PyChimera Documentation

Release 0.2.7+3.g5227d1c

Jaime Rodríguez-Guerra Pedregal

Mar 28, 2019

User Guide

CHAPTER 1

Use UCSF Chimera packages in any Python 2.7 interpreter

With PyChimera you can...

- Run scripts depending on chimera **from CLI** with `pychimera script.py`.
- Enable `import chimera` in interactive coding sessions **outside UCSF Chimera**, including IPython and Jupyter Notebooks.
- Launch a standard UCSF Chimera instance, with the benefit of importing all your `conda` or `virtualenv` packages with `pychimera --gui`.

CHAPTER 2

Projects using PyChimera

- GaudiMM
- InsiliChem Tangram
- @jrjhealey's chimera scripts & bioinfo-tools

2.1 Quickstart

PyChimera lets you use the full UCSF Chimera codebase in any Python 2.7 project.

2.1.1 Why?

UCSF Chimera is an extensible molecular visualization tool with a vast collection of modules gathered after years of development. These tools allow you to perform serious molecular modelling jobs even without Python knowledge (they also offer a very versatile text interface with custom commands).

However, all this code is only available if you use the bundled Python 2.7 interpreter to run your scripts, like this:

```
chimera --nogui --script "path/to/my/script.py arg1 arg2 ..."
```

Or, using the GUI command line toolbar, by running `runscript path/to/my/script.py arg1 arg2 ...`. This is less desirable than simply running `python script.py arg1 arg2 ...`.

With PyChimera, you can run `pychimera script.py` and forget about using the UCSF Chimera interpreter or not. It just works. Additionally, it offers some more features:

- Interactive sessions in the Python interpreter where you can just run `import chimera`, like it should be.
- Full compatibility with IPython and Jupyter Notebooks (read *Extra packages*).
- Simple API to run your Chimera-dependent script with any `python` interpreter.
- Conda recipes for UCSF Chimera and UCSF Chimera headless, for automated testing and deployment.

If you want more details, be sure to check the paper (already submitted, once published, it will be linked here).

2.1.2 Usage

Run `pychimera -h` for quick help. Basically:

Running code

To execute a script:

```
pychimera script.py
```

To launch a module that uses UCSF Chimera internally:

```
pychimera -m this
```

To execute any Python statement:

```
pychimera -c 'import chimera'
```

To know which UCSF Chimera instance is being loaded:

```
pychimera --path
```

Interactive sessions

To start an **interactive** Python session with importable UCSF Chimera modules:

```
pychimera                  # start the standard Python interpreter
pychimera -i some_file.py  # run a script and stay in the standard Python interpreter
pychimera -im module       # same, but with a Python module
pychimera -ic "string"     # same, but with a command
pychimera ipython          # launch IPython interpreter
pychimera notebook         # launch IPython notebook
```

To launch the UCSF Chimera GUI with custom packages (check InsiliChem Plume as an example!):

```
pychimera --gui
```

2.1.3 Multiplatform compatibility

UCSF Chimera is available for Linux, Mac OS X and Windows. PyChimera does its best to provide the same compatibility for all three platforms and each release is continuously tested in Travis CI (Linux, Mac OS X) and AppVeyor (Windows). Despite our efforts, some features might not be as polished in some platforms. The table below summarizes the current state of implementation:

Features	Platforms		
	Linux	Mac OS X	Windows*
`pychimera`	Y	Y	N
`pychimera script.py`	Y	Y	Y
`pychimera ipython`	Y	Y	Y**
`pychimera notebook`	Y	Y	Y
`pychimera --gui`	Y	Y	Y
`pychimera -c`	Y	Y	Y
`pychimera -i`	Y	Y	Y
`pychimera -m`	Y	Y	Y

* In Windows, use `python -m pychimera`, not `pychimera`.

** Only with `qtconsole` installed.

PyChimera has been successfully installed and tested in the following 64-bit systems:

- **Linux**
  - Arch Linux with UCSF Chimera 1.10, 1.11, 1.12
  - Ubuntu 14.04 with UCSF Chimera 1.10, 1.11, 1.12 (Travis CI's)
  - CentOS 6.3 with UCSF Chimera 1.11.2
- **Mac OS X**
  - 10.11 El Capitan with UCSF Chimera 1.12
  - 10.10 (Xcode 6.4) with UCSF Chimera 1.12 (Travis CI's)
- **Windows**
  - 7 SP1 with UCSF Chimera 1.12
  - Windows Server 2012 R2 with UCSF Chimera 1.12 (AppVeyor's)

2.2 Installation

PyChimera is a wrapper around UCSF Chimera, so it makes sense that you have to install it beforehand. Additionally, to reduce compatibility problems, PyChimera is best used within a `conda` environment.

2.2.1 Recommended steps

1. Install Miniconda

Anaconda is a Python distribution that bundles all the scientific packages you will ever need. It also provides a fancy package manager to update and install more if you need it. A stripped-down version of Anaconda that only includes the package manager (`conda`) and the essential Python packages, called Miniconda, is also available and it's the one we will use here.

Go the Miniconda webpage and download the installer for your platform. For the sake of simplicity, choose the Python 2.7 version. You can use the 3.6 version if you really want, but you will need to create a Python 2.7 environment later.

Once you have downloaded the installer, run it. Depending on the platform, you have to do it differently.

Linux and Mac OS X

Just run `bash ~/Downloads/Miniconda*.sh` and follow the wizard. Make sure to answer yes when the installer asks about modifying your `.bashrc`. When you are done, close the terminal and reopen it to apply the changes.

Note: The installer requires `chimera` to be in `$PATH`. In Linux, you can symlink `$CHIMERA_INSTALL_DIR/bin/chimera` to any location of your `$PATH` (ie, `~/.local/bin`). In OSX, make sure to use the binary located in `/Applications/Chimera.app/Contents/Resources/bin/chimera`. There are alternative locations in the same `Chimera.app` directory that might not work!

Windows

Double-click on the `.exe` and follow the steps. Make sure to add Miniconda to your %PATH% when you are asked about it.

2. Install UCSF Chimera

If you haven't already, install UCSF Chimera. If you already have it installed, skip to step 3, but make sure you satisfy the requirements detailed in step 4.

PyChimera has been tested on UCSF Chimera 1.10 and above on Linux, Mac OS X and Windows. See *Multiplatform compatibility* for more details. Go to the UCSF Chimera download page and get the latest installer for your platform (1.12 at the time of writing). You probably want the 64bit version.

Once you have it in disk, run it. Depending on the platform, this is performed differently:

Linux

You need to run these commands in a terminal:

```
cd ~/Downloads  # or whatever it is in your system
chmod +x chimera*.bin
./chimera*.bin
```

At the final step, you will be asked to create a symlink in one of the places specified in your $PATH. Choose the one that refers to the Miniconda installation.

Windows

Just double click on the `.exe` and follow the steps of the installation wizard.

3. Install PyChimera

With `conda` and UCSF Chimera in your system, this step is really easy!

```
conda install -c insilichem pychimera
```

If you installed Miniconda 3.6 or want to use a separate environment, use these ones:

```
conda create -n pychimera -c insilichem python=2.7 pychimera
source activate pychimera  # Linux and Mac OS x
activate pychimera  # Windows only
```

4. Make sure `chimera` can be found

If you followed all the steps, you should be able to run `pychimera --path` and obtain the UCSF Chimera installation directory as a result. However, if you used a different installation path and did not symlink the binary to somewhere in your PATH, `pychimera` won't be able to locate it. There's a workaround for this! Just set an environment variable called `CHIMERADIR` pointing to the UCSF Chimera installation directory.

In Linux and Mac OS X, this can be done in your `.bashrc` or equivalent.

```
export CHIMERADIR="~/.local/UCSF-Chimera"  # point to the actual location in your␣
↪system
```

In Windows, you have to search "Environment variables" in the Start menu and create a new user environment variable in the popup dialog. Remember, the variable name is `CHIMERADIR` and the value should be something like `C:\Program Files\Chimera 1.12`.

2.2.2 Alternative procedure

If you don't want (or can't) use `conda`, you can also install PyChimera with `pip`:

```
pip install pychimera
```

or directly from source:

```
# With git
git clone https://github.com/insilichem/pychimera.git
cd pychimera
python setup.py install

# With wget
wget https://github.com/insilichem/pychimera/archive/master.zip
unzip pychimera*.zip
cd pychimera-master
python setup.py install
```

While this *should* work in an ideal environment, it would probably have some rough edges due to the libraries installed in your system being different than the ones provided by UCSF Chimera. The `pychimera` conda package has been finetuned to work with the correct versions so, if possible use that. Otherwise, refer to the conda recipe to identify the correct versions.

2.2.3 Extra packages

So far, you have a barebones `pychimera` installation. If you want to make use of all the Jupyter compatibility features, you will need to install some extra packages. Namely:

- IPython support: `ipython` (in Windows, `qtconsole` is also required).
- Notebook support: `jupyter`, `notebook`.
- Interactive molecule depiction: `nglview`.

This is easily installed with conda:

```
## First, activate your environment if necessary
# source activate pychimera
## In Windows, it would be:
```

(continues on next page)

(continued from previous page)

```
# activate pychimera
conda install ipython jupyter notebook
## In Windows, you will also need:
conda install qtconsole
## For interactive visualization in the notebook:
conda install -c bioconda nglview
## might need:
# jupyter-nbextension enable nglview --py --sys-prefix
```

2.3 FAQ & Known issues

2.3.1 Numpy problems

UCSF Chimera bundles its own distribution of some popular packages, like `numpy`, and those are loaded before your env packages for compatibility reasons. Be warned if you use specific versions for your project, because you can face strange bugs if you don't take this into account.

For example:

```
RuntimeError: module compiled against API version 9 but this version of numpy is 7
```

In some platforms (Linux), this can be worked around with some work on the precendence of paths in `sys.path`, but in some of them is not as easy (OS X). The easiest and most robust way to fix this is by upgrading UCSF Chimera's `numpy`:

```
pip install --upgrade numpy -t "$(pychimera --path)/lib/python2.7/site-packages"
```

Take into account that this action will prevent outdated extensions from working again. As a far as we know, these are affected, but there might be more (please report it so we can update this list!):

- MMTK-dependent extensions: MD, Energy minimization
- AutoDock-dependent extensions: AutoDock Vina

If you use those often enough, you should have a separate, unmodified UCSF Chimera installation.

2.3.2 PyChimera GUI has ugly fonts

Anaconda-provided `tk` package is not built with truetype support (for several reasons; read here and here). Chimera does ship its own one (with correct font support), but since PyChimera loads its own Python interpreter, it ends up being replaces with `conda`'s one. All `conda` environments are created with `tk` by default, so if the fonts really bother you, you can uninstall it with `conda remove --force tk` and it will fallback to system's one if needed.

2.3.3 Setuptools problems

If you are using the development versions of `pychimera`, Chimera's `setuptools` will complain about the versioning scheme (ie, `pychimera==0.1.11+6.gc2e1fbb.dirty`). As before, the fix is to upgrade the package. You might have to remove it manually beforehand, though:

```
rm -r $(pychimera --path)/lib/python2.7/site-packages/setuptools-3.1*
pip install --upgrade setuptools -t "$(pychimera --path)/lib/python2.7/site-packages"
```

2.3.4 Chimera reports problems with libgfxinfo.so and pcrecpp

The error traceback ends with:

```
libgfxinfo.so: undefined symbol: _ZN7pcrecpp2RE4InitERKSsPKNS_10RE_OptionsE
```

This is due to an incompatibility between Chimera's `pcre` libraries and those loaded by PyChimera. Depending on how you installed PyChimera, these will be:

- Installed with `conda` (or with `pip` but inside a `conda` environment): the libraries will correspond to the `pcre` package in the conda environment. To make sure it works, you would probably have to downgrade to version 8.39 with `conda install pcre=8.39`.
- Installed with `pip` (outside a conda environment): the loaded library will be the system's one. If you can afford to downgrade to version 8.39 system-wide, do it. You will probably not, so the best option is to create a `conda` environment to execute PyChimera properly: `conda create -n pychimera -c insilichem pychimera`.
- If nothing else works, you can try (as a last resort!) to remove the `pcre` package altogether. This can have some unexpected side effects in your environment, but if you only installed one of our projects (e.g. `gaudi`), you should be fine. Do it like this: `conda remove --force pcre`.

2.3.5 dateutil version

As Numpy, the `dateutil` version shipped with Chimera is old, which can cause problems with some modern libaries. The solution is the same: upgrade the package built in Chimera.

```
pip install -U -t $(pychimera --path)/lib/python2.7/site-packages python-dateutil
```

2.4 For developers

2.4.1 Quick API

PyChimera provides access to UCSF Chimera's modules from any Python 2.x interpreter. This is achieved in two steps:

1. `patch_environ()` patches environment variables with proper paths (packages and libraries). Since the original `sys.path` is exported to `PYTHONPATH`, you can use all your virtualenv/conda packages with Chimera. This call restarts Python to inject a new `os.environ` with `os.execve`.
2. `enable_chimera()` initializes Chimera. This is done through their own routines (`chimeraInit`).

As a result, if you want to use PyChimera in your developments, you only need to execute these lines at the beginning of the script. For example, PyChimera is used programmatically in the GaudiMM CLI entry point.

```
import pychimera
pychimera.patch_environ()
pychimera.enable_chimera()
```

Calling `patch_environ()` will result in the interpreter being restarted to inject all UCSF Chimera libraries; take that into account in the logic of your program. This is why you should probably add the lines at the very beginning of the script.

Alternatively, you can leave those lines out and have your users execute the script with `pychimera` instead of `python`. Up to you, but usually you will prefer to hide the technical details...

2.4.2 Alternative methods

PyChimera also offers its interface through python `-m`. This has not been thoroughly tested, so it may not work perfectly. Add `-i` for interactive mode:

```
python -[i]m pychimera [-m another_module | -c "string" | script.py | ipython |␣
↪notebook]
```

You can also try to launch it from IPython, but, again, some things may not work. Anyway, these two commands have the same effect:

```
pychimera ipython [notebook]
ipython -m pychimera [notebook]
```

If you want to run a script with IPython and then inspect the results (`-i` flag), your best bet is to run `pychimera ipython` and then call `%run path/to/file.py` inside the interpreter.

2.4.3 How does it work?

When you run `patch_environ`, we try to locate a valid UCSF Chimera installation in the system. This is performed with three alternative strategies:

1. Check if a `CHIMERADIR` variable is set. This is normally set by the user when the automated strategies can't work right due to the system configuration. If the path is valid, use that as the UCSF Chimera installation directory. Else, try strategy #2.
2. Check if an executable called `chimera` is somewhere in `PATH`. This is done with `disutils.spawn.find_executable`. If successful, figure out the UCSF Chimera installation directory from the file path after resolving any possible symlinks.
3. If `chimera` is not in `PATH`, we can try to find the installation directory in the default locations (`~/.local` or `/opt` for Linux, `/Applications` for Mac OS X, `C:\Program Files` for Windows).

Once we have located a valid UCSF Chimera, we find the needed libraries and Python modules to patch `LD_LIBRARY_PATH`, `PYTHONPATH` and other environment variables, as specified in their own shell launcher (Linux/OSX) and cpp launcher (Windows). In this step, any additional packages and libraries installed in a `conda` environment or `virtualenv` are also injected. For all this to work, the interpreter is restarted.

After the restart, `enable_chimera` is called, which runs the UCSF Chimera initialization routines contained in `chimeraInit.py`. Depending on the CLI options, we then run a script, run IPython/Notebook or start the GUI.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

Acknowledgments

Largely based on ideas by Greg Couch at chimera-users.

CHAPTER 5

Citation

PyChimera is scientific software, funded by public research grants (Spanish MINECO's project CTQ2014-54071-P, Generalitat de Catalunya's project 2014SGR989 and research grant 2017FI_B2_00168, COST Action CM1306). If you make use of PyChimera in scientific publications, please cite it. It will help measure the impact of our research and secure future funding!

```
@article{pychimera2018,
  author = {Rodríguez-Guerra Pedregal, Jaime and Maréchal, Jean-Didier},
  title = {PyChimera: use UCSF Chimera modules in any Python 2.7 project},
  journal = {Bioinformatics},
  volume = {34},
  number = {10},
  pages = {1784-1785},
  year = {2018},
  doi = {10.1093/bioinformatics/bty021},
  URL = {http://dx.doi.org/10.1093/bioinformatics/bty021},
  eprint = {/oup/backfile/content_public/journal/bioinformatics/34/10/10.1093_
↪bioinformatics_bty021/1/bty021.pdf}
}
```