Scientific Computing

Installation of Python in Conda and first steps in Python

Peter Regner, Johannes Schmidt

Institute for Sustainable Economic Development, BOKU, Wien

2020-03-26

Latest version available on Github, this PDF is version f00728b.

1. Conda

2. Jupyter

3. Homework assignment

Conda

Download...

Please download Anaconda from
https://www.anaconda.com/distribution/

Package Manager

- A package (or library in other languages) is a collection of code that helps you accomplish tasks. In this class, when we refer to a package, we refer to Python packages. But be aware that a package, in particular under Linux, may be a complete standalone application.
- There are e.g. Python packages for machine learning, for plotting data, for working with tabular data, or for working with matrix-data. More on that later!
- A package manager is there to help you setup a stable environment for your programming system: e.g. it resolves all relevant dependencies when you install a new package.
- Conda also allows to create environments: within an environment, you are free to install a different Python version, different packages and package versions etc. This helps in having a clean seperation between projects. (But also means that you may have installed Python many times on your computer)

Package Manager of our choice

- We are going to use conda as Python package manager.
- Anaconda is a system that packages a lot of software together with conda and runs on Windows and Linux. It also has a graphical user interface. In class, we use the command line only. Anaconda provides a lot of software, also for R, you may explore it by starting the graphical user interface. (we do not dig deeper here).
- Miniconda is an alternative to use conda. It is smaller and just comes with the core libraries which are necessary to use conda.

Anaconda installation

- Run setup program
- When asked check the checkbox to set the PATH variable
- Open git bash and type

  `conda init bash`
- If this works, you successfully installed conda!

Using conda environments

- A conda environment allows you to install a seperate version of Python (or any other conda supported software) with associated packages
- List all available environments

  `conda env list`
- There should be a *base* environment available on all installations. It is automatically activated when you use conda!
- We now first use conda to *update* - conda!

  `conda update conda`
- This will download and install the newest version of conda

Creating and activating environments

- You can create a new Python environment with this command

  `conda create -n <environment-name> python=3.7 anaconda`
- *<environment-name>* can be chosen by you. We use *scientific-computing*:

  `conda create -n scientific-computing python=3.7 anaconda`

- Again list all available environments

  `conda env list`
- There should be an environment *scientific-computing* available now.
- How to work with it?

  `conda activate scientific-computing`
- Observe how the bash changes - *(scientific-computing)* is shown on the command line now!
- Not activating the correct environment is a very common source of problems! Check your environment!

Jupyter

What is a Juypter Notebook?

- In principle, you can write your code in Windows notepad and then run it on the command line
- A much better option would be to use an integrated development environment (IDE) such as pyCharm. (Syntax highlighting, code linting, etc.)
- A different form of coding environment are so called *Jupyter Notebooks*
- Here, the code is written on a webpage, sent to a server, executed there, and results are displayed in the notebook again.

Pros & Cons of Jupyter Notebooks

- Advantages
  - Integration of code and visualization of results: for data analysis crucial.
  - Interactive environment for data assessment: data (variables, objects, libraries) stay in memory and code can be dynamically adapted (Similar to R)
  - Code an be run on server with large computational capacity (if available)
- Disadvantages
  - Managing large junks of code (such as functions) becomes complicated.
  - Using Jupyter notebooks with version control is difficult, as they introduce their own XML-syntax, which is not really human readable.
  - Interactive session: very error-prone, as objects and variables can change their state depending e.g. on the number of times a certain cell is executed.
- Best of two worlds: use an IDE to manage your stable code (functions etc.) and use notebooks for exploration.
- For teaching purposes, notebooks are VERY useful. We are going to use them therefore - and may, at some point, introduce additional software such as an IDE.

Using Jupyter Notebooks

In git bash, type

```
jupyter notebook
```

This will start the Jupyter Notebook server within the current working directory. An URL is printed to the screen which can be copied to your favourite Browser. If you want to use conda environments in your notebooks, ensure that *ipykernel* is installed in your environment.

```
conda install ipykernel
```

For each conda environment that should be visible, do the following:

```
python -m ipykernel install --user --name <name-of-environment>
--display-name "Name shown in jupyter notebook"
```

Homework assignment

Assignment (I)

- If you haven't done so yet, fork the homework repository at https://github.com/inwe-boku/homework-scientific-computing
- You may have to fetch the latest changes to your repo
- Install conda and jupyter lab on your computer, run jupyter lab in the directory of the forked repository
- Use Python to create an encrypted file with the full name of the team member, the registration number (Matrikelnummer), and the git username. For that purpose, use the public key *public_key.pem* stored in the homework repository in *homework02-conda-python*
- The filename of the encrypted file should be *group-<name of group>* *<name of group>* is your self-chosen group name.
- Use the Python code on the next slides to accomplish this task.
- Add the encrypted file to your repository, commit the change, push it and issue a pull-request in our original repository to add your file to our original repository.

Assignment (II)

```
### code to encrypt your data with our public key

# imports - if they do not work,
# install package cryptography with conda
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

with open("public_key.pem", "rb") as key_file:
    public_key = serialization.load_pem_public_key(
                 key_file.read(),
                 backend=default_backend())

group_members = (b"""First Last;1234565;nickname
                 First Last;1234565;nickname""")
```

Assignment (III)

```
encrypted = public_key.encrypt(
                group_members,
                padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                slabel=None))

### important: change to your group name here!!!
filename = "group-<name of group>"

f = open(filename, 'wb')
f.write(encrypted)
f.close()
```