1 Use UCSF Chimera packages in any Python 2.7 interpreter
2 Projects using PyChimera
   2.1 Quickstart
   2.2 Installation
   2.3 FAQ & Known issues
   2.4 For developers
3 Indices and tables
4 Acknowledgments
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 1. Use UCSF Chimera packages in any Python 2.7 interpreter
Projects using PyChimera

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 
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:

<table>
<thead>
<tr>
<th>Features</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linux</td>
</tr>
<tr>
<td>pychimera</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera script.py</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera ipython</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera notebook</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera --gui</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera -c</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera -i</td>
<td>Y</td>
</tr>
<tr>
<td>pychimera -m</td>
<td>Y</td>
</tr>
</tbody>
</table>
* 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 to the [Miniconda](https://docsMINICONDA) 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.

   **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:
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:

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

In some platforms (Linux), this can be worked around with some work on the precedence 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:

```bash
pip install --upgrade numpy -t `pychimera --path`/lib/python2.7/site-packages
```

Take into account that this action will prevent MMTK-dependent extensions from working again. If you use those often enough, you should have a separate, unmodified UCSF Chimera installation.
2.3.2 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.gc2efbb.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
```

2.4 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.