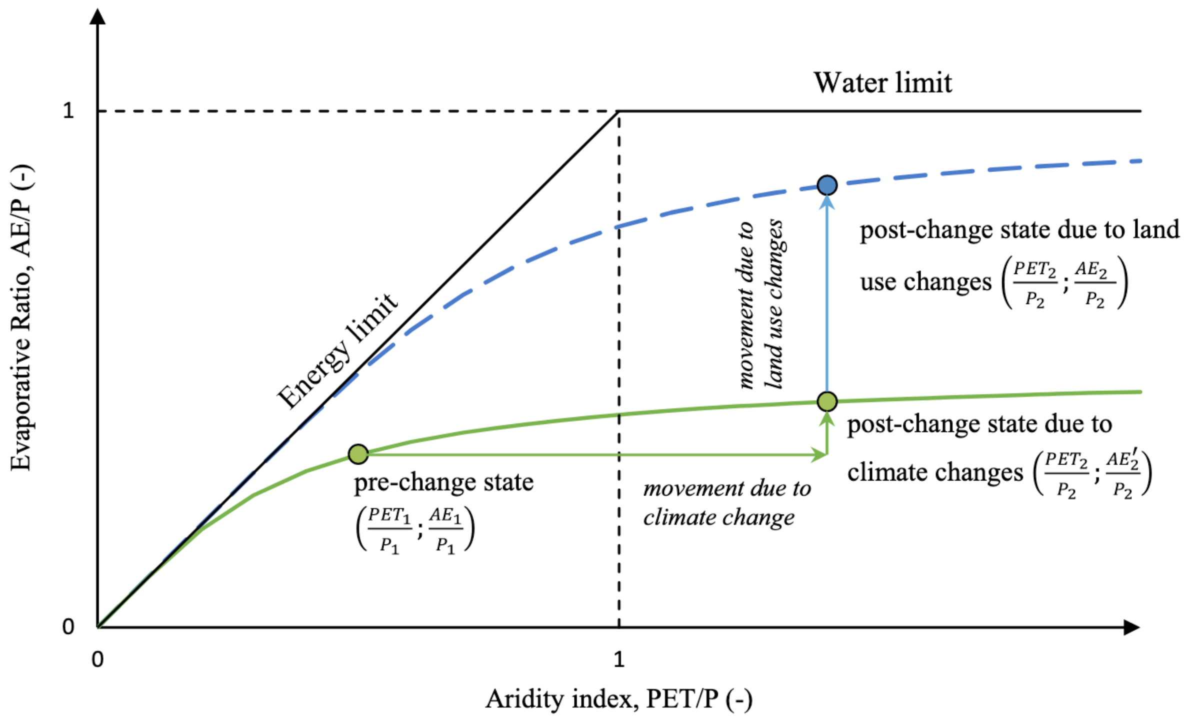


Figure 18.2: Source: Krajewski et al. (2021)

There are many alternatives to Budyko's equation. Many equations have adjustable parameters, such as Fu's equation:

$$\frac{ET}{P} = 1 + D_I - (1 + D_I^w)^{1/w},$$

where $w > 1$. Each catchment has its own specific parameter w, that may represent bio-physical/landscape features. There is no concensus regarding the interpretation of w, ranging from an effective empirical parameter, whose relationship to biophysical features can be discerned, to an arbitrary empirical constant with no *a priori* physical meaning. Source: {% cite reaver2020reinterpreting %}

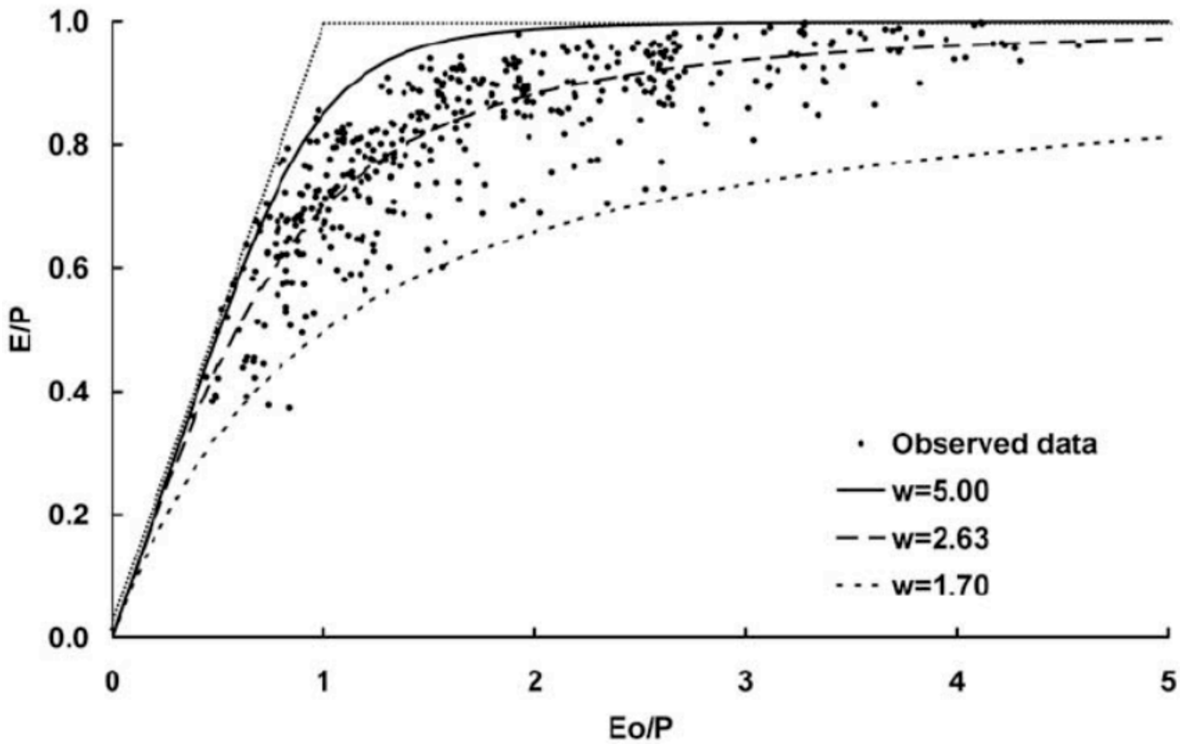


Figure 18.3: Source: Zhang et al. (2004)

18.5 Hypotheses for why dryness index controls so much the partitioning of P into ET and Q

Source: Berghuijs, Gnann, and Woods (2020)

1. The first is that the Budyko curve is accurate because landscape features (e.g., soils and vegetation) coevolve with the local climate in such a manner that precipitation partitioning into streamflow and evapotranspiration converges towards the Budyko curve
2. A second hypothesis is that catchments over time evolve towards the supply and demand limits (rather than towards a curve), because landscapes and their vegetation are unaware of the Budyko curve but do evolve to maximize their use of available resources (including water). However, because limiting factors such as climatic variability exist (which will reduce a catchment's ability to use all water because it cannot fully buffer the highly variable precipitation input), catchments will tend to not reach these limits. This may lead to an (apparent) existence of the Budyko curve which falls relatively close to the demand and supply limits.
3. A third hypothesis is that the existence of a strong universal relationship between aridity and catchment water balances might be explained by an underlying organizing principle

such as maximum entropy production because the Budyko curve may be consistent with how hydrologic systems optimally partition water and energy
4. A fourth hypothesis is that virtually any landscape and climate combination (also those in heavily disturbed landscapes: e.g., a city, agricultural lands, etc.) will fall near the Budyko curve because climate aridity will dominate precipitation partitioning largely independent of the climate-landscape configuration or any optimization principle.

18.6 Hypotheses for deviations from Budyko curve

Source: Creed and Spargo (2012)

1. Under stationary conditions (naturally occurring oscillations), catchments will fall on the Budyko Curve
2. Under non-stationary conditions (anthropogenic climate change), catchments will deviate from the Budyko Curve in a predictable manner

18.6.1 Reasons for falling off the Budyko Curve

1. Inadequate representation of P and T (Loch Vale)
2. Inadequate representation of ET (Andrews)
3. Inadequate representation of Q (Marcell)
4. Forest conversion (Coweeta)
5. Forest disturbance (Luquillo)

18.6.2 Critique

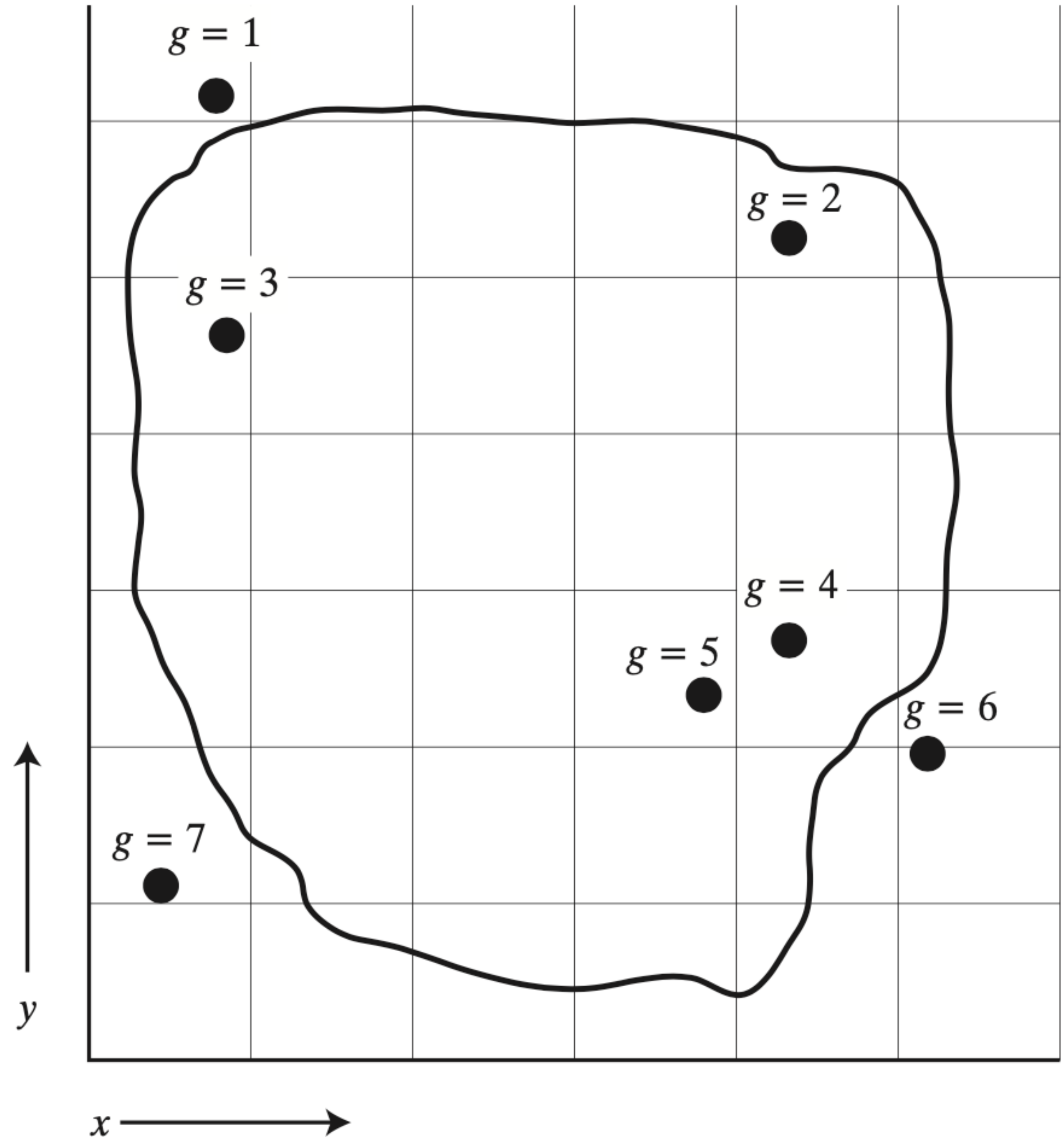
Source: Berghuijs, Gnann, and Woods (2020)

The (mathematical) specifics of such studies vary, but all approaches are founded on the assumption that catchments follow a (parametric) Budyko curve when aridity changes, and that consequently all other movements in the Budyko space are caused by other factors. The validity of this assumption remains mostly untested, which seems surprising given it underpins all of these studies' findings.

19 Spatial distribution - lecture

19.1 The problem

Let's say we want to calculate the average rainfall on a watershed, and we have data available for 7 stations, as shown in the figure below [Dingman, figure 4.26]:



There are a number of methods for calculating the average precipitation.

19.2 Thiessen method [Voronoi diagram]

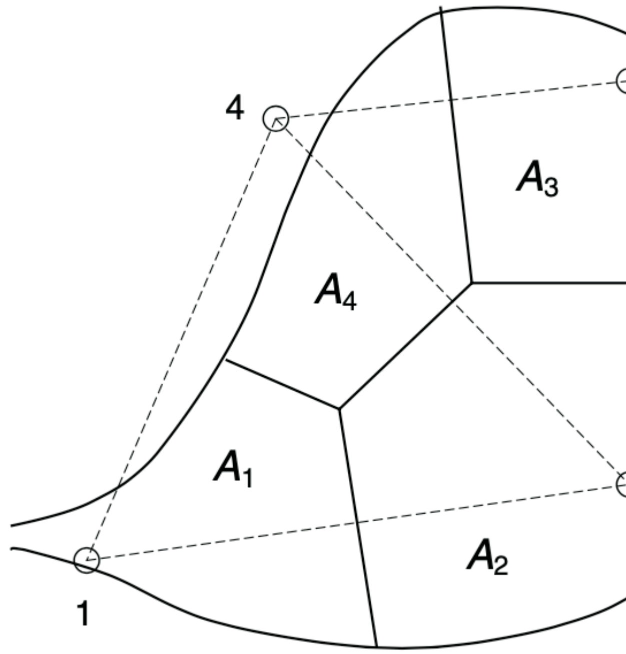


Fig. 3.11 **Sketch illustrating the application of the Thiessen polygon method to estimate the subareas A_i assigned to the precipitation gages on the map of a catchment. The subareas are bounded by the boundaries of the catchment and by the lines drawn midway between the stations. The locations of the stations are indicated by the numbered circles.**

Brutsaert (2005), Figure 3.11

How to compute the areas:

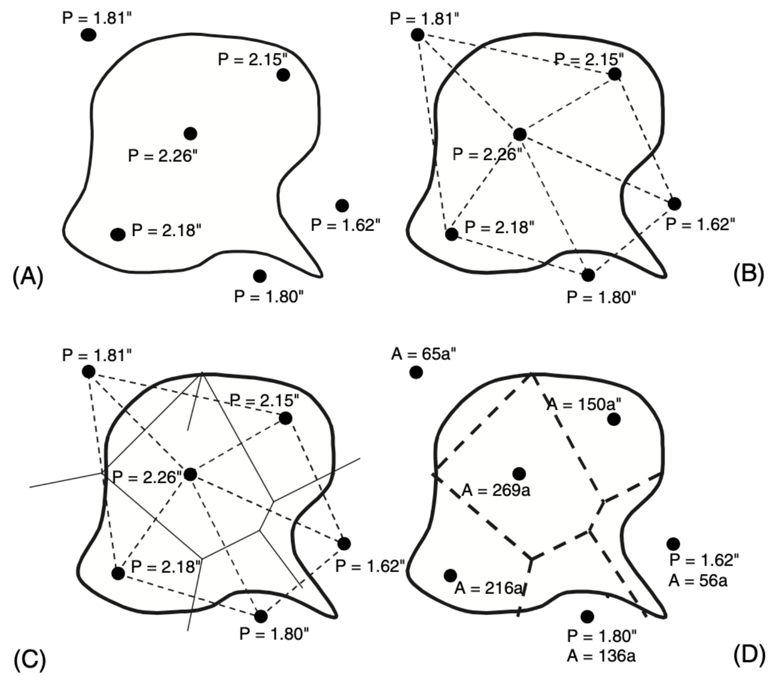
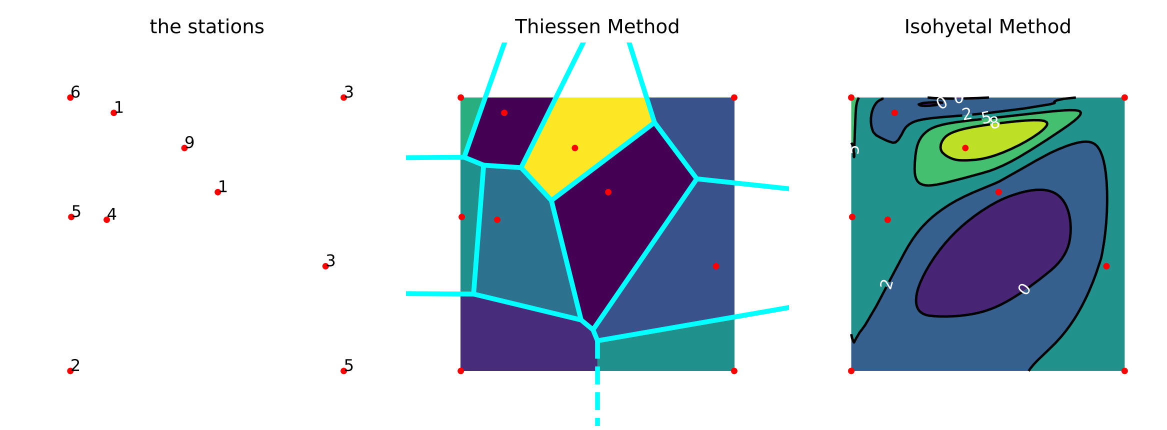


FIGURE 2.15 Use of Thiessen method to find average rainfall: (A) distribution of rain gages in a watershed locate (B) connection lines drawn between rain gage positions; (C) lines perpendicular to connection lines drawn until they form polygons; and (D) areas measured for each polygon.

Average areal precipitation is a weighted sum:

$$\langle P \rangle = \frac{\sum_i A_i P_i}{\sum_i A_i}$$

A nice way to understand the Thiessen method is depicted in the gif below (from Wikipedia):



:::

Part VII

Appendix

Welcome to extra stuff at the end of this book.

20 Gain full control of date formatting

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime
from datetime import timedelta
import seaborn as sns
sns.set(style="ticks", font_scale=1.5)
import matplotlib.gridspec as gridspec
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
```

```
import pandas as pd

start_date = '2018-01-01'
end_date = '2018-04-30'

# create date range with 1-hour intervals
dates = pd.date_range(start_date, end_date, freq='1H')
# create a random variable to plot
var = np.random.randint(low=-10, high=11, size=len(dates)).cumsum()
var = var - var.min()
# create dataframe, make "date" the index
df = pd.DataFrame({'date': dates, 'variable': var})
df.set_index(df['date'], inplace=True)
df
```

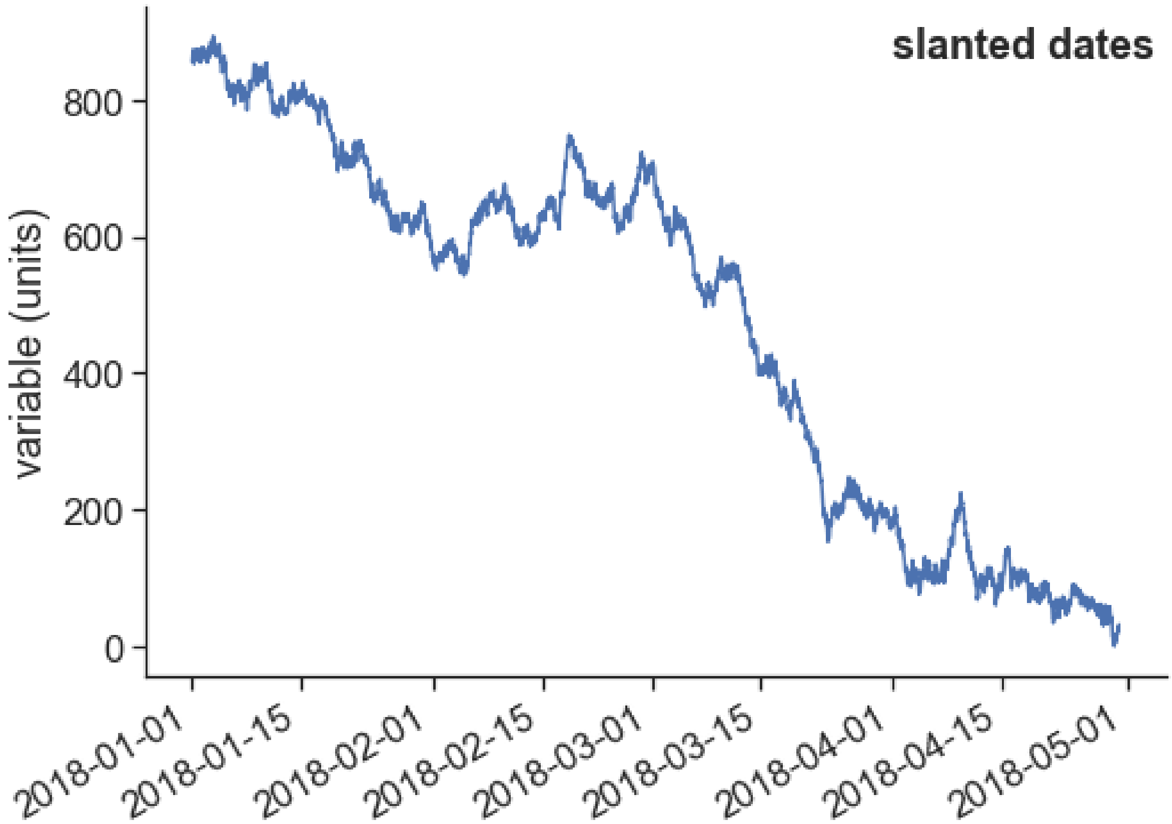
	date	variable
date		
2018-01-01 00:00:00	2018-01-01 00:00:00	856
2018-01-01 01:00:00	2018-01-01 01:00:00	863
2018-01-01 02:00:00	2018-01-01 02:00:00	867

	date	variable
date		
2018-01-01 03:00:00	2018-01-01 03:00:00	874
2018-01-01 04:00:00	2018-01-01 04:00:00	864
...	...	...
2018-04-29 20:00:00	2018-04-29 20:00:00	20
2018-04-29 21:00:00	2018-04-29 21:00:00	20
2018-04-29 22:00:00	2018-04-29 22:00:00	27
2018-04-29 23:00:00	2018-04-29 23:00:00	23
2018-04-30 00:00:00	2018-04-30 00:00:00	32

define a useful function to plot the graphs below

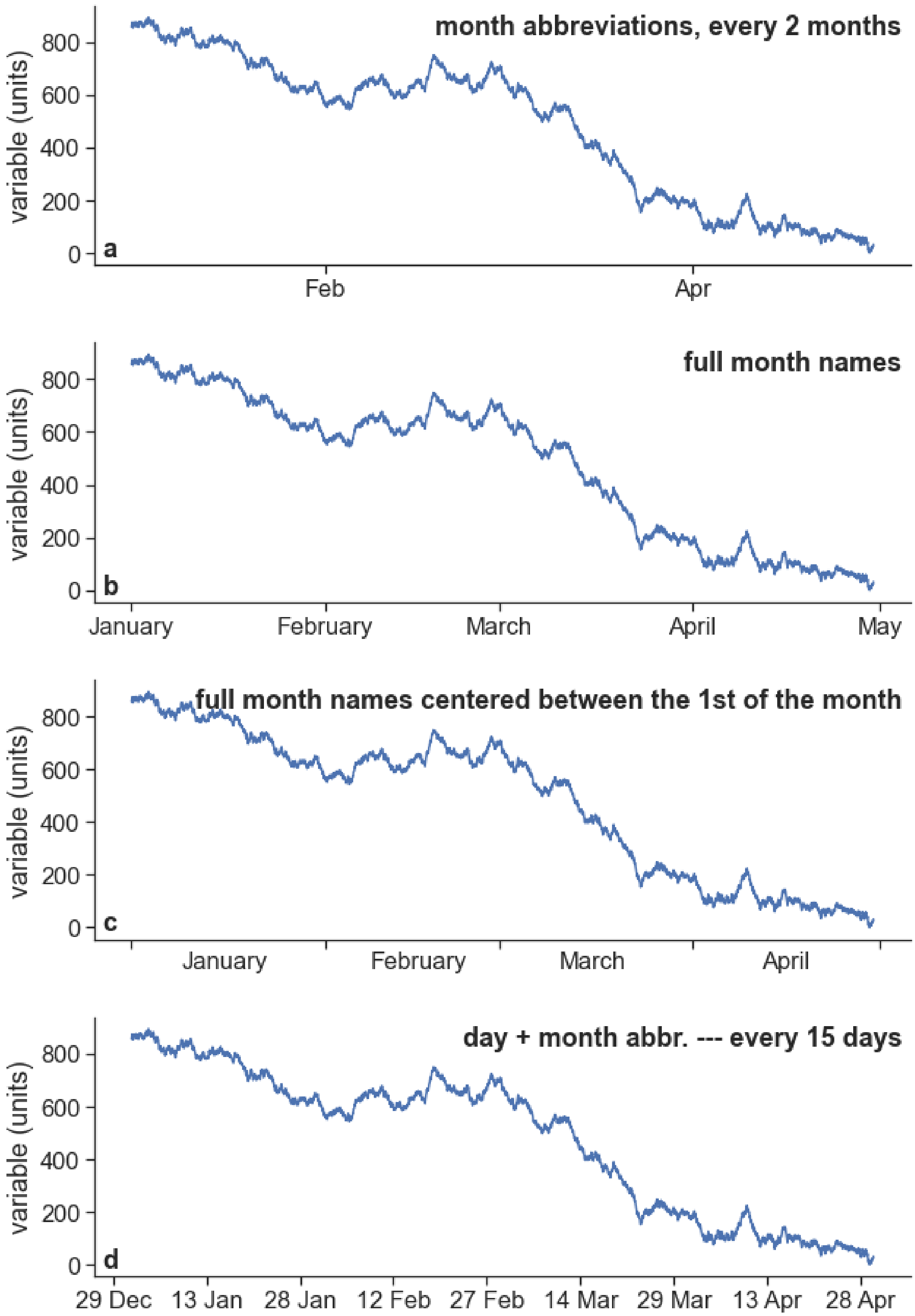
```
def explanation(ax, text, letter):
    ax.text(0.99, 0.97, text,
            transform=ax.transAxes,
            horizontalalignment='right', verticalalignment='top',
            fontweight="bold")
    ax.text(0.01, 0.01, letter,
            transform=ax.transAxes,
            horizontalalignment='left', verticalalignment='bottom',
            fontweight="bold")
    ax.set(ylabel="variable (units)")
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 6))
ax.plot(df['variable'])
plt.gcf().autofmt_xdate()  # makes slated dates
explanation(ax, "slanted dates", "")
fig.savefig("dates1.png")
```

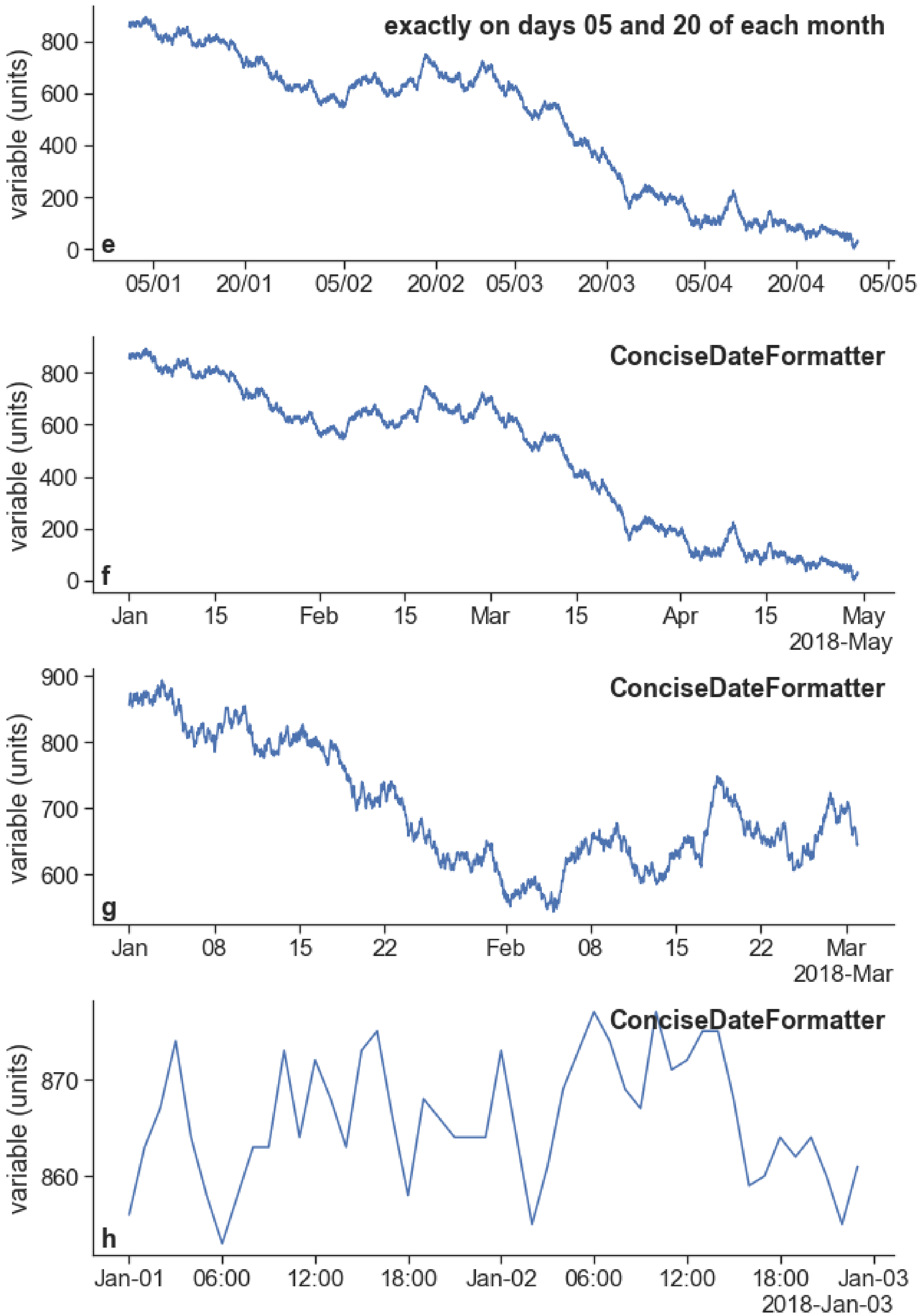



```
fig, ax = plt.subplots(4, 1, figsize=(10, 16),
                       gridspec_kw={'hspace': 0.3})

### plot a ###
ax[0].plot(df['variable'])
date_form = DateFormatter("%b")
ax[0].xaxis.set_major_locator(mdates.MonthLocator(interval=2))
ax[0].xaxis.set_major_formatter(date_form)

### plot b ###
ax[1].plot(df['variable'])
date_form = DateFormatter("%B")
ax[1].xaxis.set_major_locator(mdates.MonthLocator(interval=1))
ax[1].xaxis.set_major_formatter(date_form)

### plot c ###
ax[2].plot(df['variable'])
ax[2].xaxis.set_major_locator(mdates.MonthLocator())
```

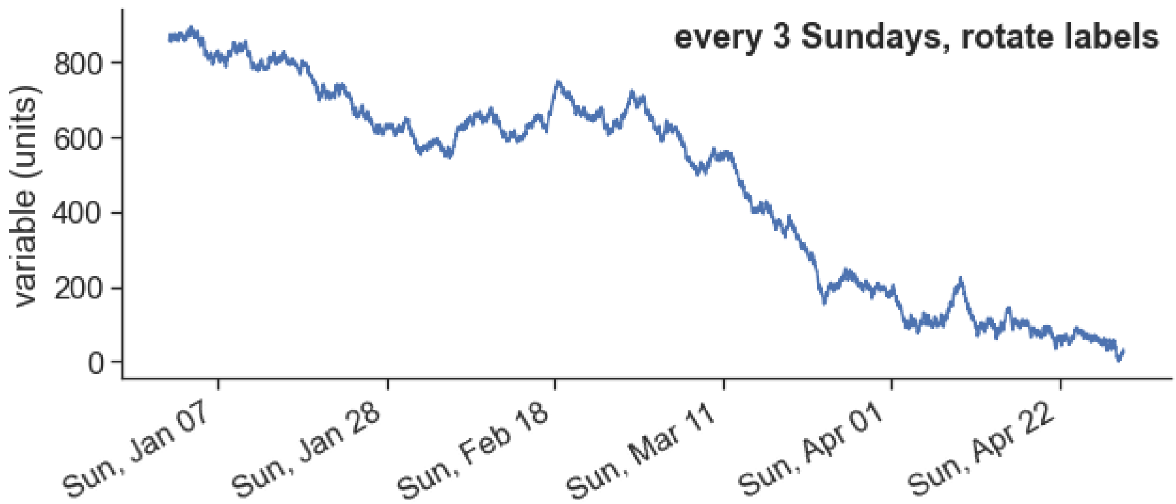
```
# 16 is a slight approximation for the center, since months differ in number of days.
ax[2].xaxis.set_minor_locator(mdates.MonthLocator(bymonthday=16))
ax[2].xaxis.set_major_formatter(ticker.NullFormatter())
ax[2].xaxis.set_minor_formatter(DateFormatter('%B'))
for tick in ax[2].xaxis.get_minor_ticks():
    tick.tick1line.set_markersize(0)
    tick.tick2line.set_markersize(0)
    tick.label1.set_horizontalalignment('center')

### plot d ###
ax[3].plot(df['variable'])
date_form = DateFormatter("%d %b")
ax[3].xaxis.set_major_locator(mdates.DayLocator(interval=15))
ax[3].xaxis.set_major_formatter(date_form)

explanation(ax[0], "month abbreviations, every 2 months", "a")
explanation(ax[1], "full month names", "b")
explanation(ax[2], "full month names centered between the 1st of the month", "c")
explanation(ax[3], "day + month abbr. --- every 15 days", "d")

fig.savefig("dates2.png")
```


month abbreviations, every 2 months
variable (units)
800
600
400
200
0
a
Feb
Apr
full month names
variable (units)
800
600
400
200
0
b
January
February
March
April
May
full month names centered between the 1st of the month
variable (units)
800
600
400
200
0
c
January
February
March
April
day + month abbr. --- every 15 days
variable (units)
800
600
400
200
0
d
29 Dec
13 Jan
28 Jan
12 Feb
27 Feb
14 Mar
29 Mar
13 Apr
28 Apr

```
fig, ax = plt.subplots(4, 1, figsize=(10, 16),
                       gridspec_kw={'hspace': 0.3})

### plot e ###
ax[0].plot(df['variable'])
date_form = DateFormatter("%d/%m")
ax[0].xaxis.set_major_locator(mdates.DayLocator(bymonthday=[5, 20]))
ax[0].xaxis.set_major_formatter(date_form)

### plot f ###
ax[1].plot(df['variable'])
locator = mdates.AutoDateLocator(minticks=11, maxticks=17)
formatter = mdates.ConciseDateFormatter(locator)
ax[1].xaxis.set_major_locator(locator)
ax[1].xaxis.set_major_formatter(formatter)

### plot g ###
ax[2].plot(df.loc['2018-01-01':'2018-03-01', 'variable'])
locator = mdates.AutoDateLocator(minticks=6, maxticks=14)
formatter = mdates.ConciseDateFormatter(locator)
ax[2].xaxis.set_major_locator(locator)
ax[2].xaxis.set_major_formatter(formatter)

### plot h ###
ax[3].plot(df.loc['2018-01-01':'2018-01-02', 'variable'])
locator = mdates.AutoDateLocator(minticks=6, maxticks=10)
formatter = mdates.ConciseDateFormatter(locator)
ax[3].xaxis.set_major_locator(locator)
ax[3].xaxis.set_major_formatter(formatter)

explanation(ax[0], "exactly on days 05 and 20 of each month", "e")
explanation(ax[1], "ConciseDateFormatter", "f")
explanation(ax[2], "ConciseDateFormatter", "g")
explanation(ax[3], "ConciseDateFormatter", "h")

fig.savefig("dates3.png")
```


exactly on days 05 and 20 of each month
variable (units)
0
200
400
600
800
e
05/01
20/01
05/02
20/02
05/03
20/03
05/04
20/04
05/05
ConciseDateFormatter
variable (units)
0
200
400
600
800
f
Jan
15
Feb
15
Mar
15
Apr
15
May
2018-May
ConciseDateFormatter
variable (units)
600
700
800
900
g
Jan
08
15
22
Feb
08
15
22
Mar
2018-Mar
ConciseDateFormatter
variable (units)
860
870
h
Jan-01
06:00
12:00
18:00
Jan-02
06:00
12:00
18:00
Jan-03
2018-Jan-03

```
fig, ax = plt.subplots(1, 1, figsize=(10, 4),
                       gridspec_kw={'hspace': 0.3})

# import constants for the days of the week
from matplotlib.dates import MO, TU, WE, TH, FR, SA, SU
ax.plot(df['variable'])
# tick on sundays every third week
loc = mdates.WeekdayLocator(byweekday=SU, interval=3)
ax.xaxis.set_major_locator(loc)
date_form = DateFormatter("%a, %b %d")
ax.xaxis.set_major_formatter(date_form)
fig.autofmt_xdate(bottom=0.2, rotation=30, ha='right')
explanation(ax, "every 3 Sundays, rotate labels", "")
```



Code	Explanation
%Y	4-digit year (e.g., 2022)
%y	2-digit year (e.g., 22)
%m	2-digit month (e.g., 12)
%B	Full month name (e.g., December)
%b	Abbreviated month name (e.g., Dec)
%d	2-digit day of the month (e.g., 09)
%A	Full weekday name (e.g., Tuesday)
%a	Abbreviated weekday name (e.g., Tue)
%H	24-hour clock hour (e.g., 23)
%I	12-hour clock hour (e.g., 11)
%M	2-digit minute (e.g., 59)

Code	Explanation
%S	2-digit second (e.g., 59)
%p	“AM” or “PM”
%Z	Time zone name
%z	Time zone offset from UTC (e.g., -0500)

21 Summary

In summary, this book has no content whatsoever.

This is Yair making changes to summary.

References

Allen, Richard G, Luis S Pereira, Dirk Raes, Martin Smith, et al. 1998. "Crop Evapotranspiration-Guidelines for Computing Crop Water Requirements-FAO Irrigation and Drainage Paper 56." *Fao, Rome* 300 (9): D05109.

Amazon Waters. 2022. "Amazon Waters." *Amazon Waters*. https://amazonwaters.org/basins.

Berghuijs, Wouter R, Sebastian J Gnann, and Ross A Woods. 2020. "Unanswered Questions on the Budyko Framework." *Journal of Hydrology* 265: 164–77.

Brutsaert, Wilfried. 2005. *Hydrology: An Introduction*. Cambridge University Press.

Budyko, Mikhail Ivanovich. 1974. "Climate and Life." *(No Title)*.

Creed, Irena, and Adam Spargo. 2012. "Budyko Guide to Exploring Sustainability of Water Yields from Catchments Under Changing Environmental Conditions." *London, Ontario. Http://Www. Uwo. Ca/Biology/Faculty/Creed/PDFs/Presentations/PRE116. Pdf*.

Daly, Edoardo, Salvatore Calabrese, Jun Yin, and Amilcare Porporato. 2019. "Linking Parametric and Water-Balance Models of the Budyko and Turc Spaces." *Advances in Water Resources* 134: 103435.

Dingman, S. L. 2015. *Physical Hydrology*. 3rd edition. Waveland Press, Incorporated.

dreamstime. 2022. "World Map of AFRICA." *Dreamstime*. https://www.dreamstime.com/world-map-africa-egypt-libya-ethiopia-arabia-mauritania-nigeria-somalia-namibia-tanzania-madagascar-geographic-xxl-chart-image154799901.

Fiona Bruce. 2015. "A Family Holiday in Lake Malawi: Zen and the Art of Paddleboarding." *The Telegraph*. https://twitter.com/hallaboutafrica/status/1203419359303159809?s=20&t=SkH17UkWrNcXzIqRF0ic_A.

Hillel, Daniel. 2003. *Introduction to Environmental Soil Physics*. Elsevier.

James Hall. 2019. "Lake Malawi." *Twitter*. https://twitter.com/hallaboutafrica/status/1203419359303159809?s=20&t=SkH17UkWrNcXzIqRF0ic_A.

Jones, Julia A, Irena F Creed, Kendra L Hatcher, Robert J Warren, Mary Beth Adams, Melinda H Benson, Emery Boose, et al. 2012. "Ecosystem Processes and Human Influences Regulate Streamflow Response to Climate Change at Long-Term Ecological Research Sites." *BioScience* 62 (4): 390–404.

Krajewski, Adam, Anna E Sikorska-Senoner, Leszek Hejduk, and Kazimierz Banasik. 2021. "An Attempt to Decompose the Impact of Land Use and Climate Change on Annual Runoff in a Small Agricultural Catchment." *Water Resources Management* 35 (3): 881–96.

leddris. 2010. "Rainfall Seasonality." *Land and Ecosystem Degradation and Desertification Response Information System*. http://leddris.aegean.gr/ses-parameters/293-rainfall-seasonality.html#:~:text=Rainfall%20seasonality%20index%20is%20a,in%20relation%20to%20water%20availability.

Margulis, Steve. 2019. "Introduction to Hydrology. eBook." https://margulis-group.github.io/textbook/.

Marty Friedlander. 2015. "Natural Springs of Israel: Seven Cool Watering Holes to Visit This Summer." *Haaretz*. https://www.haaretz.com/israel-news/travel/seven-cool-natural-springs-of-israel-1.5388627.

National Park Service. 2022. "Mississippi River Facts." *National Park Service*. https://www.nps.gov/miss/riverfacts.htm.

Raymond, Lyle S. Jr. 1988. "What Is Groundwater?" *Cornell eCommons*. https://ecommons.cornell.edu/handle/1813/3408.

Sposito, Garrison. 2017. "Understanding the Budyko Equation." *Water* 9 (4): 236.

Suma Groulx. 2015. "Water Infiltration." *Suma Groulx*. http://sumagroulx.com/water-infiltration/.

United States Department of Agriculture, Natural Resources Conservation Service. 2004. *Estimation of Direct Runoff from Storm Rainfall*. National Engineering Handbook, Part 630 Hydrology, Chapter 10. United States Department of Agriculture. https://directives.sc.egov.usda.gov/OpenNonWebContent.aspx?content=17752.wba.

Valentí Rodellas. 1988. "Evaluating Submarine Groundwater Discharge to the Mediterranean Sea by Using Radium Isotopes." *Research Gate*. https://www.researchgate.net/figure/Principal-pathways-for-submarine-groundwater-discharge-to-the-coastal-ocean-including_fig1_274590439.

Walsh, RPD, and DM Lawler. 1981. "Rainfall Seasonality: Description, Spatial Patterns and Change Through Time." *Weather* 36 (7): 201–8. https://doi.org/10.1002/j.1477-8696.1981.tb05400.x.

Ward, Andy D, and Stanley W Trimble. 2003. *Environmental Hydrology*. 2nd ed. CRC Press.

Water Science School. 2016. "Water Flowing Underground Can Find Openings Back to the Land Surface." *U.S. Geological Survey*. https://www.usgs.gov/media/images/water-flowing-underground-can-find-openings-back-land-surface.

———. 2018. "Where Is Earth's Water?" *U.S. Geological Survey*. https://www.usgs.gov/special-topics/water-science-school/science/where-earths-water.

———. 2019a. "Conceptual Groundwater-Flow Diagram." *U.S. Geological Survey*. https://www.usgs.gov/media/images/conceptual-groundwater-flow-diagram.

———. 2019b. "Groundwater Is the Area Underground Where Openings Are Full of Water." *U.S. Geological Survey*. https://www.usgs.gov/media/images/groundwater-area-underground-where-openings-are-full-water.

———. 2019c. "How Much Water Is There on Earth?" *U.S. Geological Survey*. https://www.usgs.gov/special-topics/water-science-school/science/how-much-water-there-earth.

———. 2019d. "Ice, Snow, and Glaciers and the Water Cycle." *U.S. Geological Survey*. https://www.usgs.gov/special-topics/water-science-school/science/ice-snow-and-glaciers-and-water-cycle.

———. 2019e. "Precipitation and the Water Cycle." *U.S. Geological Survey*. https://www.usgs.gov/special-topics/water-science-school/science/precipitation-and-water-cycle.

———. 2019f. “Rain and Precipitation.” *U.S. Geological Survey*. https://www.usgs.gov/special-topics/water-science-school/science/rain-and-precipitation.

———. 2019g. “The Natural Water Cycle.” *U.S. Geological Survey*. https://www.usgs.gov/media/images/natural-water-cycle-jpg.

———. 2022. “The Water Cycle.” *U.S. Geological Survey*. https://www.usgs.gov/media/images/water-cycle-png.

Zhang, Lu, Klaus Hickel, WR Dawes, Francis HS Chiew, AW Western, and PR Briggs. 2004. “A Rational Function Approach for Estimating Mean Annual Evapotranspiration.” *Water Resources Research* 40 (2).

. .2020 “ ”. *Melabes*. https://www.melabes.co.il/news/51773.