PR Computeraided Assessment of Material

Table of contents

Instructions

Welcome to the instruction page of the 724700 PR Computeraided Assessment of Materials!

The sidebar contains links to the respective exercise. For the exercise a scientific introduction is provided along with a detailed HowTo and a checklist which results should be included in the protocol.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)

x = np.linspace(-4.9,4.9,100)

# the function, which is y = x^2 here
y = x**2+10

# setting the axes at the centre
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim([-5, 35])
ax.xaxis.set_minor_locator(MultipleLocator(0.5))
ax.yaxis.set_minor_locator(MultipleLocator(2.5))

# plot the function
plt.plot(x,y, 'r', linewidth = 2.5)
plt.axvline(x = 0, ymin = 0.1, ymax = 0.9, color = 'r', linewidth =
↪  2.5)
#plt.axis('off')
plt.grid(True)

# show the plot
plt.show()
```

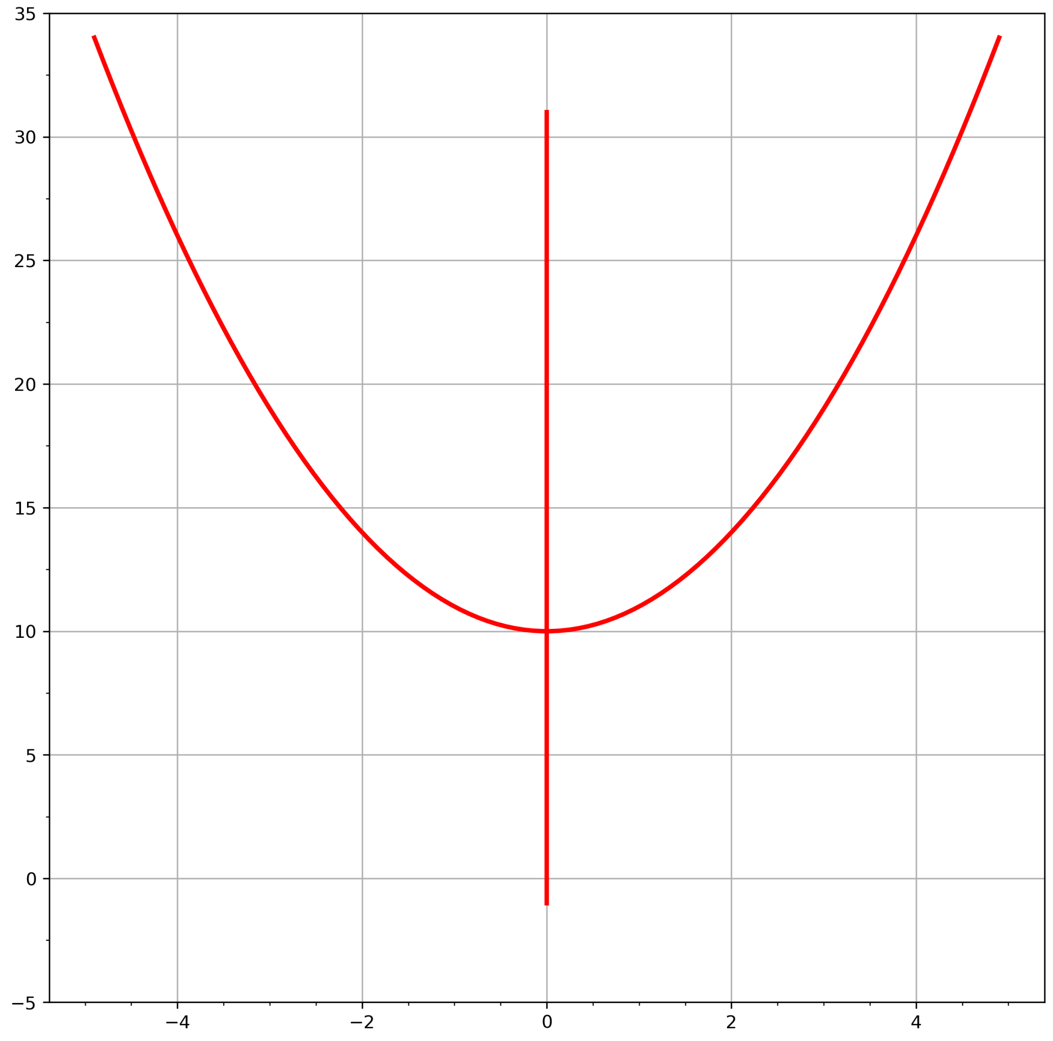


Figure 1: Example Code with corresponding Plot. Any similarity to the Greek Letter is by coincidence

Exercise: NNP-MD

Background Info

Metal-organic framework (MOFs) are hybrid crystalline materials assembled from both inorganic and organic residues containing potential voids [1]. The key advantage of MOFs over naturally occurring porous compounds such as zeolites (*i.e.* aluminosilicates) or polymers lies in their crystalline nature, which allows for systematic modification through crystal engineering principles by altering their organic linkers or metal nodes. This capability leads to an almost limitless number of possible framework topologies and properties, making them particularly appealing for technologically relevant applications.

One of the most widely discussed applications is associated to the enormous gas storage capacity of MOF materials. Storage and separation of critical green house gasses such as CO_2 and CH_4 are widely discussed [2,3]. In addition, MOFs have also emerged as highly promising candidates for the storage of carbon dioxide (CO_2) for carbon capture [4] (due to their ultrahigh internal surface area, tunable pore dimensions, and rapid kinetics for gas adsorption and desorption).

An increasing number of MOF compounds are discussed as potential carrier for green house gases such as CO_2 [4]. In this exercise, carbon dioxide storage in the comparably simple ZIF-8 (zeolitic imidazolate framework) is investigated via molecular dynamics simulations

(MD), to keep the computational effort and memory demand manageable.

The ZIF-8 system Zn(methylimidazolate)$_2$ crystallizes in the noncentrosymmetric cubic space group I43m (space group no. 217) with a lattice parameter of approx 1.7 nm. The cubic unit cell contains a total of 2 Zn^{2+} ions that are tetrahedrally coordinated by the 24 methylimidazolate linkers. The pore structure of ZIF-8 is similar compared to the topology of a prototypic sodalite zeolite (hence the name zeolitic imidazolate framework). Several studies have indeed investigated ZIF systems with respect to their CO$_2$ storage capacity [5-8].

In order to achieve fast and accurate MD simulations, the MACE-MP neural network potential (NNP) [9-11] is applied. If trained properly, NNPs provide an efficient and versatile description of a chemical system, that combines the advantages of quantum chemical and force field descriptions.

In this exercise the properties of the pristine host material (lattice parameter, thermal expansion coefficient) and the associated host-guest interaction (interaction energies, diffusion coefficient, activation energy of diffusion) will be analyzed.

[1] Yusuf, V. F.; Malek, N. I.; Kailasa, S. K. “Review on Organic Framework Classification, Synthetic Approaches, and Influencing Factors: Applications in Energy, Drug Delivery, and Wastewater Treatment.” ACS Omega 2022, 7, 44507 – 44531, DOI: 10.1021/acsomega.2c05310

[2] Li, B.; Wen, H.-M.; Zhou, W.; Chen, B. “Porous Metal–Organic Frameworks for Gas Storage and Separation: What, How, and Why?” J. Phys. Chem. Lett. 2014, 5, 3468 – 3479 DOI: 10.1021/jz501586e

[3] Li, H.; Li, L.; Lin, R.-B.; Zhou, W.; Zhang, Z.; Xiang, S.; Chen, B. “Porous metal-organic frameworks for gas storage and separation: Status and challenges” EnergyChem 2019, 1, 100006/1 – 100006/39 DOI: 10.1016/j.enchem.2019.100006

[4] Mahajan, S.; Lahtinen Ma. ”Recent progress in metal-organic frameworks (MOFs) for CO$_2$ capture at different pressures” J. Environ. Chem. Eng. 2022, 10, 108930/1 – 108930/35 DOI: 10.1016/j.jece.2022.108930

[5] Abraha, Yuel W.; Tsai, C.-W.; Niemantsverdriet, J. W. H.; Langner E. H. G. ”Optimized CO$_2$ Capture of the Zeolitic Imidazolate Framework ZIF-8 Modified by Solvent-Assisted Ligand Exchange” ACS Omega 2021, 6, 21850 – 21860 DOI: 10.1021/acsomega.1c01130

[6] Jiang, S.; Liu, J.; Guan J.; Du, X.; Chen, S.; Song, Y.; Huan, Y. ”Enhancing CO$_2$ adsorption capacity of ZIF-8 by synergetic effect of high pressure and temperature” Sci. Rep. 2023, 17584/1 – 17584/8 DOI: 10.1038/s41598-023-44960-4

[7] Kalauni, K.; Vedrtnam, A.; Wdowin, M.; Chaturvedi, S. “ZIF for CO$_2$ Capture: Structure, Mechanism, Optimization, and Modeling” Processes 2022, 10, 2689/1 – 2689/32 DOI: 10.3390/pr10122689

[8] Heinz, K.; Rogge, S. M. J.; Kalytta-Mewes, A.; Volkmer, D.; Bunzen, H. “MOFs for long-term gas storage: exploiting kinetic trapping in ZIF-8 for on-demand and stimuli-controlled gas release” Inorg. Chem. Front. 2023, 10, 4763 – 4772 DOI: 10.1039/D3QI01007D

[9] Batatia, I.; Kovacs, D. P.; Simm, G.; Ortner, C.; Csanyi G. “MACE: Higher Order Equivariant Message Passing Neural Networks for Fast and Accurate Force Fields” arXiv, 2022, DOI: 10.48550/ARXIV.2206.07697

[10] Batatia, I. et al. “A foundation model for atomistic materials chemistry” arXiv, 2024, DOI: 10.48550/ARXIV.2401.00096

[11] ACEsuit/mace-mp https://github.com/ACEsuit/mace-mp (accessed 16. 10. 2024)

HowTo

Warning

Links to the **Google Sheets** with the assigned temperatures and carbon dioxide loadings:

- Thermal Expansion
- Gas Diffusion

Each student will be assigned two specific temperature values and two different carbon dioxide loadings for the simulations. One temperature will be used for Part B.1 and the other for Part B.2. The carbon dioxide loadings will be used for the simulations in Part B.1

Important

Before you run any simulations, please make sure you know your assigned **TEMPERATURE** values.

All tools and programs are provided by the Hofer Lab. The simulaton engine is PQ. The analysis tools are written in Python and C, and are partially based on the PQAnalysis library.

The required tools and programs for this exercise are provided by executing the following command:

```
module load pq
```

Important

This command needs to be executed only once per terminal session. If you close the terminal, you will need to execute the command again when you open a new terminal.

The simulation engine PQ can be executed by running the following command:

```
PQ <input_file>
```

The analysis tools are written in an intuitive way and have a help function that can be accessed by running the script with the `--help` flag. For example:

```
<analysis_tool> --help
```

Analysis Tools

The following analysis tools are provided for this exercise:

Tool	Description
`average_a`	Averages the lattice parameter a of the provided box files.
`average_V`	Averages the volume of the provided box files.
`extract_co2`	Extracts the carbon dioxide molecules of trajectory/velocity-files and calculates the center of mass coordinates/velocities.
`msd`	Calculates the mean squared displacement of the carbon dioxide molecules from a trajectory. (Note: The carbon dioxide molecules have to be extracted first using `extract_co2`).
`rdf_zif8`	Calculates the radial distribution function of the carbon dioxide molecule from a trajectory.
`vacf`	Calculates the velocity autocorrelation function of the carbon dioxide molecule from a velocity file. (Note: The carbon dioxide molecules have to be extracted first using `extract_co2`).

Tool	Description
`linearfit`	Fits a linear function to the mean squared displacement data to calculate the diffusion coefficient according to the Einstein relation.
`integration`	Integrates the velocity autocorrelation function to calculate the diffusion coefficient according to the Green-Kubo relation.

Files

The folders encoded in the varible `ZIF_DIR` contains all necessary files for the simulations. The files are organized in the following way:

Note

```
general_input
   moldescriptor.dat
   run-01.in
   run-02.in
   run-03.in
   run-04.in
co2
   co2.rst
preeq
    248
       shake_zif8_248.top
       zif8_preeq_248.rst
    273
       shake_zif8_273.top
       zif8_preeq_273.rst
    298
       shake_zif8_298.top
       zif8_preeq_298.rst
    323
       shake_zif8_323.top
       zif8_preeq_323.rst
    348
       shake_zif8_348.top
       zif8_preeq_348.rst
    373
       shake_zif8_373.top
       zif8_preeq_373.rst
    398
        shake_zif8_398.top
        zif8_preeq_398.rst
```

Folder/File	Description
`general_input`	Contains the input files for the simulations (`moldescriptor.dat` and `run-0*.in`).
`co2`	Contains the carbon dioxide restart-file (`co2.rst`)
`preeq`	Contains the equilibration restart-files (`zif8_preeq_*.rst`) and the topology files (`shake_zif8_*.top`).

Part B.1) Gas simulation in ZIF-8

The first part of the exercise is to simulate the diffusion of carbon dioxide molecules in the ZIF-8 framework at the assigned temperature values. The carbon dioxide loading is set to the two assigned loadings per student.

> **Note**
>
> This part of the exercise is performed twice with the two assigned carbon dioxide loadings. The simulations can be run in parallel.

System Setup

The system setup is done by running the following command:

1. Copy the necessary files to the working directory

```
cp $ZIF_DIR/zif8_files/co2/* .
cp $ZIF_DIR/zif8_files/preeq/<TEMPERATURE>/* .
cp $ZIF_DIR/zif8_files/general_input/* .
```

2. Add the CO2 loading to the topology and restart files (e.g. 32 carbon dioxide molecules and 298 K)

```
add_molecules zif8_preeq_298.rst co2.rst   \
  --rst-mol-desc-file moldescriptor.dat   \
  -n 32                                   \
  > 32xco2-zif8-00.rst
```

Simulation

1. Edit the input file `run-01.in` to include your assigned temperature, generated restart file, and topology file.

```
...
# Temperature algorithm (velocity rescaling), Target T in K and Relaxation time in ps
      thermostat = velocity_rescaling; temp = XXX.XX; t_relaxation = 0.1;
...
# Files
      start_file    = XXX-00.rst;
      topology_file = XXX.top;

      file_prefix   = XXX-01;
```

> **Note**
>
> Replace `XXX` with the assigned temperature value and the corresponding restart and topology files. The `file_prefix` can be any name you choose. Please make sure to use `-00.rst` as the start file and `-01` as the file prefix. This will help to keep track of the different simulation steps.

2. Run NVT equlibration

```
PQ run-01.in
```

Start multiple input files one after the other

```
PQ run-01.in && PQ run-02.in && PQ run-03.in
```

! Important

The overall equilibration is split into two stages. The first stage is a 10 ps NVT equilibration (`run-01.in`). The second stage is a 10 ps NPT equilibration (`run-02.in`). After the equilibration, the production run is performed for 1 ns, which is split into 2 runs of 500 ps each (`run-03.in` and `run-04.in`).

Extract Carbon Dioxide Molecule

Extract the carbon dioxide molecule from the trajectory and velocity files using the `extract_co2` tool:

```
extract_co2 <trajectory_files>.xyz
```

and

```
extract_co2 <velocity_files>.vel
```

Einstein Relation

1. Calculate the mean squared displacement (MSD) of the carbon dioxide molecule using the `msd` tool:

```
msd
```

2. Fit a linear function to the MSD data using the `linearfit` tool:

```
linearfit --window 5
```

! Important

For both assigned carbon dioxide loadings, insert the corresponding self-diffusion coefficient values into the **Google Sheets**.

Green-Kubo Relation

1. Calculate the velocity autocorrelation function (VACF) of the carbon dioxide molecule using the `vacf` tool:

```
vacf
```

2. Integrate the VACF data using the `integration` tool:

```
integration
```

! Important

For both assigned carbon dioxide loadings, insert the corresponding self-diffusion coefficient values into the **Google Sheets**.

Radial Distribution Function

1. Calculate the radial distribution function (RDF) of the carbon dioxide molecule using the `rdf_zif8` tool:

```
rdf_zif8 <trajectory_file-03>.xyz <trajectory_file-04>.xyz
```

2. Calculate the RDF for `Zn` and `C` or `O` atoms (gas) using the `rdf_zif8` tool:

```
rdf_zif8 <trajectory_file-03>.xyz <trajectory_file-04>.xyz --center Zn
↪  --ligand1 X
```

Note

The RDF calculation will generate a `output` file named `rdf-C-C.out` and `rdf-Zn-C.out`, accordingly. Plot the 1st column (distance) against the 2nd column (RDF) using your preferred plotting tool.

Visualization

1. Visualize the trajectory files using the `vmd` tool:

```
vmd <trajectory_file>.xyz
```

2. Box files can be made with the `traj2box` tool:

```
traj2box <trajectory_file>.xyz --vmd > <box_file>.xyz
```

Note

The `traj2box` tool generates a box file that can be visualized in VMD. The box file contains the unit cell information with `X` atoms at the corners of the unit cell. The box file can be used to visualize the unit cell of the simulation.

3. Visualize the box file using the `vmd` tool:

`File > New Molecule > Filename (Browse to the box file) > Load (All at once)`

4. Representations can be changed in the `Graphics > Representations...` window. Good representations are:

- `Licorice` for the framework atoms (and Box atoms)
- `DynamicBonds` for the framework atoms (and Box atoms)
- `VDW` for the carbon dioxide atoms (Selected Atoms `index > 275`)
- `VDW` for the Zn atoms (Selected Atoms `name Zn`)

5. General color and setting recommendations:

- Color box atoms: Change the color of the box atoms to `Coloring Method > ColorID 8` (white) for both `Licorice` and `DynamicBonds` representations.
- Change the background color to white: `Graphics > Colors > Categories (Display) > Names (Background) > Colors (White)`
- Disable depth cueing: `Display > Depth Cueing > Off`
- Render the image: `File > Render > Tachyon (internal, in-memory rendering)`

Part B.2) Thermal Expansion of ZIF-8

The second part of the exercise is to calculate the thermal expansion of the ZIF-8 framework at the assigned temperature values.

System Setup

The system setup is done by running the following command:

1. Copy the necessary files to the working directory

```
cp $ZIF_DIR/zif8_files/preeq/<TEMPERATURE>/* .
cp $ZIF_DIR/zif8_files/general_input/* .
```

Simulation

1. Edit the input file `run-01.in` to include your assigned temperature, generated restart file, and topology file.

```
...
# Temperature algorithm (velocity rescaling), Target T in K and Relaxation time in ps
      thermostat = velocity_rescaling; temp = XXX.XX; t_relaxation = 0.1;
...
# Files
      start_file    = zif8_preeq_XXX.rst;
      topology_file = shake_zif8_XXX.top;

      file_prefix   = zif8-01;
```

Note

Replace `XXX` with the assigned temperature value. The `file_prefix` can be any name you choose. Please make sure to use `-01` as the file prefix. This will help to keep track of the different simulation steps.

2. Run NVT equlibration

```
PQ run-01.in
```

Note

The overall equilibration is split into two stages. The first stage is a 10 ps NVT equilibration (`run-01.in`). The second stage is a 10 ps NPT equilibration (`run-02.in`). After the equilibration, the production run is performed for 500 ps (`run-03.in`). Only run `run-03.in` for the thermal expansion calculation.

Analysis

1. Average the lattice parameter `a` of the ZIF-8 framework using the `average_a` tool:

```
average_a zif8-03.box
```

2. Average the volume of the ZIF-8 framework using the `average_V` tool:

```
average_V zif8-03.box
```

3. Calculate the thermal expansion coefficient using the following formula, see thermal expansion

Analysis

Log-file check

Open the `*.log` file with any editor.

Check if the log file end with this:

- ☐ "PQ ended normally"

You can also grep for this statement to verify the normal termination of the simulation for all files:

```
grep "PQ ended" *.log
```

Visual inspection

Check if the system is behaving as expected.

Open the `*.xyz` file with `VMD`.

It is a good sign if

- ☐ no atom has been shot out of the box
- ☐ the system remains intact (does not collapse or blow up)
- ☐ no unanticipated movements can be observed etc.

Troubleshooting

In the event that the simulation does not produce the anticipated results:

- ☐ Try to understand any error messages that may have occured
- ☐ Check if the simulation was set up correctly by inspecting the `*.log` file and `*.in`
- ☐ Check the starting structure
- ☐ Check the moldescriptor file

etc.

Equilibrium state of simulation

Firstly, in order to obtain meaningful physical properties from a simulation, it is essential to ensure that the system is in an equilibrium state. However, what constitutes an *equilibrium state*?

! An equilibrium state:

can be defined as a state in which macroscopic properties A (*e.g* temperture T, pressure P, energies E, volume V,…) do **NOT** undergo major changes over time.

Once these properties fluctuate around a constant pattern without any long-term trends, it can be assumed that the system reached an equilibrium state.

Arithmetic Average, Standard Deviation and Standard Error

The aforementioned constant relaxation of the properties can be verified by calculating the arithmetic mean of the properties $\langle A \rangle$

$$\langle A \rangle = \frac{1}{n} \sum_{i=0}^{n} a(t_i)$$

where $a(t_i)$ is the property a at time step t_i and n the total sampling time

as well as the standard deviation A_σ

$$A_\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(a(t_i) - \langle A \rangle)^2}.$$

and standard error A_ν of the arithmetic average

$$A_\nu = \frac{A_\sigma}{\sqrt{n}}$$

Running Average

Further the running average $\langle A \rangle_\tau$ at time τ with a window size of ω and a gap size g

$$\langle A \rangle_\tau = \frac{1}{\omega} \sum_{i=\tau-(\omega-1)g}^{\tau} a(t_i)$$

e.g. $\tau = 5$, $\omega = 3$, $g = 2$

$$\langle A \rangle_5 = \frac{1}{3}(a(t_1) + a(t_3) + a(t_5))$$

can help to estimate the fluctuation of the properties over time.

Cummulative Average

And the cumulative average is a running average that is added at each step.

$$\langle A \rangle_n = \frac{(n-1)\langle A \rangle_{n-1} + \langle A \rangle_n}{n}$$

Linear Regression Fit

Further linear regression fit helps to estimate the trend of your simulation.

For more further detail: statistics

Plot of temperature over simulation time

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import linregress
import matplotlib as mpl

mpl.rcParams["font.size"] = 15
mpl.rcParams["axes.labelsize"] = 20


physical_properties = ["SIMULATION-TIME", "TEMPERATURE", "PRESSURE",
↪  "E(TOT)", "E(QM)", "N(QM-ATOMS)", "E(KIN)",
↪  "E(INTRA)","VOLUME","DENSITY","MOMENTUM","LOOPTIME"]
physical_units = ["fs", "K","bar", "kcal/mol", "kcal/mol",
↪  "-","kcal/mol","kcal/mol","A^3", "g/cm^3", "amuA/fs", "s"]
data = pd.read_csv("../data/physical_properties.en",sep='\t+',names=⌋
↪  physical_properties)
print(data.head())
print("Shape of data: ", data.shape)
```

```
time = data["SIMULATION-TIME"]/1e6 # ns
temperature_avg = np.mean(data["TEMPERATURE"])
temperature_std = np.std(data["TEMPERATURE"])
window = 50
gap = 2
running_average = data["TEMPERATURE"].rolling(window=window).mean()
↪ #window=window,min_periods=1,center=True,step=gap

def linear_function(x,a,b):
  return a*x+b



plt.figure(figsize=(8,4))
plt.plot(time,data["TEMPERATURE"],color="black",
↪ alpha=0.5,label=r"simulation")
plt.plot(time,running_average,label=r"$T_{\mathrm{run.avg}}$")
plt.hlines(temperature_avg,xmin=np.min(time),xmax=np.max
↪ (time),linestyles="solid",color="black",label=r"$T_{\mathrm{avg}
↪ }$")
plt.hlines(temperature_avg+temperature_std,xmin=np.min(time),xmax=np.
↪ max(time),linestyles="dashed",color="black",label=r"$T_{\mathrm
↪ {std}}$")
plt.hlines(temperature_avg-temperature_std,xmin=np.min(time),xmax=np.
↪ max(time),linestyles="dashed",color="black")

plt.xlabel(r"$t$ / ns")
plt.ylabel(r"$T$ / K")
plt.legend(fontsize=12)
plt.show()
```

```
   SIMULATION-TIME  TEMPERATURE      PRESSURE         E(TOT)          E(QM)  \
0          1656002   265.867307  -9219.140442 -39593.157062 -39779.394319
1          1656004   232.575082 -10168.957588 -39594.249861 -39757.166264
2          1656006   326.845099  -6236.688599 -39600.546646 -39829.498207
3          1656008   398.202185   2882.459019 -39597.914972 -39876.851422
4          1656010   382.115074   9870.653269 -39596.205739 -39863.873337

   N(QM-ATOMS)      E(KIN)  E(INTRA)       VOLUME   DENSITY  MOMENTUM  \
0        276.0  186.237257       0.0  4946.281581  0.916867  0.000011
1        276.0  162.916403       0.0  4944.425576  0.917211  0.000011
2        276.0  228.951560       0.0  4943.049144  0.917466  0.000011
3        276.0  278.936450       0.0  4943.121195  0.917453  0.000011
4        276.0  267.667598       0.0  4944.683355  0.917163  0.000011

   LOOPTIME
0   5.08360
1   1.58753
2   3.05997
3   0.18536
4   0.17990
Shape of data:  (50000, 12)
```

exercise_b/analysis_b_files/figure-pdf/fig-temp-output-2.png

Figure 2: Temperature vs Time plot

Linear / Volumetric Thermal Expansion Coefficient

The thermal expansion coefficient is a quantity that describes the extent to which a solid substance undergoes a change in response to variations in temperature.

Therefore the system is computed in NPT ensemble at different temperatures T_i at 1 bar.

For our case we consider a temperature range from $(248 - 348)\,\mathrm{K}$ in 25 K steps.

- **Linear thermal expansion coefficient**

Linear thermal expansion coefficient at 298 K is calculated by the temperature gradient of the respective average lattice parameter $\langle a \rangle$, $\langle b \rangle$ or $\langle c \rangle$ at constant pressure P (NPT ensemble).

$$\alpha_x^{T_{298\,\mathrm{K}}} = \frac{1}{x}\left(\frac{\partial x}{\partial T}\right)_P, x = a, b, c$$

The slope can be estimated numerically in different ways. A simple way is to employ a finite difference method. There are three fundamental types of it the *forward, backward* and *central* finite difference.

In our case we use the central finite difference method with **five-point stencil**. In general it is formulated in 1D as:

$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + \frac{h^4}{30}f^{(5)(c)}, c \in [x-2h, x+2h]$$

where h is the change of x.

If you use more than five points higher stencil order can be used.

In our case the differentiation can be written as:

$$\frac{\partial x}{\partial T} \approx \frac{\left\langle x^{T_{248\,\mathrm{K}}}\right\rangle - 8\left\langle x^{T_{273\,\mathrm{K}}}\right\rangle + 8\left\langle x^{T_{323\,\mathrm{K}}}\right\rangle - \left\langle x^{T_{348\,\mathrm{K}}}\right\rangle}{12\Delta T}$$

- **Volumetric thermal expansion coefficient** It is the same formulation as for the linear thermal expansion coefficient, except that the volume V is used instead of the individual lattice parameters:

$$\alpha_V^{T_{298\,\mathrm{K}}} = \frac{1}{V}\left(\frac{\partial V}{\partial T}\right)_P$$

The volume of a orthorhombic system can be calculated as:

$$V = a \cdot b \cdot c$$

In general of a non-orthorhombic crystal sytems the volume is calculatd by:

$$V = abc\sqrt{1 - \cos^2(\alpha) - \cos^2(\beta) - \cos^2(\gamma) + 2(\cos(\alpha)\cos(\beta)\cos(\gamma))}$$

Plot Lattice Parameter vs. Temperature

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

import matplotlib as mpl

mpl.rcParams["font.size"] = 15
mpl.rcParams["axes.labelsize"] = 20

temperature = np.array([248.15,273.15,298.15,323.15,348.15])
a = np.array([25.8, 25.77, 25.73, 25.70, 25.66])
a_err = np.array([0.2,0.1,0.15,0.2,0.23])

plt.figure(figsize=(8,4))
plt.errorbar(temperature,a,a_err,fmt='ok',label=r"$\left < a \right
↪  >$")
plt.xlabel(r"$T$ in K")
plt.ylabel(r"$\left < a \right > $ in $\mathrm{\AA}$")
plt.xlim(223,373)
plt.xticks(temperature)
# plt.legend(fontsize=12)
plt.show()
```

exercise_b/analysis_b_files/figure-pdf/fig-lattice-output-1.png

Figure 3: T vs a plot

Experimental Measurement

X-ray diffraction (XRD) powder X-ray diffraction (PXRD) neutron powder diffraction (NPD)

Interaction Energy

The interaction energy is a measurement to estimate the energy share of interaction between the guest and host system. It can be calculated by substracting the single energy shares of the individual systems (U_{guest} and U_{host}) from the energy of the total system ($U_{\mathrm{guest@host}}$):

$$U_{\mathrm{int}} = \langle U_{\mathrm{guest@host}} \rangle - \langle U_{\mathrm{host}} \rangle - \langle U_{\mathrm{guest}} \rangle \, .$$

In doing so you take the average total energy $U = E_{\mathrm{kin}} + U_{\mathrm{pot}}$ of the respective simulated systems (guest@host, host, guest) at a long enough equilibrated trajectories.

Radial Distribution Functions (RDFs)

Distribution Functions (RDFs)

The radial distribution function (RDF) is a simple measurement to get structural information of the system.

It describes the probability $P_{ab}(r)$ to find a target particle type b at a distance r away from a reference particle type a *e.g.* the probability to find $H_2$from Zn^{2+} clusters in a distance r:

$$P_{ab}(r) = \int_0^{r'} 4\pi r'^2 g_{ab}(r') dr'$$

where $g_{ab}(r)$ is the RDF. The RDF formulats the average local density $\langle\rho_{ab}(r)\rangle$ normalized by $\rho = (N_a N_b)$

$$g(r) = \frac{\langle\rho_{ab}(r)\rangle}{\rho}.$$

The two-particle density correlation function is defined as:

$$\rho_{ab}(r) = \sum_{i=1}^{N_a}\sum_{j=1}^{N_b}\langle\delta(|\vec{r_i} - \vec{r_j}| - r)\rangle\,.$$

The average number of particles b that can be found in the shell of distance r can be determined by multiplying the density ρ with the probability $G(r)$

$$N_{ab}(r) = \rho G_{ab}(r)$$

Dirc-Delta Function

$$\delta(x - x_0) = \begin{cases} 0 & x \neq x_0 \\ \infty & x = x_0 \end{cases}$$

$$\int_L \delta(x - x_0)dx = 1, x_0 \in L$$

Self-Diffusion Coefficient

In order to describe the dynamic of guests in host systems, it is necessary to calculate transport properties. A transport property would be for example the *diffusion*, *viscosity*, *electrical or thermal conductivity.*

In general transport properties can be described as a property coefficient γ which depends on microscopic variable A (*e.g.* positions $\Delta\vec{r}$ or the velocities $\vec{v}$) that is written as an infinite time integration of a non-normalized time correlation function $\left\langle \dot{A}(t)\dot{A}(t_0)\right\rangle$:

$$\gamma = \int_0^\infty dt\left\langle \dot{A}(t)\dot{A}(t_0)\right\rangle.$$

This is also known as **Green-Kubo** relation [@Kubo1957] which can also written as equivalent **Einstein** expression

$$\gamma = \lim_{t\to\infty}\frac{\left\langle (A(t) - A(t_0))^2\right\rangle}{2t} = \frac{1}{2}\lim_{t\to\infty}\frac{d}{dt}\left\langle (A(t) - A(t_0))^2\right\rangle$$

In case of self-diffusion coefficient D_s the variable A is the molecular position $\vec{r}_\mathrm{cm}$ which in general is the center of mass position of the particles:

$$\vec{r}_\mathrm{cm} = \frac{\sum_{i=1}^N m_i\vec{r}_i}{\sum_{i=1}^N m_i}$$

where m_i is the atomic mass, $\vec{r}_i$ is the atomic positions and N the number of atoms in the particle.

As well as $\dot{A}$ is the molecular velocity which should be also considered as center of mass.

Einstein-Relation

The Einstein Relation can be therefore written as slope of the mean-squared-displacement $(MSD(\tau))$ over time origins τ

$$D_\mathrm{s} = \frac{1}{2\cdot d}\lim_{t\to\infty}\frac{d}{dt}MSD(\tau)$$

where $d = 1, 2, 3$ is the dimensionality.

Mean-Squared Displacement

The mean-squared displacement describes the the temporally displacement of the particle from a time origin t_0 averaged over the time interval τ

$$MSD(\tau) = \left\langle \frac{1}{N} \sum_{i=1}^{N} |\vec{r}_i(t) - \vec{r}_i(t_0)|^2 \right\rangle_\tau$$

where N is the number of the particle, $\vec{r}_i$ is the center-of-mass position vector of the particle i.

Looking at the Figure Figure **??** (a), we see that the beginning of the MSD is not linear. Only at higher correleation times the normal diffusion (see Figure Figure **??** (b)) behaviour appear.

Question:

- How do you get the self-diffusion coefficient out of the MSD regarding the Einstein-relation?
- How do you get the gradient/slope of the MSD?
- What does the limes in that formular mean?

Answer:

The self-diffusion coefficient is the slope of the MSD divided by $2 \cdot d$.
But this is only true if correlation time is long enough. We can say that in the linear regime we fulfill this limes due to unchanging slope which represents the difffusion coefficient.
Therefore it is important to have really linear behaviour in order to apply the Einstein relation.
You get the slope by fitting a linear regression at the last linear section.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import matplotlib as mpl

mpl.rcParams["font.size"] = 15
mpl.rcParams["axes.labelsize"] = 20

data = np.genfromtxt("../data/diff.out")
steps = data[:,0]
time = steps*0.002
MSD_x = data[:,1]
MSD_y = data[:,2]
MSD_z = data[:,3]
MSD_xyz = MSD_x + MSD_y + MSD_z

def ballistic_regime(x,a):
  return a*x**2

def superdiffusion(x,a):
  return a*x**1.5

def subdiffusion(x,a):
  return a*x**0.5

def normaldiffusion(x,a):
  return a*x
```

```
def confined_diffusion(x,a,b):
  return a*(1-np.exp(-b*x))

def linear_regression(x,a,b):
  return a*x + b

popt, pcov = curve_fit(linear_regression,time[-500:],MSD_xyz[-500:])

d_einstein = popt[0]/6
time_regime= np.linspace(0,10,1000)

fig, axs = plt.subplots(2,1,figsize=(4,8),gridspec_kw={'hspace': 0.4})
axs[0].plot(time,MSD_x,label=r"$MSD_x$",color="C0")
axs[0].plot(time,MSD_y,label=r"$MSD_y$",color="purple")
axs[0].plot(time,MSD_z,label=r"$MSD_z$",color="grey")
axs[0].plot(time,MSD_xyz,label=r"$MSD_{x+y+z}$",color="k")
axs[0].vlines(time[-500],0,500,color="k",linestyle="dotted",label=⌟
↪  r"fitting
↪  regime")
axs[0].plot(time[-500:],linear_regression(time[-500:],*popt),color=⌟
↪  "red",linestyle="dashed",label=r"linear fit $y=a\cdot x + b$: $a=
↪  {:.2f}$, $b= {:.2f}$ ".format(popt[0],popt[1]))
axs[0].text(-0.5, 1.05, '(a)', transform=axs[0].transAxes,
↪  fontsize=14, verticalalignment='top')
axs[0].text(1.0,400, r"$D_s=%2.2f\,\mathrm{\AA^2 ps^{-1}}$"
↪  %(d_einstein),fontsize=14)
axs[0].legend(fontsize=12,loc='upper left', bbox_to_anchor=(1.05,
↪  1.05))


axs[1].plot(time_regime,ballistic_regime(time_regime,100),label=⌟
↪  r"ballistic regime $\propto \tau^2$", color="gold",
↪  linestyle="dashed")
axs[1].plot(time_regime,superdiffusion(time_regime,100),label=⌟
↪  r"superdiffusion regime $\propto \tau^{\alpha}, \alpha>1$",
↪  color="darkred", linestyle="dashed")
axs[1].plot(time_regime,normaldiffusion(time_regime,100),label=⌟
↪  r"normal diffusion regime $\propto \tau$", color="red",
↪  linestyle="dashed")
axs[1].plot(time_regime,subdiffusion(time_regime,100),label=⌟
↪  r"subdiffusion regime $\propto \tau^{\alpha}, \alpha<1$",
↪  color="salmon", linestyle="dashed")

axs[1].plot(time_regime,confined_diffusion(time_regime,50,2),label=⌟
↪  r"confined diffusion regime $\propto$ const.", color="hotpink",
↪  linestyle="dashed")


axs[0].set_xlabel(r"correlation time $\tau$ in ps")
axs[0].set_ylabel(r"$MSD(t)$ in $\mathrm{\AA^2}$")

axs[0].set_ylim(0,500)


axs[1].set_xlabel(r"correlation time $\tau$ in ps")
axs[1].set_ylabel(r"$MSD(t)$ in $\mathrm{\AA^2}$")
axs[1].set_xlim(0,5)
axs[1].set_ylim(0,500)
```

```
axs[1].text(-0.5, 1.05, '(b)', transform=axs[1].transAxes,
↪  fontsize=14, verticalalignment='top')

axs[1].legend(fontsize=12,loc='upper left', bbox_to_anchor=(1.05,
↪  1.05))
plt.show()
```

exercise_b/analysis_b_files/figure-pdf/fig-msd-output-1.png

Figure 4: (a) MSD(t) plot, (b) diffusion regimes

Green-Kubo Relation

The Green-Kubo formalism needs a numerical integration of the velocity-autocorrelation function $VACF(t)$ to get the self-diffusion coefficent D_s

$$D_\mathrm{s} = \frac{1}{d}\int_0^\infty VACF(t)dt$$

where $d = 1, 2, 3$ is the dimension.

Veloctiy-Autocorrelation Function

The Velocity-Autocorrelatin Function $VACF(\tau)$ is averaged over a time interval τ

$$VACF(\tau) = \left\langle \frac{1}{N}\sum_{i=1}^{N} |\vec{v}_i(t)\vec{v}_i(t_0)|^2 \right\rangle_\tau$$

where $\vec{v}_i$ is the velocity of the particle i and t_0 labels the time origin.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simpson
from scipy.integrate import cumulative_trapezoid
import matplotlib as mpl

mpl.rcParams["font.size"] = 15
mpl.rcParams["axes.labelsize"] = 20

data = np.genfromtxt("../data/green_kubo.out")
time = data[:,0]
VACF_x = data[:,1]
VACF_y = data[:,2]
VACF_z = data[:,3]
VACF_xyz = data[:,4]


integral_VACF_xyz = simpson(VACF_xyz, x=time)
cumulative_integral = cumulative_trapezoid(VACF_xyz, time, initial=0)
d_green_kubo = integral_VACF_xyz / 3

fig, axs = plt.subplots(1,figsize=(4,4))
axs.plot(time,VACF_x,label=r"$VACF_x$",color="C0")
axs.plot(time,VACF_y,label=r"$VACF_y$",color="purple")
axs.plot(time,VACF_z,label=r"$VACF_z$",color="grey")
axs.plot(time,VACF_xyz,label=r"$VACF_{x+y+z}$",color="k")
axs.plot(time, cumulative_integral, label=r'Cumulative Integral of
↪  $VACF_{x+y+z}$',color="red")

axs.text(0.05, 0.95, r"$D_s = %.2f \, \mathrm{\AA^2 ps^{-1}}$"
↪  %(d_green_kubo), transform=axs.transAxes, fontsize=14,
↪  verticalalignment='top')


axs.set_xlabel(r"correlation time $\tau$ in ps")
axs.set_ylabel(r"$VACF(t)$ in $\mathrm{(\AA ps^{-1})^{2}}$")
axs.legend(fontsize=12,loc='upper left', bbox_to_anchor=(1.05, 1.05))
plt.show()
```

exercise_b/analysis_b_files/figure-pdf/fig-vacf-output-1.png

Figure 5: VACF(t) plot

Activation Energy

The diffusion coefficient is directly depending on the activation energy E_a by an exponential decay

$$D_\mathrm{s} = D_0 e^{-\frac{E_\mathrm{a}}{RT}}$$

where D_0 is factor of the exponential function, $R = 8.314\,\mathrm{JK^{-1}mol^{-1}}$ is the gas constant and T is the temperature.

With this formalism we can apply an Arrhenius equation to fit a linear function to get the activation energy E_a

$$\ln(D_\mathrm{s}) = -\frac{E_\mathrm{a}}{RT} + \ln D_0$$

exercise_b/analysis_b_files/figure-pdf/fig-ea-output-1.png

Figure 6: (a) D(T), (b) D(n), (c) ln(D(1000/T) and (d) Ea(n))

Checklist

All data generated along with a discussion of results fit easily on a single page. The images and graphs to be plotted are grouped into three subsections, which should also be grouped together in the protocol.

Subsection 1 - System Setup and Equilibration:

- ☐ A screenshot of the CO2@ZIF-8 hybrid system including the simulatin box
- ☐ A graph showing the lattice parameter of the pristine (= empty) ZIF-8 against the simulation time a(t)

Subsection 2 - Gas Diffusion:

- ☐ A graph showing the mean-square-displacement (MSD) of the CO2 molecules vs time including a fit
- ☐ A graph showing the Green-Kubo correlation function vs time and the associated integral

Subsection 3 - Group Work:

- ☐ A graph showing the lattice parameter of the pristine (= empty) ZIF-8 at the five different temperatures (248.15 K - 348.15 K) and the resulting linear thermal expansion coefficient
- ☐ A graph showing the Arrhenius plot of the Einstein diffusion coefficient at the five different temperatures (298.15 K - 398.15 K) including a fit
- ☐ A graph showing the Arrhenius plot of the Green-Kubo diffusion coefficient at the five different temperatures (298.15 K - 398.15 K) including a fit

Templates

Matplotlib

```python
import numpy as np

data = np.genfromtxt("../data/data.dat",delimiter="   ",
↪  skip_header=2)
print(data)
```

```
[[ 100.          4.36417563   -1.19      ]
 [ 200.          3.97386645   31.19      ]
 [ 300.          3.81670518   43.19      ]
 [ 400.          3.63958609   60.1      ]
 [ 500.          3.36586221  108.77      ]
 [ 600.          3.30685375  123.16      ]
 [ 700.          2.95904139  192.74      ]
 [ 800.          2.7512791   233.53      ]
 [ 900.          2.73239376  235.13      ]
 [1000.          2.37106786  284.71      ]
 [1100.          2.22010809  301.51      ]]
```

Two axis plot

```python
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator,FixedLocator


# global matplotlib settings
plt.rcParams.update({
    'legend.fontsize': 18,
    'figure.figsize': (4, 4),
    'axes.labelsize': 25,
    'xtick.labelsize': 20,
    'ytick.labelsize': 20}
)

# create a figure with a subfigure to create 2 different y-axes
fig, ax1 = plt.subplots()
```

```
# create second subfigure with with sharing the x-axis of ax1
↪  subfigure
ax2 = ax1.twinx()
# plot the first data as line plot as ax1 subfigure
# set color, linewidth, marker symbol and label
ax1.plot(data[:,0], data[:,1], color="grey", linewidth=3,
↪  marker="x",label="data1")
# define the color of the y-axis
ax1.tick_params(axis='y', labelcolor="grey")

# plot the second data as the ax2 subfigure by using scatter plot
ax2.scatter(data[:,0], data[:,2],color="black",
↪  linewidths=3,label="data2")
ax2.tick_params(axis='y', labelcolor="black")

# add all labels in one
lines1, labels1 = ax1.get_legend_handles_labels() # get labels of ax1
↪  subfigure
lines2, labels2 = ax2.get_legend_handles_labels() # get labels of ax2
↪  subfigures
ax2.legend(lines1 + lines2, labels1+ labels2, loc=4) # location of
↪  legend

# set x- and y-limits of the axes
ax1.set_xlim(0,1500)
ax1.set_ylim(-0.1,5)
ax2.set_ylim(-0.1,400)

# set the x- and y-labels of the axes
ax1.set_xlabel(r"$x$ / a.u")
ax1.set_ylabel(r"$y_1$ / a.u.",color="grey") # define also the color
↪  of the labels
ax2.set_ylabel(r"$y_2$ / a.u.",color="black")

# create a vertical line at 800 starting from 0 until 5
ax1.axvline(800,0,5,color="red",linewidth=3)

# define manjor and minor axes ticks
ax1.xaxis.set_major_locator(FixedLocator(np.arange(0, 2000, 500)))
ax1.xaxis.set_minor_locator(AutoMinorLocator(2))
ax1.yaxis.set_major_locator(FixedLocator(np.arange(-1, 6, 1)))
ax1.yaxis.set_minor_locator(AutoMinorLocator(2))
ax2.yaxis.set_major_locator(FixedLocator(np.arange(-100, 500, 100)))
ax2.yaxis.set_minor_locator(AutoMinorLocator(2))

# save the figure as png
# plt.savefig("figure.png", dpi=300)
# show the picture
plt.show()
```

templates/matplotlib_files/figure-pdf/cell-3-output-1.png

Line spectra with Gaussian broadening

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

# create random frequencies between 20 and 5000 cm-1
vib = np.random.rand(10)*(5000-20)
# with random intensity
intensity = np.random.rand(10)

# define gaussian broadening
def spectrum(vib,intensity,sigma,v):
    gvib=[]
    for vibi in v:
        tot=0
        for vibj,I in zip(vib,intensity):
            tot = tot + I*np.exp(-((((vibj-vibi)/sigma)**2)))
        gvib.append(tot)
    return gvib

# create the broaded function with smooth frequency values
v=np.linspace(0,5000, num=10000, endpoint=True)
# use different sigma
sigma1=100
sigma2=200
sigma3=400

gvib1=spectrum(vib,intensity,sigma1,v)
gvib2=spectrum(vib,intensity,sigma2,v)
gvib3=spectrum(vib,intensity,sigma3,v)

fig,ax=plt.subplots(figsize=(4,4))

# plot the gaussian broadenings
ax.plot(v,gvib1,"--k", label=r"$\sigma_1=100\,\mathrm{cm^{-1}}$")
ax.plot(v,gvib2,linestyle="--", color="grey",
↪   label=r"$\sigma_2=200\,\mathrm{cm^{-1}}$")
ax.plot(v,gvib3, linestyle="dashed", color="lightgrey",
↪   label=r"$\sigma_3=500\,\mathrm{cm^{-1}}$")

# plot the line spectra
for v,I in zip(vib,intensity):
    ax.plot((v,v),(0,I),c="red")

# or use vlines
ax.vlines(v,0,I,color="red")

# set x- and y-axis limits
ax.set_xlim(0,6000)
ax.set_ylim(0,3)

# set minor ticks
ax.xaxis.set_minor_locator(AutoMinorLocator(2))
ax.yaxis.set_minor_locator(AutoMinorLocator(2))

# invert the x-axis
plt.gca().invert_xaxis()

# set the axes labels
```

```
plt.xlabel('frequency / cm$^{-1}$')
plt.ylabel('intensity / a.u.')

# plot the legend
plt.legend(fontsize=15)
# save the figure
# # plt.savefig('random_line_spectra.png')

plt.show()
```

templates/matplotlib_files/figure-pdf/cell-4-output-1.png

Band Plot

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

# define the kpoints and the kpoint_labels
kpoints = np.array([1,21,66,76])
kpoint_labels = ["Z",r"$\Gamma$","X","P"]

# an example of random band data
bands = np.genfromtxt("../data/band_tot.dat")
```

```
EFermi = -0.5681 #eV taken from detailed.out
# size of the band data
shape = bands.shape

# put all data in one array to calculate the band gap
y = np.array([])
for i in range(1,shape[1]):
    y = np.append(y,bands[:,i])

x = np.array([])
for i in range(1,shape[1]):
    x = np.append(x,bands[:,0])
# use the convention of E-EFermi
yF = y - EFermi

# the maximum energy of the valence band
max_valence = np.max(yF[yF<0])
# the minimum energy of the conduction band
min_conduction = np.min(yF[yF>0])

k_max_valence = x[np.argmin(yF[yF>0])]
k_min_conduction = x[np.argmax(yF[yF<0])]

# band gap
E_gap = max_valence - min_conduction
```

```
plt.figure()
# plot the bands
for i in range(1,19):
    plt.plot(bands[:,0], bands[:,i]-EFermi, color="black",
 ↪  linewidth=3)
# plot the vertical lines for the k-points of the Brillouin zone
for j in kpoints:
    plt.axvline(j, color="grey", linestyle="dashed")

# set some axes limits
plt.xlim(1,76)
plt.ylim(-6.5,3.5)

# plot Egap
plt.axhline(max_valence, color="black", linestyle="dashed")
plt.axhline(min_conduction, color="black",
 ↪  linestyle="dashed",label=r"$E_{\mathrm{gap}}=%2.3f\,\mathrm{eV}$"
 ↪  %E_gap)

 ⌋
 ↪  plt.plot([k_min_conduction,k_max_valence],[max_valence,min_conduction],
 ↪  color="red",label=r"$E_{\mathrm{indirect,gap}}$")

plt.xticks(kpoints,kpoint_labels)
plt.ylabel(r"$E-E_{\mathrm{Fermi}}$ / eV")
plt.legend(fontsize=12,loc=4)
plt.show()
```

templates/matplotlib_files/figure-pdf/cell-7-output-1.png

```
import numpy as np
import matplotlib.pyplot as plt
# it is used for constants
from scipy import constants
# it is used for showing the latex equations
from IPython.display import display, Latex

# define constants
eV = constants.eV
hbar = constants.hbar
mol = constants.Avogadro
cal = constants.calorie
kB = constants.k
u = constants.u
A = constants.angstrom

print("eV = ",eV, "J")
print("hbar = ",hbar, "J s")
print("mol = ",mol, "mol")
print("cal = ",cal, "J")
print("kB = ",kB, "J K-1")
print("u = ",u, "kg")
print("A = ",A, "m")

display(Latex( r"$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}}$ in cm
↪  $^{-1}$"))

display(Latex(r"$\mu = \frac{m_1 \cdot m_2}{m_1 + m_2}$ in g $\cdot$
↪  mol $^{-1}$"))
```

```
eV =  1.602176634e-19 J
hbar =  1.0545718176461565e-34 J s
mol =  6.02214076e+23 mol
cal =  4.184 J
kB =  1.380649e-23 J K-1
u =  1.66053906892e-27 kg
A =  1e-10 m
```

$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}}$ in cm $^{-1}$

$\mu = \frac{m_1 \cdot m_2}{m_1 + m_2}$ in g $\cdot$ mol $^{-1}$

Plot functions

```
# plt.switch_backend("TkAgg")
# it is used for plotting the figures in the jupyter notebook as
↪  interactive figures
# %matplotlib widget
plt.rcParams.update({
    'legend.fontsize': 9,
    'figure.figsize': (2,2),
    'axes.labelsize': 10,
    'xtick.labelsize': 9,
    'ytick.labelsize': 9}
)

# array of r values from 0 to 10 with 100 points
r = np.linspace(0, 15, 1000)
# force constant is extracted
k = 3 # check the units of k it should be in kcal mol-1 A-2

# translation of the units



# harmonic potential
def harmonic_potential(r, k, r0):
    return 0.5 * k * (r-r0)**2

def morse_potential(r, D, a, r0):
    return D * (1 - np.exp(-a*(r-r0)))**2






fig  = plt.figure(dpi=300)
plt.plot(r, harmonic_potential(r, k, 5), label=r"harm. pot.")
plt.plot(r, morse_potential(r, 15,0.3, 5), label=r"Morse pot.")
```

```python
plt.xlabel(r"$r$ / $\AA$")
plt.ylabel(r"$V(r)$ / kcal mol$^{-1}$")
plt.xlim(0, 15)
plt.ylim(0, 20)
plt.legend(fontsize=6)
plt.tight_layout()
plt.show()
```

templates/matplotlib_files/figure-pdf/cell-9-output-1.png

Fit a function to data

```python
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
import numpy as np
import matplotlib.pyplot as plt
# global matplotlib settings
plt.rcParams.update({
    'legend.fontsize': 9,
    'figure.figsize': (2,2),
    'axes.labelsize': 10,
    'xtick.labelsize': 9,
    'ytick.labelsize': 9}
)
# define the function to fit
def harmonic_potential(r, k, r0):
    return 0.5 * k * (r-r0)**2

# generate some data
r = np.linspace(0, 15, 1000)
k = 3
r0 = 5
data = harmonic_potential(r, k, r0) + np.random.normal(0, 0.5, r.size)

# fit the data
popt, pcov = curve_fit(harmonic_potential, r, data)
# calculate the R2 score
r2 = r2_score(data, harmonic_potential(r, *popt))

# print the fit parameters
print("k = ", popt[0], " +/- ", np.sqrt(pcov[0,0]))
print("r0 = ", popt[1], " +/- ", np.sqrt(pcov[1,1]))
print("R2 = ", r2)



plt.figure(dpi=300)
# plot the data and the fit
plt.plot(r, data, label="random data")
plt.plot(r, harmonic_potential(r, *popt), label=r"$V(r) = \frac{1}{2}
 ↪  k (r-r_0)^2$" + "\n" + r"$k = %2.4f \pm %2.4f$" %(popt[0],
 ↪  np.sqrt(pcov[0,0])) + "\n" + r"$r_0 = %2.4f \pm %2.4f$" %(popt[1],
 ↪  np.sqrt(pcov[1,1])) + "\n" + r"$R^2 = %2.4f$" %r2)
plt.xlabel(r"$r$ / $\AA$")
plt.ylabel(r"$V(r)$ / kcal mol$^{-1}$")
plt.tight_layout()
plt.legend(fontsize=3)
plt.xlim(0, 15)
plt.ylim(0, 20)
plt.show()
```

```
k =  3.0001519238121075  +/-  0.0015751184414068474
r0 =  5.000995310066044  +/-  0.0019478807815955332
R2 =  0.9998536203850723
```

templates/matplotlib_files/figure-pdf/cell-10-output-2.png

Poster Templates

Here are some poster templates:

- UIBK Poster template
- Overleaf Templates

Think about how to create the poster. Figures and tables should be informative and readable. Choose a suitable font size and style. Divide the poster in reasonable sections.

Guides

Statistical Excursion

Regression analsis helps to understand the relationship between independent and dependent variables.

The regression describes a model e.g. linear regression is using a linear function $f(x_i, \alpha)$

$$f = \alpha_0 + \alpha_1 x_i$$

which should describe your relationship between your dependent y_i and the independent x_i variables

$$y_i = f(x_i, \alpha) + \epsilon_i$$

within small error terms ϵ_i.

To estimate the parameter of your model α_i so that the error terms ϵ_i are small as possible the most time an ordinary least squares fit is performed:

$$(\alpha_0, \alpha_1) = \text{argmin}(g(\alpha_0, \alpha_1))$$

where

$$g(\alpha) = \sum_i \epsilon_i^2 = \sum_i (y_i - \alpha_0 + \alpha_1 x_i)^2$$

so that α_0 and α_1 can computed as

$$\alpha_0 = \bar{y} - \alpha_1 \bar{x}$$

$$\alpha_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x}^2)} = \frac{\sum_i \Delta x_i \Delta y_i}{\sum_i \Delta x_i^2}$$

where $\bar{x}$ and $\bar{y}$ are the respective averages and Δx_i Δy_i are the respective deviations.

Linux Quickstart Guide

This guide is for people new to Linux. The aim of this documents is to help with everyday challenges when working with Linux systems. Generally, manual pages (*e.g.* `man sed` gives the terminal manual of sed) are almost everytime helpful and also `-h` (or `--help`) flags after commands quickly display helpful information. Obviously, the internet often helps as well and most of the time new things are learned doing. A good overview of the command line interface on linux is on the following github repository (there is also a german version, but reading the english one is recommended):

https://github.com/jlevy/the-art-of-command-line

Probably most used linux commands:

command [options]	description
cd /path/to/directory | Change to directory	
mkdir [options] directory	Create a new directory
touch filename	Create an empty file with the specified name
cp [options] source destination	Copy files and directories
mv [options] source destination	Rename or move file(s) or directories
ls [options]	List directory contents
pwd	Display the pathname for the current directory
rm [options] directory	Remove (delete) file(s) and/or directories
rm -r [options] directory	Delete directories
cat [filename]	Display file's contents to the standard output device
less [options] [filename]	View the contents of a file one page at a time
tail [options] [filename]	Display the last n lines of a file (the default is 10)
head [options] [filename]	Display the first n lines of a file (the default is 10)
grep [options] pattern [filesname]	Search files or output for a particular pattern
chmod [options] mode filename	Change a file's permissions (u+x to make executable)
find [pathname] [expression]	Search for files matching a provided pattern.
sort [options] file	sort the file via [option]
ps [options]	Display a snapshot of the currently running processes
top	Displays the resources being used on your system. Press q to exit
kill [options] pid	Stop a process. If the process refuses to stop, use kill -9 pid

More commands are:

Table 4: Basic Linux Commands (modified version of source: here)

command [options]	description
man [command]	Display the help information for the specified command
echo [options]	Print string, variable, etc.
tar [options] filename	Store and extract files from a tarfile (.tar) or tarball (.tar.gz or .tgz)
bc	basic calculator (e.g. echo “3+2”| bc gives 5 for floats use flag -l)
chown [options] filename	Change who owns a file
clear	Clear a command line screen/window for a fresh start
date [options]	Display or set the system date and time
df [options]	Display used and available disk space
du [options]	Show how much space each file takes up
ln [options] source [destination]	Create a shortcut.
locate filename	Search a copy of your filesystem for the specified filename.
passwd [name [password]]	Change the password
ssh [options] user@machine	Remotely log in to another Linux machine, over the network
who [options]	Display who is logged on
> [oufile]	Write stdout (text written to terminal) to outfile (e.g. dftb+ > dftb.out)

You can use `Tab` for autocompletion of commands.

Commands can usually be combined with the pipe: `|`. To write a pipe press `Alt Gr + <|>`-key (usually located on the left of Y on German keyboards) in between. For example:

```
grep "Done" dftb.out | cat -n
```

To send your job to the background, stop it with `CTRL+Z` and then enter `bg`. Afterwards, you can logout with `CTRL+D` and the process will still be running (can be observed with `top` when relogging in) after you closed the terminal.
On the desktop computers in the office you can search the history with `PageUp` and `PageDown` keys. The prefix you entered will then be autocompleted by searching in the history for matching commands you entered before.

Note

If you want quick access to files or directories in your home folder, the `~`-symbol is an alias for your home directory `/home/user` (= `\$HOME`).

How to: edit files

For editing files you can use any editor that you like *e.g.*:

- `VSCode` (code)
- `nedit`
- `gedit`
- `Emacs` (emacs)

For quick changes in files `vim` is a good option as it runs in the terminal (no `-X` in `ssh` required). To learn the basic commands the vimtutor (just type "vimtutor" in your terminal) is highly recommended. **To exit vim, just type :q and Enter.**

Vim cheat sheet

How to: gawk

Basic usage of `gawk`:

```
gawk '{ [commands] }' file
```

Example: print the first and third column of the file `example.dat`. Be sure to add some spaces in between the columns using "   ":

```
gawk '{print $1"    "$3}' example.dat
```

Example: print the first column and $500 * x^2$ of the first column column of the file `x.dat` to calculate a harmonic potential and output then data to the file `potential.dat`:

```
gawk '{print $1"    " 500*$1*$1}' x.dat > potential.dat
```

For more advanced usage, also `printf` and `if/else` statements can be used similar in the language `C`. And `sed` (see below) can be used to substitute characters with white spaces which then generate new columns that can be printed with `gawk`.

How to: sed

Basic usage of `sed`:

```
sed -flags '[commands]' file
```

Example: every occurrence (g for global at the end) of "apple" in `fruit.dat` will be substituted (`s` at the beginning) by "banana":

```
sed 's/apple/banana/g' fruit.dat
```

To extract lines between two strings (print every line between "pear" and "kiwi"), use:

```
sed '/pear/,/kiwi/p' fruit.dat
```

Note

Note that symbols, numbers, words, strings can be replaced with a white space (" ") to separate columns which can be printed with `gawk`. The combination of sed and gawk is very powerful!

How to: for-loops in bash

A simple for loop looks like:

```
for i in [list]
    do
    [commands]
    done
```

For every item in [list] the commands between the lines `do` and `done` will be executed. For example to print all the items in the fruit list:

```
for i in pear banana apple
    do
    echo $i
    done
```

or print a sequence of numbers (here 1-10):

```
for i in $(seq 1 1 10)
    do
    echo $i
    done
```

or iterate over output files (here grep for the Total Energy of a QM calculation):

```
for i in $(ls *.out)
    do
    grep "Total Energy" $i
    done
```

How to: ssh & scp

Note the ~ symbol is an alias for `/home/\$USER` where `\$USER` is your username. To connect with a remote pc via `ssh` (no colon!)

```

ssh username@remote_host
```

To enable streaming of windows (*e.g.* `nedit`) to your computer add `-X` (when accessing from home, `-CY` might be faster and more stable):

```
ssh -X username@remote_host
```

Copy file from a remote host to local host (your computer):

```
scp   username@from_host:one_file.txt   some/folder/
```

Copy file from local host (your computer) to a remote host:

```
scp   one_file.txt   username@to_host:/some/folder/
```

Copy directory from a remote host to local host:

```
scp -r   username@from_host:/some/folder/    other/folder/
```

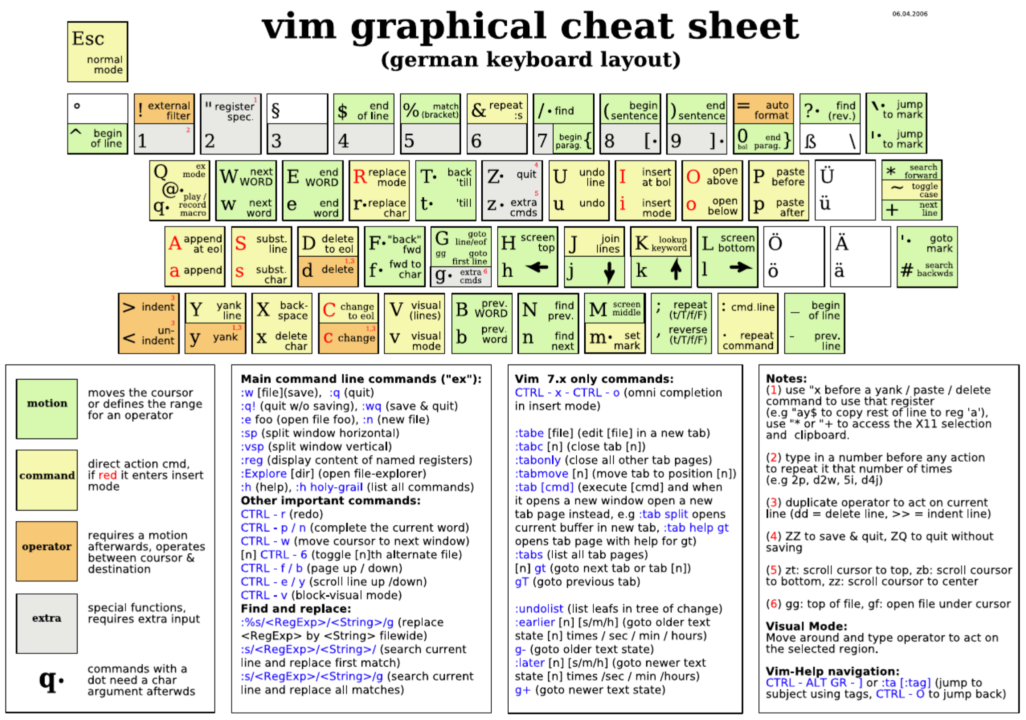
Copy directory from local host to a remote host:

```
scp -r   some/folder/   username@to_host:/other/folder/
```

The use of rsync is recommended. See:
https://www.tecmint.com/rsync-local-remote-file-synchronization-commands/

To login into remote computers without password authentication run *ssh-keygen* (just press *Enter* a few times) and *ssh-copyid* $USER@$HOST and enter the password. This is especially convenient for accessing the clusters.

Appendix: Vim Cheat Sheet



moves the coursor or defines the range for an operator

direct action cmd, if red it enters insert mode

requires a motion afterwards, operates between coursor & destination

special functions, requires extra input

commands with a dot need a char argument afterwds

Main command line commands ("ex"):
:w [file](save), :q (quit)
:q! (quit w/o saving), :wq (save & quit)
:e foo (open file foo), :n (new file)
:sp (split window horizontal)
:vsp (split window vertical)
:reg (display content of named registers)
:Explore [dir] (open file-explorer)
:h (help), :h holy-grail (list all commands)
Other important commands:
CTRL - r (redo)
CTRL - p / n (complete the current word)
CTRL - w (move coursor to next window)
[n] CTRL - 6 (toggle [n]th alternate file)
CTRL - f / b (page up / down)
CTRL - e / y (scroll line up /down)
CTRL - v (block-visual mode)
Find and replace:
:%s/<RegExp>/<String>/g (replace <RegExp> by <String> filewide)
:s/<RegExp>/<String>/ (search current line and replace first match)
:s/<RegExp>/<String>/g (search current line and replace all matches)

Vim 7.x only commands:
CTRL - x - CTRL - o (omni completion in insert mode)

:tabe [file] (edit [file] in a new tab)
:tabc [n] (close tab [n])
:tabonly (close all other tab pages)
:tabmove [n] (move tab to position [n])
:tab [cmd] (execute [cmd] and when it opens a new window open a new tab page instead, e.g :tab split opens current buffer in new tab, :tab help gt opens tab page with help for gt)
:tabs (list all tab pages)
[n] gt (goto next tab or tab [n])
gT (goto previous tab)

:undolist (list leafs in tree of change)
:earlier [n] [s/m/h] (goto older text state [n] times / sec / min / hours)
g- (goto older text state)
:later [n] [s/m/h] (goto newer text state [n] times /sec / min /hours)
g+ (goto newer text state)

Notes:
(1) use "x before a yank / paste / delete command to use that register (e.g "ay$ to copy rest of line to reg 'a'), use "* or "+ to access the X11 selection and clipboard.

(2) type in a number before any action to repeat it that number of times (e.g 2p, d2w, 5i, d4j)

(3) duplicate operator to act on current line (dd = delete line, >> = indent line)

(4) ZZ to save & quit, ZQ to quit without saving

(5) zt: scroll cursor to top, zb: scroll coursor to bottom, zz: scroll coursor to center

(6) gg: top of file, gf: open file under cursor

Visual Mode:
Move around and type operator to act on the selected region.

Vim-Help navigation:
CTRL - ALT GR -] or :ta [:tag] (jump to subject using tags, CTRL - O to jump back)

Figure 7: Vim Cheat Sheet

Visual. Quickstart Guide

xmgrace

A tutorial can be found here:
https://universe.bits-pilani.ac.in/uploads/Pilani/navin/ReadPDFDOC/xmgrace.pdf

gnuplot

A lot of example plots can be seen here (an example plot is also shown in the appendix):
http://gnuplot.sourceforge.net/demo/

Gnuplot template for pdf plots (resulting example plot shown below):

```
#!/usr/bin/gnuplot
reset
# plot a pdf
set terminal pdfcairo enhanced font "Times New Roman,12.0" rounded
set output 'confinement_plot.pdf'
set border linewidth 1.0
set size ratio 0.55
# set plot line styles
set style line 1 linecolor rgb 'black'   linetype 1 linewidth 1
set style line 2 linecolor rgb '#FFB85F' linetype 1 linewidth 1
 ↪  #yellow
# set legend position
set key left top
# set axis properties
set ylabel 'V(r)'
set xlabel 'r / Bohr'
set xtics nomirror
set ytics nomirror
set xr[0:10]
set yr[0:9]
#variables
w = 8.5
r0 = 4.0
a = 2.0
r1=2.5
#functions
f (x) = w / ( 1 + exp (a * (r0 - x)))
g (x) = (x/r1)**2
# plot funcitons
plot  f(x)  title 'Wood-Saxon Confinement' w lines linestyle 1, \
          g(x)  title 'Power Confinement'      w lines linestyle 2

### for plotting data files:
# plot "name_of_datafile.dat" w lines
```

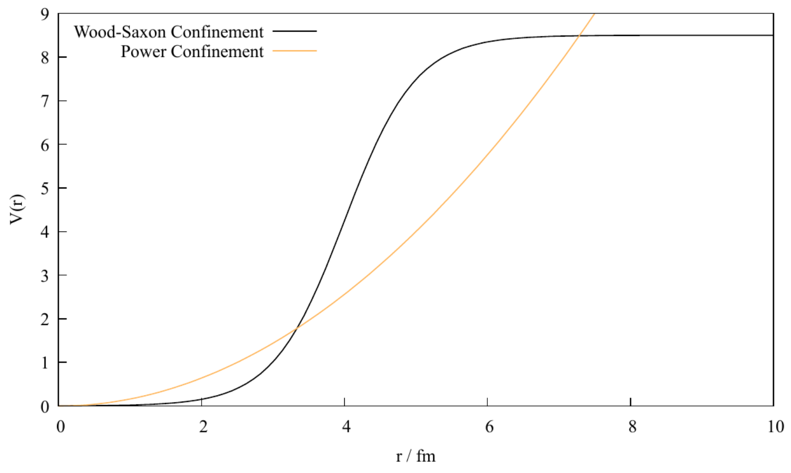


Figure 8: gnuplot example plot

matplotlib

Quickstart guide:
https://matplotlib.org/stable/users/getting_started/index.html

julia

Found on the net (not testet yet, but seems useful):
https://nextjournal.com/leandromartinez98/tips-to-create-beautiful-publication-quality-plots-in-julia

How to: vmd eye candy

- Set background color to white (a good .vmdrc file is shown in the appendix)

- Adjust camera to a nice perspective of the molecule and set the resolution of atoms and bonds in graphical representations to 25
- File -> Render ...
- Select "Tachyon"
- add to the the command before the -o flag "-res 3000 3000" so that i looks like "... TARGA -res 3000 3000 -o ..."
- autocrop and scale the image to a smaller size it with gimp, done!

Good default `VMD` settings (stolen from Bernhard, copy content end save it in `.vmdrc` in your home directory):

```
color Display Background white
color Axes Labels black
color Labels Bonds black
color Labels Angles black
display depthcue off
## position and turn on menus
menu main move 5 225
menu display move 395 30
menu graphics move 395 500
menu main on
menu graphics on
mol delrep 0 top
mol representation Licorice 0.100000 12.000000 1.000000
mol addrep top
mol representation DynamicBonds 1.600000 0.100000 12.000000
mol addrep top
```

Latex Quickstart Guide

- A Short Introduction to LATEX
- The Not So Short Introduction to LATEX 2_ϵ
- Overleaf Documentation A list of LaTeX editors:

- TeXStudio
- Kile
- Any texteditor with a LaTeX plugin (*e.g.* VScode, (neo)vim, Kile, ...)
- Overleaf (online tool) - it is not recommended if you want to compile a very large project like a thesis because it is getting very slow.

Python Quickstart Guide

- A good free option to learn python is `python` for Everybody
- Excersim offers programming challenges (various languages) Udemy courses have a good reputation as well.

Python Tutorial

Welcome to the `python` tutorial! In this tutorial, we will cover the basics of the `python` programming language, including variables, data types, loops, and functions.

Variables

In `python`, a variable is a container that holds a value. To create a variable, you simply assign a value to it using the assignment operator (=). For example:

```
x = 5
```

This creates a variable called `x` and assigns it the value `5`. You can then use the variable `x` in your code to refer to the value `5`.

Data Types

`python` has several built-in data types, including:

- Integers: whole numbers, like `5`
- Floats: decimal numbers, like `3.14`
- Strings: sequences of characters, like `"hello"`
- Booleans: values that can be either `True` or `False`

You can use the `type()` function to check the data type of a variable. For example:

```
x = 5
print(type(x)) # Output: <class 'int'>
```

Loops

Loops are used to repeat a block of code multiple times. In `python`, there are two types of loops: `for` loops and `while` loops.

`For` loops are used to iterate over a sequence of values, such as a list or a string. For example:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
  print(fruit)
```

This will output each fruit in the list, one at a time.

`While` loops are used to repeat a block of code while a certain condition is true. For example:

```
x = 0
while x < 5:
  print(x)
  x += 1
```

This will output the numbers `0` through `4`, one at a time.

Functions

Functions are reusable blocks of code that perform a specific task. To define a function in `python`, you use the `def` keyword followed by the name of the function and a colon. For example:

```
def greet(name):
  print("Hello, " + name + "!")
```

This defines a function called `greet` that takes a single argument called `name`. You can then call the function using the name of the function followed by parentheses containing any arguments. For example:

```
greet("Alice") # Output: Hello, Alice!
```

This will call the `greet` function with the argument `"Alice"`, which will print the message "Hello, Alice!".

Conclusion

That's it for the `python` tutorial! We've covered the basics of variables, data t

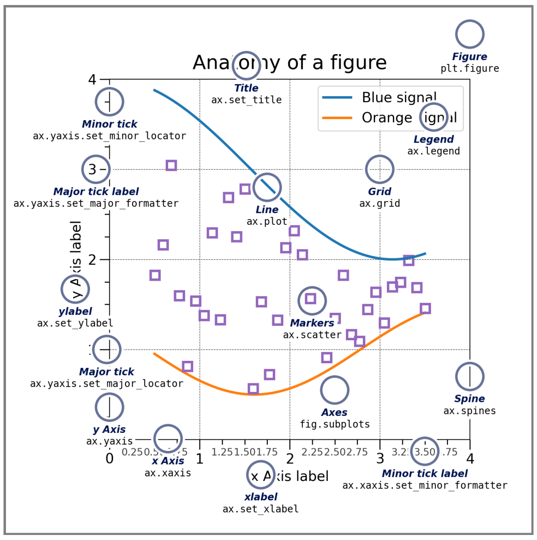
Data Visualization

Matplotlib is the main plotting library in Python. It is a very powerful tool for creating high-quality plots and figures.

More informations can be found under https://matplotlib.org/

```
from matplotlib import pyplot as plt
# change style to default
plt.style.use('default')
```

Components of Matplotlib Figure:



Global settings

```
import matplotlib as mpl
mpl.rcParams['font.size'] = 20
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.linestyle'] = '-'
mpl.rcParams['figure.figsize'] = (3,3)
```

Figure

```
fig = plt.figure()
plt.xlabel('x') # x label
plt.ylabel('y') # y label
plt.title('Title') # title
# plot line with x and y data , label for legend
plt.plot([1, 2, 3], [4, 5, 6], label='line1')
plt.xlim(0, 4) # x axis limits
plt.ylim(4, 6) # y axis limits
plt.xticks([1, 2, 3, 4]) # x axis ticks
plt.yticks([ 4, 5, 6]) # y axis ticks
plt.grid(True) # show grid
plt.legend() # show legend
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-4-output-1.png

Axes

```
fig, ax = plt.subplots()
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Title')
ax.plot([1, 2, 3], [4, 5, 6], label='line1')
ax.set_xlim(0, 4)
ax.set_ylim(4, 6)
ax.set_xticks([1, 2, 3, 4])
ax.set_yticks([ 4, 5, 6])
ax.legend()
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-5-output-1.png

Use different styles

You can use different styles for your plots. The following code shows how to use the `ggplot` style.

```
plt.style.use('ggplot')
```

```
fig = plt.figure()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Title')
plt.plot([1, 2, 3], [4, 5, 6], label='line1')
plt.xlim(0, 4)
plt.ylim(4, 6)
plt.xticks([1, 2, 3, 4])
plt.yticks([ 4, 5, 6])
plt.grid(True)
plt.legend()
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-7-output-1.png

Other style can be found under https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

Subplots

You can create subplots in a figure. The following code shows how to create a figure with 2x2 subplots.

```
plt.style.use('default')
```

```
fig, axes = plt.subplots(2, 2, sharex=False, sharey=False, figsize=(5,
↪  5),gridspec_kw={'hspace': 0.3, 'wspace': 0.2})
axes[0, 0].set_title('Title 1')
axes[0, 0].plot([1, 2, 3], [4, 5, 6], label='line1')
axes[0, 0].set_xlim(0, 4)
axes[0, 0].set_ylim(4, 6)
axes[0, 0].set_xticks([1, 2, 3, 4])
axes[0, 0].set_yticks([ 4, 5, 6])
axes[0, 0].legend()
axes[0, 1].set_title('Title 2')
axes[0, 1].plot([1, 2, 3], [-4, -5, -6], label='line1')
axes[0, 1].set_xlim(0, 4)
axes[0, 1].set_ylim(-6,-4)
axes[0, 1].set_xticks([1, 2, 3, 4])
axes[0, 1].set_yticks([ -4, -5, -6])
axes[0, 1].legend()
axes[1, 0].set_title('Title 3')
axes[1, 0].plot([-1, -2, -3], [4, 5, 6], label='line1')
axes[1, 0].set_xlim(-4,0)
axes[1, 0].set_ylim(4, 6)
axes[1, 0].set_xticks([-1, -2, -3, -4])
axes[1, 0].set_yticks([ 4, 5, 6])
axes[1, 0].legend()
axes[1, 1].set_title('Title 4')
axes[1, 1].plot([-1, -2, -3], [-4, -5, -6], label='line1')
axes[1, 1].set_xlim(-4,0)
axes[1, 1].set_ylim( -6,-4)
axes[1, 1].set_xticks([-1, -2, -3, -4])
axes[1, 1].set_yticks([ -4,- 5, -6])
axes[1, 1].legend()
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-8-output-1.png

You can also use `gridspec` to create subplots.

```
from matplotlib.gridspec import GridSpec
fig = plt.figure()
gs =  GridSpec(2,2 ,width_ratios=[1, 2], height_ratios=[4, 1])
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])
ax3 = fig.add_subplot(gs[1, 0])
ax4 = fig.add_subplot(gs[1, 1])
fig.suptitle('Title') # title for the entire figure
for i,ax in enumerate(fig.get_axes()):
    ax.set(xlabel='x', ylabel='y')
    ax.plot([1, 2, 3], [4, 5, 6],label = "ax%d" %i)

plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-9-output-1.png

Colors and Color maps

Matplotlib provides a large number of color maps. Here your can find a list of all available color maps: https://matplotlib.org/stable/tutorials/colors/colormaps.html and colors: https://matplotlib.org/stable/tutorials/colors/colors.html and https://matplotlib.org/stable/gallery/color/named_colors.html.

Texts and Annotations

```
import numpy as np
x = [0.2,0.4,0.6,0.8,1.0]
y = [0.2,0.4,0.6,0.8,1.0]
plt.text(0.5, 0.5, 'This is a text!', fontsize=12, ha='center')
plt.annotate('Data!', xy=(0.2, 0.2), xytext=(0.5,
↪  0.4),arrowprops=dict(facecolor='black', shrink=0.05))
plt.scatter(x,y, color='red',marker='x',s=100)
plt.xlabel(r'$x$ / mm') # using LaTeX syntax
plt.ylabel(r'$\frac{y}{z}$ / mm$^{-1}$') # using LaTeX syntax
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-10-output-1.png

Logarithmic scale

```
y = np.random.normal(loc=0.5,scale=0.4,size=10000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))
plt.figure()
```

```
# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)

# log
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)

# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthresh=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95,
↪  hspace=0.25,
                    wspace=0.35)

plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-11-output-1.png

Scatter / Line plot

```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 4, 9, 16, 25])
y2 = y + 10
```

```
plt.scatter(x, y, s=100, c='green', edgecolor='black', linewidth=1,
↪  alpha=0.75, marker='o', label='scatter')
plt.plot(x, y2, color='red', marker='o', markersize=10,
↪  markerfacecolor='blue', linestyle='--', linewidth=2,label='line')
plt.legend()
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-13-output-1.png

Error bars

```
plt.style.use('default')
plt.errorbar(x, y, yerr=5, fmt='o', color='black', ecolor='lightgray',
↪  elinewidth=3, capsize=10)
# fmt is the format of the marker, ecolor is the color of the error
↪  bar, elinewidth is the width of the error bar line, capsize is the
↪  size of the error bar cap
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-14-output-1.png

Save figure

```
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y, label='line1')
# bbox_inches is the bounding box in inches.
# If 'tight', it will fit the figure to the plot area.
plt.savefig('../data/test.png', dpi=300, bbox_inches='tight')
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-15-output-1.png

Histogram

```
x = np.linspace(0, 200,1000)
y = np.random.normal(0,1,1000)
fig, ax = plt.subplots()
ax.hist(y, bins=30, color='lightblue', edgecolor='black')
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-16-output-1.png

```python
# 2D histogram
x = np.random.normal(0,1,1000)
y = np.random.normal(0,1,1000)

plt.hist2d(x, y, bins=30, cmap='Blues')
colorbar = plt.colorbar()
colorbar.set_label('counts in bin')
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-17-output-1.png

Density plot

```python
x = np.linspace(0, 100, 1000)
y = np.linspace(0, 100, 1000)*2
# Create a 2D array of shape (1000,1000)
X, Y = np.meshgrid(x, y)
# Z is a 2D array of shape (1000,1000)
Z = np.sin(X) + np.cos(Y)
print("Shape of X:", X.shape)
print("Shape of Y:", Y.shape)
print("Shape of Z:", Z.shape)
```

```
Shape of X: (1000, 1000)
Shape of Y: (1000, 1000)
Shape of Z: (1000, 1000)
```

```python
fig, ax = plt.subplots()
# Create a contour plot
plt.contour(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-19-output-1.png

```python
# 2D grid is interpreted as an image with imshow
plt.imshow(Z, extent=[0, 100, 0, 100], origin='lower', cmap='RdGy')
plt.colorbar()
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-20-output-1.png

```
# contour plot with labels
X = np.linspace(-5, 5, 100)
Y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))

contours = plt.contour(X,Y,Z,3, colors='black') #
↪  plt.contour([X,Y],Z,[levels])
plt.clabel(contours, inline=True, fontsize=6)
```

```
<a list of 16 text.Text objects>
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-21-output-2.png

Seaborn

`Searborn` is based on `Matplotlib`and is made for statistical graphics. It is comparable with `R`'s `ggplot2` library.

It works best with `Pandas` DataFrames. You can easly create complex plots with only a few lines of code by grouping and aggregating data.

More informations can be found under https://seaborn.pydata.org/

```
import seaborn as sns
```

Example

```
# Use the default data set from seaborn
tips = sns.load_dataset('tips')
# Create a boxplot
sns.boxplot(x='day', y='total_bill', data=tips)
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-23-output-2.png

```
import matplotlib.pyplot as plt
import seaborn as sns


sns.set_theme(style="darkgrid")
iris = sns.load_dataset("iris")

# Set up the figure
f, ax = plt.subplots(figsize=(8, 8))
ax.set_aspect("equal")

# Draw a contour plot to represent each bivariate density
sns.kdeplot(
    data=iris.query("species != 'versicolor'"),
    x="sepal_width",
    y="sepal_length",
    hue="species",
    thresh=.1,
)
```

```
<Axes: xlabel='sepal_width', ylabel='sepal_length'>
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-24-output-2.pn

3D plots

Matplotlib can also be used to create 3D plots but it is not the best tool for this.

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

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111, projection='3d')
x = np.random.standard_normal(100)
y = np.random.standard_normal(100)
z = np.random.standard_normal(100)
ax.scatter(x, y, z, c='r', marker='o')
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.tight_layout() # adjust the plot to the figure
plt.show()
```

guides/VisualizationPythonPackages_files/figure-pdf/cell-25-output-1.pn

Better tools for 3D plots are `Mayavi` and `Plotly`.

Other plotting libraries

- Plotly
- Mayavi
- Bokeh
- Altair
- ggplot
- pygal
- pandas