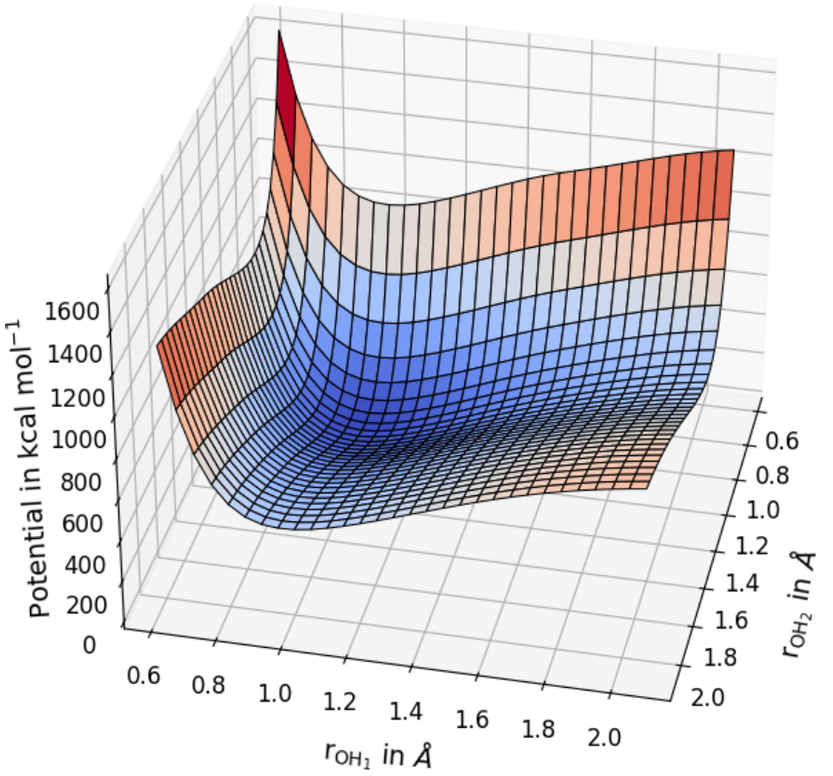
# A Data Science and Visualization Primer

Specials – 3D Visualisation and Interactive Plots

assoc. Prof. Dr. Thomas Hofer
Department of Theoretical Chemistry
Email: T.Hofer@uibk.ac.at

Stefanie Kröll, MSc
Department of Theoretical Chemistry
Email: Stefanie.Kroell@uibk.ac.at

# Welcome to the Fourth Exercise Block

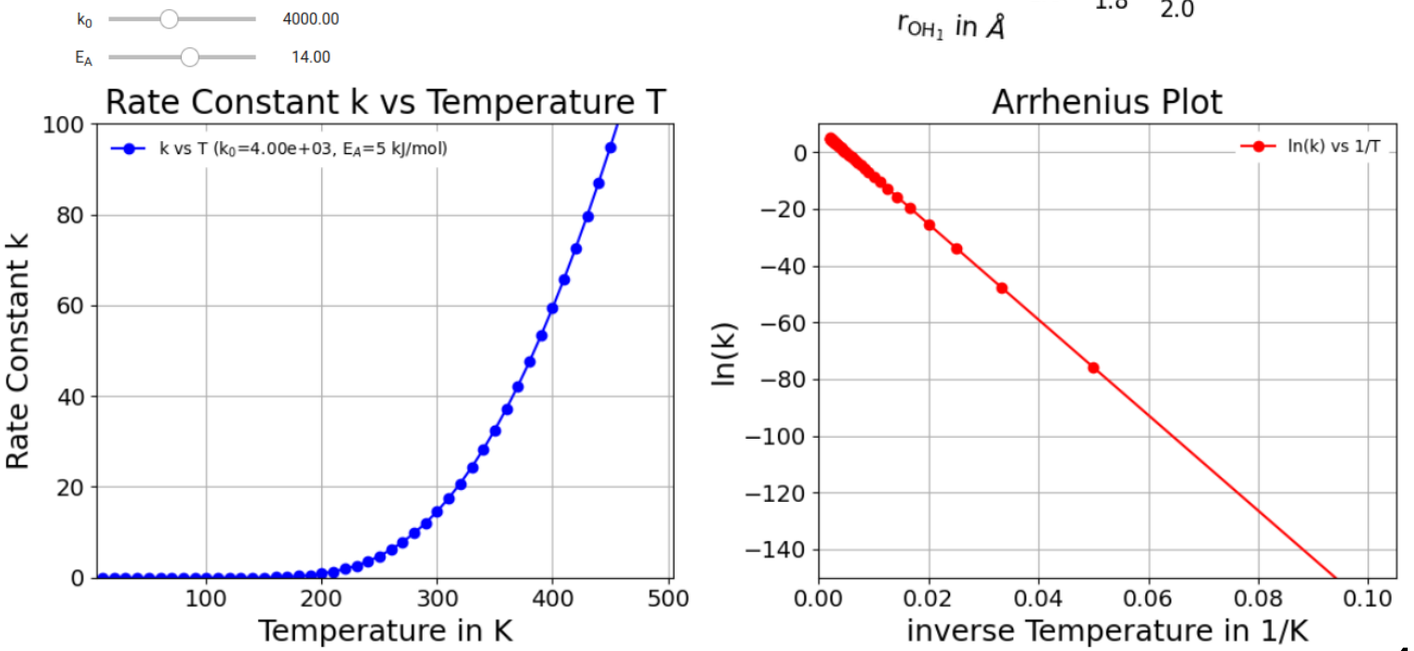



In this slide set, we will outline some special topics, that can be useful in some cases.

The first aspect we will discuss is the generation of 3D data, which is typically challenging in different software packages. Matplotlib is just one of the tools capable of generating compelling visualizations of 3D data.

Another useful tool are interactive diagrams, in which the influence of parameter changes can be visualized on the fly.

This can be useful in many situation, *e.g.* in the classroom, when presenting complex data in meetings and even at conference talks.



# 3D Visualization (*intermediate difficulty*)

In this exercise we read the data *via* the package Pandas (short for **pan**el **da**ta). This is a package specialized for data management.

In large data set it is not uncommen that some entries are missing, which would then show as `na` (not available) or `nan` (not a number).

The pandas module `data.dropna()` is useful to eliminate these missing data entries.

Using `data.to_numpy()` provides a convenient way to transfer the data to Numpy arrays.

In order to generate the 3D grid for visualization, a 2D base grid has to be available:

- First we have to extract the unique $x$- and $y$-positions from the 3D data using `np.unique()`.
- Using these the 2D base grid is created *via* `np.meshgrid()`.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.interpolate import griddata

data = pd.read_csv('grid_data.dat',
                   header=None,
                   skipinitialspace=True,
                   sep=' ')

data = data.dropna()

data = data.to_numpy()

x=data[:,0]
y=data[:,1]
z=data[:,2]

x_unique = np.unique(x)
y_unique = np.unique(y)

X, Y = np.meshgrid(x_unique, y_unique)

Z = griddata((x, y), z, (X, Y), method='cubic')

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

ax.view_init(elev = 32.5, azim = 12.5)

ax.plot_surface(X, Y, Z, cmap='coolwarm',
                linewidth = 0.75, edgecolor='k')
```

# 3D Visualization (*intermediate difficulty*)

Once the 2D base grid has been constructed, the associated z-values are generated using SciPy's `griddata()` command. In case data points are missing, they are added *via* cubic interpolation.

To plot the newly generated data a 3D plot has to be requested using `plt.axes(projection='3d')` when defining the figure.

The orientation of the camera relative to the axes is controlled *via*:

- the elevation angle relative to the $xy$-plane (top view: `elev = 0` side view: `elev = 90`)
- the azimuth angle controlling the $z$-rotation ($x$-aligned: `azim = 0` $y$-aligned: `azim = 90`)

The data is finally plotted via `ax.plot_surface()` with `cmap` selecting the colormap and `edgecolor` the color of the grid lines.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.interpolate import griddata

data = pd.read_csv('grid_data.dat',
                   header=None,
                   skipinitialspace=True,
                   sep=' ')

data = data.dropna()

data = data.to_numpy()

x=data[:,0]
y=data[:,1]
z=data[:,2]

x_unique = np.unique(x)
y_unique = np.unique(y)

X, Y = np.meshgrid(x_unique, y_unique)

Z = griddata((x, y), z, (X, Y), method='cubic')

fig, ax = plt.figure(figsize=(8, 8)),
                     plt.axes(projection='3d')

ax.view_init(elev = 32.5, azim = 12.5)

ax.plot_surface(X, Y, Z, cmap='coolwarm',
                  linewidth = 0.75, edgecolor='k')
```

# 3D Visualization (*interm. difficulty*)

Adjusting the axis labels requires some fine tuning.

First, the automatic rotation of the labels is turned off using `ax.xaxis.set_rotate_label(False)`.

The orientation of the labels is then manualyl adjusted using:

- `labelpad`: distance between axis and label
- `rotation`: angle of the label

Fun Fact: `axis='both'` in `ax.tick_params()` controls the ticks of all three axis in the plot.

ProTipp: Using the LaTeX non-breaking space symbol '~' enables us to shift the z-label a little bit up along the axis.

To enter LaTeX math mode raw strings `r'...'` is used together with dollar symbols, *i.e.* `$~~~~$`

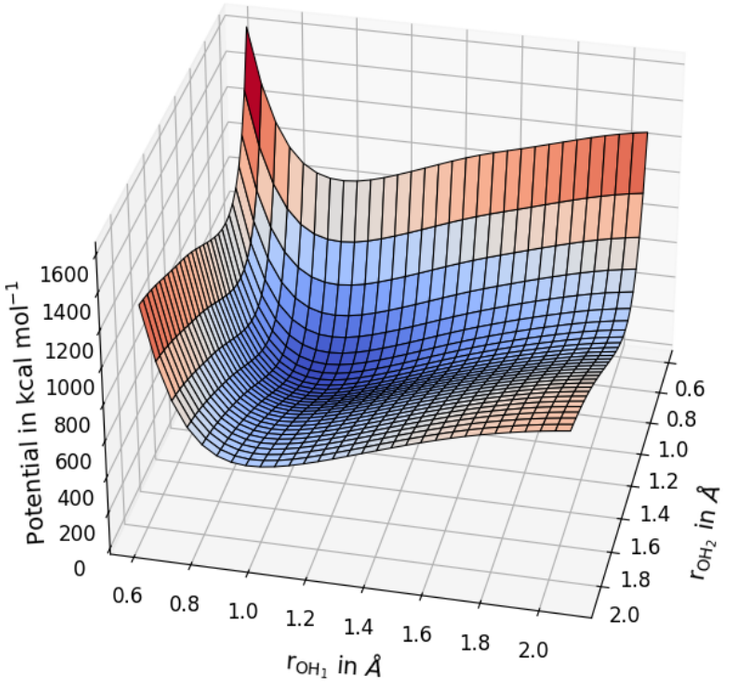
```
ax.xaxis.set_rotate_label(False)
ax.yaxis.set_rotate_label(False)
ax.taxis.set_rotate_label(False)

ax.set_xlabel('r$_\mathrm{{OH_2}}$ in $\AA$',
               fontsize = 14, labelpad = 15,
               rotation = 80)

ax.set_ylabel('r$_\mathrm{{OH_1}}$ in $\AA$',
              fontsize = 14, labelpad = 10,
              rotation = -5)

ax.set_zlabel(r'$~~~$Potential in kcal mol$^{{-1}}$',
              fontsize = 14, labelpad = 6,
              rotation = 94)

ax.tick_params(axis='both', labelsize=12)
```



# Interactive Plots (*interm. difficulty*)

Sometimes it may prove useful to generate plots that have an interactive element, to visualize concepts to the audience.

This can be achieved with the `interact()` module in the package `ipywidgets`.

To do this, the target function and the associate have to be defined in a custom function using the `def` command.

Using `np.linspace()` an interval with a given number of points is created for the x-axis.

The magic then happens inside the function `interact()`. With the help of `FloatSlider()` interactive sliders for all parameters can be defined.

That way any function can be transformed into a compelling, interactive presentation.

```
from ipywidgets import interact, FloatSlider

def van_deemter(A=1.0, B=1.0, C=0.1):
    u = np.linspace(0.01, 10, 500)
    H = A + (B / u) + (C * u)
    H_min = np.min(H)

    plt.axhline(H_min, color='r', linestyle='--',
                linewidth=0.8)

    plt.plot(u, H, color='k',
             label=f'Van Deemter (A={A}, B={B}, C={C})')

    [...]

interact(van_deemter,
         A = FloatSlider(min=0.1, max=10.0, step=0.5,
                         value=1.0, description='A'),
         B = FloatSlider(min=0.1,  max=5.0,  step=0.2,
                         value=4.0, description='B'),
         C = FloatSlider(min=0.01, max=5.0,  step=0.1,
                         value=2.0, description='C'))
```