Draft version February 14, 2018
Typeset using LaTeX **preprint** style in AASTeX61

PYTHON: A COOKBOOK

Knox S. Long, Christian Knigge, Nick Higginbottom, Stuart Sim and
James Matthews, Sam Mangham, Mandy Hewitt, Ed Parkinson

(Dated: February 14, 2018)

Abstract

Python is a multi-dimensional Monte Carlo ionization and radiative transfer code that simulates the passage of photons through supersonic astrophysical flows. It can therefore provide self-consistent estimates for both the ionization states and the spectral signatures of such flows. Python has been designed particularly with *outflows* in mind, and several flexible, parameterized models of stellar and accretion disk winds are built into the code. Python has already been used to calculate synthetic spectra for a wide range of wind-driving astrophysical systems, including accreting white dwarfs, young stellar objects and active galactic nuclei. This *Cookbook* is intended to quickly bring potential users to the point where they can use python for their own applications. We therefore provide only brief introductions to the code and its built-in wind models. However, we describe fully how to obtain and install python, provide step-by-step instructions for designing and running different types of models, and also show how the results of such runs can be examined.

Contents

1. INTRODUCTION

PYTHON is a program which we have developed to simulate the spectra of disk systems with winds, originally cataclysmic variables and YSOs, but more recently AGN.[1] Although our focus was on situations where multiple dimensions are important, the code is also capable of modeling the spectra of spherical systems, such as stars.

The program was initially described by Long & Knigge (2002). Significant improvements to the code have been made since then, and these are described by Sim, Drew, & Long (2005) and by Higginbottom et al. (2013, 2014). The code can be run either in a single processor or in a multiprocessor environment.

The purpose of this cookbook describe how to use PYTHON in sufficient detail that users can install PYTHON on their machines and begin to explore what PYTHON can do. Many details are omitted. At some point, we expect to write a complete description of the program for now, but for now our various refereed publications will have to suffice.

The basic concept of PYTHON is (a) to define a set of sources of radiation, a star, a disk, etc and the kinematic parameters of a wind, density, velocity, etc. whose ionization structure needs to be calculated, (b) to then calculate the ionization conditions in the wind following photons packets generated over a wide wavelength range through the wind, and (c) once the ionization conditions are established, to calculate the emergent spectrum of the wind in specific directions over a narrower wavelength range. The primary output of PYTHON is therefore a spectrum of a source viewed at a specific inclination angle (and in some cases orbital phase), although we also provide information about the calculated ionization and temperature state of the wind.

PYTHON is hosted on github at https://github.com/agnwinds/python. The installed version of PYTHON contains three main executables "py" which carries out the MC radiative transfer calculation, and two helper routines, "py_wind" and "windsave2table" which provide ways to examine the ionization structure of the wind.

PYTHON remains a program in development. Some of these development efforts are exposed as options in the input files. Over time, we will try to weed out the less useful of these options, and fully test the ones we think are useful.

We are interested in feedback on your experience with the program. If you encounter problems, either in the installation or use of PYTHON, please contact us by sending email to long@stsci.edu, or, where appropriate use the issues page on the github sight.

2. INSTALLATION

PYTHON is written in C and should run on most linux or Mac systems. Python is hosted as a project on Github.com at https://github.com/agnwinds/python. PYTHON can be compiled with gcc (clang on Macs), icc and with mpicc. It uses the GNU Scientific Libraries (gsl). If mpicc is already installed on your machine, it will be compiled, by default, with mpicc to enable multiprocessing. If not, it will be installed with your single processor compiler.

After PYTHON is installed, and environment variables set up, there will be one executable, `py`, which carries out the radiative transfer calculation and two helper executables `py_wind` and

[1] The name PYTHON was adopted before the Python programming language became as popular as it is today.

windsave2table},which allow one to inspect the structure of the wind. All of the executables are located in a $PYTHON/bin.

2.1. *Quiek Install*

The simplest way to install python is to issue the following commands:

```
$ git clone https://github.com/agnwinds/python.git
$ cd python
$ git clone https://github.com/agnwinds/data data
```

Create an environment variable PYTHON for the main python directory, and add $PYTHON/bin to your path.

```
export PYTHON=/path/to/python/
export PATH = $PATH:$PYTHON/bin
```

or for the csh and variants

```
setenv PYTHON /path/to/python/
setenv PATH "${PATH}:${PYTHON}"
```

And then install the software with the following instructions

```
$ cd $PYTHON
$ ./configure
$ make install
$ make clean
```

At this point, one can proceed to verifying the installation as discussed in Section 2.2. Alternatives and more explanation of what the installation steps are described in the next few sections.

2.2. *Verifying your installation*

Once you have added PYTHON to your path, you should be set to run PYTHON.

To verify this, create a test directory. And then, in this directory execute the following commands.

```
$ cp $PYTHON/examples/regress/cv_standard.pf .
$ Setup_Py_Dir
$ py cv_standard
```

The code will now run a very simple cataclysmic variable model!

If you want to test the installation in multiprocessor mode with 3 processors, `py cv_standard` would be replaced by `mpirun -np 3 py cv_standard`.

2.2.1. *Updating the code and options for recompiling*

As stated previously, the installed PYTHON directories contain two git repositories. The main repository can be updated from any part of the directory structure, except the data directory. The secondary repository containing the atomic data and various other data files must be updated from within the data directory.

Currently, our primary branch for the main repository is 'dev' and our primary branch for the data repository is 'master'. To update both of these repositories one can issue the following commands:

```
cd $PYTHON
git pull origin dev
cd data
git pull origin master
```

In practice, data is changed less often than the main repository. At this point the code base is up-to-date, but the source code needs to recompiled. This can be accomplished with the following commands:

```
cd $PYTHON/source
make clean
make python
make py_wind
make windsave2table
```

By default these "make" commands will attempt to use mpicc for compilation if mpicc is available. If one prefers a different compiler the make commands have several command line switches, e.g

```
make D CC=gcc py
```

where D compiles the code for use with a debugger such as gdb and CC allows one to specify a what type of c compiler to use.

Note that two copies of the executables are always created and placed in $PYTHON/bin directory. One of the two copies is simply the executable name, py for the radiative transfer routine, and one with the version number attached, e.g py82 for version 82 of the program. This is intended to make it easy to compare results for various versions of the code.

3. BASICS OF A PYTHON CALCULATION

As shown in Fig. 1, PYTHON is carried out in three phases, consisting of (a) a data gathering phase in which the astrophysical system to be modeled is defined and the atomic data are read in to the program, (b) calculation of the ionization state of the wind in a series of ionization cycles, and (c) calculation of a detailed spectrum for the object over the wavelength range of interest for comparison to data.

3.1. *Specification of the model geometry*

PYTHON carries out radiation transfer through a wind in a spherical, cylindrical or polar grid. All wind geometries are axially symmetric A disk, if it exists is in the xy plane. The pole of the system is along the z axis. All models are symmetric with respect to the xy plane.

To define the model geometry, one must define one or more radiation sources and one or more wind regions.

The radiation sources typically are a central object, that is a star or a black hole, and/or a disk. For a binary system, a secondary is defined but it only serves to absorb photons that enter its Roche lobe. Each radiation source has to be completely defined both in terms of its size and the type of radiation it emits, e.g a blackbody of a certain temperature or as a power law or as a stellar photosphere with certain temperature and gravity. Disks can be given an arbitrary run of temperature with radius, but more usually are defined in terms of a standarad Shakua-Sunyaev disk and so need a central object mass, an inner radius, and a mass accretion rate.

Photons, or more properly photon packets, are (initially) generated by the radiation sources and then traverse the wind region or regions. Our terminology for a wind region is a "domain". Each domain occupies a region of space, where the density, velocity and atomic abundances are specified. Each domain has its own coordinate grid. PYTHON contains a number of kinematic models defining the density and velocity, but one can read in a wind model as well, as long as it maps on one of the coordinate systems PYTHON understands. The wind regions need not occupy all of space, so for example, one can define a wind region that is a simple biconical flow arising from a limited region of the disk. A model consisting of two domains might involve a fast flow originating from the inner disk, and a distinct slower flow originating from further out in the disk

3.2. *Ionization cycles*

After the the data gathering phase, and before one simulate spectra for a given model geometry, one must determine the ionization state of the wind.

PYTHON begins by assuming that the plasma has a constant temperature, and that ion abundances are set by the Saha equation. Photon packets representing the spectrum of each of the radiation

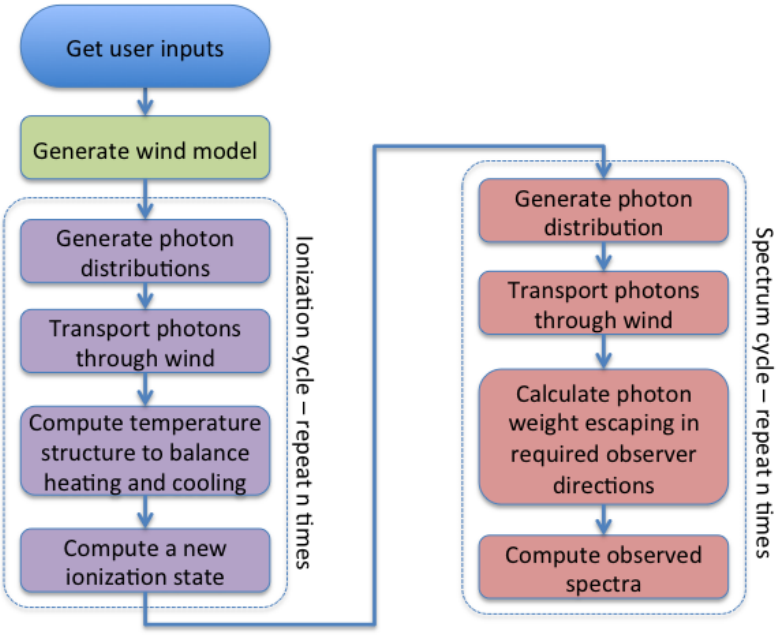

Figure 1. Code flow

sources (over a very large frequency range) in the system are generated and these packets are tracked throughout the cells of the wind. As they pass through these cells and are scattered or absorbed, quantities, such as the number of photoionization of each ion in each cell, required to make a better estimate of the ionization structure are captured. Then at the end of an ionization cycle the ionization structure and temperature of cells in the wind recalculated. This changes the opacity of the wind for future cycles.

This process is repeated multiple times (as determined by the user) with the goal of reaching a point where the wind ionization structure reaches a stable state and heating and cooling are balanced throughout the wind. PYTHON tracks the fraction of cells that are stable (converged). How the quality of the result, depends on the (patience of the) user, since s/he determines the number of cycles and the number of photon packets generated for each cycle.

3.3. *Spectral Cycles*

Once the ionization cycles are complete, next step is to generate simulated spectra (in specifc wavelength ranges and in the case of a disk system, at specific inclination angles.) To accomplish this, the ionization structure of the wind is fixed and a new set of photon packets are generated from each radiation source, but now over a narrow range that exceeds by only a little the range of the desired spectrum. The wavelength range has to exceed the spectral range of the desired spectrum somewhat to allow for Doppler shifts due to scattering in the wind.

Note that the term spectral cycles has a quite different meaning than ionization cycle in the context of PYTHON. In a spectral cycle, all of the photon packets count in the generation of the final spectra.

ksl believes that the section above should have several figures, one showing a typical wind geometry, one showing how the ionization structure of some ion evolves in a model calculation, and some spectra.

4. RUNNING PYTHON

If you have installed PYTHONand checked verified it as discussed above, then you have already run your first PYTHON model. The simplest way to explore PYTHON is to follow the procedure described for verification, Section 2.2, but with one of the other input, or ".pf" files in the examples directory. The input files can e modified with a simple text editor.

The executable for running the last compiled version of python is "py"[2]. When running in single processor mode, one runs PYTHON with the command:

```
$ py model.pf
```

or, equivalently,

```
$ py model
```

or, if running in multiprocessor mode

```
$ mpirun -n x py model
```

[2] If you look in the bin directory, particularly after you have been using PYTHON for awhile and updated from git, you will see the various versions of PYTHONthat have been compiled. To run a specific compiled version of python, simply replace "py" with one of these, e.g."py82d" for version 82d

where x is the number of parallel threads you would like to run or, if running in multiprocessor mode

```
$ mpirun -n x py model
```

where x is the number of parallel threads you would like to run

PYTHON has a number of command line options that can be used to control various aspects of how a program is executed. A summary of these options can be obtained by issuing `python -h`.

5. INPUTS

PYTHON is intended as a general purpose program for calculating radiative transfer through winds in spherical or axially-symmetric systems and as such each model run involves specifying a large number of variables. The required inputs are specified in a parameter, or `.pf` file. The parameter files consist of a series of lines containing keywords and values which define values. For example,

```
Central_object.mass(msol)     52.5
```

implies that the mass of the star (actually more generically the mass of the central object in the model) is 52.5 $M_\odot$.

PYTHON can be run using an old or modification of an old parameter file, or PYTHON can generate a parameter file "on the fly", by querying the user about what parameter values he or she would like to use. Modification of an existing ".pf" file can be carried out in one's favorite editor. If one tries to use an old parameter file, but the current version of PYTHON needs more information, PYTHON will drop into an interactive mode for that particular parameter.

PYTHON also uses atomic data and, in some cases, tabulated spectra for radiation sources that are (usually contained) in the d`$PYTHON/data` directory. The setup of the atomic data files is described in Section A. It is possible to create custom sets of atomic data, but here we assume you will use one of the sets of atomic data we have constructed, and limit our discussion of input files to the parameter, or `.pf` file. At the end of the input process, PYTHON writes a new ".pf" file to disk, to serve as the basis for the next time this particular model is run.

The recommended way to set up a completely model in python is first to set up the ".pf" file with the following command:

```
py --dry-run my_new_model.pf
```

which will execute the portions of the code that generates the .pf file, and then exit, so one can inspect the "pf" file for reasonableness. PYTHON will suggest an answer to most questions; one can except this answer with a carriage return. Initially, one will want to specify a small small number of ionization and spectrum cycles to see that everything is working as expected.

Next one the program interactively in single processor mode, e.g.

```
py my_new_model.pf
```

Once satisfied, one would modify some of the parameters, the number of cycles, the number of photons per cycle, the dimension of the wind, etc., in the .pf file for a more serious, likely multiprocessor run.

If the .pf file already exists, then PYTHON will use, or try to use that file. Normally, PYTHON will simply run to completion in this case. If for any reason (usually because the underlying source code has changed, PYTHON expects a parameter which is not in the .pf file, then PYTHON will drop into interactive mode for that particular keyword. It will attempt to use as much of the previous .pf file as possible. Once the input process is complete, PYTHON will write out a new parameter file, called whatever.out.pf. The original file, whatever.pf, will remain unchanged.

All this means that once one has set up a basic model, one can generally take an existing parameter file `model.pf`, copy it to a file with a new name, e.g. `new_model.pf`, edit a few of the parameters in it, and run a second model.

6. OUTPUT FILES

PYTHON produces a variety of output files in the directory in which the program is run, and additional diagnostic files in a subdirectory created for holding diagnostic information. The files all have the same root names as the name of the parameter file, and extensions that indicate the file type:

- model.spec: An ascii file containing the detailed spectra of the system at specific inclination angles created during the spectral cycles.
- model.spec_tot: An ascii file containing spectra of photon bundles that escaped the system during the last ionization cycle over the entire wavelength region for which photons were created.
- model.log_spec_tot Identical to the model.spec_tot file except that the spectra have binned logarithmically
- model.wind_save A binary file that contains the structure of the wind, including ionization fractions

All of the ascii files produced by PYTHON begin with a series of comment lines that contain a copy of the parameter file that was used to create the spectrum as well as a line giving the version number of the code.

The first two columns of the real data are the frequency and wavelength of the spectral bins, and the remaining columns provide spectra. Columns three through seven are diagnostic. They represent the angle averaged spectra which have escaped from the system (or hit a surface) as it would have been observed at a distance of 100 pc. Subsequent columns in the `model.spec` files are the spectra at specific inclination angles (and phase angles of the system). So for example, a column labeled A45P0.50 would be the spectrum for a system at 45° inclination viewed when the secondary (if one exists) is behind the primary again at a distance of 100 pc. The fluxes presented are either F_λ or F_ν depending on what was specified by the keyword `spec.type`. The default is F_λ.

The `.wind_save` is, as noted, a binary file that contains all of the cell dependent quantities used by PYTHON to calculate the spectra. These include the basic physical conditions calculated for each cell T_e, T_r, ion and electron densities, velocities, etc. One can use the program `py_wind` to display many of these variables and to write them to ascii or fits files.

Still need to describe the diagnostic files and also the wind file, referencing the latter to the PY_WIND discussion later.

6.1. *Convergence and Diagnostics*

PYTHON logs a considerable amount of information about its progress in a set of diagnostic files which are placed in a directory `diag_model`, where model is the rootname of the parameter file. The primary files have names like verb model_x.diag, where x refers to diagnostics from thread x. In single processor mode the file will be named `model_0.diag`. These files contain a running history of the program.

ksl: We probably need a section on determing whether a model has converged, and whether there are an exorbitant nubmer of errors.

6.2. *Reading and Analyzing Outputs*

Shipped with the install we provide a number of scripts for analysing the data from python runs. These scripts are provided in the `py_progs/` directory, and are documented on the github wiki: https://github.com/agnwinds/python/wiki/reading-and-processing-outputs. Individual function documentation is produced via standard docstrings and "pydoc", and can be found in html format in `py_progs/html_docs/`.

7. USING PY_WIND OR WINDSAVE2TABLE TO INSPECT THE PROPERTIES OF THE WIND

PYTHON generates a binary file which contains many of the quantities used in the generation of spectra. These include physical quantities such as electron and ion densities, radiation and electron temperatures, and a great deal of diagnostic information, including which cells converged and which did not, and the number of photon bundles that passed through each cell. These quantities can be accessed through the routine PY_WIND or WINDSAVE2TABLE.

WINDSAVE2TABLE is a hardwired routine that prints out a fixed set of ascii files that summarize information about the model and certain ions the wind, currently Carbon, Nitrogen, Oxygen and Fe. It also prints out a table whatever.master.txt which contains basic information like velocities, electron densities, and temperatures in each cell of the wind. There is one file of each type for each wind domain.

Domains need to be explained

PY_WIND is an interactive routine (though it has an option to print out a fixed set of files. It allows one to print information about any ion, as well as information like densities and temperatures. Almost, any variable int the structures that are sued in PYTHON to describe the wind are available, or can be made availabe using PY_WIND.

A snippet of one of the files generated by PY_WIND follows. Note that we again adopt Astropy conventions for comment headings. This choice is intended to make it straightforward to read the files as Astropy tables, and then to create plots of the various portions various variables, using, for example, magplotlib.

```
# TITLE= "sv_macro.ionC4.dat"
# Coord_Sys CYLIND
x          z          var    inwind   i    j
1.4000e+09 3.5000e+08 0.00e+00  -2    0    0
1.4000e+09 8.0805e+08 0.00e+00  -1    0    1
1.4000e+09 1.0575e+09 0.00e+00  -1    0    2
```

```
1.4000e+09 1.3840e+09 0.00e+00  -1   0   3
1.4000e+09 1.8113e+09 0.00e+00  -1   0   4
1.4000e+09 2.3705e+09 0.00e+00  -1   0   5
```

The non-commented section consists of columns listing the position of the cell in physical space, the value of the parameter, a flag indicating that the cell is actually in the wind, and the cell number in x and z. Cells in the wind have a value of 1 in column 4.

One can also generate a file called model.complete, which summarises most of the important quantities in the wind, such as ion fractions, electron densities and so on. This is best utilized using the python scripts contained in py_progs.

8. REFERENCES

REFERENCES

Long, K. S., & Knigge, C. 2002, ApJ, 579, 725
Higginbottom, N., Knigge, C., Long, K. S., Sim, S. A., & Matthews, J. H. 2013, MNRAS, 436, 1390
Higginbottom, N., Proga, D., Knigge, C., et al. 2014, ApJ, 789, 19
Proga, D., Stone, J. M., & Drew, J. E. 1999, MNRAS, 310, 476 /
Sim, S. A., Drew, J. E., & Long, K. S. 2005, MNRAS, 363, 615

APPENDIX

A. ATOMIC DATA

There are various sources of information about atomic data including in some of our papers. These include James' description on github. The scripts used to generate the data files themselves can be found here.

The atomic data used by PYTHON is conventionally contained in the `data` directory. All of the data is in ascii format and so can in principle be modified by the user. All of the atomic data is read in by the routine `get_atomic_data.c`.

A complete description of the atomic data goes beyond the scope of this CookBook, but the following basics information may be helpful. The `get_atomic_data` routine first reads a master file which the names of all of the atomic data files. It then reads each of these files in turn (as if they were just one long file).

Various 'masterfiles' exist in the data directory including a series of files with names like "standard79" which access the detailed data files in use when the developers were at Version 79 of PYTHON. Those beginning with "h10" or "h20" have H or H and He set up as macro-atoms. Here, as an example if the masterfile called standard79

```
#This is the standard set of data files created for python79 - auger ionization data include
data/atomic79/elem_ions_ver_73.py
data/atomic79/topbase_levels_h.py
data/atomic79/topbase_levels_he.py
data/atomic79/topbase_levels_cno.py
#data/atomic79/topbase_levels_fe.py
data/atomic79/levels_ver_2.py
data/atomic79/lines_linked_ver_2.py
data/atomic79/excited.py
data/atomic79/recomb.data
data/atomic79/topbase_h1_phot_extrap.py
data/atomic79/topbase_he1_phot_extrap.py
data/atomic79/topbase_he2_phot_extrap.py
data/atomic79/topbase_cno_phot_extrap.py
data/atomic79/topbase_fe_phot_extrap.py
#Next is VFKY PI cross sections, supplemented with VY data out to 50kev.
data/atomic79/vfky_outershell_tab.data
#Next is VY inner shell photoionization cross sections
data/atomic79/vy_innershell_tab.data
#Now we have electron yield data from Kaastra and Mewe, matching the inner shells
data/atomic79/kaastra_electron_yield.data
#Now we have fluorescent photon yield data from Kaastra and Mewe, matching the inner shells
data/atomic79/kaastra_fluorescent_yield.data
data/atomic79/coll.data
#Next is dielectronic recombination data from chianti.
data/atomic79/chianti_dr.dat
```

```
#Next is badnell style total radiative recombinaion rate
data/atomic79/chianti_rr.dat
#Next is badnell style ground state radiative recombinaion rate
data/atomic79/Badnell_GS_RR.txt
#Necxt if gaunt factor data from sutherland (1997)
data/atomic79/gffint.dat
#Next is direct ionization from dere
data/atomic79/di_dere.dat
```

The first of the detailed files (conventionally names something containing `elements_ions` identifies what elements and what ions of each element are to be included in the calculation along with the abundances of each element. The next few files give levels for various ions, some taken from Topbase, and others from Verner. The "lines_linked" file provides information about the transitions between the various levels that the doce uses. Photoionization and more detailed information about other atomic processes follow. The names of the files provides hints at the data sources.

The remaining files contain more detailed information about each ion, including energy levels and cross sections. The structure is hierarchical in the sense that if you have to include an ion first as an element for any of the ion information to be read in, and you have to include an ion for any of the level information to be read in. Thus, for example, if you have a set of atomic data files which include H, He, C, N, and O, and you want to calculate a model with just H, and He, all you need to to do is comment out (#) the other elements in the `elements_ions` file.

ksl: The alternative approach here would be to define standard choices for this e.g. standard73, and write a separate document describing the details of how to set up ones own database. We could indicate how to comment out lines JM: I think this is probably the right approach Knox. At the moment it's a little complex.