Exercises

Exercises may be the most important part of this module container. We suggest that you do them actively and in small groups. Really, the only way to learn Python is to do it.

0 Exercises 0: Getting ready

0.1 Installation and development environment

Python is an open-source environment with significant input from the user community. Many of the developments are packaged in libraries designed for specific tasks. These need to be installed prior to usage which at times can be a bit tricky because of dependencies between different libraries. In order to alleviate you from installation difficulties that we have experienced in the past, we provide a fully installed python environment in a virtual environment that can be run on any laptop. If you use your own Laptop for the course this can be useful. Follow these steps to download and run the environment on your own computer:

1. Download and install VirtualBox for your System: `https://www.virtualbox.org/`
2. Copy the virtual machine files onto your system (external hard drive, or `https://esdynamics.geo.uni-tuebingen.de/nextcloud/s/rebnNeCwWFdTwwc`).
3. Run VirtualBox and select *Machine* → *Add* from the menu.
4. The virtual machine (vm) is now available on your system. You can start it by selecting the vm on the left side and click on the start button on the top right.

Figure 1: Add a virtual machine to VirtualBox

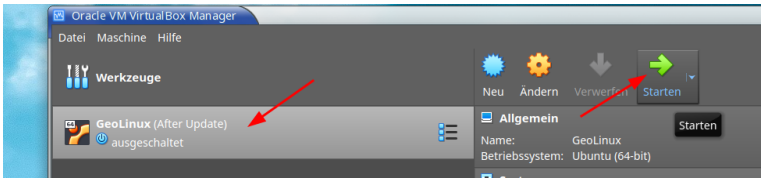

Figure 2: Start the virtual machine

It is best if you do that BEFORE the first class, then we can hit the road running. If you prefer to run your own installation (let's say with Anaconda), feel free to do it. Required packages are:

- Matplotlib
- Numpy
- Pandas

In Order to verify that Python is working correctly you can try to run the following example:

Test Python environment

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

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()

plt.show()
```

Src/Ex0/PlotExample.py

And this is what it should look like:

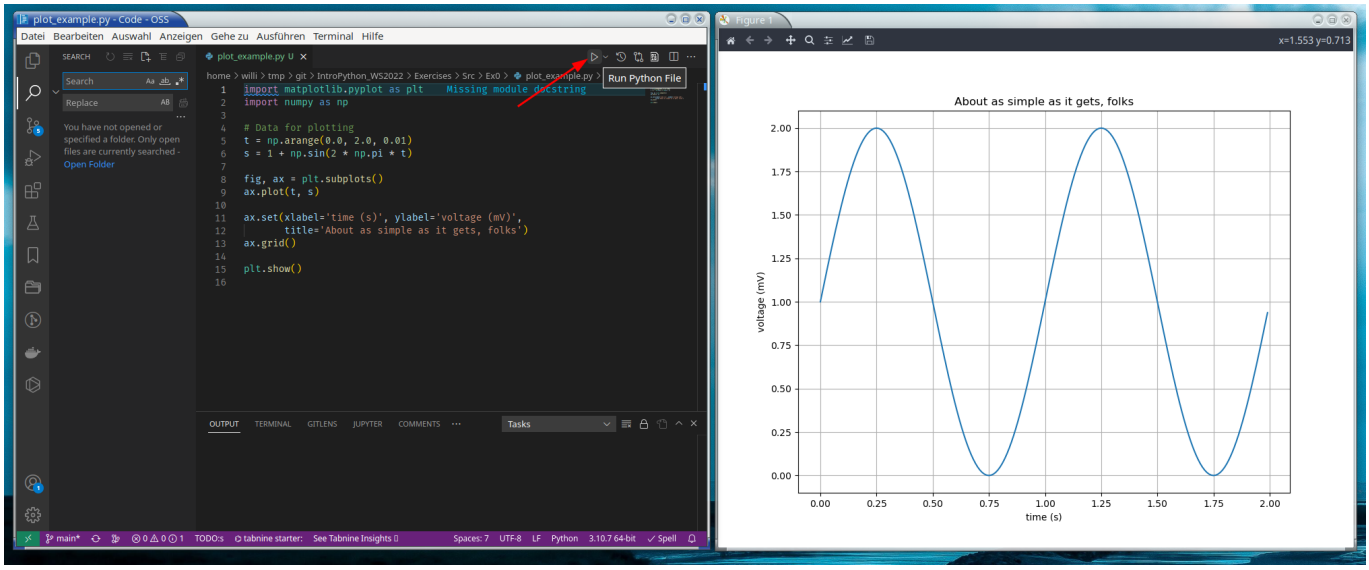

Figure 3: Test your Python environment

1 Data types and Visualization

After the first subtle introduction, you should be able to fill out this table (< 5 mins).

1.1 Data Types

Data Type	How to write in Python?	Geo- Env. Context
Integer		
Float		
Boolean		
List		
Tuple		
Dictionary		
String		
Numpy Array [Vector]		
Numpy Array [Matrix]		

What is the difference between an element of an array, and the index of an array?

Solutions

Data Type	How to write in Python?	Geo- Env. Context
Integer	`a = 1`	Sample number
Float	`t = 20.9`	Temperature value
Boolean	`True, False`	Above or Below threshold?
List	`[20.9, 23.3, 23.1]`	Temperature time series
Tuple	`MLoc = ("Location", -71.1, 8.4)`	Measurement Location
Dictionary	`s = {"Name": "samp1", "Year": 2022}`	Data structure
String	`"Guten Tag!"` `'Hello World!'`	Error message
Numpy Array [Vec.]	`np.array[(43.1, 2.09, 1)]`	Time series
Numpy Array [Mat.]	`np.array[(43.1, 2], [.09, 1)]`	Image

The element of an array is the value for a given index (indices are only integers). The first element of an array is evaluated with the index 0. The last one with the index -1.

1.2 Loading and visualization of ASCII-txt input data

Load the file `monthly_in_situ_co2_mlo_ready4loading.txt` which is the Keeling curve. If you don't know what this is please inform yourself. The data are decimal years in the first column and CO2 in ppm in the second column. A lack of data is marked with negative numbers.

- Load the data (e.g., using `numpy.loadtxt()`).
- How many datapoints are in the time series? What are the dimensions of the data array?
- Visualise the data (e.g., using `matplotlib`) with meaningful x- and y- limits and axis labels. Find out if the data are better visualised as points, curves or both. Change the color and symbols of the points.
- Using python functions calculate basic quantities such as the mean, minimum and maximum of your time series. Are those values meaningful?
- Visualize only the first ten elements, the last ten elements, and elements from 20 to 30.

Solutions

Keeling raw data

```python
import numpy as np
import pylab as plt

Data = np.loadtxt('../../Data/monthly_in_situ_co2_mlo_ready4loading.txt')
print(f'The dimensions of the time series are {Data.shape}')
print(f'We have {len(Data[:,0])} data values.')
print(f'The mean value of Co2 is at the moment {np.mean(Data[:,1])} ppm which seems low.')
print(f'The standard deviation Co2 is at the moment {np.std(Data[:,1])} ppm which seems high.')

fig,ax = plt.subplots()
ax.plot(Data[:,0],Data[:,1])
ax.set(xlabel='Time in years', ylabel='Co2 in ppm', title='Keeling Curve')

fig,ax = plt.subplots()
ax.plot(Data[0:10,0],Data[0:10,1],'r-x')
ax.set(xlabel='Time in years', ylabel='Co2 in ppm', title='Keeling Curve first ten samples')

fig,ax = plt.subplots()
ax.plot(Data[-11:-1,0],Data[-11:-1,1],'r-x')
ax.set(xlabel='Time in years', ylabel='Co2 in ppm', title='Keeling Curve last ten samples')
plt.show()
```

Src/Ex1/KeelingRawData.py

1.3 Visualization of mathematical functions

Make use of some numpy functions (arange) and get a feel for some vector manipulation.

- Visualize $f(t) = \sin(\omega t + \phi_0)$ for a given frequency $\omega = 1.2$Hz and phase shift ϕ_0 over the time interval $t = 0...10$ at discrete time intervals $dt = 0.001$ s.
- Visualize $f^2(t)$. What happens if you reduced dt to 1 s ?
- Visualize a Gaussian peak:

  $$f(x) = Ae^{-\frac{(x-x_0)^2}{\sigma_x}}$$

  where x is the independent variable and the factors x_0, A, and σ_x determine the shape of the function. Make sure that you understand what each parameter pertains to.

1.4 Basic filtering and data manipulation

Load the Keeling curve from **??** again. We have understood that no data values are marked with negative numbers. Let's remove those with a for loop (there are better ways at a later stage). Take your time, if this is your first time of writing a loop this can take some time.

- Print all CO_2 values on the screen using a for loop. (How could this be done much easier?)
- Print only the time intervals where no data are available
- Create a new vector where the no-data values are removed.
- Visualize the new vector. Anything different compared to **??**?

Start a new for loop block and calculate the time derivative of the time series. Visualize it.

Solutions

Kelling filter

```python
import numpy as np
import pylab as plt

Data = np.loadtxt('../../Data/monthly_in_situ_co2_mlo_ready4loading.txt')
NumberOfSamples = len(Data[:,0])
#Filter negative values. Later use functions for this, e.g., np.where()
ListFilter=[]
for kk in np.arange(0,NumberOfSamples):
    if Data[kk,1]<0:
        print(f'There is a negative value in row {kk}.')
        ListFilter.append(kk)
print(f'Remove all negative values from time series.')
Data = np.delete(Data,ListFilter,0)
NumberOfSamples = len(Data[:,0])

#Approximate time derivative with forward differencing.
#This will have one data point less.
d_dt = np.zeros((NumberOfSamples,2))

for kk in np.arange(0,NumberOfSamples-1):
    d_dt[kk,0] = Data[kk,0]
    d_dt[kk,1] = (Data[kk+1,1]-Data[kk,1])/((Data[kk+1,0]-Data[kk,0]))
d_dt[NumberOfSamples-1,1]=d_dt[NumberOfSamples-2,1]
d_dt[NumberOfSamples-1,0]=Data[NumberOfSamples-2,0]

#Visualize
fig,ax = plt.subplots()
ax.plot(Data[:,0],Data[:,1])
ax.set(xlabel='Time in years', ylabel='Co2 in ppm', title='Keeling Curve')

fig,ax = plt.subplots()
ax.plot(d_dt[:,0],d_dt[:,1])
ax.set(xlabel='Time in years', ylabel='Co2 rate per year', title='Keeling Curve')
plt.show()
```

Src/Ex1/KeelingFilter.py

1.5 Loops continued.

Visualize a 2D Gaussian peak:

$$f(x,y) = Ae^{-\frac{(x-x_0)^2}{\sigma x} - \frac{(y-y_0)^2}{\sigma y}}$$

using a nested for loop. This is harder. In essence you will need to fill a 2D array and visualize it with colors using `plt.pcolormesh()`. Then do the same but replace the nested for loops using numpys `"meshgrid"`. What is easier?

2 Functions and loops

2.1 Linear regression

Load the Keeling curve from **??**. Fit a first and second order polynomial to the dataset. We understand that this type of fitting must have been solved by somebody else already and therefore use functions to do the task, specifically numpy's polyfit and polyval combination. Visualize the seasonality by subtracting the fit from the observations. Which models (i.e. first or second order polynomial) fits better and what does that mean?

2.2 Writing your own functions

Calculating the time (or spatial) derivative of a 1D dataset is something that may occur quite frequently. Occuring tasks are best written in functions for many reasons (name at least two). Write a function called *ForwardDifferencingXY* that takes a 2 x n array as input where the independent variable (e.g., time) is stored in the first column, and the dependent variable (e.g., CO_2) in the second column. The output should also be a 2 x n array with the independent variable in the first column and the derivative of the dependent variable in the second colum.
Here you should do this for the Keeling curve from **??** and for the Gausspeak from **??**.

- Use your function *ForwardDifferencingXY* to calculate rates of change for the Keeling curve **??**, the sinsoid and the 1D Gauss Peak **??**
- Make your function more robust so that it catches wrong use interactions (e.g., passing on a n x 2 array instead of 2 x n array)
- Make your user function more user friendly by providing and optional figure with sub-panels showing the original data on top, and the time derivative at the bottom.
- Add noise to your input data using numpys `randn` function, what does that do to your derivatives. Why?

3 More loops

3.1 Endless loops, reading user input

Almost all programming languages that support loops can also have endless (infinite) loops, that is loops that never end. Besides a `for` loop Python also has a `while` loop. The syntax is very easy:

While loop 1

```
x = 0

while x < 10:
    x = x + 1
    print(f"x: {x}")
```

Src/Ex3/While1.py

To finish a loop in Python just use the `break` keyword:

While loop 2

```
x = 0

while True:
    x = x + 1
    print(f"x: {x}")
    if x > 9:
        break
```

Src/Ex3/While2.py

In order to read an input from the user via keyboard Python has the `input()` function:

User input 1

```
name = input("What is your name? ")
print(f"Your name is {name}")
```

Src/Ex3/Input1.py

To convert a string into a number Python has two functions: `int()` and `float()`.

Now write a function that reads in numbers from the user and calculates the total sum and prints this value at each iteration. The program should also print out all the numbers sorted (ascending) at the end. In order to quit the program the user has to enter `'quit'` instead of a number.

Solutions

User input 2

```python
all_numbers = []
total_sum = 0

while True:
    print(f"Total sum: {total_sum}")
    user_in = input("Enter a number: ")

    if user_in == "quit":
        break

    number = int(user_in)
    total_sum += number
    all_numbers.append(number)

all_numbers.sort()
print(f"numbers: {all_numbers}")
```

Src/Ex3/UserInput.py

3.2 Recursion

There is a third way to do loops in Python (and other programming languages as well): If you define a function you can call that function recursively:

Recursion with Fibonacci

```python
def Fibonacci(n):
    if n < 0:
        print("Negative numbers are not allowed.")
    elif n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)

print(f"F0: {Fibonacci(0)}")
print(f"F1: {Fibonacci(1)}")
print(f"F2: {Fibonacci(2)}")
print(f"F3: {Fibonacci(3)}")
print(f"F9: {Fibonacci(9)}")
```

Src/Ex3/Fibonacci1.py

Mathematically the Fibonacci numbers are defined as:

$$F_n = \begin{cases} 0, & n == 0 \\ 1, & n == 1 \\ F_{n-1} + F_{n-2}, & n > 1 \end{cases}$$

Now write the Fibonacci function in a non-recursive way using a `for` loop or a `while` loop.

Recursion with Fibonacci

```python
def Fibonacci(n):
    numbers = [0, 1, 1, 2]

    for i in range(4, n + 1):
        fib_2 = numbers[i - 2]
        fib_1 = numbers[i - 1]
        numbers.append(fib_1 + fib_2)
    return numbers[n]

def Fibonacci2(n):
    fib_2 = 0 # n - 2
    fib_1 = 1 # n - 1
    fib_0 = 1 # n

    if n == 0:
        return 0

    for _ in range(2, n + 1):
        fib_0 = fib_1 + fib_2
        fib_2 = fib_1
        fib_1 = fib_0

    return fib_0

print(f"f0: {Fibonacci(0)}")
print(f"f1: {Fibonacci(1)}")
print(f"f2: {Fibonacci(2)}")
print(f"f3: {Fibonacci(3)}")
print(f"f9: {Fibonacci(9)}")

print("\n")

print(f"f0: {Fibonacci2(0)}")
print(f"f1: {Fibonacci2(1)}")
print(f"f2: {Fibonacci2(2)}")
print(f"f3: {Fibonacci2(3)}")
print(f"f9: {Fibonacci2(9)}")
```

Src/Ex3/Fibonacci2.py

3.3 Reading data from multiple files

At some time you have to read in multiple files one after each other and combine all the data. One way of doing this is to specify the files in a list:

Multiple files 1

```
import pandas

files = ["file1.csv", "file2.csv", "more_data.csv", "old_data.csv", "unknown.csv"]

result = pandas.DataFrame()

for file in files:
    data = pandas.read_csv(file)
    result = pandas.concat([result, data])

print(result)
```

Src/Ex3/Files1.py

Another way is to specify a file pattern and to load all the files that have that pattern in their name:

Multiple files 2

```
import pandas
import glob

files = glob.glob("*.csv")

result = pandas.DataFrame()

for file in files:
    data = pandas.read_csv(file)
    result = pandas.concat([result, data])

print(result)
```

Src/Ex3/Files2.py

So now try to do it by yourself: Load in some data files, merge the data and make a nice plot.

4 Heat transport

In class we have introduced the diffusive heat equation as a partial differential equation:

$$\frac{d}{dt}T = -\kappa \frac{d^2}{dx^2}T$$

It determines how a given temperature profile diffuses (here one dimensionally) from a fixed initial state T(t=0,:) depending on the thermal diffusivity κ and over the spatial coordinates x. As discussed in class, this type of differential equation occurs over a wide range of relevant Earth processes. It can be solved numerically explicilty by discretizing the derivatives:

$$T_i^{n+1} = T_i^n + k\frac{\Delta t}{\Delta x^2}\left(T_{i-1}^n - 2T_i^n + T_{i+1}^n\right)$$

Take a minute an understand why this equation can be discretized in this way using the time index n and the spatial index i.
Simulate a one dimensional soil profile with $\kappa = 5^{-7}m^2s^{-1}$ over 10 m. Assume initially an isothermal profile of $10C^\circ$, a surface temperature of $20C^\circ$ and a bottom temperature of $5C^\circ$. Solve the heat evolution in a nested loop. Choose small time steps. Visualize time steps in-between. What is the steady-state temperature profile? Why? Once this runs well, you can start including day-night cycles in the top boundary condition. How far do these propagate into the soil? How useful is the bottom boundary condition and how could this be changed? (Keyword Neumann and Dirchilet conditions.)

Solutions

I cannot find code in the train, but it is not if you know what you are doing. `https://people.sc.fsu.edu/~jburkardt/py_src/fd1d_heat_explicit/fd1d_heat_explicit.py`

Figure 4: View downstream of Priestley Glacier. At the far end you can see the ocean. The ice starts to float on ocean water approximately at the end of the mountain chain located at the glaciers true left side.

5 More advanced exercises

5.1 GPS Data analysis

Today's topic is focused on Antarctica. The data file *XYShirase_GPS_Small.txt* is the output of GPS measurements located at Priestley Glacier feeding into the Nansen Ice Shelf. Ice shelves are the floating extensions of the Antarctic Ice Sheet. Beneath them is only water and consequently the ice lifts up and down with ocean tides. The data file contains 6 columns (1: Position in polar stereographic x, 2: Position in polar stereographic y, 3: Longitude, 4: Latitude, 5: Elevation, 6: Time). The polar stereographic projection (EPSG:3031) is a rectangular coordinate system with units meters (sort of comparable to UTM coordinates). The time is given in modified Julian days which are days since November 17, 1858 (just for information, not needed in exercise). Please complete the following tasks. Some are easier, some are harder.

- Visualize the movement of the GPS station in a x-y scatter plot (easy)
- Visualize the vertical GPS position as a time series (easy). Which tidal regime is visible here?
- Calculate the mean horizontal GPS velocity in the observational period Pythagoras is your friend. Always. (medium).
- The vertical position has a fair amount of scatter. This is normal, as GPS measurements are about three times worse in the vertical compared to the horizontal direction. Smooth the data with your own moving average filter with variable window size. This filter should average the vertical position at time t as a mean value for a given number of positions recorded close to t. Assume that the data

are regularly spaced in time. Cut-off the smoothed vector at the beginning and the end. It can be smaller than the data file (medium, involves for loops).

- Find functions in python that do a similar job and compare your results. (easy)
- Discuss the pitfalls of filtering and suggest improvements. Memorize that a moving average filter is usually not the best way to go for noise reduction. Better options are weighted moving average or bandpass filters.
- Design your own FIR low-pass filter, e.g., following `https://tomroelandts.com/articles/how-to-create-a-simple-low-pass-filter`. This is advanced and outside the scope of this course. Only do this if everything else is easy for you. We will not discuss the theory here.

5.2 Cellular Automata: Abelian Sandpile Model

Self-organized criticality is a fascinating subject that occurs in a number of geo- and environmental processes. Your job is to simulate an Abelian Sandpile Model which has in fact some quite realistic insights into the nature of landslides. The rules of this system are (Wikipedia entry has a more formal version of it):

- Choose a random location on your grid and drop a grain of sand (i.e. add +1)
- Continue unless one of your grid points as more than 3 grains.
- If a grid point as more than 3 grains then empty that cell, and distribute all grains to gridpoints in the surrounding (this is a landslide). Then continue.
- Grains fall of the boundaries are lost.

Implement this set of rules on a grid and visualize its evolution.

5.3 Cellular Automata: Langton's Ant as a 2D example

Imagine you are in a 2D universe that is intially completely white on consists out of 100 x 100 cells that you can walk on. Your life depends on two rules only:

- At a white square, turn 90 right, flip the color of the square, move forward one unit
- At a black square, turn 90 left, flip the color of the square, move forward one unit

Will you ever leave your world? The rules are simple enough, but an answer to this question is very difficult if not impossible. However, you can find the answer in Python. Go for it (with our help)! Cellular automata such as this have many applications in Geosciences. Another important model is, e.g., the sandpile model.