PR Theoretical Chemistry and Computer-Chemistry (Advanced)

Table of contents

Instructions

Welcome to the instruction page of the PR Theoretical Chemistry and Computer-Chemistry (Advanced)!

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

In the QM part of this course, one poster has to be created for each of the exercises (i.e. 2 posters).

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

# 100 linearly spaced numbers
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()
```

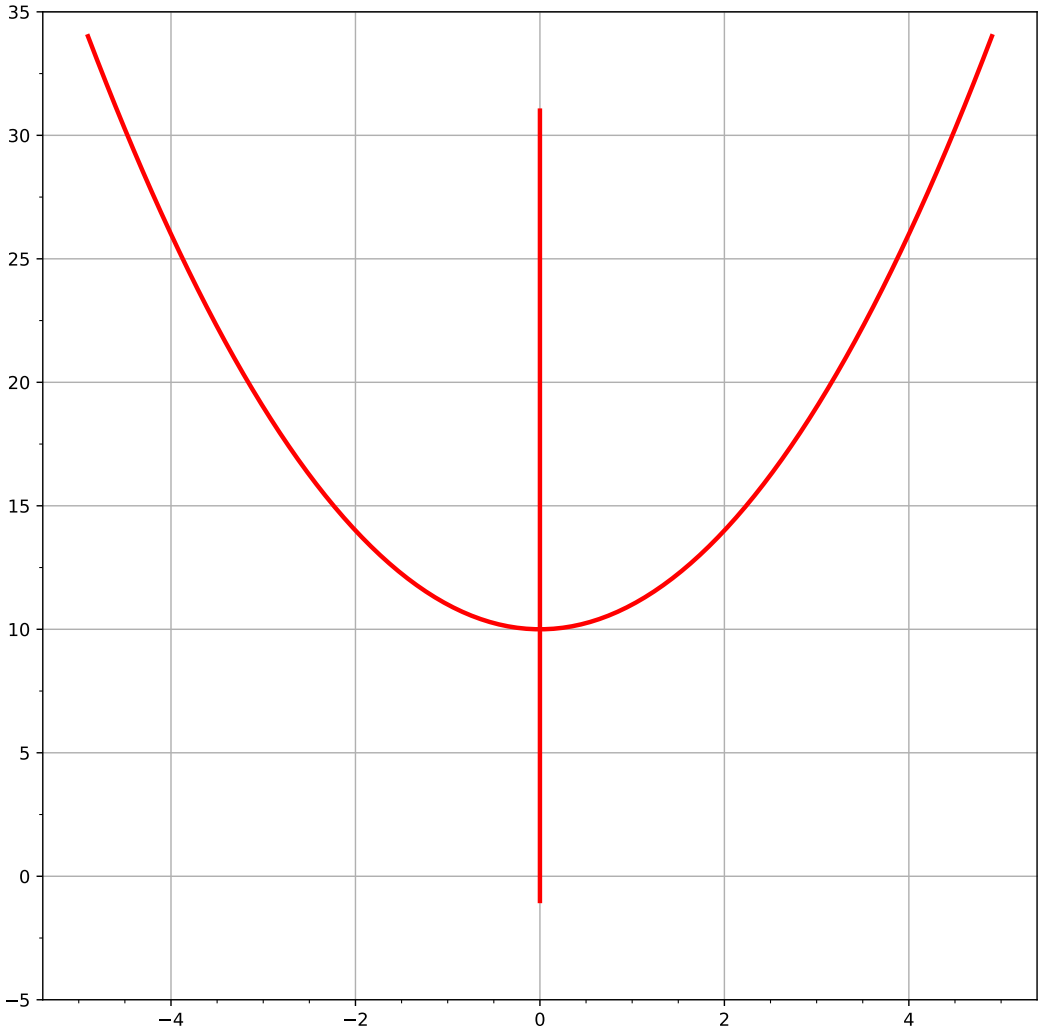


Figure 1: Example Code with corresponding Plot. Any similarity to the Greek Letter Ψ is by coincidence

Timeschedule for the next two weeks

Week 1	Morning : 09:00 - 13:30	Afternoon : 13:30-18:00
Monday	MM-MD 1 Introduction	MM-MD 1 Setup
Tuesday	**Exercise A: Numerov Introduction**	**Exercise A: Numerov Setup**
Wednesday	**Exercise B: NPP-MD Introduction**	**Exercise B: NPP-MD Setup**
Thursday	**Exercise A: Numerov Analyse**	**Exercise A: Numerov Analyse**
Friday	MM-MD 2 Introduction	MM-MD 2 Setup

Week 2	Morning : 09:00 - 13:30	Afternoon : 13:30-18:00
Monday	**Exercise B: Analyse Introduction**	**Exercise B: Analyse**
Tuesday	**Exercise B: Analyse**	**Exercise B: Analyse**
Wednesday	MM-MD Analyse	MM-MD Analyse
Thursday	MM-MD Analyse	MM-MD Analyse
Friday	MM-MD Analyse	MM-MD Analyse

Posters Submission Date

> ! Important
>
> **Submission until: XX.XX.XX**

Excercise A - Vibration Spectroscopy

Background Info A

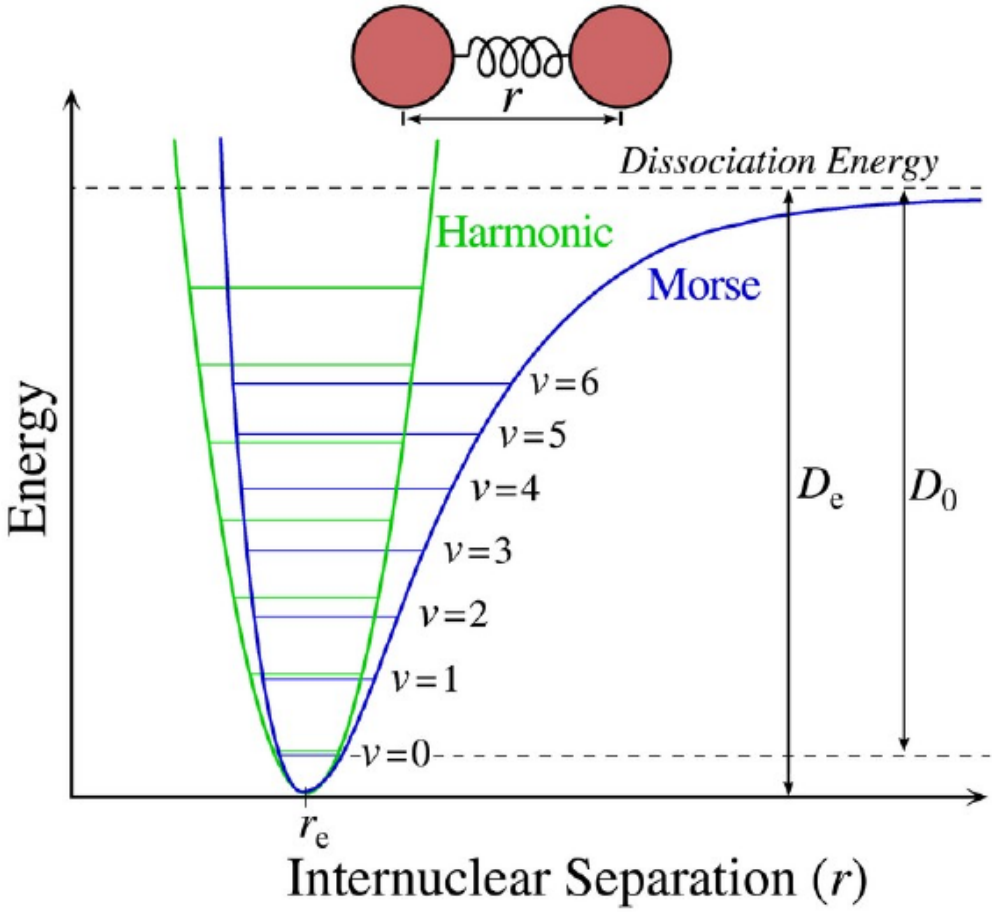
Exercise A – Vibrational spectroscopy using nuclear wave functions

The simplest way to generate vibrational spectra is the **harmonic approximation**: all second derivatives of the energy with respect to the nuclear degrees of freedom are computed at a pre-optimized minimum structure, yielding the Hessian:

$$H_{ij} = \frac{\partial^2 \langle E \rangle}{\partial q_i \partial q_j}$$

After extraction of the force constants k of each vibrational mode from the Hess-matrix (via diagonalisation), the harmonic frequencies ν can be obtained via

$$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}}$$



with μ being the associated reduced mass. However, a short-coming of this approach is the **neglect of anharmonicity**, **mode coupling** and **nuclear quantum effects**. In some cases these contributions cancel, while for other systems the error in the harmonic frequencies can be up to $300\,\mathrm{cm}^{-1}$.

An elegant alternative is the calculation of vibrational wave functions by numerically solving the nuclear Schrödinger equation. One method to achieve this is a method according to **Boris V. Numerov** requiring the potential curve as input in form of a discretised grid. The latter can be obtained from a potential energy scan executed for instance using the QM program `gaussian`. The difference between the eigenvalue of the ground state ($n = 0$) and the first ($n = 1$) and second ($n = 2$) excited states correspond to the expected vibrational frequency of the fundamental and first overtone, when divided by Planck's constant:

$$\nu_{n0} = \frac{E_n - E_0}{h}$$

From the frequencies ν_{n0} (in s^{-1}) the respective wave numbers $\bar{\nu}_{n0}$ (in cm^{-1}) have to be calculated and reported. Knowledge of the wave function also enables the calculation of the oscillator strengths f_{n0}, providing an accurate estimate of the expected IR absorption intensities.

$$f_{n0} = \frac{4\pi m_\mathrm{e}}{3\mathrm{e}^2\hbar c}|\mu_{n0}|^2\bar{\nu}_{n0} = 4.702 \cdot 10^{-7}\,\frac{\mathrm{cm}}{\mathrm{D}^2}|\mu_{n0}|^2\bar{\nu}_{n0}$$

To calculate the oscillator strength, the wave numbers $\bar{\nu}_{n0}$ and the respective transition dipole moment μ_{n0} is required. The latter is given as:

$$|\mu_{n0}|^2 = |\mu^x_{n0}|^2 + |\mu^y_{n0}|^2 + |\mu^z_{n0}|^2$$

with

$$|\mu^x_{n0}| = \langle\psi_0|\mu^x|\psi_n\rangle = \int_{-\infty}^{\infty} \psi_0 \mu^x \psi_n dx$$

While this integral appears intimidating, it can be easily evaluated in an approximate way using the trapezoidal rule:

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}(y_1 + 2y_2 + 2y_3 + ... + 2y_{n-1} + y_n)$$

$$\approx \frac{\Delta x}{2}\sum_{i=1}^{n-1}(y_i + y_{i+1})$$

The calculation of the trapezoidal integration is actually very simple and can be easily implemented in a spreadsheet editor such as `libreoffice` or `excel` (or computed using data science software such as `matlab`, `octave`,`python`, `R`, etc.)

A.1) Diatomic molecule

In this part the vibration of a diatomic molecule is investigated. After drawing the molecule using `gaussview`, an energy minimisation (= geometry optimisation) and a harmonic frequency calculation has to be carried out. In the next step a potential energy scan has to be performed in the range of approx. -0.5 Å to +1.5 Å using a step length $\Delta r = 0.01$ Å. This yields about 150 individual potential points.

The energy and dipole moment information can be extracted in the terminal using the provided tool:

```
./extract_data.sh  scan-logfile  >  data-file
```

The data file contains the distance information in Å, the QM energy in Hartree, and the components of the dipole moment in x, y and z direction in Debye. Next, the force constant k according to the harmonic approximation has to be determined via finite differences from the energy column of the data-file (second column).

$$k \approx \frac{E_1 - 2E_{\text{min}} + E_{+1}}{\Delta r^2}$$

with E_{min} being the minum energy and $E_{\pm 1}$ corresponding to the energy values of the two neighbouring points. Please report the force constant in kcal $\cdot$ mol^{-1}Å^{-2}:

Because the program gaussian provides a wrong reduced mass μ in case of diatomic systems, this property has to be calculated manually in this part of the exercise.

$$\mu = \frac{m_1 m_2}{m_1 + m_2}$$

Using the reduced mass μ and the force constant k the frequency (in s^{-1}) and corresponding wave number (in cm^{-1}) should be calculated and compared to the result obtained from gaussian.

> **Note**
>
> Extra care is required to apply all factors to obtain the correct unit of the frequency and wavenumber (i.e. just dividing the force constant k in kcal $\cdot$ mol^{-1}Å^{-2} by the reduced mass μ given in g $\cdot$ mol^{-1} will obviously yield a wrong result …)

In the final step the Numerov calculation is carried out, requiring the data file and the reduced mass as input:

```
./numerov    data-file    mass    >  numerov-outfile
```

The output of the Numerov procedure contains the coordinate and potential information (in units Å and kcal $\cdot$ mol^{-1}) as well as the wave-functions for the five lowest states. The energy eigenvalues are given as comment line (also in kcal $\cdot$ mol^{-1}) on top of the Nuermov output file. The respective energy differences have to be transformed into cm^{-1}.

The report for this part should include:

- A screenshot of the equilibrium geometry (white background!).
- The calculation of the force constant k and the reduced mass μ.
- Calculation of the frequency and wave number showing ALL(!) required conversion factors.
- A figure showing the QM energy as a function of distance and the Numerov wave functions plus the energy graph showing the harmonic approximation (all in the same figure).
- A table comparing the wavenumbers obtained via the Numerov procedure with the results obtained from the harmonic approximation for n = 1 - 4.
- For all points a short description/interpretation should be included as text.

Note

In this part, the transition dipole moments and oscillator strengths are **NOT(!)** required.

A.2) Stretch vibration in a polyatomic molecule

In this example, the X-H stretch motion of a polyatomic molecule (X=O,N) provides a suitable approximation to the actual normal mode. In order to assess the influence of quantum effects, both hydrated and deuterated species (i.e. X-H and X-D) have to be investigated. After the geometry optimisation and frequency computation, the potential energy scan of the X-H distance in the range of approx. 0.5 Å to 2.0 Å has to be carried out, using again a step length of 0.01 Å.

Careful:

Be sure to select the correct bond for the potential energy scan! (This is a **very** frequent error in this excercise.)

As before the energy and dipole moment information can be extracted in the terminal using the provided tool:

```
./extract_data.sh    scan-logfile    >    data-file
```

The resulting data file contains the coordinate, energy and dipole moment information (in units of Å, Hartree and Debye) and can be directly used as input for the Numerov programm:

```
./numerov    data-file    mass    > numerov-outfile
```

In addition to the data file, the reduced mass of the vibrational mode has to be provided as a second input. For polyatomic molecules the reduced mass provided in the gaussian output file is correct! To identify the correct mode, a visual inspection in `gaussview` is required.

Careful:

Be sure to make a separate check in the deuterated case – the order of vibrations is sometimes different (again a **very** common error in this excercise).

As before the output of the Numerov procedure contains the coordinate and potential information (in units Å and kcal $\cdot$ mol^{-1}) as well as the wave-functions for the five lowest states. The energy eigenvalues are given as comment line (also in kcal $\cdot$ mol^{-1}) on top of the file. The respective energy differences have to be transformed into cm^{-1}. Prior to computing the transition dipole moments it should be verified that the wave functions are orthonormal for the states $i, j = 0 - 2$.

$$\langle \psi_i | \psi_j \rangle - \delta_{ij} = 0$$

Next, the oscillator strength f_{n0} (for $n = 1, 2$) should be computed from the transition dipole moments and reported relative to f_{10}. (*i.e.* the oscillator strengths are divided by f_{10}, thus always giving a number of 1.0 for the ground state transition).

The report for this part should include:

- A screenshot of the equilibrium geometries (white background!).
- Two figures comparing the potential energy and the first five wave functions including also the potential energy of the harmonic approximation for both the hydrated and deuterated species.
- A graph showing the components of the dipole moment as a function of the distance.
- Results of the orthonormality check between the states i, $j = 0, 1, 2$ in form of a table.
- A table comparing the frequencies and oscillator strengths of the fundamental and first overtone modes obtained via the Numerov procedure and the harmonic approximation.
- For all points a short description/interpretation should be included as text.

A.3) Protocol:

The protocol in this section should be designed as an A0-Poster (similar to those Ph.D. students typically prepare for their presentations in the Atrium of the CCB building), see poster templeates. All data generated along with a discussion of results fit easily on a single page. Please include your name and matriculation number, and provide the protocol as pdf-file including your surname in the name of the file.

HowTo A

Exercise A – Vibrational spectroscopy using nuclear wave functions

The required scripts and programs for this exercise are provided in the folder:

```
cd /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che
↪ mie/vibration/
```

Note

This HowTo assumes you remember how to
i) create new directories and
ii) copy/move/rename files.

If you are unsure please look at this Linux quickstart guide.

Part A.1) Vibrational spectroscopy of a diatomic molecule at the example of H2

Tasks

- Execute an energy minimisation, a frequency calculation and a potential energy scan using `gaussian`
- Extract the potential energy data from the output file
- Execute a Numerov calculation

Results

- Numeric evaluation of the force constant from the potential plot
- Calculation of the harmonic vibrational wave numbers from the force constant (compare this value to the gaussian result for verification – deviation typically about 1-2 cm^{-1})
- Calculation of the anharmonic vibrational wave numbers from the Numerov eigenvalues
- 1 plot V_{QM} vs bond length including the vibrational wave function from the Numerov output and the harmonic approximation to the potential energy
- Table comparing the harmonic and anharmonic wave numbers for the states 01 to 04
- A screenshot of the equilibrium geometry (use a white background color!)

HowTo Part A.1)

1) Draw the structure using `gaussview` (command `gv`) and save the file, *e.g.* as `h2_opt.com`.

```
gv
```

GaussView

draw stuff
File > Save
Be sure to **NOT** save cartesian coordinates, so "Write Cartesian" must be unticked!

2) Open the File in your favourite editor (`code`, `nedit`, `gedit`, `vi`, …) and change the command line (`#`) to

h2_opt.com

```
# B3LYP/6-311++G(3df,3pd) OPT=TIGHT SCF=VERYTIGHT INT=ULTRAFINE
```

! Important

It is important to include **one (and only one) blank line** after the command line (`#`-line), the title-line as well as the atom list and at least one blank-line after the last input for all `gaussian` input file!

3) Execute the geometry optimisation:

```
g16 h2_opt.com
```

4) Open the log-file (`code`,`nedit`, `gedit`, `vi`, …) and get the equilibrium bond distance,*e.g.*:

```
nedit h2_opt.log
```

h2_opt.log

[...]
[...]
Final structure in terms of initial Z-matrix:
H
H,1,B1
Variables:
B1= 0.74273528
[...]
[...]

5) Copy the in-file and change the command line to that of a frequency calculation, *e.g.*:

```
cp h2_opt.com h2_freq.com
```

h2_freq.com

```
# B3LYP/6-311++G(3df,3pd) FREQ SCF=VERYTIGHT INT=ULTRAFINE
```

Also set the bond length to the minimum distance, *e.g.*

h2_freq.com

[...]
0 1
H
H 1 B1

B1 0.74273528
[...]

6) Execute the frequency calculation and get the harmonic frequency from the input-file

```
g16 h2_freq.com
```

```
nedit h2_freq.log
```

h2_freq.log

[...]
[...]
Harmonic frequencies (cm**-1), IR intensities (KM/Mole), Raman scattering
activities (A**4/AMU), depolarization ratios for plane and unpolarized
incident light, reduced masses (AMU), force constants (mDyne/A),
and normal coordinates:
1
SGG
Frequencies -- **4410.4726**
Red. masses -- ~~1.0078~~
Frc consts -- ~~11.550~~
IR Inten -- 0.0000
[...]
[...]

! Be careful

For diatomics the force constant and the reduced mass are not given correctly! (according to the developers that's a feature, not an error …) This is the reason why in this exercise, the force constant has to be determined from the potential energy scan! Do **NOT** use the force constant from the output file.

7) Copy the in-file and change the command line for the potential energy scan, *e.g.*:

```
cp h2_freq.com h2_scan.com
```

h2_scan.com

```
# B3LYP/6-311++G(3df,3pd) SCAN SCF=VERYTIGHT INT=ULTRAFINE
# GEOM=NOCROWD NOSYMM POP=ALWAYS
```

8) Change the starting value of the bond length by subtracting 0.5 Å, and set up the scan points *e.g.*:

Note

$0.7427352 - 0.5 = 0.2427352$

h2_scan.com

```
[…]
0  1
H
H     1     B1

B1    0.24273528  150  0.01
[…]
```

This will perform a potential energy scan from 0.2427352 Å, increasing the H-H distance 150 times by 0.01 Å, thus the scan will end at a H-H distance of 1.7427352 Å. Don't forget to include a final blank line in the gaussian input.

9) Execute the potential energy scan and grab a coffee:

```
g16 h2_scan.com
```

10) Copy the extraction-script and the Numerov analysis code to your working directory:

```
cp /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che⌋
 ↪  mie/vibration/extract_data.sh
 ↪  .
cp /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che⌋
 ↪  mie/vibration/numerov
 ↪  .
```

11) Extract the potential energy and dipole moment information from the log-file and use **>** to redirect the output:

```
./extract_data.sh h2_scan.log > scan_data.dat
```

> **Warning**
>
> Careful – if told, the redirect `>` will overwrite existing files and there is no undo if this happens!

The file `scan_data.dat` contains five data columns:

scan_data.dat

- column 1 : Distance in Å
- column 2 : Energy in Hartree
- columns 3 to 4 : x, y and z-component of the dipole moment

This file can be directly imported into graphic-programs, *e.g.* `office`,`xmgrace`, `gnuplot`, …

The dipole moment data is only required in part A.2 of this practical course.

12) Calculate the reduced mass of the vibration and execute the Numerov analysis:

- In this course always the reduced mass formula for two particles should be used.

> **Warning**
>
> Be sure to use the mass of the correct isotopes, *e.g.* $^{1}H_2$, $^{2}D_2$, ^{12}C, ^{18}O and do **NOT** use the averaged mass given in the periodic table.

- The Numerov-program requires two inputs, being the data-file from step 11 above and the reduced mass, e.g. in case of $^{1}H_2$:

```
./numerov scan_data.dat 0.50391251605 > numerov.dat
```

Again, be careful while redirecting `>` to not overwrite any files.

The file `numerov.dat` contains a headline (`#`) and twelve data columns. Columns 1 to 7 are to generate the required figure, columns 8 to 12 are for data analysis:

numerov.dat

- # headline – energy eigenvalues E of the five lowest wave functions Ψ_0 to Ψ_4 in kcal/mol
- column 1 : Distance in Å
- column 2 : Energy in kcal/mol and shifted to zero at the minimum
- columns 3 to 7 : wave functions Ψ_0 to Ψ_4 shifted by the respective eigenvalue
- columns 8 to 12: wave functions Ψ_0 to Ψ_4 without shift

This file can be directly imported into graphic programs (e.g. office, xmgrace, gnuplot, …) as well as data software (`office`, `matlab`, `octave`, `R`, `bash`, `origin`, …) to perform the analyses required in A.2.2.

> **Note**
>
> Which software you choose for the analysis part (or if you want to write your own code) is entirely up to you. There are different ways to obtain the requested data, *e.g.* the numerical integration can be carried out i) in an `excel`-sheet, ii) using `phyton`/`R` datascience routines, iii) on the command line using a (very simple) `awk`-script or iv) even graphically in the program `xmgrace`. You can choose whichever method works best for you.

Part A.2) Stretch vibration in a polyatomic molecule

Tasks

i) Execute an energy minimisation, a frequency calculation and a potential energy scan using gaussian
ii) Extract the potential energy data from the output file
iii) Execute a Numerov calculation
iv) Analyse both a hydrated (^{1}H) and deuterated system (^{1}D)

Results

- Numeric evaluation of the force constant from the potential plot
- Calculation of the harmonic vibrational wave numbers from the force constant for ^{1}H and ^{2}D (compare these values to the gaussian results for verification – deviation typically about $1-2$ cm^{-1})
- Calculation of the anharmonic vibrational wave numbers from the Numerov eigenvalues
- 2 plots V_{QM} vs bond length including the vibrational wave function from the Numerov output and the harmonic approximation to the potential energy
- 1 plot showing the components of the dipole moment and the total dipole moment as a function of the distance.
- Results of the orthonormality check between the states $i, j = 0, 1, 2$ should be provided in a table.
- A table comparing the frequencies and oscillator strengths of the fundamental and 1$^{\mathrm{st}}$ overtone modes obtained via the Numerov procedure and the harmonic approximation should be included.
- A screenshot of the equilibrium geometry (use a white background color!)

Follow steps 1) to 12) of part 2.1 for the polyatomic system. Redo the calculations for the deuterated system, but be smart - not all calculations of the steps i) to iii) above have to be repeated for the deuterated system. Just consider which calculation steps are mass-independent and which one is not.

In order to change the isotope from ^{1}H and ^{2}D in the gaussian calculation, the iso-keyword should be applied, *e.g.* in case of a frequency calculation for D_2:

d2_scan.dat

```
# B3LYP/6-311++G(3df,3pd) FREQ SCF=VERYTIGHT INT=ULTRAFINE

D2 Frequency calculation (at minimum geometry)

0 1
H  (iso=2)
H  (iso=2)    1    B1

B1    0.74273528
```

! Common errors/pitfalls in this exercise

- Be **VERY** careful which bond to scan – in many cases the correct bond is NOT `B1` (a very common error!). Which bond to scan depends on the way the molecule is drawn and the resulting order of atoms.
- If you start to draw the molecule starting with the functional group of interest (*e.g.* start with the OH- group in case of methanol), the y-component of the dipole moment will be zero. This means less work in the analysis of the transition dipole moment and the oscillator strengths.
- Be aware of the units of the different properties and take your time to perform a dimensional analysis. For instance, typical units of the force constants k and the reduced mass μ are kcal $\cdot$ mol^{-1} $\cdot$ Å^{-2} and g/mol, respectively. It is not

possible to simple use the associated values to calculate the frequency without first making the units compatible:

$$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}}$$

Here, the units kcal, $Å^{-2}$ and g need to be converted to obtain the unit of s^{-1} for the frequency.

Checklist A

The protocol in this section should be designed as an A0-Poster (similar to those Ph.D. students typically prepare for their presentations in the Atrium of the CCB building). All data generated along with a discussion of results fit easily on a single page. Please include your name and matriculation number, and provide the protocol as pdf-file including your surname in the name of the file.

The report for exercise A.1 should include:

- ☐ A screenshot of the equilibrium geometry (white background!).
- ☐ The calculation of the force constant k and the reduced mass μ.
- ☐ Calculation of the frequency and wave number showing **ALL(!)** required conversion factors.
- ☐ A figure showing the QM energy as a function of distance and the Numerov wave functions plus the energy graph showing the harmonic approximation (all in the same figure).
- ☐ A table comparing the wavenumbers obtained via the Numerov procedure with the results obtained from the harmonic approximation for $n = 01$ - 04.

> **Note**
>
> In this part, the transition dipole moments and oscillator strengths are **NOT(!)** required.

The report for exercise A.2 should include:

- ☐ A screenshot of the equilibrium geometries (white background!).
- ☐ Two figures comparing the potential energy and the first five wave functions including also the potential energy of the harmonic approximation for both the hydrated and deuterated species.
- ☐ A graph showing the components of the dipole moment as a function of the distance.
- ☐ Results of the orthonormality check between the states $i, j = 0, 1, 2$ in form of a table.
- ☐ A table comparing the frequencies and oscillator strengths of the fundamental and first overtone modes obtained via the Numerov procedure and the harmonic approximation.
- ☐ For all points a short description/interpretation should be included as text.

> **Note**
>
> In this part, the transition dipole moments and oscillator strengths **ARE(!)** required for the analysis.

Excercise B - NNP Molecular Dynamics

Background Info B

Exercise B - NNP MD Simulations of H2@ZIF-8

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. alumino- silicates) 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, 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 CO2 and CH4 are widely discussed [2,3]. In addition, MOFs have also emerged as highly promising candidates for the storage of molecular hydrogen as a green energy carrier [4] (due to their ultrahigh surface area, tunable pore dimensions, and rapid kinetics for hydrogen adsorption and desorption).

An increasing number of MOF compounds are discussed as potential hydrogen carrier [4]. In this exercise, H2 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 crystalizes in a cubic unit cell (space group no. 217, I43m) with a lattice parameter of approx 1.7 nm. The cubic unit cell contains a total of 12 Zn2+ 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 hydrogen storage capacity [5-8].

In order to achieve fast and accurate MD simulations, the MACE-MP neural network potential (NNP) 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 coefficent, 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 DOI: 10.1016/j.enchem.2019.100006*

[4] Li, Y.; Guo, Q.; Ding, Z.; Jiang, H.; Yang, H.; Du, W.; Zheng, Y.; Huo, K; Shaw, L. L. “MOFs-Based Materials for Solid-State Hydrogen Storage: Strategies and Perspectives” *Chem. Eng. J.* **2024**, 485, *149665 DOI: 10.1016/j.cej.2024.149665*

[5] Wu, H.; Zhou, W.; Yildirim, T. “Hydrogen Storage in a Prototypical Zeolitic Imidazolate Framework-8” J. Am. Chem. Soc. **2007**, 129, *5314 – 5315 DOI: 10.1021/ja0691932*

[6] Pinjari, S.; Bera, T.; Kjeang, Erik “Room temperature hydrogen storage enhancement in copper-doped zeolitic imidazolate frameworks with trioctylamine” *Sustain. Energy & Fuels* **2023**, 7, *3142 – 3151 DOI: 10.1039/D3SE00277B*

[7] Balderas-Xicohtencatl, R.; Villajos, J. A.; Casabán, J.; Wong, D.; Maiwald, M. Hirscher, M. "ZIF-8 Pellets as a Robust Material for Hydrogen Cryo-Adsorption Tanks" *ACS Appl. Energy Mater.* **2023**, 6, *9145 – 9152 DOI: 10.1021/acsaem.2c03719*

[8] Villajos, J. A.; Balderas-Xicohténcatl, R.; Al Shakhs, A. N.; Berenguer-Murcia, Á.; Buckley, C. E.; Cazorla-Amorós, D.; Charalambopoulou, Georgia; Couturas, F.; Cuevas, F.; Fairen-Jimenez, D.; Heinselman, K. N.; Humphries, T. D.; Kaskel, S.; Kim, Hyunlim; Marco-Lozar, J. P.; Oh, H.; Parilla, P. A.; Paskevicius, M.; Senkovska, I.; Shulda, S.; Silvestre-Albero, J.; Steriotis, T.; Tampaxis, C.; Hirscher, M.; Maiwald, M. "Establishing ZIF-8 as a reference material for hydrogen cryoadsorption: An interlaboratory study" *ChemPhysChem* **2024**,25, *e202300794 DOI: 10.1002/cphc.202300794*

[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)

Analysis B

Preanalysis

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 like expected.

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

It is a good sign if

- ☐ no atom has been shoot out of the box
- ☐ box was not collapsing or blowing up
- ☐ no unexpected movements was happening

ect.

Troubleshooting

If the simulation is not computed as expected some first-aid steps can be done:

- ☐ Try to understanding any error messages that may have arisen
- ☐ Check in the `*.log` file and `*.in` if the simulation was setup correctly.
- ☐ Check the starting structure
- ☐ Check the moldescriptor

ect.

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

```
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=p
 ↪  hysical_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),linest
 ↪  yles="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"$T_{\mathrm{s
 ↪  td}}$")
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)
```

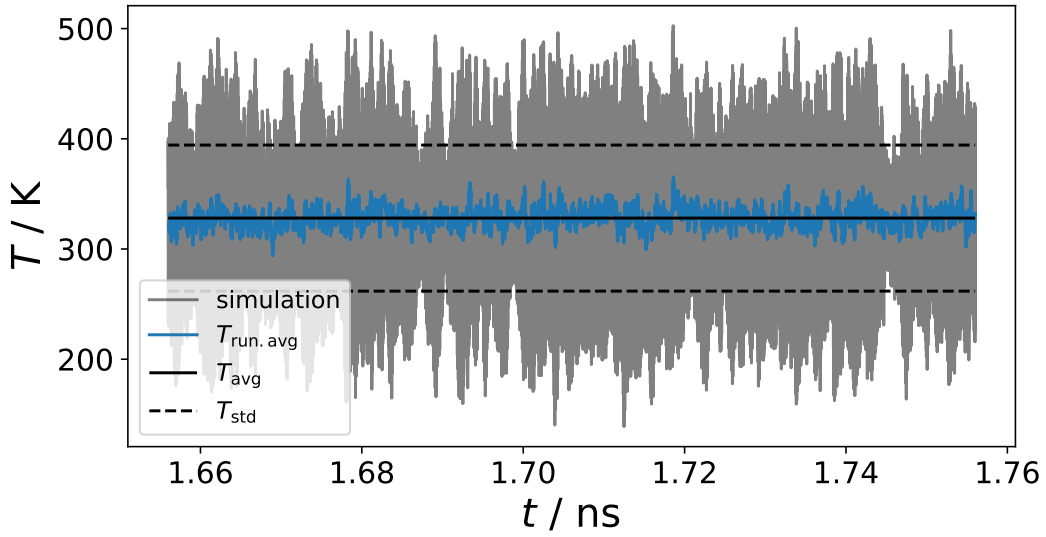


Figure 2: Temperature vs Time plot

Linear / Volumetric Thermal Expansion Coefficient

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 *e.g.* i ={248.15 K, 273.15 K, 298.15 K, 323.15 K, 348.15 K} at 1 bar

- **Linear thermal expansion coefficient**

Linear thermal expansion coefficient at 298 K is calculated by the temperature T 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:

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

- **Volumetric thermal expansion coefficient** It is the same formulation like in the linear thermal expansion coefficient with the difference that the volume V is used instead the individual lattice parameter:

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

The volume of a cubic 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

```
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()
```

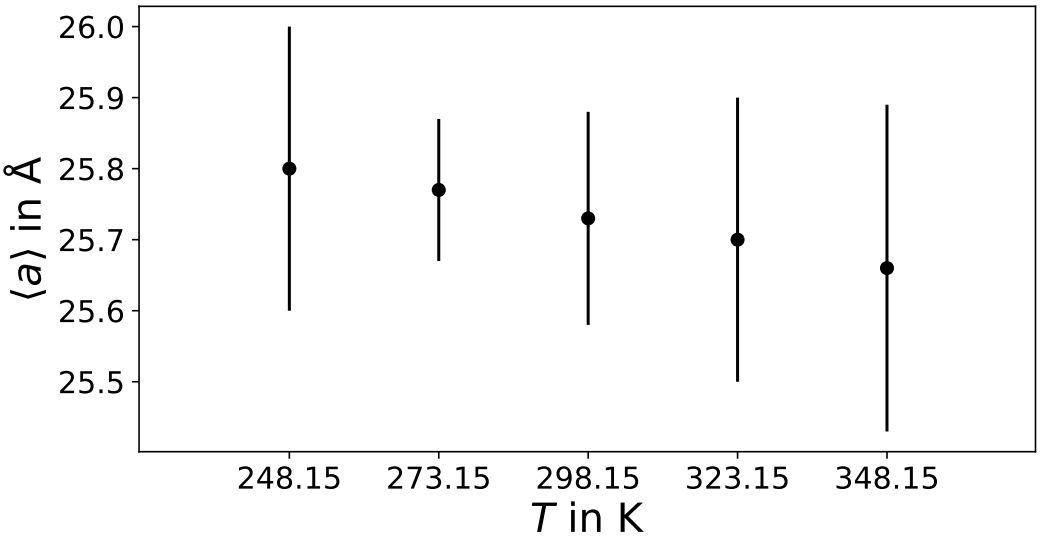


Figure 3: T vs a plot

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 ($U_{\text{guest@host}}$):

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

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

Ensemble

Pay attention:
The $\langle U_{\text{guest@host}} \rangle$ and $\langle U_{\text{host}} \rangle$ energy are determined by a NPT simulation in contrast $\langle U_{\text{guest}} \rangle$ energy has to be calculated in a NVT ensemble. A single gas molecule in a periodic box has to be calculated NVT because NPT would cause into crash of the simulation.

Radial Distribution Functions (RDFs)

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

!RDF-plot

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:

$$\rho_{ab}(r) = \sum_{i=1}^{N_a} \sum_{j=1}^{N_b} \left\langle \delta(|\vec{r}_i - \vec{r}_j| - r) \right\rangle .$$

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

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

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 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.

If we look at the Figure Figure 4 (a) we realize the beginning of the MSD is not linear. Only with higher correleation time, normal diffusional (see Figure Figure 4 (b)) behaviour appear in the linear region.

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 diffusion 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"f
 ↪  itting
 ↪  regime")
axs[0].plot(time[-500:],linear_regression(time[-500:],*popt),color="r
 ↪  ed",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"bal
 ↪  listic regime $\propto \tau^2$", color="gold",
 ↪  linestyle="dashed")
axs[1].plot(time_regime,superdiffusion(time_regime,100),label=r"super
 ↪  diffusion regime $\propto \tau^{\alpha}, \alpha>1$",
 ↪  color="darkred", linestyle="dashed")
axs[1].plot(time_regime,normaldiffusion(time_regime,100),label=r"norm
 ↪  al diffusion regime $\propto \tau$", color="red",
 ↪  linestyle="dashed")
axs[1].plot(time_regime,subdiffusion(time_regime,100),label=r"subdiff
 ↪  usion 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()
```

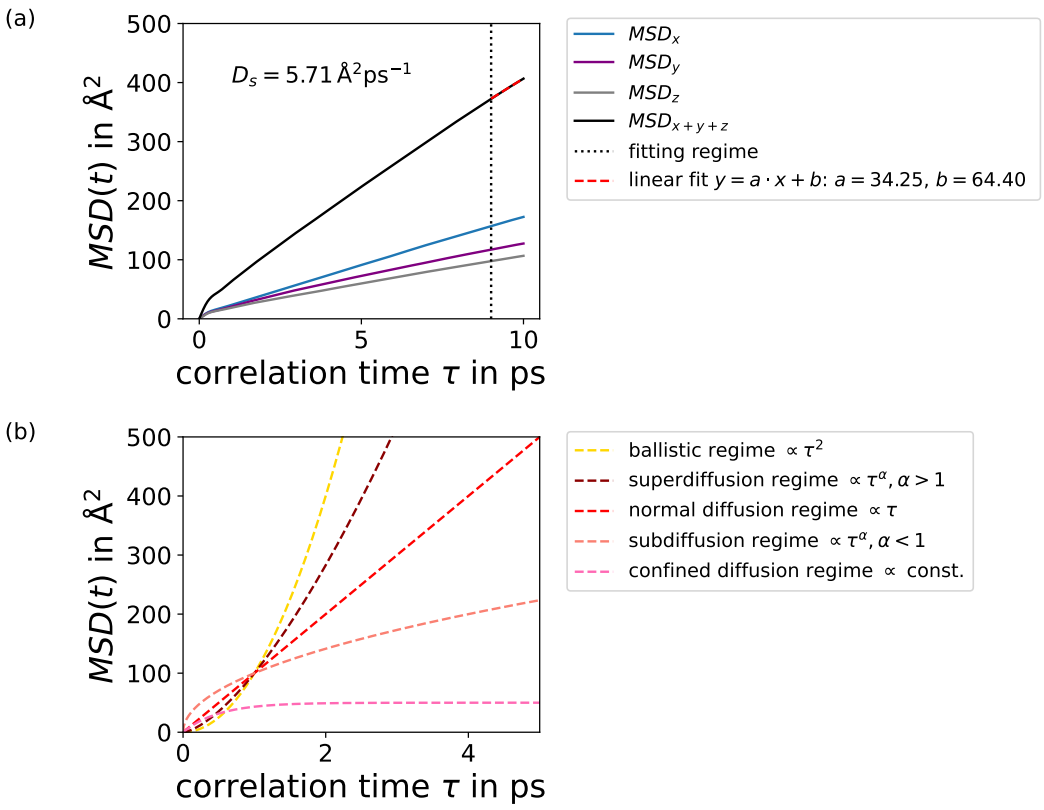


Figure 4: (a) MSD(t) plot, (b) diffusion regimes

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()
```

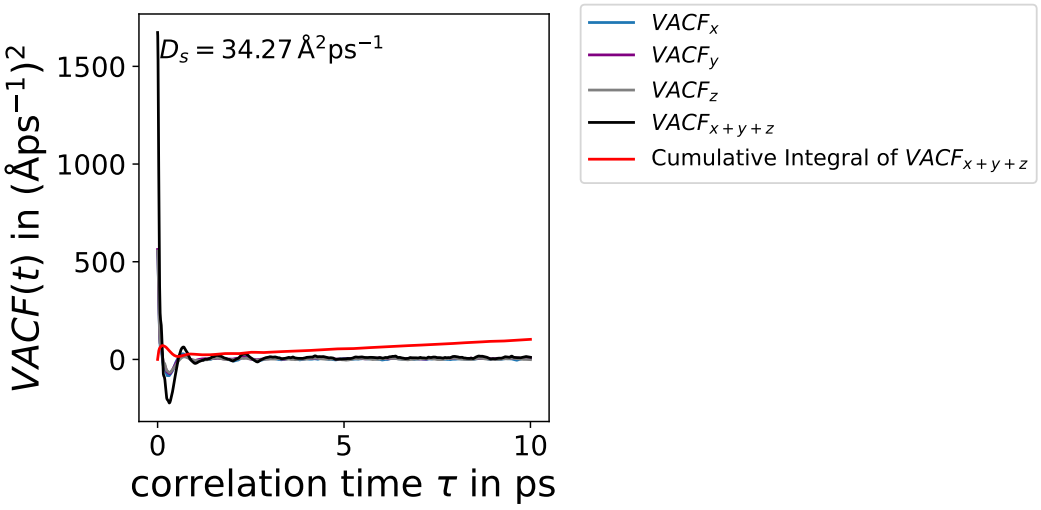


Figure 5: VACF(t) plot

Activation Energy

The diffusion coefficient is directly depending on the actiation energy E_a by an exponential decacy

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

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 respective simulation temperature.

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

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

```
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

diff_4 = np.array([2.5,2.7,2.8,2.9,3.2])
diff_8 = np.array([4.2,5.8,4.7,5.0,5.3])
diff_12 = np.array([5.2,5.8,5.7,6.0,6.3])
diff_16 = np.array([5.2,5.9,6.1,6.5,7.0])
temp = np.array([300,323,348,373,398])
diff_300 = np.array([2.5,4.2,5.2,5.2])
diff_323 = np.array([2.7,5.8,5.8,5.9])
diff_348 = np.array([2.8,4.7,5.7,6.1])
diff_373 = np.array([2.9,5.0,6.0,6.5])
diff_398 = np.array([3.2,5.3,6.3,7.0])
loading = np.array([4,8,12,16])



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

popt_4, pcov_4 = curve_fit(linear_regression,np.log(diff_4),1000/temp)
popt_8, pcov_8 = curve_fit(linear_regression,np.log(diff_8),1000/temp)
popt_12, pcov_12 =
 ↪  curve_fit(linear_regression,np.log(diff_12),1000/temp)
popt_16, pcov_16 =
 ↪  curve_fit(linear_regression,np.log(diff_16),1000/temp)


R=8.313
Ea_4 = popt_4[0]*R
Ea_8 = popt_8[0]*R
Ea_12 = popt_12[0]*R
Ea_16 = popt_16[0]*R

fig, axs = plt.subplots(2,2,figsize=(8,8),gridspec_kw={'hspace':
 ↪  0.4,'wspace': 0.4})
axs[0,0].scatter(temp,diff_4,label=r"$n=4$")
axs[0,0].scatter(temp,diff_8,label=r"$n=8$")
axs[0,0].scatter(temp,diff_12,label=r"$n=12$")
axs[0,0].scatter(temp,diff_16,label=r"$n=16$")
axs[0,0].set_xlabel(r"$T$ in K")
axs[0,0].set_ylabel(r"$D_{\mathrm{s}}$ in $10^{-8}ms^{-1}$")
axs[0,0].legend(fontsize=8,loc='upper right')


axs[0,1].scatter(loading,diff_300,label=r"$T=300\,{\mathrm{K}}$")
axs[0,1].scatter(loading,diff_323,label=r"$T=323\,{\mathrm{K}}$")
axs[0,1].scatter(loading,diff_348,label=r"$T=348\,{\mathrm{K}}$")
axs[0,1].scatter(loading,diff_373,label=r"$T=373\,{\mathrm{K}}$")
axs[0,1].scatter(loading,diff_398,label=r"$T=398\,{\mathrm{K}}$")
axs[0,1].set_xlabel(r"$n$")
axs[0,1].set_ylabel(r"$D_{\mathrm{s}}$ in $10^{-8}ms^{-1}$")
axs[0,1].legend(fontsize=8,loc='lower right')
```

```
axs[1,0].scatter(1000/temp,np.log(diff_4),label=r"$n=4$")
axs[1,0].scatter(1000/temp,np.log(diff_8),label=r"$n=8$")
axs[1,0].scatter(1000/temp,np.log(diff_12),label=r"$n=12$")
axs[1,0].scatter(1000/temp,np.log(diff_16),label=r"$n=16$")
axs[1,0].set_xlabel(r"$1000/T$ in K$^{-1}$")
axs[1,0].set_ylabel(r"$\mathrm{ln}(D_{\mathrm{s}})$")
axs[1,0].legend(fontsize=8,loc='upper right')

axs[1,1].scatter(4,Ea_4)
axs[1,1].scatter(8,Ea_8)
axs[1,1].scatter(12,Ea_12)
axs[1,1].scatter(16,Ea_16)
axs[1,1].set_ylabel(r"$E_{\mathrm{a}}$ in kcal mol$^{-1}$")
axs[1,1].set_xlabel(r"$n$")
axs[1,1].legend(fontsize=8,loc='upper right')

plt.show()
```

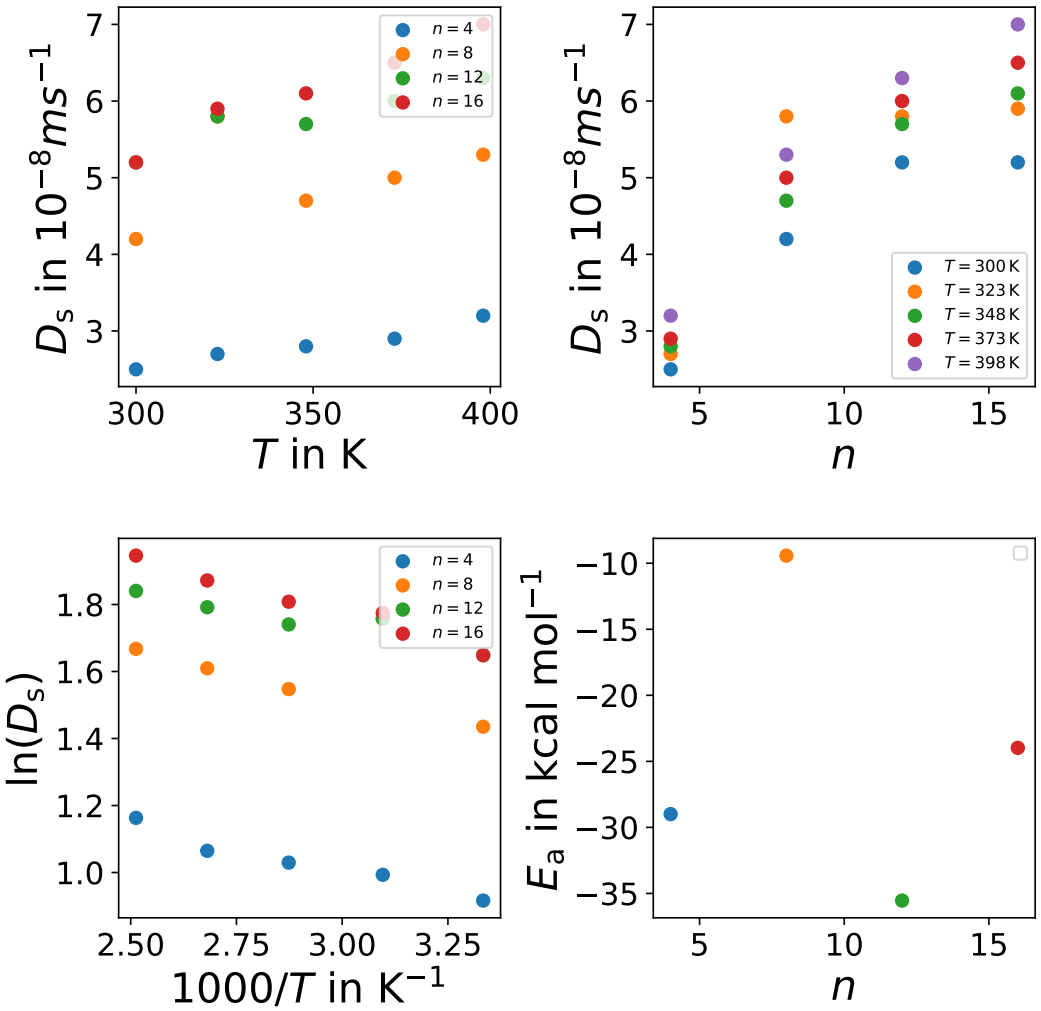


Figure 6: (a) D(T), (b) D(n), (c) ln(D(1000/T) and (d) Ea(n))

HowTo B

> ⚠ Warning
>
> Links to the **Google Sheets** with the assigned temperatures and hydrogen loadings:
> - Thermal Expansion
> - Gas Diffusion

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

Tools and Programs

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_h2`	Extracts the hydrogen molecules of trajectory/velocity-files and calculates the center of mass coordinates/velocities.
`msd`	Calculates the mean squared displacement of the hydrogen molecules from a trajectory. (Note: The hydrogen molecules have to be extracted first using `extract_h2`).
`rdf`	Calculates the radial distribution function of the hydrogen molecule from a trajectory.

Tool	Description
`vacf`	Calculates the velocity autocorrelation function of the hydrogen molecule from a velocity file. (Note: The hydrogen molecules have to be extracted first using `extract_h2`).
`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 `/media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Chemie/zif8-h2/zif8_files` contains all necessary files for the simulations. The files are organized in the following way:

File Structure

```
general_input
    moldescriptor.dat
    run-01.in
    run-02.in
    run-03.in
    run-04.in
h2
    h2.rst
    shake_h2.top
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`).
`h2`	Contains the hydrogen restart-file (`h2.rst`) and the topology file (`shake_h2.top`).
`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 hydrogen molecules in the ZIF-8 framework at the assigned temperature values. The hydrogen loading is set to the two assigned loadings per student.

> **Note**
>
> This part of the exercise is performed twice with the two assigned hydrogen 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 /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che
 ↪  mie/zif8-h2/zif8_files/h2/*
 ↪  .
cp /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che
 ↪  mie/zif8-h2/zif8_files/preeq/<TEMPERATURE>/*
 ↪  .
cp /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che
 ↪  mie/zif8-h2/zif8_files/general_input/*
 ↪  .
```

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

```
add_molecules zif8_preeq_298.rst h2.rst   \
  --rst-mol-desc-file moldescriptor.dat   \
  --top-file shake_zif8_298.top           \
  --added-top-file shake_h2.top           \
  --output-top-file shake.top             \
  -n 32                                   \
  > 32xh2-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
```

> **!** 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 Hydrogen Molecule

Extract the hydrogen molecule from the trajectory and velocity files using the `extract_h2` tool:

```
extract_h2 <trajectory_file>.xyz -n 276
```

and

```
extract_h2 <velocity_file>.vel -n 276
```

> **i** Note
>
> The `-n` flag specifies the number of atoms of the empty ZIF-8 framework.

Einstein Relation

1. Calculate the mean squared displacement (MSD) of the hydrogen 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 hydrogen loadings, insert the corresponding self-diffusion coefficient values into the **Google Sheets**.

Green-Kubo Relation

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

```
vacf
```

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

```
integration
```

> **!** Important
>
> For both assigned hydrogen loadings, insert the corresponding self-diffusion coefficient values into the **Google Sheets**.

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 /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che⌋
 ↪  mie/zif8-h2/zif8_files/preeq/<TEMPERATURE>/*
 ↪  .
cp /media/TC_Lehre/03_PR_Fortgeschrittene_Uebungen_zu_TC_und_Comp_Che⌋
 ↪  mie/zif8-h2/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;
```

> **i** 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
```

> **i** 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

Checklist B

The protocol in this section should be designed as an A0-Poster (similar to those Ph.D. students typically prepare for their presentations in the Atrium of the CCB building). All data generated along with a discussion of results fit easily on a single page. Please include your name and matriculation number, and provide the protocol as pdf-file including your surname in the name of the file. The images and graphs to be plotted are grouped into three subsections, which should also be grouped together on the poster.

Subsection 1 - System Setup and Equilibration:

- ☐ A screenshot of the H2@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 H2 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

How to read data into python:

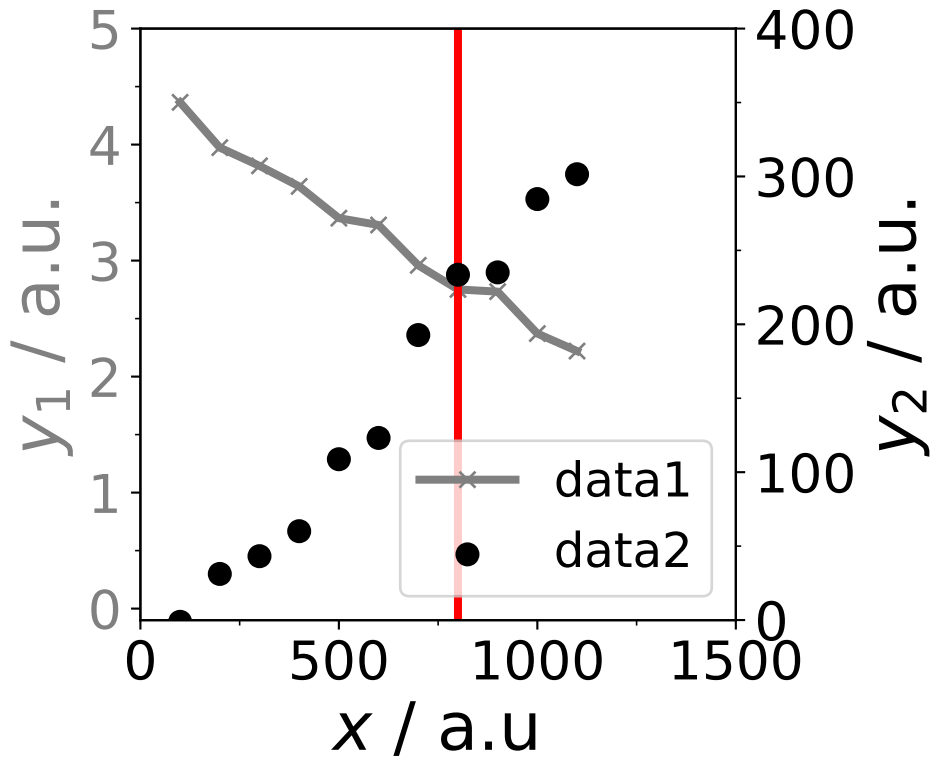
```
import numpy as np

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

```
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
```

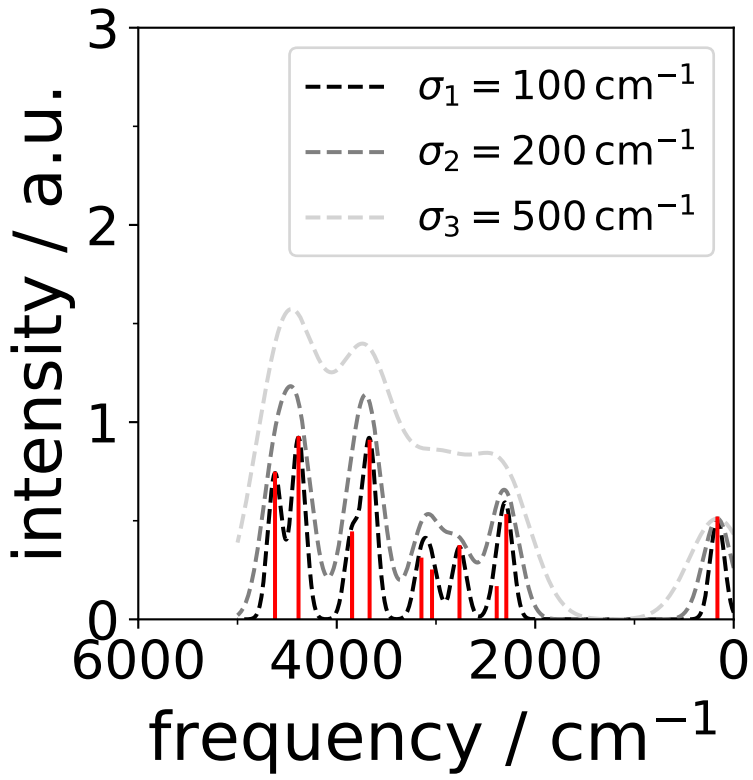
```
ax1.set_xlabel(r"$x$ / a.u")
ax1.set_ylabel(r"$y_1$ / a.u.",color="grey") # define also the color
 ↪  of the labels
ax2.set_ylabel(r"$y_2$ / a.u.",color="black")

# 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()
```



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()
```



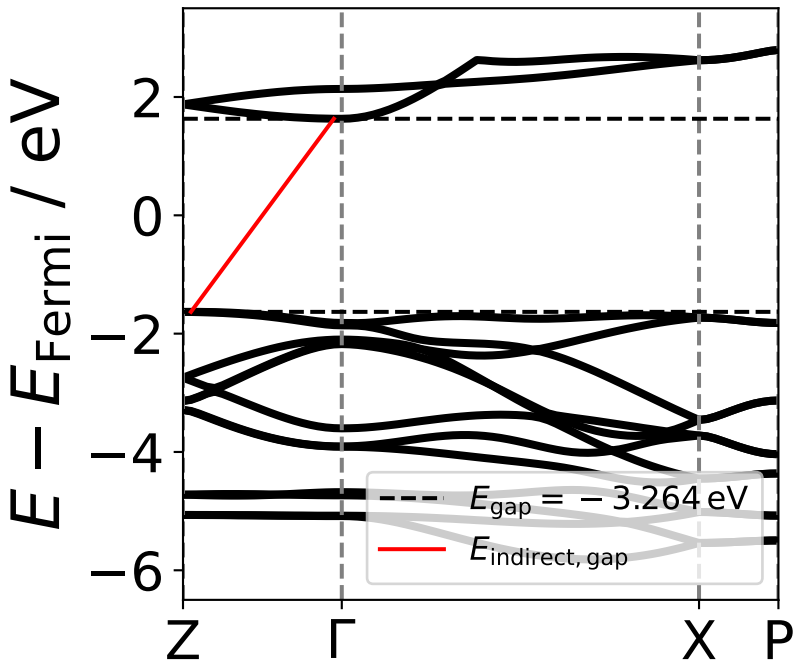
Band Plot

```
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()
```



```
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 =  1.6605390666e-27 kg
A =  1e-10 m
```

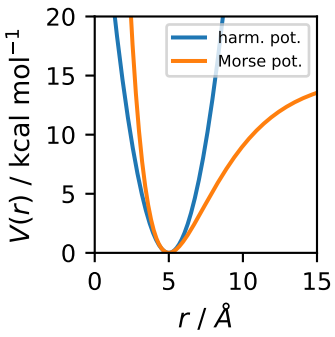
$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}}$ in cm $^{-1}$

$\mu = \frac{m_1 \cdot m_2}{m_1 + m_2}$ 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.")

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()
```

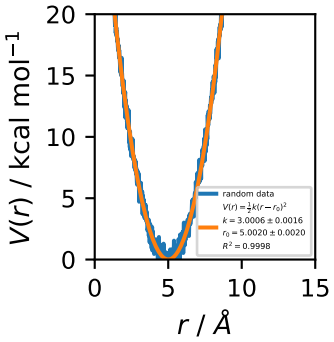


Fit a function to data

```
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 =  3.00063900925827  +/-  0.00159570827887112
r0 =  5.002026481436358  +/-  0.001972764208047263
R2 =  0.999849445867121
```



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

Linear Regression:

$$y = ax + b$$

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$b = \frac{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i}{n}$$

Linux Quickstart Guide

Introduction

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)

command [options]	description
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 6: 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

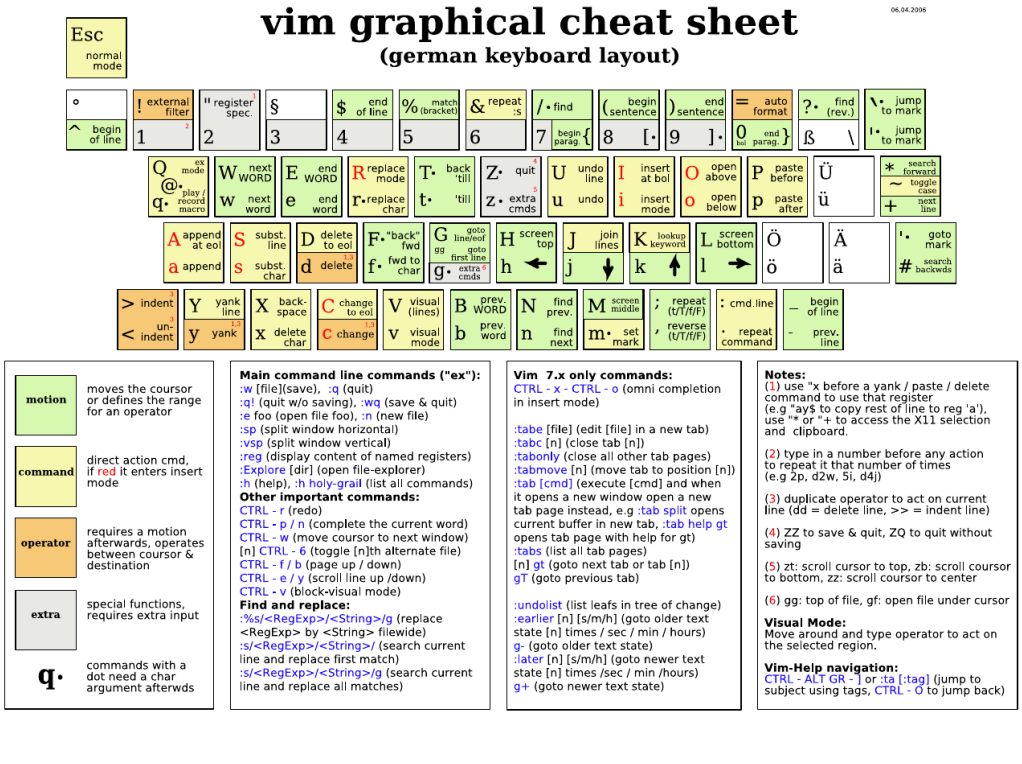


Figure 7: Vim Cheat Sheet

Visualization Quickstart Guide

How to: scientific plots

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
```

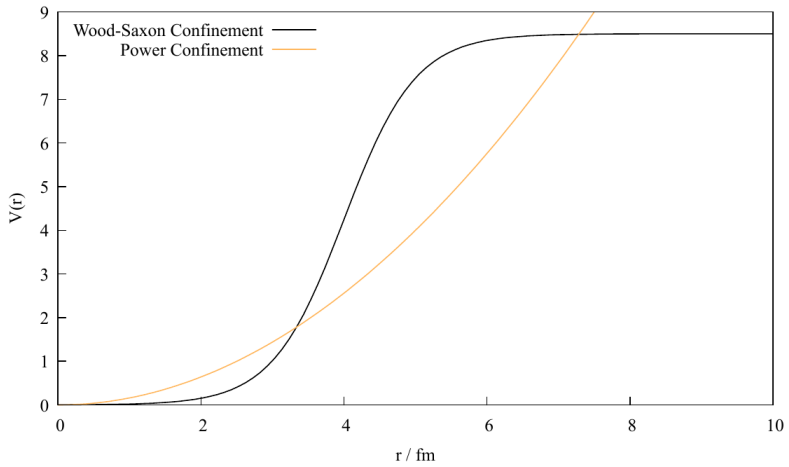


Figure 8: gnuplot example plot

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

How to: LaTeX

- 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

How to learn programming

- 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

Python Modules for Data Visualization

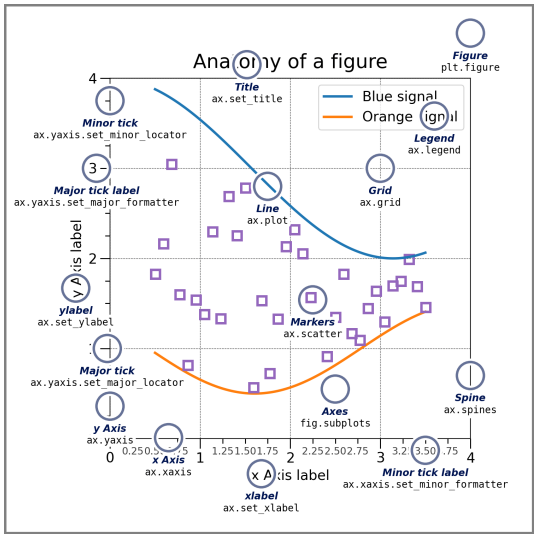
Matplotlib

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:

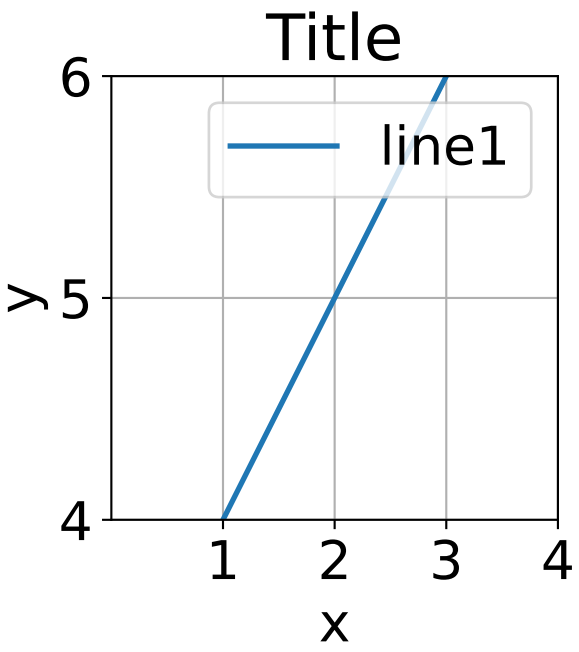


Global settings

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

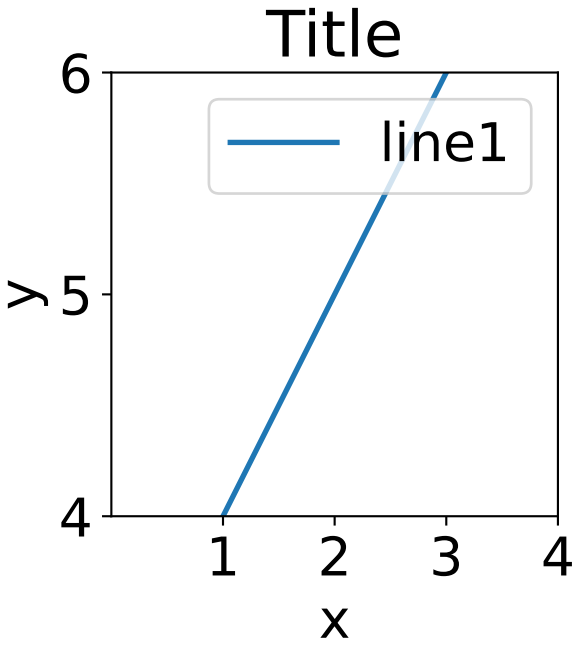
Figure

```
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()
```



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()
```

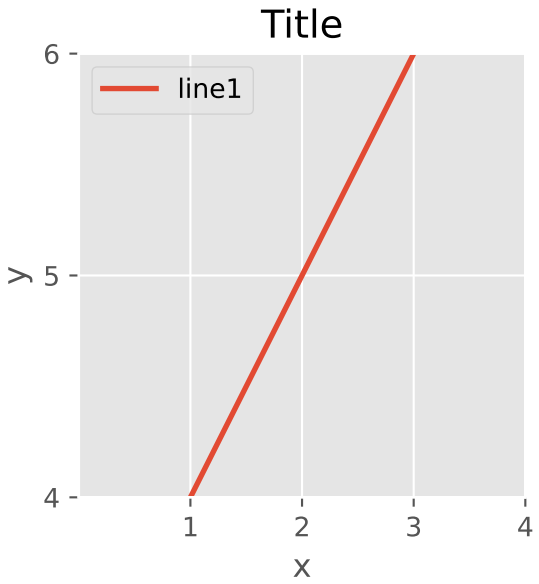


Use different styles

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()
```



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

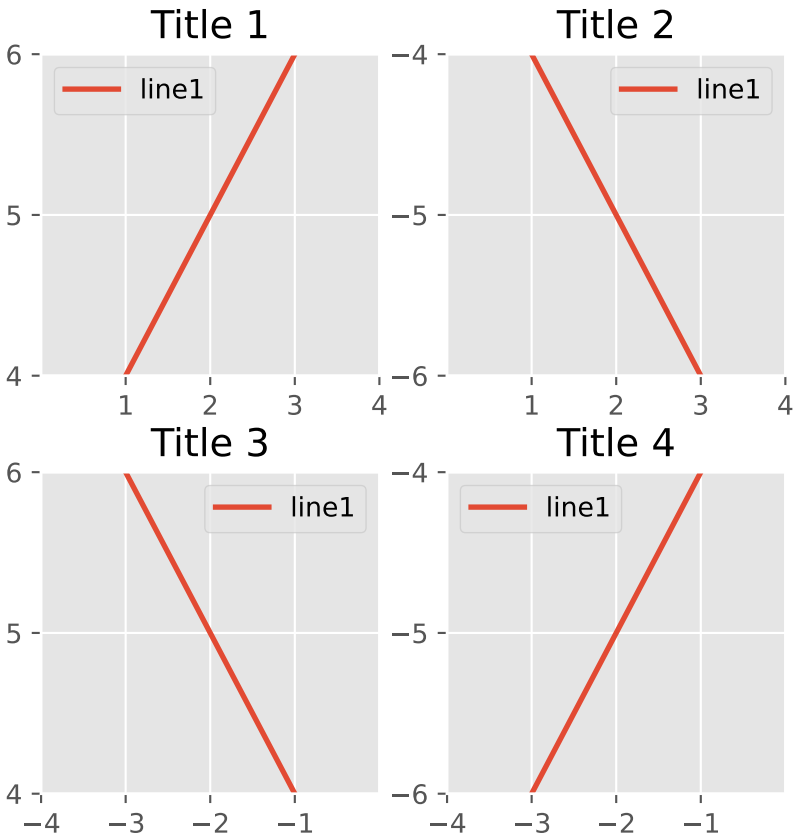
Subplots

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])
```

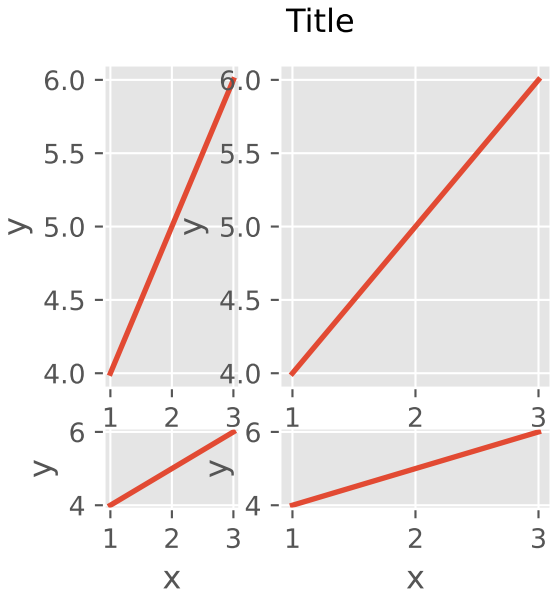
```
axes[0, 0].legend()
axes[0, 1].set_title('Title 2')
axes[0, 1].plot([1, 2, 3], [-4, -5, -6], label='line1')
axes[0, 1].set_xlim(0, 4)
axes[0, 1].set_ylim(-6,-4)
axes[0, 1].set_xticks([1, 2, 3, 4])
axes[0, 1].set_yticks([ -4, -5, -6])
axes[0, 1].legend()
axes[1, 0].set_title('Title 3')
axes[1, 0].plot([-1, -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()
```



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()
```

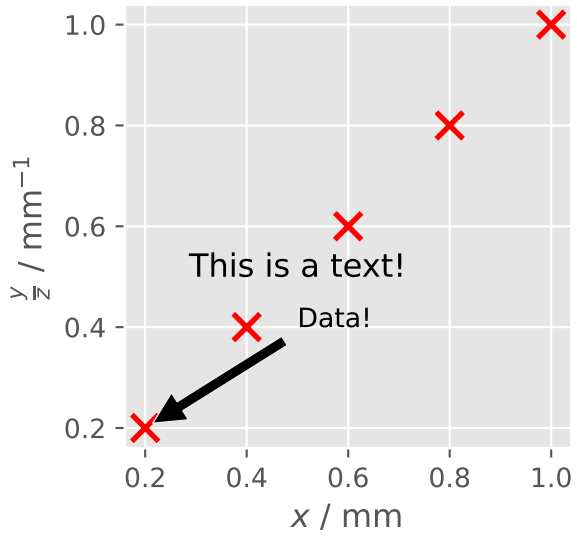


Colors and Color maps

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

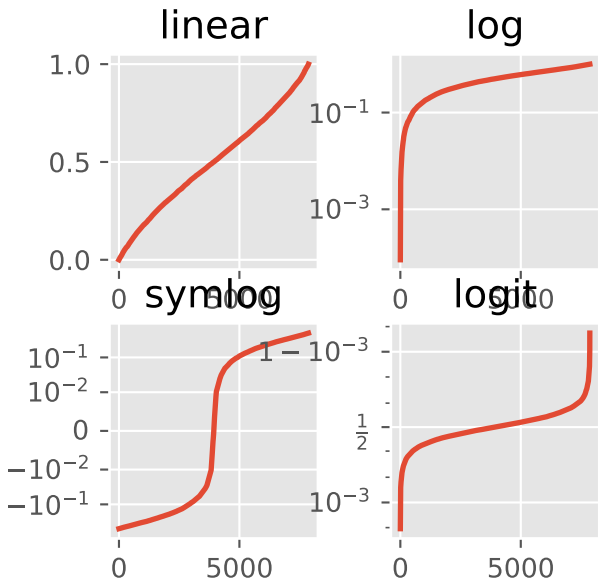
```
import numpy as np
x = [0.2,0.4,0.6,0.8,1.0]
y = [0.2,0.4,0.6,0.8,1.0]
plt.text(0.5, 0.5, 'This is a text!', fontsize=12, ha='center')
plt.annotate('Data!', xy=(0.2, 0.2), xytext=(0.5,
 ↪  0.4),arrowprops=dict(facecolor='black', shrink=0.05))
plt.scatter(x,y, color='red',marker='x',s=100)
plt.xlabel(r'$x$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



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)
```

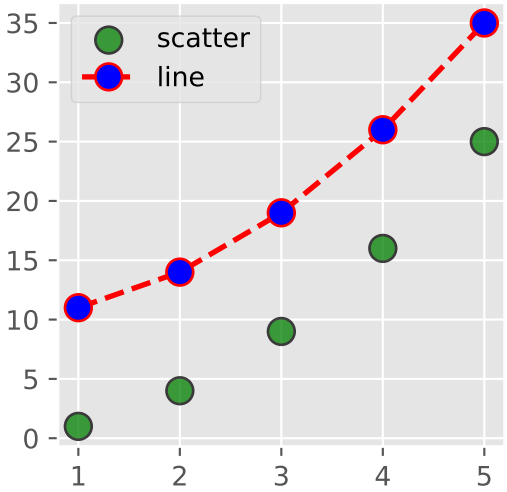
```
plt.show()
```



Scatter / Line plot

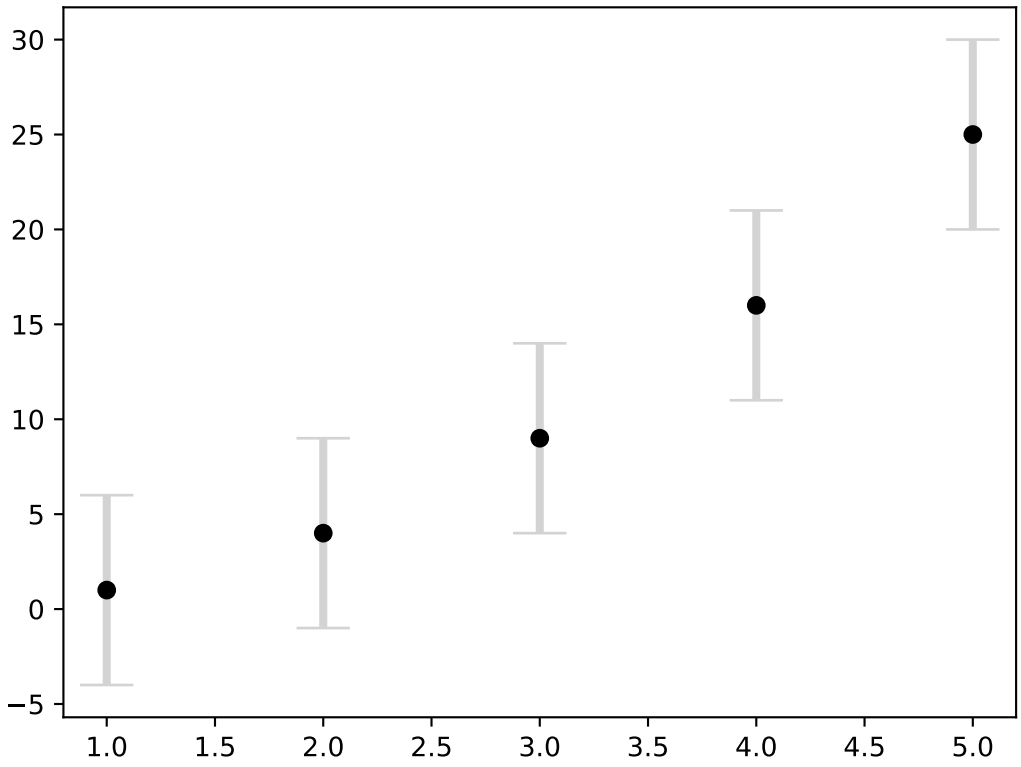
```
import numpy as np
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 4, 9, 16, 25])
y2 = y + 10
```

```
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()
```



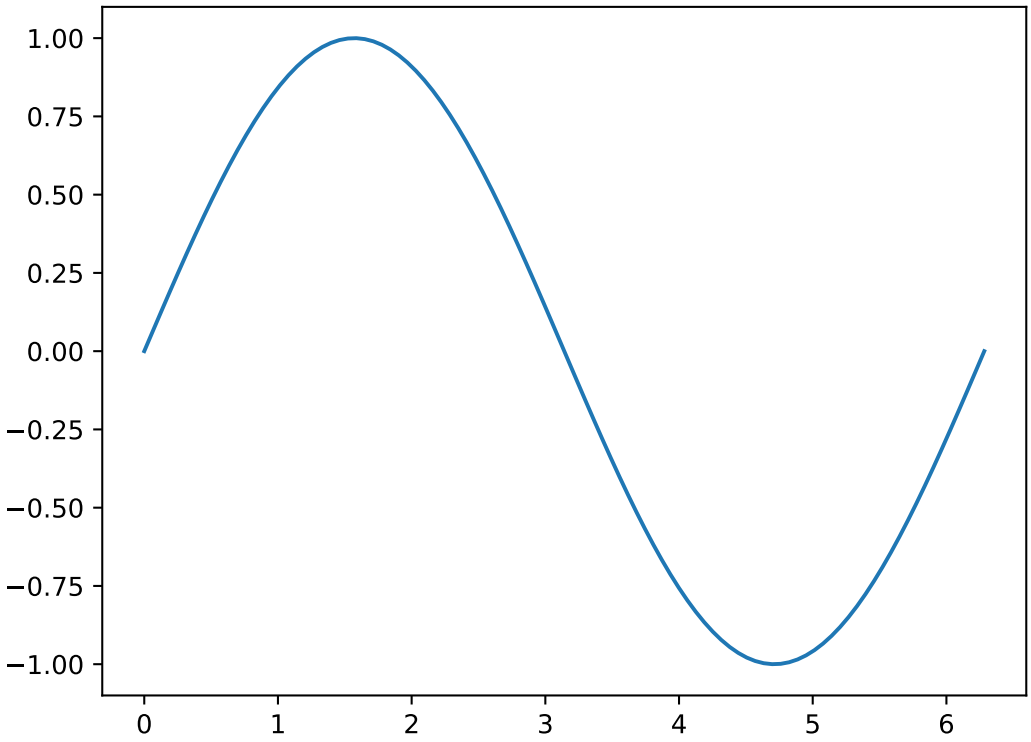
Error bars

```
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()
```



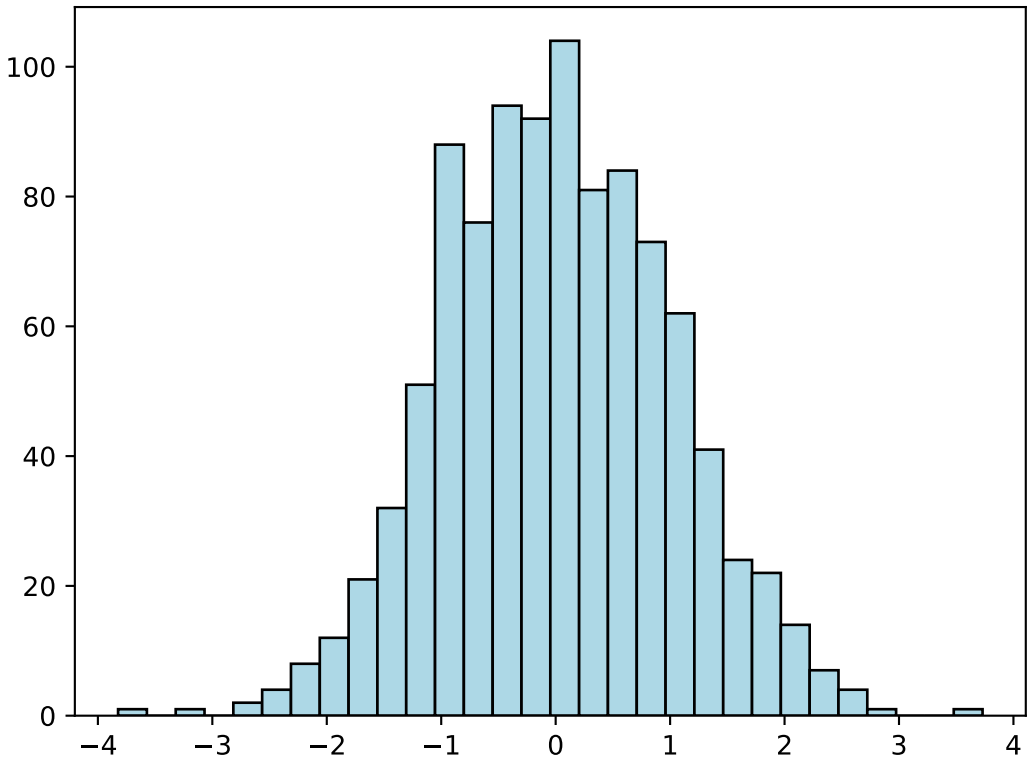
Save figure

```
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()
```

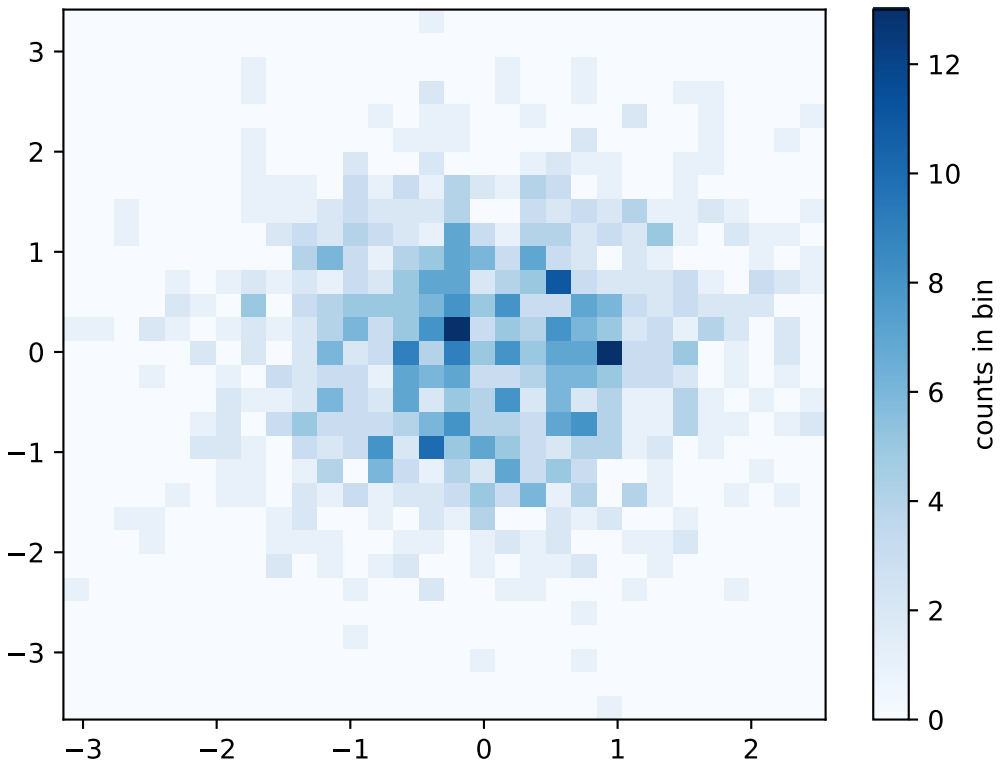


Histogram

```
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()
```



```
# 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()
```

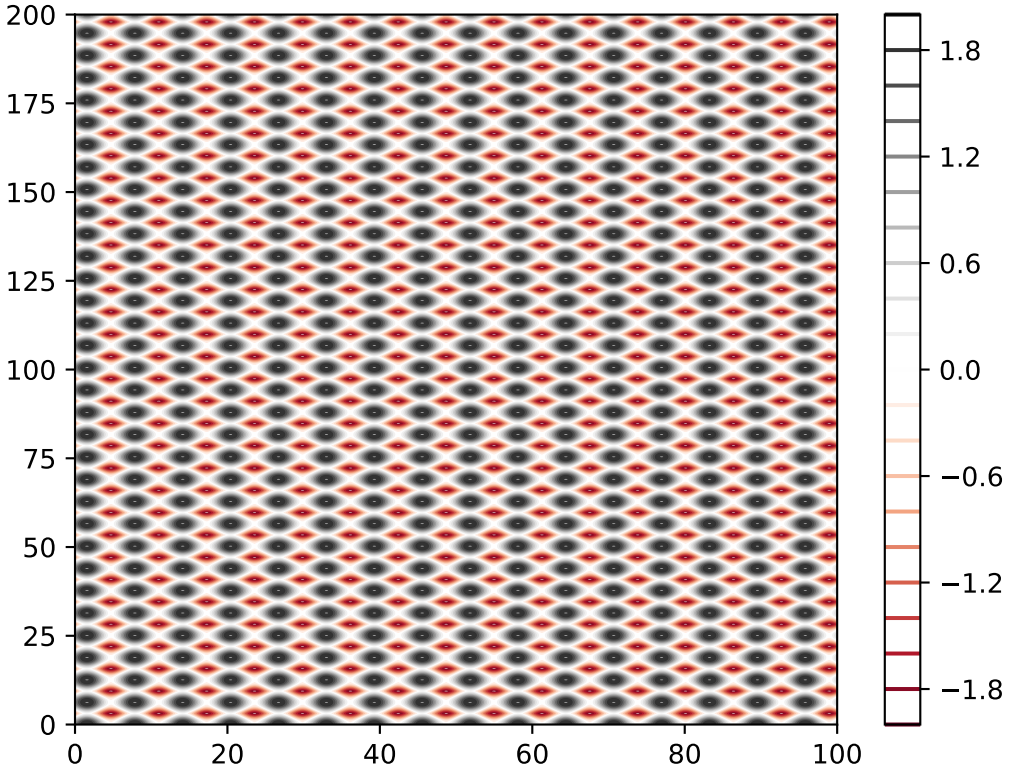


Density plot

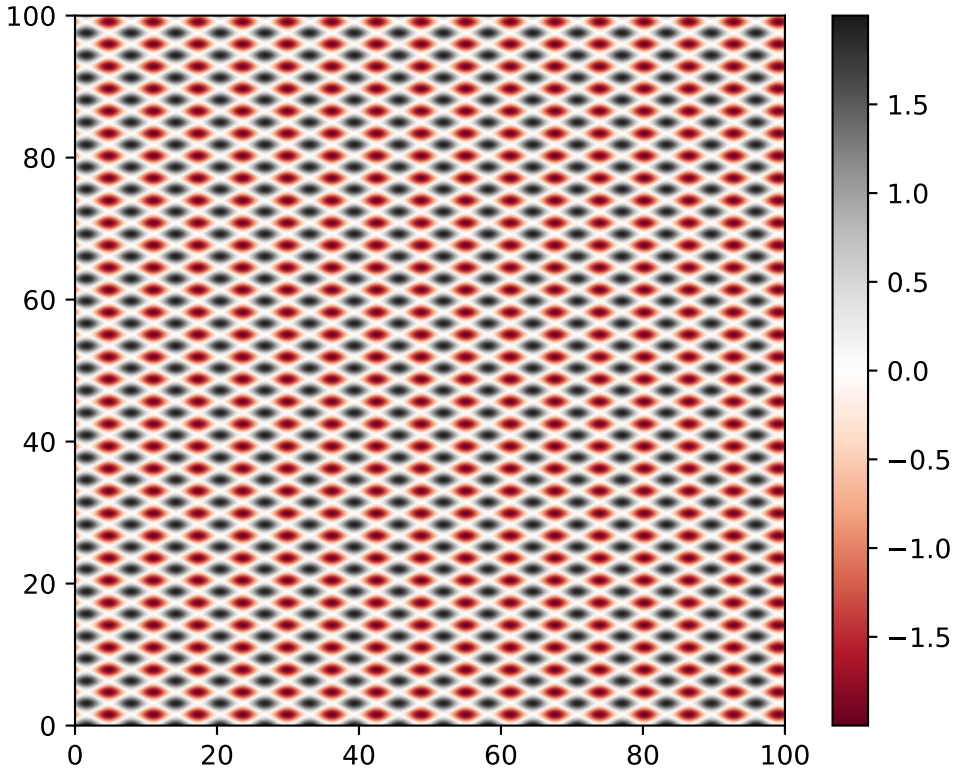
```
x = np.linspace(0, 100, 1000)
y = np.linspace(0, 100, 1000)*2
# Create a 2D array of shape (1000,1000)
X, Y = np.meshgrid(x, y)
# Z is a 2D array of shape (1000,1000)
Z = np.sin(X) + np.cos(Y)
print("Shape of X:", X.shape)
print("Shape of Y:", Y.shape)
print("Shape of Z:", Z.shape)
```

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

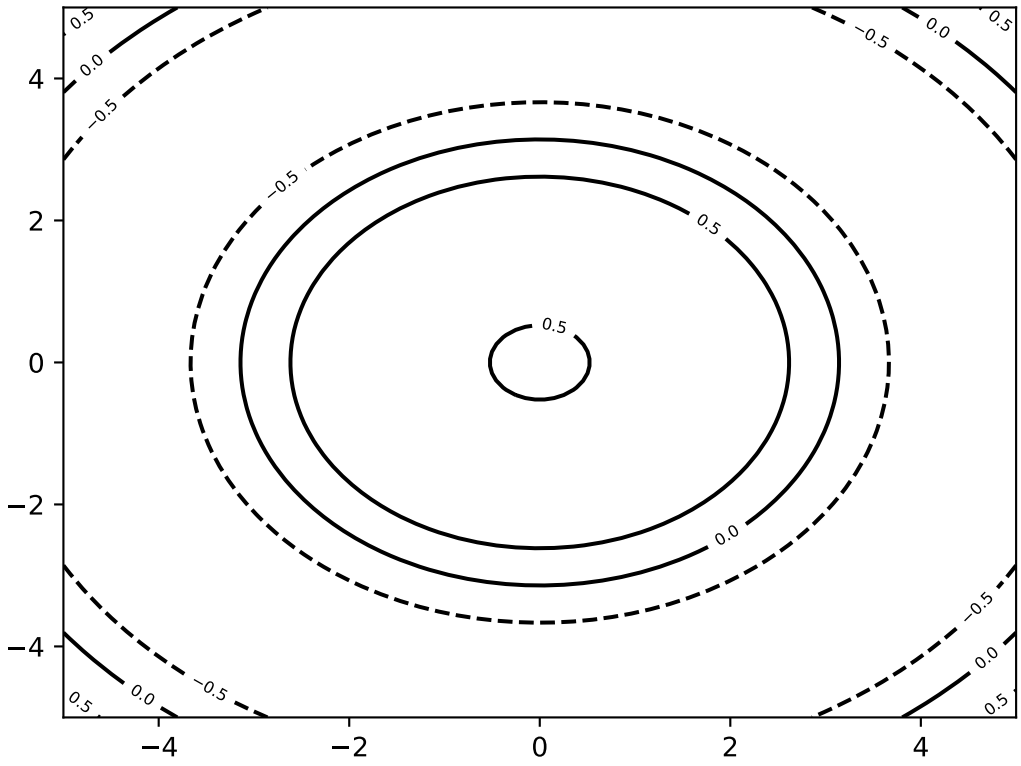
```
fig, ax = plt.subplots()
# Create a contour plot
plt.contour(X, Y, Z, 20, cmap='RdGy')
plt.colorbar()
plt.show()
```




```
# 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()
```



```
# 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)
```



Seaborn

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

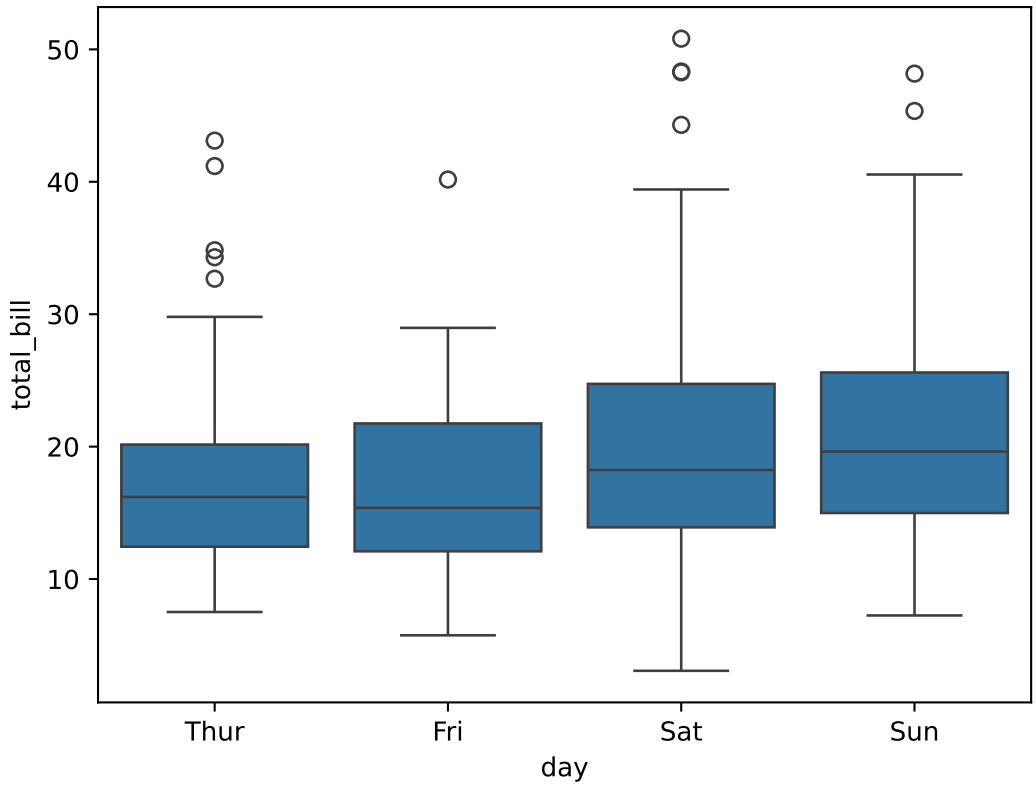
It works best with `Pandas` DataFrames. You can easly create complex plots with only a few lines of code by grouping and aggregating data.

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

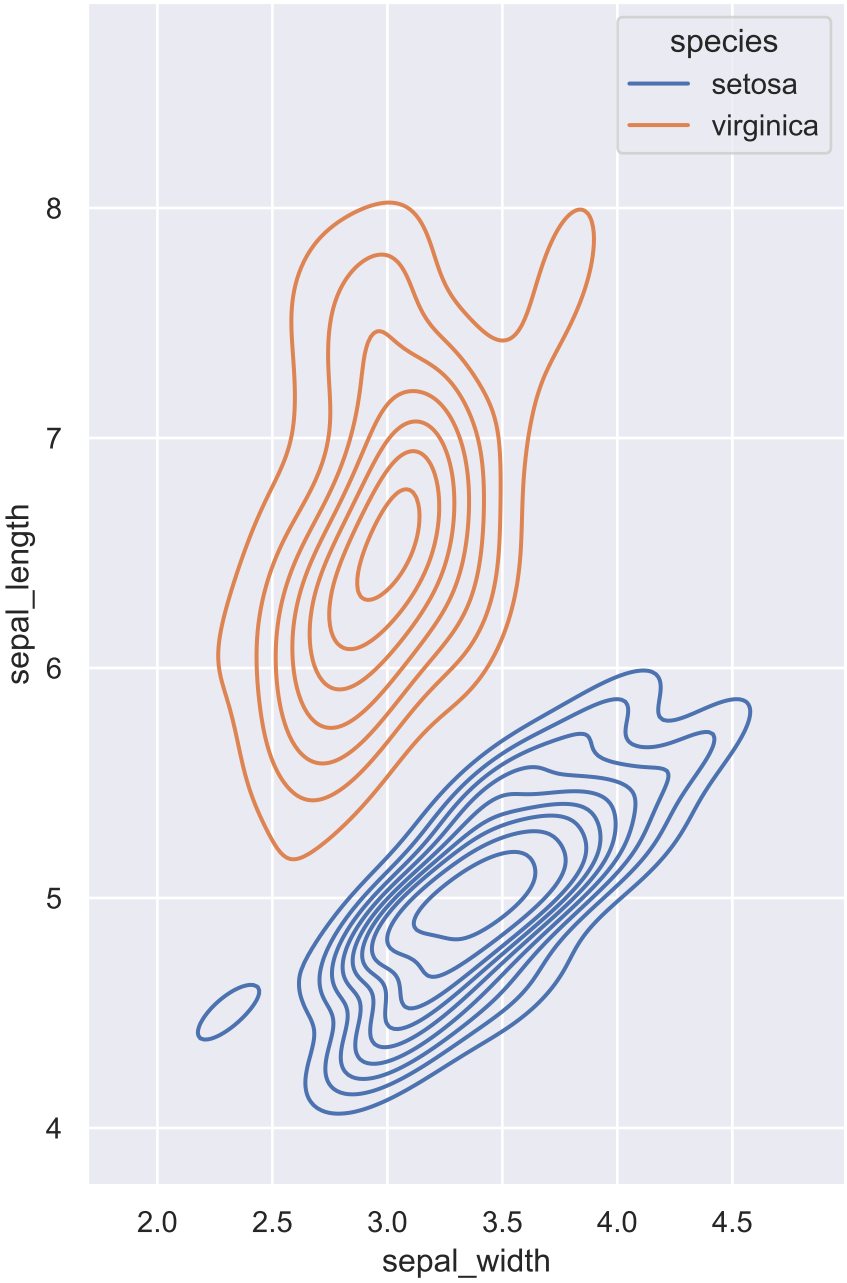
```
import seaborn as sns
```

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)
```




```
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,
)
```

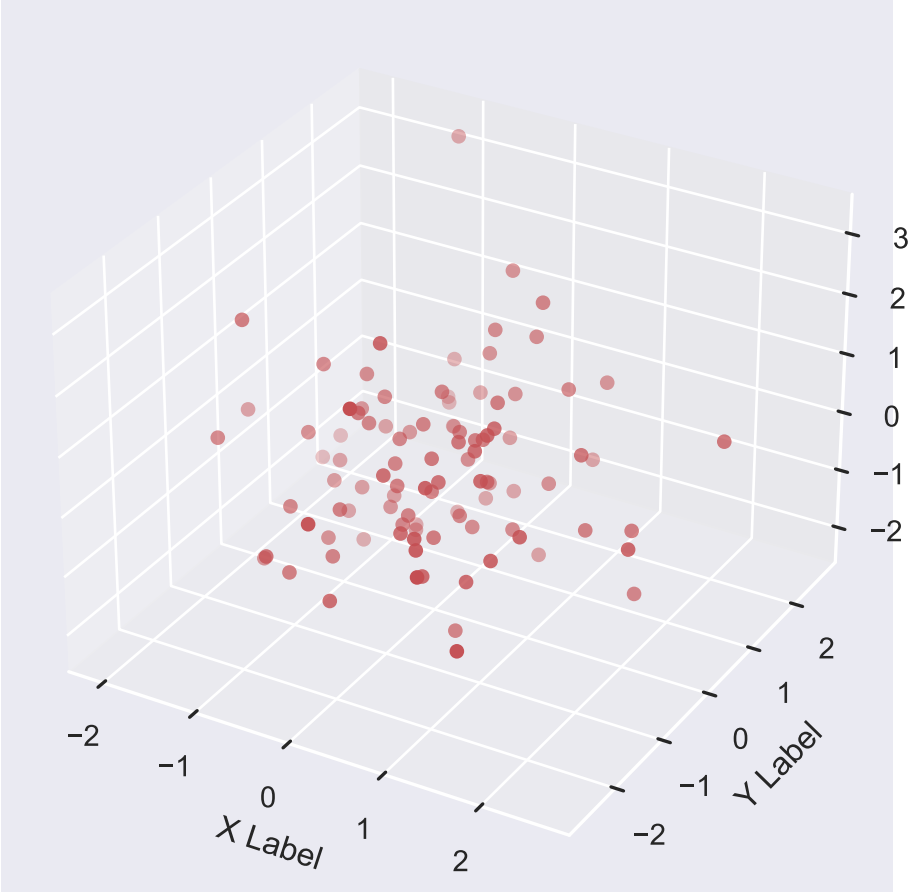


3D plots

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()
```



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

Other plotting libraries

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