Appendices

A Code Management

Coding in a programming language is a process of trial-and-error, leading to a working control program or modelling code in many incremental steps. When working on this individually, but more so when coding in a team of code contributors, tracking progress is mandatory. Also, keeping track of the coding progress in multiple locations reduces the risk of information or code losses.

A.1 Github or Bitbucket account

For optimal collaboration, programmers communicate their last achievement to their colleagues or key users by *committing* them in a *repository* on their local PC, then *pushing* the changes to a central copy of the repository in the cloud. Before starting their work, colleagues can *pull* the last changes, so that their local copy of the repository is up-to-date. The updated local copy is then used for adding new contributions or trying out the updated features. Changes or improvements can be committed locally and pushed to the cloud repository by each member of the team.

For establishing a team and implementing the workflow above, each contributor must have an account on a Git repository cloud service.

- Github: Go to `https://github.com/` and *sign up* with the button in the upper right corner of the web page.

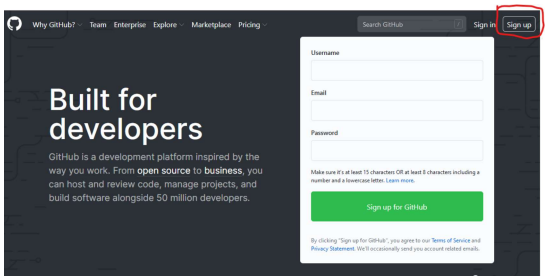


Figure A.1: GitHub sign-up

- Bitbucket: Go to `https://bitbucket.org/` and sign up with the "Get it free" button in the upper right corner of the web page.

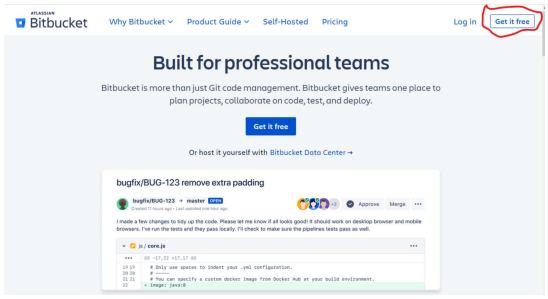


Figure A.2: Bitbucket sign-up

Do not forget to store your usernames and passwords. The use of a password *vault* is strongly recommended: `https://keepass.info/`

Note: Strictly, a GitHub or Bitbucket account is only necessary for all access to *private* repositories or for *contributing* to *public* cloud repositories. Read-only access to *public* repositories is possible without an account.

A.2 Versioning

The starting point for version control is the installation of a version control system on your PC. Historically, a number of systems have been invented. Most of them still exist. Among those, CVS, SVN, Bazaar and Mercurial (hg). In the last few years, however, the Git versioning system seems to gain a major "market share". It has been designed by the inventor of Linux, Linus Torvalds, and has a robust performance. The possibilities of Git are a bit overwhelming for the beginning user, which asks for a careful introduction.

A.2.1 Installation of Git

On Windows, installation of the *Git for Windows* package is the most convenient option. Download the installer at https://gitforwindows.org/.

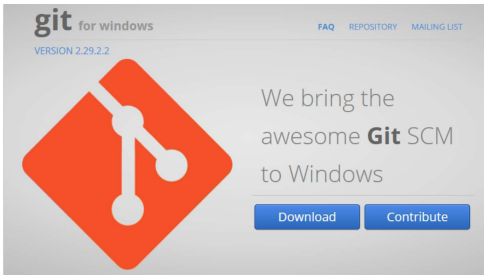


Figure A.3: Git for Windows

- Download the latest version for your OS, which is probably 64-bit Git-2.xx.y.z-64-bit.exe
- On execution of the installer, all *default options* can be chosen.

On a computer with a Linux OS, there are several options, e.g.:

- sudo apt install git (Ubuntu, Debian, RPi)
- sudo pacman git (ArchLinux)

For Apple Macintosh installation, consult https://git-scm.com/download/mac.

A.2.2 Installation of a Git client

Using Git can be done from the command line (Windows Command Prompt, Git bash or Linux bash). On Linux, this is the preferred option. On Windows, there are useful Git client programs, which make versioning easier after a while. In the beginning, they may just confront you with the overwhelming capacities of Git. Going through this phase merits the effort, however. A preferred client can be downloaded at https://tortoisegit.org/download/.

- Download and run the installer: TortoiseGit-2.13.0.1-64bit.msi.
- Install with all *default options*.

This client integrates with Windows Explorer as an addition to the context menus under the right-mouse-click button. Installation with the default options is straightforward. Check if the context menus are extended with Git options.

A second client, with extended functionality can be found at https://www.sourcetreeapp.com/. Recommended for regular Git users, who work in teams with more than three collaborators, or with collaborators at external reasearch partners. Sourcetree is also available for Mac, but (unfortunately) not for Linux.

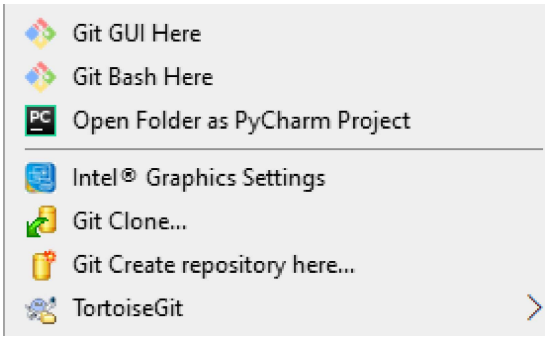


Figure A.4: Git options in context menu

A.2.3 Working with Git

After installation of Git and a Git client, you can start working with Git *repositories*. A repository can be *created* locally, on the harddisk of your PC. Alternatively, a repository from the cloud can be *cloned* to your harddisk.

After these initial operations, your contributions to the code can be *staged* and *committed* locally and *pushed* to the cloud. Contributions committed and pushed by colleagues can be *pulled* to your local harddisk.

Regular pulling (before you start coding) and staging-committing-pushing (after you finished coding) cycles will thus assure, that the repositories of all contributors will be synchronized with each other and with the repository in the cloud. The repository in the cloud is often called *origin*.

Note: Each local Git repository should be placed in a separate *local* folder on the physical HDD or SSD. For clarity, it is recommendable to choose a folder name: ⟨*repository_name*⟩-git. (no spaces in the folder name).

Each created or cloned repository folder contains a hidden bookkeeping folder named .git. This bookkeeping is not compatible with file sychronization mechanisms. So, please do not create repository folders on Microsoft OneDrive, SharePoint, Teams, Google Drive, OneDrive or SurfDrive shared folders.

Repository folders are possible on WebDAV, NFS or SMB mounted shared folders, **if** only one single user has access to the shared folder (from several personal computers, tablets or phones).

A.3 Git workflow

- Creating a repository locally: https://tortoisegit.org/docs/tortoisegit/tgit-dug-create.html
- Cloning a repository from GitHub or Bitbucket; https://tortoisegit.org/docs/tortoisegit/tgit-dug-clone.html
- Pushing to GitHub or Bitbucket: https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html
- Pulling the changes from the cloud to your local repository: https://tortoisegit.org/docs/tortoisegit/tgit-dug-pull.html
- Committing and pushing your local changes to the cloud: https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html

B Installation of Python and Miniconda

Installation of the software for programming in Python on a computer is easiest with a *package manager*. The recommended package manager for Python is named conda. It can be used as a Windows console (Anaconda Prompt), which is very similar to the implementation in a Linux or MacOS X bash console. The console program is called Miniconda and can be downloaded at:

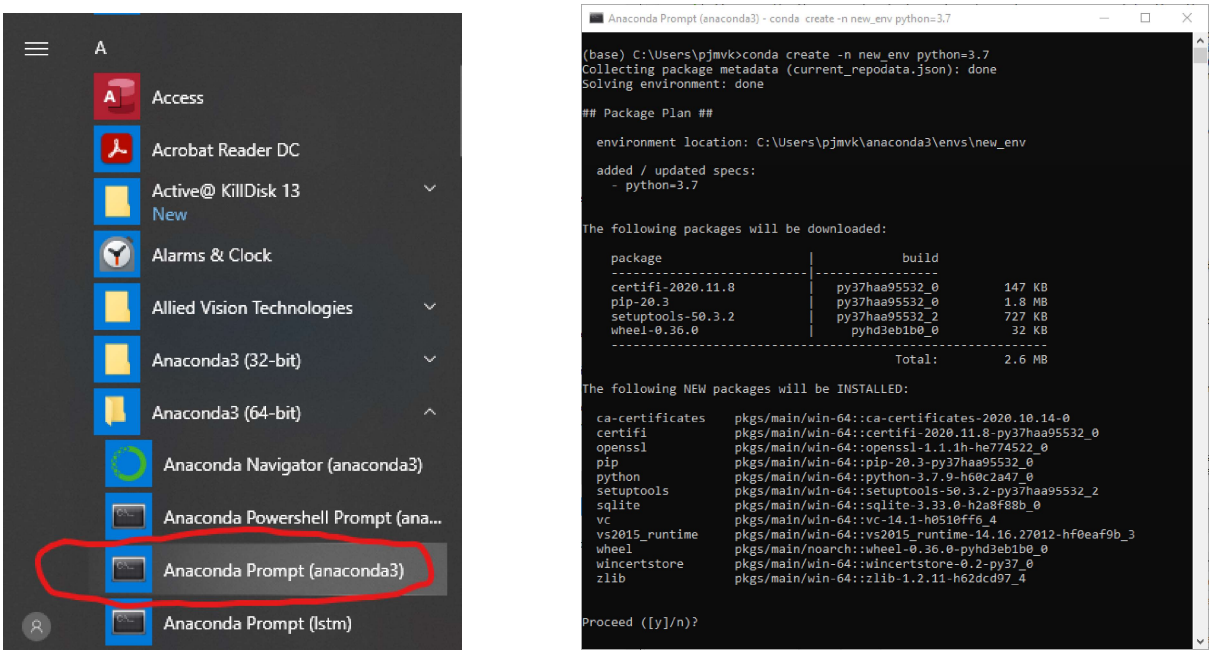
https://docs.conda.io/en/latest/miniconda.html.

Miniconda is available for Windows, Linux and MacOS. In general, nowadays, one would choose the latest Miniconda Python 3.x version 64-bit installer: Miniconda3-latest-Windows-x86_64.exe on Windows, or the analogous installers for Linux and MacOS X.

- Download and execute your installer
- Install "Just for me" and create C:\Users\⟨*username*⟩\miniconda3. Note: ⟨*username*⟩ should NOT contain spaces. Do not install Miniconda system-wide in the "C:\Program Files" directory.
- Take default options for PATH and REGISTRY update

B.1 Anaconda Prompt

After installation of Anaconda (Miniconda) the Windows Start Menu contains a new group "Anaconda3". A detailed view is shown below:



(a) Anaconda Start Menu **(b)** Anaconda Prompt

Figure B.1: Anaconda (Miniconda) tools installed on Windows

The Start Menu entry "Anaconda Prompt (miniconda3)" will open a Windows Command Prompt ("MS-DOS window") with special properties. The current directory name (standard prompt) is preceded by (base). This is the name of the default conda *environment*. The (base) environment is *NEVER* used for code development. After installation of the package manager, it contains the latest version of the Python interpreter, which is kept up-to-date with the command:

```
conda update -n base -c defaults conda
```

The Python interpreter in the (base) environment may be used by other Windows programs, which will then ask you for the location of the default Python interpreter.

The standard location is: C:\Users\⟨*username*⟩\miniconda3\python.exe.

B.2 Conda configuration

Before creating any dedicated environments, it is recommended to optimize the conda configuration [3, 4]. In the previous command, the conda *package* was updated in the base *environment*. The package was read from the defaults *channel*. Other channels exist, that often contain more up-to-date versions of important Python packages. The channel conda-forge is keeping a more recent copy of many packages, including the opencv package, which is a key package used for image analysis. Therefore we add:

- conda config –show this shows the configuration file. Look for "channels:".
- conda config –add channels conda-forge this adds the conda-forge channel before the defaults channel (top of the list).
- conda config –set channel_priority strict this guarantees that channel priority is maintained according to the list.

The config file now shows:

...
channel_priority: strict
channels:
- conda-forge
- defaults
...

See also:
https://stackoverflow.com/questions/39857289/should-conda-or-conda-forge-be-used-for-python-environments
https://conda-forge.org/docs/user/tipsandtricks.html#using-multiple-channels

B.3 Conda environments

A dedicated environment can be made with the command:

conda create -n ⟨*env_name*⟩ python=3.x

The next commands are:

activate ⟨*env_name*⟩

conda info –envs or conda info -e: note the active environment with asterisk *

conda install ⟨*package_name*⟩ or pip install ⟨*package_name*⟩

The result can be seen with: conda list : note the packages are all compatible with your Python version

The commands can be stored in a textfile:

Code Listing 10: Example textfile with conda commands for new environment

```
conda config --add channels conda_forge
conda config --set channel_priority strict
```

```
conda create -n impedance python=3.8
activate impedance
conda info -e

conda install matplotlib
conda install scipy
conda install pandas
conda install openpyxl
conda install pyyaml
conda install -c conda-forge lmfit

conda list
```

Environments are stored in the private directory of the user. On Windows: C:\Users\⟨*username*⟩\miniconda3\envs

On Linux: /home/⟨*username*⟩/miniconda3/envs

Note: Code development or storage is NEVER done in C:\Users\⟨*username*⟩\miniconda3 NOR in any of its subfolders.

B.4 Python from the console

The Anaconda Prompt program on Windows is a specialized version of the general Windows Command Prompt, formerly known as "MS-DOS window" or "console". The advantage of Anaconda Prompt is that all file paths to the Miniconda installation are automatically set and that the prompt indicates the currently active Python environment. On Linux and MacOS, Anaconda Prompt is integrated in the command-line terminal. Thus Anaconda Prompt looks very similar on all operating systems.

Running a Python script in the console is as simple as:

activate ⟨*env_name*⟩

conda info –envs or conda info -e: note the active environment with asterisk *

cd ⟨*repository_root*⟩

python ⟨*script_name*⟩

For an overview of Python command-line features, see:

https://realpython.com/python-command-line-arguments/#the-command-line-interface

B.5 Python IDE

B.5.1 Installation

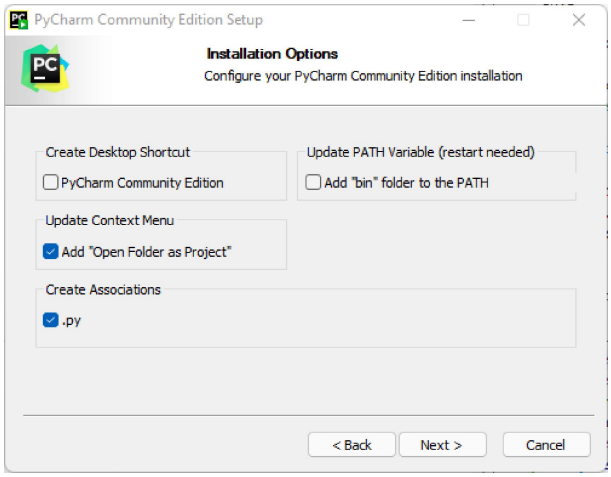


Figure B.2: PyCharm IDE installation on Windows

After creation of the conda environment with the right selection of Python *packages* for the analysis of the impedance measurements, it is recommended to develop the Python software in a dedicated IDE, where Python scripts can be edited, run, and debugged. The recommended program for this purpose is PyCharm, which can be downloaded here:

https://www.jetbrains.com/pycharm/download/#section=windows

https://www.jetbrains.com/community/education/#students

- Download the "Community" installer pycharm-community-2021.3.2.exe and run it.
- Install in default directory.
- Check options according to Figure B.2.
- after succesful installation, navigate to the *repository root* folder of your project and right-click on the canvas of the folder window (not on a file or subfolder icon). Choose the "Open Folder as PyCharm Project" item. The PyCharm IDE will start and open the project and subfolders.
- Once a project has been opened in PyCharm, it creates a (hidden) subfolder named .idea in the repository root folder. This folder contains the bookkeeping of the IDE. Do not change anything in this .idea subfolder. If anything goes wrong, delete it entirely, and PyCharm will create a fresh one.
-

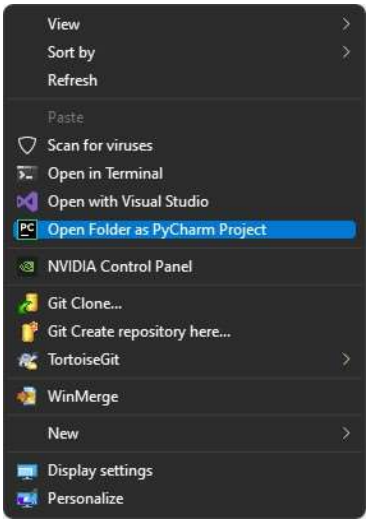
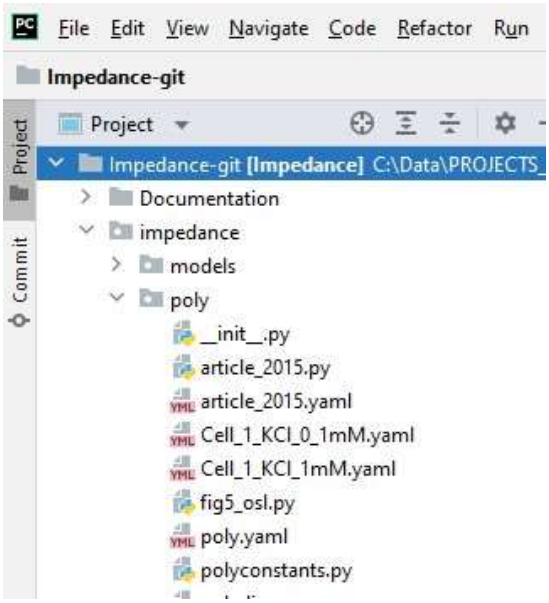


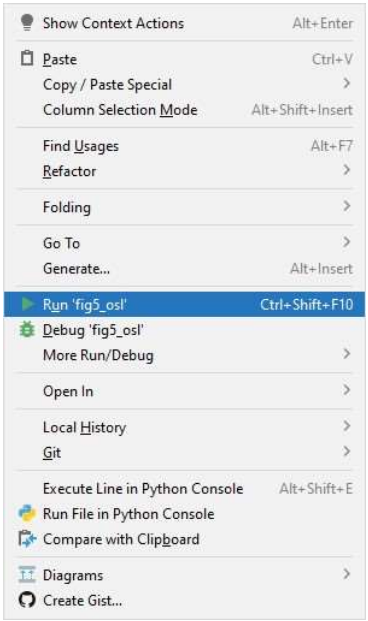
Figure B.3: PyCharm context menu after right-click on root folder canvas

B.5.2 Configuration

- To work with your Python code in PyCharm, you need to configure a Python interpreter. Assign the Python interpreter from your (existing!) conda environment created following the guidelines in Appendix B.3 to the open project. See: https://www.jetbrains.com/help/pycharm/configuring-python-interpreter.html.
- After configuring the interpreter, the IDE needs some time to load all the packages from your environment. A progress bar is visible on the bottom of the IDE window.



(a) Project tab with root folder and subfolders



(b) Context menu for right-click

Figure B.4: PyCharm project configuration and running

B.5.3 Running and debugging Python scripts

- Selecting a Python script in the Project tab can be done with a left-click. Right-clicking on a Python (*.py) module opens the context menu of Fig. B.4b. Clicking the green arrow in the context menu starts

executing the selected script.

- Double-clicking a *.py module in the Project tab opens the module in the editor window of the IDE. Right-clicking on the canvas of the editor window also opens the context menu of Fig. B.4b.