Computational Radiometry Work Package

Document Number	Equipment or Sub-System

Subject

00-Installing-Python-pyradi

Distribution

Conclusions/Decisions/Amendments

Author	Signature
CJ Willers	

Date	Date	Date August 23, 2021
Previous Package No.	Superseding Package No.	Current Package No.

1 0 Installing Python and pyradi

This notebook forms part of a series on computational optical radiometry[1]. The notebooks can be downloaded from Github[2]. These notebooks are constantly revised and updated, please revisit from time to time.

The date of this document and module versions used in this document are given at the end of the file.

Feedback is appreciated: neliswillers at gmail dot com.

1.1 Short summary

Install these to get started quickly. This is our current preference of tools. There are alternatives, but these are our current favourites (assuming a Windows install):

1. Anaconda https://www.anaconda.com/products/individual.
2. Visual Studio Code https://code.visualstudio.com/download.
3. VS Code Python extension https://code.visualstudio.com/docs/languages/python.
4. Git for Windows (git and bash command line) https://gitforwindows.org.
5. Tortoise git https://tortoisegit.org/download/
6. Terminal console https://conemu.github.io/ (alternative use the pre-installed Windows Terminal)

Next install pyradi[3] and make sure that Python know where to look for it[4].

You may want to know about conda environments[5]. To enable Jupyer to work with different environments[6]

1.2 Python and tools

When asked which books she would recommend for newcomers to Python, my better half immediately said: find a real problem and good tools, then use Google search to solve the specific questions you run into. This made me think. Yes, Python is a computer programming language Yet, in its broader ecosystem is also so much more than just a language with syntax. Think of Python, its libraries and development environment as a set of problem solving tools, more than just a language. To use the tool set you need to know the language syntax, but you will find yourself quickly moving beyond the focus on a language towards focusing on solving the problem. It is almost as is the language will disappear as the focus shifts.

To be able to make the shift towards problem solving, you need a few building blocks in place:

1. Do get to know the language syntax, especially the built-in data structures such as lists, dictionaries, tuples and sets (also study the advanced specialized container data types in the `collections` module). Build a solid insight into common programming idioms, often referred to the 'the Pythonic way' (https://docs.python-guide.org/writing/style/) (this is very different from programming in traditional languages).

2. The really powerful tools in the Python toolset are the very many specialist packages (aka modules). Depending on your need these packages could be the scientific and data processing tools (numpy, scipy, sympy, pandas, etc.), web and database tools (django, flask, alchemy, etc.), machine learning (scikit-learn, PyTorch, Tensorflow), or specialist packages in astronomy, genomics, and many more fields. Most technical applications would benefit from a good grounding in numpy, scipy, sympy, and pandas, thereafter extending to other tools.

3. The 'best' development tool set can greatly aid in solving the problem. For scientific work the electronic lab book called Jupyter Lab is very popular. More conventional integrated development environment (IDE) include tools such as PyCharm (not free) and the free tools such as Spyder (available in Anaconda), or Visual Studio Code (do install the free Python extension by Microsoft). At a more basic level any text editor can be used, because Python uses standard ASCII files for scripting.

4. Python is most often installed as part of a bigger distribution that includes many more tools and packages than just the core Python interpreter. For scientific work the Anaconda distribution is the most popular — being very complete, it is also very large with a more than a gigabyte footprint on the hard disk. The minimal installation which includes only the core Python functionality which is powerful but excludes the specialist packages.

5. Modern software development is almost unthinkable without a version control repository. The most popular tool here is git, with alternatives being subversion and mercurial. Do use a repository for development work, even if just locally on your computer. If you require off-site repository storage consider the likes of github or gitlab.

6. Take the trouble to learn the Markdown markup language, it is the de-facto standard in the code development world today. Scientific workers may also find benefit in using LaTeX for more advanced scientific and mathematical publication.

7. Python is most at home in a command window. Linux has several such terminals. Windows users can surely move beyond the standard `cmd` terminal to something like https://conemu.github.io/ or PowerShell.

1.3 Programming editors and Python IDEs

Before discussing the Python installation procedure, consider potential development environments. These tools are critical to your success and positive experience with using Python.

Which integrated development environment (IDE) to install? You can write Python scripts with any ASCII editor (even notepad) and run the script in a command console. Some people find it more productive to work in a more powerful IDE environment. Finally, to build more comprehensive 'lab books' of your work with code, text, graphs, etc., use the IPython notebook.

There are a number of options listed here[7], starting from simple text editors, all the way to comprehensive IDE environments.

We will focus on the Windows OS here, because Linux users already know which tools they want to use.

A lightweight way is to use a standard text editor (to write the script) and a command window (to execute the script). Good editors include:

`http://www.sublimetext.com/`[8] general purpose text editor, costs money.

http://notepad-plus-plus.org/download/[9] free, general purpose editor.

https://atom.io/[10] free, general purpose editor.

https://wiki.python.org/moin/Vim[11] if you know what vim is you don't need this. If your don't know about vim, don't bother.

There are several command windows to choose from. Windows users can use a third party command window (see here: http://aarontgrogg.com/blog/2015/07/31/a-better-windows-command-line-experience-comparing-powercmd-vs-console2-vs-consolez-vs-conemu-vs-cmder/). My current preference is conemu available here: https://conemu.github.io. Alternatively Windows users can use Microsoft's cmd.exe or Powershell. Linux users probably already have their favourite command window.

Specalised IDEs include (some might be included in large Python distributions):

http://pythonhosted.org/spyder/[12] good IDE already included in Anaconda.

https://www.jetbrains.com/pycharm/[13] free community version.

https://pytools.codeplex.com/[14] (Visual Studio plugin).

http://sourceforge.net/projects/pydev[15] (Eclipse plugin).

The IPython notebook is a very powerful tool to record all your work and finally produce a report with all your documentation, code and results in one file. For more information on the IPython notebook see here:

http://nbviewer.ipython.org/github/NelisW/ComputationalRadiometry/blob/master/01-IPythonHintsAndTips.ipynb

1.4 The Python Ecosystem

Some Python installations are not kind to spaces in install folder names. Avoid spaces in any folder or filename to prevent issues.

Any Python path change only comes to effect when a new command window is opened. So if you make a change to the *.pth file, close the currently open Python session or Jupyter notebook, then close the command window, and finally open a new command window and run a new Python session.

The Python ecosystem comprises - The Python core language that contains the language and standard library. You can see this as a minimal, but already powerful language package. - Specialist packages that you can install if and when required. One such specialist application would be science and engineering, which would require NumPy, SciPy, Matplotlib, and some other packages.

You have two main options: - Install the core Python package and then install the specialist packages manually. - Load everything in one big installation https://stackoverflow.com/questions/15762943/anaconda-vs-epd-enthought-vs-manual-installation-of-python

I would recommend the second option. There are a number of fairly complete distributions, including: https://store.continuum.io/cshop/anaconda/ (my preferred distribution), https://www.enthought.com/products/epd/ https://code.google.com/p/pythonxy/, http://www.sagemath.org/> (not very well known as a Python distribution).

Which to install: Python 2.7.X or Python 3.X? If you have no compelling reason to use 2.7 (i.e., compatibility with some project), use 3.X.

Which to install: 32-bit or 64-bit? If you want to remain compatible with other 32-bit installation

packages or linking in with other 32-bit binary code, you must install the 32-bit version. If these restrictions do not apply, install the 64-bit version on a 64-bit PC.

pyradi is only tested with Python 3.7 and 3.8, it is no longer supported on 2.7

1.4.1 Difference between conda and pip

https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/

1.4.2 Installing Anaconda

Install Anaconda for Python 3.X[16].

http://conda.pydata.org/docs/_downloads/conda-cheatsheet.pdf

1.4.2.1 Conda package manager

http://conda.pydata.org/docs/test-drive.html

https://youtu.be/UaIvrDWrIWM

1.4.3 Anaconda environment path settings

Some users prefer to have the Anaconda/Python paths permanently in the environment. This enables you to open Python from any command window, without using the Anaconda menus settings (When using the Anaconda menu, the paths are set for the one command window instance only).

Some installations had difficulty finding the required version of `libiomp5md.dll`. It appears that one installation had two (different?) version of the file:

```
where libiomp5md.dll
C:/ProgramData/Continuum/anaconda3/Library/bin/libiomp5md.dll
C:/Program Files (x86)/Common Files/Intel/Shared Files/cpp/Bin/Intel64//libiomp5md
```

Make sure that the Anaconda path comes first, above the other instances. Note that this may break the other application using the other instance.

1.4.4 Multiple versions of Python in Conda environments

Please read this excellent overview of conda environments:

https://www.freecodecamp.org/news/why-you-need-python-environments-and-how-to-manage-them-with-conda-85f155f4353c/

If you need to run different versions of Python (e.g., 3.3 and 3.6), consider using the conda environment infrastructure. A Conda environment is a new Python installation that lives alongside any other base installations or environment installations you already have. Essentially every Conda environment is a separated Python installation. When you want to use the specific python in the environment,

you activate the environment. Activation prepares a command window and sets up the paths to the Python version in that specific environment.

http://conda.pydata.org/docs/using/envs.html

The underlying principle here is that you create a thin root installation environment that has the bare essentials to support conda and not much more. You then create conda environments[17] where you can install full anaconda versions of the the appropriate Python revisions. You do the real work in the environments by activating the environment before running the python and packages in that environment. When you create environments in conda, it creates a more or less complete Python installation in the ..../envs/ directory in an existing miniconda or anaconda installation. So the multiple versions are installed side by side.

Activating an environment must be done in a command window; where the process of activation simply modifies the system level paths from the root environment to the newly activated environment. The new environment is active in the present command window until the environment is deactivated, which means that the original paths to the Python folders are restored back to the root environment.

It is recommended practice to keep a small root environment, solely as a tool to create new environments - all the real work is done in the specially created environments. This is a means to control which package versions for development, testing and deployment.

Install miniconda[18] (Python version 3.7). Note that the choice of which Miniconda is installed only affects the root environment. Regardless of which version of Miniconda you install, you can still install many different Python-version environments (even Python 2.7!).

On Windows - it seems that `win32api` is used during a full anaconda install, so you might have to install

```
conda install pywin32
```

- it seems that installing conda requires admin rights for some of the tasks during the installation. The installation does complete, but there are warnings that for some tasks admin rights are required. Within the scope of my Python work, there never was a problem, but to install without any warning, start the command console as admin before installing conda/anaconda.

Then create environments for the versions you want to install (use whatever names you want, like py36, pymc, myenv):

```
conda create --name pymc python=3.7 anaconda
conda create --name py36 python=3.6 anaconda
```

To activate the environment in a command window

```
activate py36
```

To deactivate the environment in a command window

```
decativate
```

To list all environments

```
conda info --envs
conda env list
```

To list modules in current environment

```
conda list -e
```

To search to see if a package is available in the anaconda repository (not your disk)

```
conda search numpy
```

Update an environment by activating it first and then updating - but note that there are two methods:

1. `conda update anaconda` - this command updates the anaconda meta-package as a list of packages tested against each other, known to work together.
2. `conda update --all` - this command updates individual packages to the individual latest versions, irrespective of compatibility status. It is likely that these packages will work together, but it is not tested and verified.

   `conda update --all` - this command updates individual packages to the individual latest versions, irrespective of compatibility status. It is likely that these packages will work together, but it is not tested and verified.

   activate py36

   conda update anaconda

After being used for a while, conda can accumulate a lot of disk space, because it doesn't remove old unused packages. To remove unused packages (-p), index caches (-i) and tarballs (-t), run

```
conda clean -pit
```

To upgrade/update to a specfic new version of anaconda in one of the environments, run

```
deactivate
conda update conda
activate py36
conda install anaconda=2.4.1
```

Conda loads the new environments in the `*/Miniconda*/envs` folder, inside the base environment (unless you instructed it to install elsewhere).

To remove a conda environment use:

```
conda remove --name py36 --all
conda info --envs
```

1.4.5 List package versions in conda (aka requirements.txt)

Sometimes it is necessary to recreate a new environment with exactly the same versions as an existing environment. The packages and versions can be captured in the `requirement.txt` file for the environment. This can be done in conda using the following command:

```
conda list --export > requirements.txt
```

When you are ready to distribute your package to other users, they can easily duplicate your environment and the associated dependencies with

```
conda create --name <envname> --file requirements.txt
```

1.4.6 Installing Anaconda on Linux

Download the installer. In your terminal window type one of the below and follow the instructions:

Python 3.x:

`bash Anaconda3-4.0.0-Linux-x86\_64.sh`

NOTE: Include the `bash` command even if you are not using the bash shell

`https://www.continuum.io/downloads`[19]

1.4.7 conda-forge

https://conda-forge.org/ contains user-supplied packages not considered as part of the central anaconda distribution. A package in conda-forge can be installed with one of the following commands, which tells conda to look in the conda-forge repository:

```
conda install --channel https://conda.anaconda.org/conda-forge packagename
conda install -c conda-forge packagename
```

where `packagename` is the name of the package to be installed. The packages available on conda-forge are shown here: http://conda-forge.org/feedstocks/

You can add the conda-forge channel to the list of available channels searched by conda when installing:

`conda config --add channels conda-forge`

1.4.8 conda package versions and 'pinning'

The most commonly-used packages are installed in the base `anaconda` install. You may want to later install packages that are not part of the base installer, but are available in the `anaconda` library.

This is done with the command (in this example case to install mayavi):

```
conda update conda
conda install mayavi
```

The new package install will automatically also install all depencency packages. Normally this is acceptable, but in some cases, the new package requires an earlier version of an already installed package - in which case the installed dependency package will downgrade the installed package to an older version. This may cause problems if another application requires the later version.

You can 'pin' package numbers to some release number to prevent automatic install changes to the release number, see here[20]. If a package revision number is pinned, it will stay at the pinned revision, irrespective of all upgrade or dependency attempts to change it.

The `--no-deps` flag prevent automatic changes/upgrade/downgrade of intalled package dependencies. Note that this option will prevent and and all dependencies from downloading, so the newly installed package may not work correctly.

Conda attempts to install the newest versions of the requested packages. To accomplish this, it may update some packages that are already installed, or install additional packages. To prevent existing

packages from updating, use the `--no-update-deps option`. This may force conda to install older versions of the requested packages, and it does not prevent additional dependency packages from being installed (http://conda.pydata.org/docs/commands/conda-update.html)

To force conda to install a specific version of a package, use the form `conda install scipy=0.17.0`, where the required version number is given after the equal sign.

If you wish to skip dependency checking altogether, use the `--force` option. This may result in an environment with incompatible packages, so this option must be used with great caution.

The `conda update` command also have these two dependency flags: `--update-dependencies`, `--update-deps`: Update dependencies (default: True). `--no-update-dependencies`, `--no-update-deps` Don't update dependencies (default: False).

1.4.9 Updating Anaconda from behind a proxy in Windows

Create a `.condarc` configuration file in your user home directory (`C:/Users/[yourusername]`), follow the instructions in the websites below. Place the following text in the `.condarc` file, but replace with the information relevant to your situation:

```
proxy\_servers:
    http: http://[yourusername]:[yourpassword]@[your proxy server]:[proxy port]
    https: https://[yourusername]:[yourpassword]@[your proxy server]:[proxy port]
```

The difficult part is not creating the file, but finding out what the details for your situation are.

`http://conda.pydata.org/docs/config.html`[21]

`https://groups.google.com/a/continuum.io/forum/#!msg/anaconda/WQBn1c-spQ4/HVa1v83zAlEJ`[22]

Please note that this

1.4.10 Jupyter lab Widgets Extension

Some of the notebooks use the Jupyter widgets. The widget manager must be installed manually after the anaconda install

`jupyter labextension install @jupyter-widgets/jupyterlab-manager`

https://www.npmjs.com/package/@jupyter-widgets/jupyterlab-manager

https://jupyterlab.readthedocs.io/en/stable/user/extensions.html

1.4.11 Multiple Python environment in the Jupyter (IPython) notebook

Jupyter supports multiple environments by running different kernels for each version. So you can run conda environments for Python 3.6 and Python 3.7 notebooks from the same server, just by selecting the kernel.

It seems that jupyter (ipython notebook) only runs from the root and not from an environment. So we follow a procedure here of installing jupyter in the root installation and then create different kernels for each of the different Python versions installed in the environments.

The idea here is to install multiple ipython kernels[23] and here[24]. Here are instructions if you are using anaconda:

Install miniconda[25] or anaconda[26]. If you want only a basic Python capability in the root installation, install miniconda[27]. On the other hand if you want a full anaconda capability in the root environment, install anaconda[28]. Note that you will be installing full anaconda versions in the different environments. The only reason not to use miniconda is to have the full anaconda suite available at any moment, without activating a conda environment[29].

Install jupyer in the miniconda root

```
deactivate
conda update conda
conda install --upgrade jupyter
```

Suppose we want a new conda environment[30] (named py36) with python 3.6 and set up Jupyter to work with py36.

you might already have done the installations, don't repeat:

```
conda create -n py36 python=3.6 anaconda
conda install --upgrade <<any packages you need in this environment>>
```

now continue with ipykernel installation in py36

```
activate py36
conda update anaconda
conda install notebook ipykernel
ipython kernel install
deactivate
```

Repeat the the last procedure for all environments you want to use in Jupyter.

Note Early in 2020 the above command `ipython kernel install` did not have the desired effect. In the end the following command did the job (`mort` is the environment name in this case):

```
python -m ipykernel install --user --name mort --display-name "Python (mort)"
```

The `--name` and `--display-name` options to ipython kernel install is the name of the kernel and the display name in the notebook. See ipython kernel install `--help` for more information or here[31].

To see which kernels are available open jupyter and start a new notebook. Alternatively use the command:

```
jupyter kernelspec list
```

which will display the list of available kernels similar to the following:

```
Available kernels:
  mort       C:/Users/NWillers/AppData/Roaming/jupyter/kernels/mort
  python3    C:/ProgramData/Anaconda3/share/jupyter/kernels/python3
```

These lines are paths to where the kernel is defined. Note that the two kernels are not defined in the same folder. In this case `mort` was created by the above user command. The Python3 kernel was created by the installer.

To remove a kernel use the command

`jupyter kernelspec uninstall <kernel name>`

where `<kernel name>` would be something like `mort` for the above list. *Don't remove the kernel installed by the installer.*

When set up this way you can also use any environment by simply activating the respective version[32].

1.4.12 Jupyter Kernel Error

On occasion you may find that, when loading a notebook, you get a kernel error

`The system cannot find the file specified`

It seems to be created after environments are removed. I tried to run the following (as given in previous section), but it had no effect:

```
conda install notebook ipykernel
ipython kernel install
```

After google searches I found this solution, which worked:

`python -m ipykernel install --user`

this seems to (re?)install the kernel for the current user, using python instead of the ipython command show in the previous section.

https://github.com/jupyter/notebook/issues/2301 (nice detail here)

https://github.com/jupyter/notebook/issues/4079 (involves changing a json file, but the above worked, so I did not try this)

1.4.13 Links to local libraries that are not installed in the Python directory structure

Python looks for installed packages in the `/Lib/site-packages` folder. If the package is not found in one of the many `/Lib/site-packages` folders, Python will fail importing the package.

It is possible to keep a copy of your package elsewhere, outside of any `/Lib/site-packages` folder, but then you have to tell Python about the package name and where the package is installed on your PC. To do this, we need to find the `/Lib/site-packages` folder and then place a `*.pth` file in this folder.

1.4.13.1 Locating /Lib/site-packages folders

The python distribution e.g., Anaconda, can be installed in a number of different locations. To determine where the conda environments on your PC is located, open a command window and type the following:

```
conda env list
```

Now we have to determine the location of the `site-packages` folder in the environments, from which Python can locate the available modules. This location could be one of the following (or similar):

```
C:/Users/YourUserName/Anaconda3/Lib/site-packages
C:/Miniconda3/Lib/site-packages
C:/ProgramData/Anaconda3/Lib/site-packages
```

In the conda environments the packages are installed in the `..../envs` folder, inside the base environment, at locations with the following name structure:

`'<where installed>/./envs/<enironment name>/./Lib/site-packages'`

i.e., something like

```
C:/Users/YourUserName/Anaconda3/envs/py36VTK/Lib/site-packages
C:/Users/YourUserName/Anaconda3/envs/landlab/Lib/site-packages
C:/ProgramData/Anaconda3/envs/pymc/Lib/site-packages
```

1.4.14 The *pth file

To tell Python about your special installation folder, place a `<module name>.pth` file in the `site-packages` directory in the environment where this package must be available.

The name of the file must correspond to the name of the module `<module name>` and the file extension must be `.pth`. The file must contain the absolute path to the module on the first line. BTW the `pth` extension is short for path, which is what is in the file.

For example, if I cloned the `https://github.com/NelisW/pyradi` module from github to my hard disk at `V:/work/WorkN/pyradi`, I can place a file in the root and or environment `site-packages` folder(s) with the name `pyradi.pth` that contains the line `V:/work/WorkN/pyradi/pyradi`. Note that in this case `pyradi` is installed at a particular location, but the path written to the file is further down the folder structure with an additional `pyradi`. This is because the module is actually one level deeper.

If you use the `.pth` method, Python does not know if it is a local folder structure, a repository, or a pypi installation, just so long as it has legal Python module code, all is fine.

I have not done this yet, but in principle you can have many different versions of a module in different directories on your PC (e.g. for testing with different versions of Python, or with different functionality for whatever reason). You can then create different conda environments where each environment has its own unique and different `.pth` file, pointing to the specific code you want to use with that specific environment. The different `.pth` files must then be placed in each environment's own `site-packages` directory, to be used with that specific environment.

References:

`https://docs.python.org/2/library/site.html`[33]

`http://bob.ippoli.to/archives/2005/02/06/using-pth-files-for-python-development/`[34

`https://grahamwideman.wikispaces.com/Python-+site-package+dirs+and+.pth+files`[35]

`http://stackoverflow.com/questions/15208615/using-pth-files`[36]

1.5 Installing pyradi

You only need to do this if you want to use pyradi for computational radiometry.

pyradi is no longer supported on PyPi.

1.5.1 Best option: use Git

You should really consider using a software version control process for your coding and writing work. There are several alternatives, but I prefer to use the current mainstream tool Git. So install git, use it to manage pyradi on your PC, but more importantly, *use it to manage your own work afterwards.*

If you are not already using git consider the following resources:

http://lmgtfy.com/?q=starting+git

https://git-scm.com/book/en/v1/Getting-Started

https://rogerdudler.github.io/git-guide/

http://sirupsen.com/starting-with-git/

https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

Search for git videos on YouTube.

There are also good books on Git.

If you use Linux, you probably already have git on the system and might also know how to use it.

Working on Windows I would recommend installing git from https://gitforwindows.org/, and then installing the Tortoise Git Explorer shell interface from https://tortoisegit.org/download/. Tortoise creates context menus on your Windows Explorer which makes it very easy to use for most git tasks.

To checkout the pyradi repo[37] from github you *do not* need a github account. You only need to create a github account if you want to create repositories on GitHub. So initially, no need to get an account.

Go to https://github.com/NelisW/pyradi and look for the green button on the right side, near the top of the page and click on it. Then click on the little clipboard icon to load the link from the textbox to your local PC clipboard.

Open a Windows Explorer and create a new folder (e.g., `your-master-folder`) where you want to clone the pyradi repo (the folder can be anywhere you want, just be careful to not have spaces in the pathname). Right-click in the empty folder and then in the context menu, click on `git clone...`. This should open up a dialog box that already shows the pyradi URL pasted from the clipboard. The directory where the repo will be cloned is also shown. Don't change anything and click on the OK button. If all goes well a new window appear that shows the progress with cloning (downloading) the repo.

Also clone the pyradi data from https://github.com/NelisW/pyradi-data and (optionally) the pyradi docs from https://github.com/NelisW/pyradi-docs, using the same procedure as above. Clone in the same folder where pyradi is installed. Use exactly the same names for pyradi and pyradi-data. So when you are done there should be two or three folders depending on where you installed docs.

```
your-master-folder
    pyradi
```

```
    pyradi-data
    pyradi-docs
```

At this point you have pyradi on your PC but your Python installation does not know about it yet. Continue with the procedure below to tell Python about your pyradi files' location in the `pyradi.pth` file.

If you want to update the locally cloned version later (and we recommend that you do this from time to time), right-click on the folder where pyradi is installed and click on the `TortoiseGit` line and then the `Pull` line. A dialog box should appear where you click on OK. This way you will always have the latest version of pyradi.

1.5.2 Second best option: download a zip file

This option is second best because to get later versions you have to download the zip file(s) again and again. If you use Git, you can just 'pull'.

Go to https://github.com/NelisW/pyradi and look for the green button on the right side, near the top of the page; click on the green button. At the bottom of the small popup window, click on the `Download Zip` link to download the zip file to your PC.

Also download the pyradi data from https://github.com/NelisW/pyradi-data and (optionally) the pyradi docs from https://github.com/NelisW/pyradi-docs, using the same zip file download procedure as above.

Open a Windows Explorer and create a new folder (e.g., `your-master-folder`) where you want to unzip the pyradi repo (the folder can be anywhere you want, just be careful to not have spaces in the pathname). Unzip all two or three zip files into one this folder. So when you are done there should be two or three subfolders. Keep the three subfolder names exactly as shown, don't change them.

```
your-master-folder
    pyradi
    pyradi-data
    pyradi-docs
```

At this point you have pyradi on your PC but your Python installation does not know about it yet. Continue with the procedure below to tell Python about your pyradi files' location in the python.pth file.

If you later want to instal updated versions, download the zip files and copy over the current files in the same directories.

1.5.2.1 Tell Python about your pyradi files' location in the python.pth file.

Follow the procedure in the previous section to create a `pyradi.pth` file. The file must be named `pyradi.pth` and the first (and only line) in the file must be the path to where the pyradi module is available. On my PC this is `V:/work/WorkN/pyradi/pyradi` - note the second pyradi in the link to point further down the repository tree.

You do not have to tell Python about the documentation or data.

Now if you now start up a Python in a new command window, Python will locate the pyradi module by looking up the location on your PC from the pyradi.pth file.

1.5.3 Mayavi in pyradi

pyradi should work without any additional package installations beyond Anaconda, except for the optional use of Mayavi[38]. Mayavi is only required for the three-dimensional spherical plots, you may ignore the requirement otherwise. Most sources advise users to install Mayavi as part of a Python distribution, such as the Enthought Tool Set[39] or Anaconda[40] distributions. This is certainly an option for newcomers, but these distributions are not always current with the latest code package versions. Of the two, Anaconda appears to be quicker with the updates. Look for Anaconda[41] (under the supervision of Travis Oliphant, one of the driving forces behind Numpy) or Enthought Tool Set[42], both of which contains contains Mayavi and other tools. You can find Mayavi (and other) installation packages on this site[43].

1.6 PYTHONPATH and Executing Python Scripts

The PYTHONPATH environmental variable tells python where to look for modules when importing, *it does not define the path to a *.py script to be executed*. If you have a module in a directory outside the standard Python install directory (e.g., an experimental module in your HOME directory), add the path to the module to the PYTHONPATH environmental variable. The general advice is to not change the environmental variable permanently on operating system level, because of potential conflicts with other installed packages.

1.6.1 Adding to PYTHONPATH for the current session only

From inside Python execute the following command to add a new path to your current PYTHONPATH, which will not add the path permanently:

```
import sys
sys.path.append('/path/to/your/python/module')
```

To insert your new path at the front of PYTHONPATH (useful if you want to look in your local directories first, overriding existing installs):

```
sys.path.insert(0,'/path/to/your/python/module')
```

To get a list of the paths currently in PYTHONPATH:

```
import sys
print(sys.path)
```

1.6.2 Adding to PYTHONPATH permanently for all sessions

If you are using/developing a module that is not located in `.../site-packages/`, but you want a path to the module on a more permanent basis in your PYTHONPATH, use a path configuration file. Path configuration files are usually placed in the `.../site-packages/` directory (or a few other places) and can have any name, but with an extension of .pth. Each line must contain a single path that will be appended to sys.path. New paths are *appended* to sys.path, modules in the added directories will not override standard modules.

It is not clear from the documentation, but the paths to the modules listed in the pht file must be *relative* to one of the site packages directories. These directories are where Python would look for modules. Absolute paths do not seem to work (at least on Windows, the are Linux examples that do have absolute paths). If your module is located on the same disk as `.../site-packages/`, then it appears necessary to provide a path relative to this `.../site-packages/`, i.e., something like `../../../pathtomodule`. If the module is not on the same disk/drive as `.../site-packages/`, you should probably set up a secondary site-packages directory on the other disk (using `addsitedir`).

`https://docs.python.org/2/library/site.html`[44]

`https://docs.python.org/2/library/site.html#site.USER\_SITE`[45]

`https://docs.python.org/2/install/`[46]

`https://stackoverflow.com/questions/15208615/using-pth-files`[47]

`http://bob.ippoli.to/archives/2005/02/06/using-pth-files-for-python-development/`[48]

`https://groups.google.com/a/continuum.io/forum/#!topic/anaconda/RO\_h0wD7sKE`[49]

`https://stackoverflow.com/questions/10693706/creating-a-secondary-site-packages-di`

`https://stackoverflow.com/questions/19914527/using-python-pth-files-assuming-no-ac`

`https://docs.python.org/2/library/site.html#site.addsitedir`[52]

1.6.3 Anaconda

Use any of the above methods

https://stackoverflow.com/questions/17386880/does-anaconda-create-a-separate-pythonpath-variable-for-each-new-environment

1.6.4 Canopy

Canopy users please see this

`https://support.enthought.com/entries/23665767-How-do-I-set-PYTHONPATH-and-other-e`

1.6.5 General information:

`https://stackoverflow.com/questions/19917492/how-to-use-pythonpath`[54]

`https://stackoverflow.com/questions/4580101/python-add-pythonpath-during-command-l`

`http://www.stereoplex.com/blog/understanding-imports-and-pythonpath`[56]

`https://stackoverflow.com/questions/1489599/how-do-i-find-out-my-python-path-using`

`http://users-cs.au.dk/chili/PBI/pythonpath.html`[58] http://docs.continuum.io/anaconda/ide_integ

```
import sys
print(sys.path)
```

```
['K:\\WorkN\\ComputationalRadiometry', 'C:\\Users\\nwillers\\Anaconda3↵
   \\python38.zip', 'C:\\Users\\nwillers\\Anaconda3\\DLLs', 'C:\\Users↵
   \\nwillers\\Anaconda3\\lib', 'C:\\Users\\nwillers\\Anaconda3', '', '↵
   C:\\Users\\nwillers\\Anaconda3\\lib\\site-packages', 'K:\\WorkN\\↵
```

```
python-bibtexparser', 'K:\\WorkN\\pyradi\\pyradi', 'C:\\Users\\↵
nwillers\\Anaconda3\\lib\\site-packages\\win32', 'C:\\Users\\↵
nwillers\\Anaconda3\\lib\\site-packages\\win32\\lib', 'C:\\Users\\↵
nwillers\\Anaconda3\\lib\\site-packages\\Pythonwin', 'C:\\Users\\↵
nwillers\\Anaconda3\\lib\\site-packages\\IPython\\extensions', 'C:\\↵
Users\\nwillers\\.ipython']
```

1.6.6 Executing a Python script from any directory

To execute a python script from anywhere on your system, use the PATH environmental variable.

On Linux:

Add your script path to the system's PATH environmental variable:

`export PATH=$PATH:/home/user/path/to/python/script.py`

Add the following line as the first line (shebang line) of the script to tell Linux which interpreter to use:

`#!/usr/bin/env python`

Set up the script as an executable:

`chmod x /home/user/path/to/python/script.py+`

`https://stackoverflow.com/questions/19917492/how-to-use-pythonpath`[59]

On Linux: If the file is created in Windows, the line ending is not Linux friendly. Windows uses carriage return and line feed `/r/n`, whereas Linux uses just a line feed `/n`. This means that if the file is created on Windows and executed on Linux the operating system is searching to execute the file `python/r`, which is very different from the intended `python`. I tried converting the file from Windows to Linux format by using the `fromdos` command, but that did not fix the line ending problem. Opening and closing the file in `nano` did not help either. The only remedy was to (1) open the file in Linux (e.g. in `nano`), move to the top of the file, (2) manually type in the whole first line followed by `Enter` a few times, making very sure that a new Linux line ending is introduced, and (3) then remove the old line with the bad line ending.

On Windows:

Add the path to your script to your user or system PATH environmental variable. The Rapid Environment Editor[60] tool makes it very easy. Once the path is set up, you can just double click on the script or run it in a command window.

1.7 Template project creation

https://medium.com/@rrfd/cookiecutter-data-science-organize-your-projects-atom-and-jupyter-2be7862f487e

cookiecutter seems to lead the pack:

https://cookiecutter.readthedocs.io/en/latest/

A command-line utility that creates projects from cookiecutters (project templates), e.g. creating a Python package project from a Python package project template.

`conda install -c conda-forge cookiecutter`

https://cookiecutter.readthedocs.io/en/latest/first_steps.html

https://github.com/mikeckennedy/cookiecutter-course

https://medium.com/worldsensing-techblog/project-templates-and-cookiecutter-6d8f99a06374

Some templates: https://cookiecutter.readthedocs.io/en/latest/readme.html http://cookiecutter-templates.sebastia

https://nsls-ii.github.io/scientific-python-cookiecutter/ – nice overview

https://drivendata.github.io/cookiecutter-data-science/

https://github.com/mandeep/cookiecutter-pyqt5

https://github.com/selimb/cookiecutter-latex-article

https://docs.microsoft.com/en-us/visualstudio/python/using-python-cookiecutter-templates https://docs.microsoft
us/visualstudio/python/quickstart-04-python-in-visual-studio-project-from-cookiecutter

1.8 Python and module versions, and dates

```
try:
    import pyradi.ryutils as ryutils
    print(ryutils.VersionInformation('matplotlib,numpy,pyradi,scipy,↵
       pandas'))
except:
    print("pyradi.ryutils not found")
```

```
Software versions
Python:    3.8.3 64bit [MSC v.1916 64 bit (AMD64)]
IPython:    7.21.0
OS:    Windows 10 10.0.19041 SP0
matplotlib:    3.3.4
numpy:    1.18.5
pyradi:    1.1.4
scipy:    1.6.1
pandas:    1.2.3
Mon Aug 16 20:08:24 2021 South Africa Standard Time
```

BIBLIOGRAPHY

[1] [Online]. Available: https://github.com/NelisW/ComputationalRadiometry#computational-optical-radiometry-with-pyradi

[2] [Online]. Available: https://github.com/NelisW/ComputationalRadiometry#computational-optical-radiometry-with-pyradi

[3] [Online]. Available: #installpyradi

[4] [Online]. Available: #linkstolocallibs

[5] [Online]. Available: #condaenvironents

[6] [Online]. Available: #jupyterinultipleenvironments

[7] [Online]. Available: https://wiki.python.org/moin/PythonEditors

[8] [Online]. Available: http://www.sublimetext.com/

[9] [Online]. Available: http://notepad-plus-plus.org/download/

[10] [Online]. Available: https://atom.io/

[11] [Online]. Available: https://wiki.python.org/moin/Vim

[12] [Online]. Available: http://pythonhosted.org/spyder/

[13] [Online]. Available: https://www.jetbrains.com/pycharm/

[14] [Online]. Available: https://pytools.codeplex.com/

[15] [Online]. Available: http://sourceforge.net/projects/pydev

[16] [Online]. Available: https://www.anaconda.com/products/individual

[17] [Online]. Available: http://conda.pydata.org/docs/using/envs.html

[18] [Online]. Available: https://docs.conda.io/en/latest/miniconda.html

[19] [Online]. Available: https://www.continuum.io/downloads

[20] [Online]. Available: https://www.continuum.io/blog/developer/advanced-features-conda-part-1

[21] [Online]. Available: http://conda.pydata.org/docs/config.html

[22] [Online]. Available: https://groups.google.com/a/continuum.io/forum/#!msg/anaconda/WQBn1c-spQ4/HVa1v83zAlEJ

[23] [Online]. Available: http://stackoverflow.com/questions/30492623/using-both-python-2-x-and-python-3-x-in-ipython-notebook

[24] [Online]. Available: https://ipython.readthedocs.io/en/stable/install/kernel_install.html

[25] [Online]. Available: http://conda.pydata.org/miniconda.html

[26] [Online]. Available: https://www.continuum.io/downloads

[27] [Online]. Available: http://conda.pydata.org/miniconda.html

[28] [Online]. Available: https://www.continuum.io/downloads

[29] [Online]. Available: http://conda.pydata.org/docs/using/envs.html

[30] [Online]. Available: http://conda.pydata.org/docs/using/envs.html

[31] [Online]. Available: https://ipython.readthedocs.io/en/stable/install/kernel_install.html#kernels-for-different-environments

[32] [Online]. Available: http://conda.pydata.org/docs/using/envs.html

[33] [Online]. Available: https://docs.python.org/2/library/site.html

[34] [Online]. Available: http://bob.ippoli.to/archives/2005/02/06/using-pth-files-for-python-development/

[35] [Online]. Available: https://grahamwideman.wikispaces.com/Python-+site-package+dirs+and+.pth+files

[36] [Online]. Available: http://stackoverflow.com/questions/15208615/using-pth-files

[37] [Online]. Available: https://github.com/NelisW/pyradi

[38] [Online]. Available: http://code.enthought.com/projects/mayavi/

[39] [Online]. Available: http://www.lfd.uci.edu/~gohlke/pythonlibs/#ets

[40] [Online]. Available: https://store.continuum.io/cshop/anaconda/

[41] [Online]. Available: https://store.continuum.io/cshop/anaconda/

[42] [Online]. Available: http://www.lfd.uci.edu/~gohlke/pythonlibs/#ets

[43] [Online]. Available: http://www.lfd.uci.edu/~gohlke/pythonlibs/#scikits-image

[44] [Online]. Available: https://docs.python.org/2/library/site.html

[45] [Online]. Available: https://docs.python.org/2/library/site.html#site.USER_SITE

[46] [Online]. Available: https://docs.python.org/2/install/

[47] [Online]. Available: https://stackoverflow.com/questions/15208615/using-pth-files

[48] [Online]. Available: http://bob.ippoli.to/archives/2005/02/06/using-pth-files-for-python-development/

[49] [Online]. Available: https://groups.google.com/a/continuum.io/forum/#!topic/anaconda/RO_h0wD7sKE

[50] [Online]. Available: https://stackoverflow.com/questions/10693706/creating-a-secondary-site-packages-directory-and-loading-packages-from-pth-fil

[51] [Online]. Available: https://stackoverflow.com/questions/19914527/using-python-pth-files-assuming-no-access-to-site-packages-directory

[52] [Online]. Available: https://docs.python.org/2/library/site.html#site.addsitedir

[53] [Online]. Available: https://support.enthought.com/entries/23665767-How-do-I-set-PYTHONPATH-and-other-environment-variables-for-Canopy-

[54] [Online]. Available: https://stackoverflow.com/questions/19917492/how-to-use-pythonpath

[55] [Online]. Available: https://stackoverflow.com/questions/4580101/python-add-pythonpath-during-command-line-module-run

[56] [Online]. Available: http://www.stereoplex.com/blog/understanding-imports-and-pythonpath

[57] [Online]. Available: https://stackoverflow.com/questions/1489599/how-do-i-find-out-my-python-path-using-python

[58] [Online]. Available: http://users-cs.au.dk/chili/PBI/pythonpath.html

[59] [Online]. Available: https://stackoverflow.com/questions/19917492/how-to-use-pythonpath

[60] [Online]. Available: http://www.rapidee.com/en/about