Computer lab tools - installation and use

Anaconda · Python · Jupyter Notebook · ASE · OVITO

Lab teachers: Erik Bowall, Bojana Kocmaruk

September, 2022

Contents

1 Introduction

The material covered in this instructions tutorial will go over the installation process and the basic concepts related to the tools we will be using in the lab sessions throughout this course. Besides being useful for the course itself, the tools covered can have a wide range of other applications (not just in the realm of computational chemistry).

Almost all software packages that we will be using are not natively supported in Windows. Most codes for data handling and computation are designed to run under Linux since nowadays virtually all High Performance Centers (HPC) operate under the Linux environment. Regardless, the manual has been written with the main assumption that you are using a Windows machine, and we will be using a walk-around so that everything can be installed and run smoothly under Windows as well. In order to avoid issues, it is highly advised that you download the materials for this lab from Studium and keep them all in the same folder, preferably on your Desktop (but anywhere, as long as it is under `C:\Users\(username)` is fine).

For the lab sessions, we have booked a computer room at Ångström Laboratory and you are allowed to use the computers from there. If you decide to do that, you should

know that the Anaconda distribution has already been installed on these computers, and you can skip that step of this tutorial (do not skip the part with the installation of the required packages, though!)

In case you are not a Windows user, most of the installation part of the manual is applicable in your case as well, except that you will be running the commands in your terminal window and using the appropriate Linux paths for your files. In case you have any issues and would like support in this process, feel free to contact your lab teacher.

2 The Anaconda distribution

The most important step in setting up your environment is installing the Anaconda distribution. It is a widely popular open-source platform which includes a large number of Python libraries ("packages") with tools useful for data manipulation, analysis and visualization, machine learning, etc. Through Anaconda and the Anaconda prompt, you will also be able to install packages that are not a part of Anaconda. In order to install Anaconda, go to https://www.anaconda.com/products/individual, click the "Download" button. Then download the Python 3.8 version of Anaconda that is suitable for your computer architecture and install it.

Typing "anaconda" into the start menu gives you a preview of the available features. The most important are the Anaconda Prompt, Anaconda Navigator and Jupyter notebook.

- The Anaconda Navigator gives you an overview of all the packages that are installed (available under "Environments"), and their versions. You can also install packages with the Navigator, but we will not use it for this purpose.
- The Anaconda Prompt is where you willy type in commands for installing new packages and configuring your environment.
- Jupyter Notebook is an app that we will predominantly use to run Python code.

3 Introduction to Python

Python is currently one of the most popular programming languages, and also quite big in the science community as well. Its power lies in its flexibility and easy to read syntax. In addition, there are more than 200.000 open-source Python packages (for basically everything you can think of) and a large number of programs, that are written in Python. The Python community is also very active and supportive towards newbies, which makes it a common choice of a language for getting started with programming.

There are many packages inside of Python that natively accompany each installation, but it is also easy to install many others alongside. If you are planning to use tools that are parts of these packages (both the ones that come with your Python installation and the ones that were additionally installed), they need to be imported first before they are used in the code. This is usually done at the beginning of each script (with "import statements"), because it is the easiest way to keep track of the loaded tools.

Python handles data (objects)differently depending on the *type* of that data (whether it is a number, text(so-called "strings"), a list, or something entirely different.

Python is a dynamically typed language. Loosely speaking this means that Python implicitly takes care of object typisation for you and you don't need to explicitly state the type of each object you create. Python offers four primitive data types (Strings, Booleans, Integers and Floats) and a series of in-built non-primitive datatypes, like e.g. Tuples, Lists and Sets, which allow you to bundle data together. It is even possible to create own data types (via classes; for details have a look at the documentation https://docs.python.org/3/tutorial/classes.html)

3.1 Running Python code

By installing Anaconda, you basically installed Python on your system. There are several options for running the Python interpreter.

3.1.1 The Python prompt

This can be done by opening the Anaconda Prompt and simply typing `python` and hitting Enter. This opens a so-called Python prompt. The prompt is used to perform quick and simple tasks that are usually a few lines long, maximum. It is a useful tool to test whether a package you just installed is available. For example, we can test whether a package called "math" is installed on our system by typing

```
import math
```

If this command passed without an error occurring, it means that the package is there and ready for use. The "math" package is actually part of the Python Standard Library. Therefore, the command above will always run successfully if you installed the Python interpreter correctly.

3.1.2 Running Python scripts through Anaconda Prompts

The functionality of Python can be fully taken advantage of by writing more elaborate code. This is usually done by writing Python scripts. These are files with the extension

"py". Running Python scripts can be done directly through the Anaconda Prompt, too. For example, if we want to run a Python script we must use the command "python" and give it the file location and name. The following line illustrates a command that would run a file "test.py", that is inside the `C:\Users\bojana\Desktop` folder:

```
python C:\Users\bojana\Desktop\test.py
```

3.1.3 Jupyter notebook

In the labs however, we will be running Python code in the Jupyter Notebook environment. This functionality is a part of the Jupyterlab package that is installed automatically with Anaconda as a Jupyter Notebook "app". You can start up Jupyter Notebook by accessing that app through the Startup menu. This opens a tab in your web browser where you will get a list of all the folders in your home folder. In both labs you will be using the notebooks that will be written for you and available for download from the course page on Studium. In order to access the notebooks that you downloaded for this lab session, simply navigate to the folder where you had previously saved them. You can also create new notebooks by clicking on the "New" tab and choose "Python 3".

Here is some information for getting started with using Jupyter notebooks:

- To run the cells with code in them, press ctrl+Enter.
- The first cell at the top of the page imports the necessary packages and tools that will be used further in the exercises. Without these dependencies, the code will not work so make sure you run this cell first and don't edit it.
- The cells should generally be run in the order in which they are arranged to prevent Python objects from overwriting one another.
- Sometimes it may take a while for the code inside the cell to finish running. While it is running, a "*" sign will be shown to the cell's upper left side. While a cell is running, do not run another one - wait for it to finish first. Otherwise your Jupyter kernel may get stuck.
- In case the kernel does get stuck, you can restart the whole notebook environment. Go to Kernel-Restart & Clear output to do this.

The Jupyter notebook can contain text for additional explanation of what has been calculated or produced by the code that had been run. One can insert figures, equations in addition to text. This makes it possible to produce lab reports inside the Notebook,

answering questions inserting text alongside the code you had used to produce results. This is done in Markdown cells instead of regular cells that run Python. One can create a Markdown cell by adding a new cell and then selecting "Cell" at the top of the page, then "Cell Type" and "Markdown". The shortcut for this action is just selecting a cell and pressing "M".

3.2 Python packages

Most of the packages that we will be using in the labs for this course are already installed on your systems alongside Anaconda (NumPy, Pandas, Matplotlib, Seaborn). The rest of the packages used for this and the following labs we will install manually using the file `requirements.txt` (which is just an appropriately formatted list of the packages we will use. To be on the safe side, this requirements.txt file contains a few lines that instruct Anaconda to check whether these preinstalled packages are the correct version, and reinstall them if they are not. The installation of the packages and configuring your environment is done by running Anaconda Prompt and typing the commands:

```
conda config --add channels conda-forge
```

and

```
conda create -n recreated\env --file C:\Users\bojko530\Desktop\Lab1\
PythonIntroduction\Lab\Python\Introduction\requirements.txt
```

For Linux users: You use the same method one above, just using the file "requirements.txt" location to the requirements file where you stored it. If you used /home/bojana/Desktop as the place where you downloaded the lab materials, the command would then be:

```
pip install --user -r /home/bojana/Desktop/Lab1\_materials/requirements.txt}
```

Now that you have stuff set up on your computer, it is time to run some code. We will do this through Jupyter Notebook. In your Jupyter Notebook tab in your browser navigate to the folder where you placed the data downloaded from Studium, and open the "getting_started.ipynb" file. In this notebook we cover a few extremely popular Python packages for mathematics, calculations, analysing data and plotting. The Anaconda distribution already contains them, so there is no need to install anything at this point. The following subsections will contain some information about these packages.

In case you are using one of the student computers from the computer lab, make sure that the location where you downloaded the files is exactly c:\Users\your _username\Desktop.

It is not enough to just download them onto the visible workspace (because that "Desktop" you see there actually has a location on disk x:.

3.2.1 Numerics with Numpy

NumPy's webpage: https://numpy.org/

NumPy is a package that most scientific Python tools rely on. It handles data by representing it as an N-dimensional array object and that way, instead of running a function on each individual element of an array, you run the function on the array itself, on each of the elements in parallel. On top of that NumPy takes advantage of low level languages like C, which together with the former often leads to a major improvement over Python lists in terms of computational efficiency. Besides being useful for scientific purposes, NumPy can also be used as an efficient multi-dimensional container of data. In addition to the common math capabilities, it also includes some basic linear algebra functions, Fourier transforms and it has high quality random number generators.

3.2.2 Data Analysis with Pandas

Pandas webpage: https://pandas.pydata.org/

The most notable feature of Pandas is the "DataFrame" object designed for data manipulation and operations for handling numerical tables. Using Pandas you can create databases by merging datasets, which you can then reshape by adding and subtracting extra columns and rows, and reorganize and filter your data in an efficient manner. It is also able to provide some statistics on the data contained in the DataFrame as well as handle missing data in sophisticated ways. You can create the DataFrame object by loading data from an external file, and you can also write the data contained in the DataFrame object into a file. The file formats that are compatible with Pandas include Excel files, SQL or csv.

Since Pandas is based on NumPy it is compatible with NumPy's array objects. You can use NumPy arrays to create Pandas DataFrames, and vice-versa (create an array out of the DataFrame or its particular rows and columns)

3.2.3 Data Visualization with Matplotlib

Matplotlib's webpage: https://matplotlib.org/3.1.1/index.html#

The most widely used plotting package for Python is Matplotlib. It shares a lot of features with Matlab, which served as inspiration for its development. It generates

publication-quality plots, and offers many options for customizing your plot, adding objects to it, etc. Besides the standard "scatter" and "line" plots, you can plot pie-charts, histograms, as well. You can include grids, change the tick settings, add arrows, text boxes, add mathematical symbols into titles and labels, and many many more. The learning curve for mastering Matplotlib is quite high though, so we will go through some of its most basic features in this course.

Matplotlib is also compatible with both Pandas and NumPy, meaning that you can use DataFrames and array objects and plot the data contained in them directly.

3.2.4 Beyond Matplotlib - Seaborn

Seaborn's webpage: https://seaborn.pydata.org/

Seaborn is a widely used successor of Matplotlib designed for statistical data visualization. It is well-integrated with all the previously covered tools, especially Pandas. Since with Pandas you are able to easily load in a data set from an external file (Excel for example) into a Jupyter notebook, it is therefore rather easy to make use of Seaborn's tools for generating all sorts of nice plots for your data. The major advantage Seaborn has over Matplotlib is that it uses significantly less code to generate the same plots, which is another reason for its integration into this manual. And as most other data analysis tools, it gets installed along with the Anaconda distributions so it is already readily available for use on your computers.

4 Handling and visualizing atomic structures

The tasks in Lab 2 will be on crystal structures, the different atomic arrangements on different surfaces, adsorption sites on surfaces and nanoparticle shapes. In order to perform these exercises, we will use a tool for creating, manipulating and displaying these different concepts.

4.1 Atomic Simulation Environment (ASE)

ASE's webpage: https://wiki.fysik.dtu.dk/ase/

This is a highly useful library for computational chemistry and material modeling. It is a rather simple to use Python package, well documented and full of useful tools for handling atomic, molecular, surfaces objects, etc. You can also import structure files that you downloaded off the internet into ASE, and view and edit those, too. The convenient thing about ASE for us is that its functions and visualization features can be

used inside Jupyter notebooks, which makes ASE the obvious choice for us to use in the labs. The ASE's visualization library that we are going to be using in the lab offers a way to visualize the structures that were created during the exercises. It also provides some functionality with manipulating these structures, adding new atoms, repeating the structures in x, y an z directions, constraining certain atoms, etc. You can save the images of your structures in the ASE gui window that pops up when you run the "view" command. This is done by clicking on Tools-Render scene.

To test ASE we will open the "ase_ examles.ipynb" notebook and go through the code.

4.2 OVITO

OVITO's webpage: https://www.ovito.org/about/features/
Download OVITO: https://www.ovito.org/windows-downloads/

We have seen that ASE can be used to visualize molecules and materials in the previous section. However, the visualization is pretty basic, and these structures don't exactly look too pretty. So in this section we will explore another nice visualization tool, OVITO, which is also a Python-based open source tool, with enhanced visualization capabilities. Besides being a good alternative for ASE, it can also provide you with the opportunity to visualize the different concepts that ASE can not. We will be using it for some of its features in Lab 2 as well, so you should install it beforehand. The actual OVITO tools we will use will be explained in the Lab 2 manual along with the actual tasks.

Since this is a Windows program, you can download and install it as you would any other.