PYTHON: A Cookbook

Knox S. Long, Christian Knigge, Nick Higginbottom, James Matthews, & Stuart Sim

September 23, 2014

Abstract

PYTHON is a multi-dimensional Monte Carlo ionization and radiative transfer code 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 fora wide range of wind-driving astrophyical 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.

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, although these codes are typically much more specialized.

The program was intially 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 an be run either in a single processor or in a multiprocessor enviroment.

The purpose of this users guide is describe how to use PYTHON in sufficent detail that users can install PYTHON on their machines and begin to use PYTHON for their work. Many details are omitted, but we hope we have captured the most important information.

The basic concept of PYTHON is (a) to define a set of sources of radiation, a star, a disk, etc and a the kinematic parameters of a wind, density, velocity, etc, whose ionization structure needs to

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

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 narower wavelength range. The primary output of PYTHON is therefore a spectrum of a source viewed at a specific inclination andle (and in some cases orbital phase), although we also provide information about the calculated ionization and tempeature state 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 ???.

2. Installation

PYTHON is written in c and should run on most linux or Mac systems. It can be compiled with gcc (clash on Macs) and with mpicc. It uses the GNU Scientific Libraries (gsl). If mpicc is already installed on your machine, it will be compiled in mpicc. If not, it will be installed with with your single processor compiler.

Check that this is actually true. I have a suspicion that a change has to be made in the Makefile first.

After python is installed, and environment variables set up, there will be two primary executables, `py` and `py_wind` located in a `$PYTHON/bin`. The routine `py` will be used to calculate model spectra; `py` allows you to extract information about the physical conditions of the wind at the end of the ionization cycles.

2.1. Retrieving the Software

We use git for version control sofware, and Python is a project on Github.com. Most likely git will already installed on your machine, but if we recommend you install it[2]

The first step in the installation, is to set up the directory structure and retrieve the necessary makefiles. To do this, simply issue the following command:

```
$ git clone https://github.com/agnwinds/python.git -b structure
```

[2]**Provide information about git.** If is is not possible to install git on your machine, contact us.

This will create a top-level directory "python", and several subdirectories (so that you want to make sure you do not already have a directory called python at the location you wish to install PYTHON):

You then need to cd to the new directory and set your environment variables

```
$ export PYTHON=/path/to/python/
$ cd $PYTHON
$ make install
$ make clean            # this can take a while, cleaning gsl is a little laborious
```

These command lines need to be altered so that logs are maintained.

note that export syntax is for bash. For csh use

```
$ setenv PYTHON /path/to/python/
```

These commands and in particular the `make`install command will retrieve the remaining files needed by PYTHON, and then compile the gsl scientific libraries, Bill Pences's cfitsio package, and python itself.

At this point the PYTHON installation will have its final, fairly simple, directory structure.

- progs: location of source code for various fully debugged version of the code (Note that the progs directory is created as part of the make process.)
- data: location for all datafiles. Files that are mainly for reference should be gzipped to save space. Such files are not recreated in
- bin: The location of the executables. (It is a good idea to put this directory in your path)
- software: This directory contains libraries which are used in in python that must be recompiled when creating an installation on a new machine, primarily Bill Pence's cfitsio package and the GNU scientific library gsl
- example: A directory with a few examples of python runs. (Note that the input files will have changed and so one may not be able to run these examples without some changes in the input files.)and put the code, the atomic data

140901 ksl: When I tried a raw installthis is not quite the directory structure that was produced

2.2. Setting up environment variables

The only environment variables that is needed is to define PYTHON. For bash or sh users simply place the following:

```
export PYTHON=/path/to/python/
```

where `$HOME}`is you would modify `$HOME}`to read something else if you are not installing putting this in your top level directory. You also need to add `$PYTHON/bin}`to your path, e.g::

```
PATH = $PATH:$PYTHON/bin
```

for c shell, use the following commands

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

2.3. Verifying your installation

Once you have added PYTHON to your path, copy the script `Setup_Py_Dir}` to `$PYTHON/bin/},`e.g.

140901 ksl: When I tried a raw install it looked like to me that this next command was unneeded because the setup command was already in bin

```
$ cp $PYTHON/py_progs/setup_scripts/Setup_Py_Dir $PYTHON/bin
```

To verify this, create a test directory. And then directory execute the following commands. in that directory go to a directory where you want to do a run with parameter file root.pf

```
$ cp $PYTHON/Examples/sv.pf .
$ Setup_Py_Dir
$ py sv.pf
```

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

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

ksl: We do not have this set up yet and so these steps do not work, but this is what I think we should do. I'm unsure James, how to modify the directory structure on Python or to put a file there

3. Basic code structure: ionization cycles vs spectral cycles

As shown in Fig. 1, PYTHON is carried out in three phases, consiting 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 serios of ionization cycles, and (c) calculation of a detailed spectrum for the object over the wavelength range of interest for comparison to data.

After the the data gathering phase, and initial determination of the ionization conditionsin the plasma are calculated assuming a constant temperature defined as one of the input variables. Photon packets are then generated from all of the radiation sources in the system and these packets are tracked throughout the cells of the wind. As they pass through these cells and are scattered or absorbed, the quantities reqired to make a better estimate of the ionization structure are captured. Then at the end of an ionization cycle the ionization structure is recalculated, changing the opacity of the wind for future cycles. The cycle is repeated multiple times (as determined by the user), and to a point where (hopefully) the wind ionization structure is no longer changing since heating and coolling are balanced throughout the wind. The fraction of cells that are stable (converged) is tracked, but the nunber of ioniztion cycles is user defined.

Once the ionization cycles are complete, the ioniztion structure of the wind is fixed and the detailed spectrum is calculated, using photons packets generated within a speciefied and some times very narrow range.

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 photons packets count in the generation of the final spectra.

4. Inputs & Outputs

4.1. Inputs

PYTHON obtains data for models from two sources, a parameter file, which conists of a series of lines contain keywords and values which define values and large sets of atomic data which is contained in the `$PYTHON/data` directory. There are various sets of atomic data which one can use depending on the type of model one is trying to run, and it is fairly easy for the expert to create his/her own set of data files, or modify an existing one, but we will defer a discussion of how to modity the existing files to another document. 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.

The usual way to run python is with the command:

```
$ py model.pf
```

or

```
$ py model
```

In either case, PYTHON looks for the file `model.pf`. If this file is absent from the directory from which PYTHON is being run, then the user will be queried for all the parameters which will define the model and the run, before the code calculates the model and the simulated spectra for the model.

A typical query from PYTHONto the user will look like:

```
System_type(0=star,1=binary,2=agn) (0) :
```

where in this case, `System_type` is the keyword, the wording in firset of parentheses is a hint about possible replies, and the value in the second set of parenthesis is the current default response.

To accept the default, simply hit a carriage return. To subsitute a new value, enter it (2 if you would like to calculate a model for an gan). (To get the progrm to exit immediatly so you can have a think, hit Ctl-D.)

In the case where the program is set up interactively, python will write a file `model.pf` that contains all of the parameters once they have been entered. On the other hand, if `model.pf` exists, then PYTHON will simply run the model. This means that 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 paremeters in it, and run a second model.

As a result, the recommened way to set up a model in python is to run it interactively, with a small number of ionization and spectrum cycles to see that everything is working as expected, and then to modify some of the paramers in the file for a more serious run.

The syntax for the lines in the parameter file are very simple. For example,

```
mstar(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 $M_{\odot}$.

In editing, the parameter files produces by PYTHON a few facts about the way PYTHON handles parameters files are worthwhile to remember:

- Order matters. PYTHON reads the lines in the parameter files only once, and expects them to be in a certain order. If there is a line that PYTHON does not expect, PYTHON will simply skip that line and look for the desired keyword in the next line. If the keyword is never found, PYTHON will switch to interactive mode, but then you will have to answer all of the questions from that point on.
- Keywords can appear multiple times. Thus for example if you ask for spectra at 4 inclination angles, there will be a line in the parameter file for each incliantion angle
- The information is parenthesis in the first word is advisory information.
- A change you make in one line can cause new keywords to appear. A simple example of this is the situation in which you had run a model of a star with a stellar wind. PYTHON calculates blackbody models internally, but if you want to use a stellar atmosphere model for the star, PYTHON will ask for the list of files that contain the spectra calculated for a stellar atmosphere.

4.2. 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 detiled 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 wavelenght 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 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 specta 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 quantiies 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 diagnositc files and also the wind file, referencing the latter to the PY_WIND discussion later.

5. Running PYTHON

The easiest way to become familiar with PYTHON is to run a few simple models. This should give you a sense of how PYTHON works, and how to interact with it.

5.1. A spherical wind model for a star

A simple test model that can be run with PYTHON in a short time is a spherical wind for a star succh as Zeta Pup. PYTHON does not have alll of the physics of modern models for stars, but such a model is useful for seeing how much the missing physics and atomic data affects python.

The first line of the input file simply allows you to select the what type of model, in this case a spherical system, with a central object and a spherically outflowing wind

```
Wind_type(0=SV,1=Sphere,2=Previous,3=Proga,4=Corona,5=knigge,6=homologous,7=yso,8=elvis,9=shel
```

The next four lines descibe what set of atomic data one wishes to use for the calcuation, how many photon packets are included in each ionization or spectral cycle, the number of ionization cycles and the number of spectral cycles.

```
Atomic_data     atomic/standard73
photons_per_cycle       100000
Ionization_cycles       20
spectrum_cycles 20
```

The next two lines establish what type of coordiante system we want to calculate the spectrum in, and the number of cells in the calculation. Had we chosen a cylindrical or spherical polar coordinate system, we would hae been asked for the dimension in each of the two directions.

```
Coord.system(0=spherical,1=cylindrical,2=spherical_polar,3=cyl_var)     0
Wind.dim.in.x_or_r.direction    30
```

The next line defines which of the various ways to calculate the ionization will be used.

```
Wind_ionization(0=on.the.spot,1=LTE,2=fixed,3=recalc_bb,6=pairwise_bb,7=pairwise_pow)   3
```

The next two lines describe what line transfer mode we want to use, and whether we want to include adiabatic expansion as a cooling term in our calculation. The first three line transfer mode options are for diagnostic purposes, and our atomic data contains data for macro atoms, so one would normally choose option 3 for the radiative transfer mode

SWM140923 - Will these diagnostic purposes be described? They're potentially useful for building up an understanding of a model you've made.

```
Line_transfer(0=pure.abs,1=pure.scat,2=sing.scat,3=escape.prob,6=macro_atoms,7=macro_atoms+anis
Thermal_balance_options(0=everything.on,1=no.adiabatic)                         0

System_type(0=star,1=binary,2=agn)                        0
Star_radiation(y=1)     1
Disk_radiation(y=1)     0
Boundary_layer_radiation(y=1)   0
Wind_radiation(y=1)     1
Rad_type_for_star(0=bb,1=models)_to_make_wind   0
Model_file                       kurucz91/kurucz91.ls

mstar(msol)     52.5
rstar(cm)       1.32e12
tstar   42000

disk.type(0=no.disk,1=standard.flat.disk,2=vertically.extended.disk)    0
Torus(0=no,1=yes)                                       0
wind.radmax(cm) 1e13
wind.t.init     40000

stellar_wind_mdot(msol/yr)      5.1e-6
```

```
stellar.wind.radmin(cm) 1.32e12
stellar.wind_vbase(cm)  2e+07
stellar.wind.v_infinity(cm)     2.25e+08
stellar.wind.acceleration_exponent      1
```

At this point, essentially all of the parameters for the ionization cycle have been specified and we move on to defining the parameters for the spectral cycles.

```
Rad_type_for_star(0=bb,1=models,2=uniform)_in_final_spectrum    0
Model_file                    kurucz91/kurucz91.ls
spectrum_wavemin        850
spectrum_wavemax        1850
no_observers    1
angle(0=pole)   45
```

The next line is an example of a line in the parameter file that is really not used in calculation.

```
phase(0=inferior_conjunction)   0.5
```

The next few lines are primarily for diagnositc purposes. Normally one chooses the defaults.

```
live.or.die(0).or.extract(anything_else)        1
Select_specific_no_of_scatters_in_spectra(y/n)  n
Select_photons_by_position(y/n) n
spec.type(flambda(1),fnu(2),basic(other)        1
Extra.diagnostics(0=no) 0
Use.standard.care.factors(1=yes)                    1
```

PYTHONincorpoates stratified sampling to reduce the number of photon bundles required in the ionization cycle. If the source to have spectra that resemble and ensemble of blackbodies, one should choose the cv option.

```
Photon.sampling.approach(0=T,1=(f1,f2),2=cv,3=yso,4=user_defined,5=cloudy_test,6=wide,7=AGN,8=
```

5.2. A biconical disk wind model for an accreting white dwarf

5.3. A biconical disk wind model for a quasar

5.4. An externally generated hydrodynamic model of a line-driven quasar disk wind

6. Using PY_WIND 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 hsyicsal quanties 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 quanties can be accessed through the routine PY_WIND.

A snippet of one of the files generated by PY_WIND follows

```
# TITLE= "sv_macro.ionC4.dat"
# Coord_Sys CYLIND
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 postion 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.

7. Wind models

PYTHON comes with three built-in ”kinematic” (parameterized) outflow models, as well as with the facility to read in externally provided flow models. The type of wind model that will be assumed by the code is specified by the `Wind\_type` parameter. Each of the example runs described in the previous section used a different wind model and thus provides a practical use case for that particular model. In this section, we provide brief, but complete descriptions of the wind models and define all of the model parameters that can be specified by the user.

PY_WIND simply reads and then displays portions of the wind file created by python. It can select various parameters from the wind and it can write them to files so thatthe variables can be plotted.

The normal mode of running PY_WIND is to run it interactively. As you run it interactively the variables you select are displayed ont the screen. The variables that you display can also be written to files (depending on the answer to the question Make_files)

The commands that were executed in the interactive will be stored in py_wind.pf (if you end with a "q", and not an EOF response to the choice question) EOF terminates the program at that point before the command file is written to py_wind.pf

The py_wind.pf file is useful if you want to run the exact same set of commands on another windfile. You should rename py_wind.pf to something.else.pf and run PY_WIND on that data set using the -p something.else.pf option

The command line switches -d and -s are intended to produce a standard set of output files sufficient for many purposes.

7.1. Spherical stellar winds

The simplest outflow model provided in PYTHON describes a purely radial wind that emanates from a sphere of radius R_{min} (`Wind_type=1`) When modelling stellar winds, the continuum source in the model will usually be a star of radius R_*, so if the wind is driven continuously from the surface of this star, R_{min} should be set to R_* (the default value for R_{min} in PYTHON). However, a value of $R_{min} > R_*$ may be useful for modelling expanding spherical *shells* associated with outflows that are no longer active.

Following Castor & Lamers (1979), the wind speed at distance r from the origin is taken to be

$$v(r) = v_0 + (v_\infty - v_0)(1 - R_*/r)^\beta, \tag{1}$$

where v_0 sets the initial wind speed, v_∞ is the terminal wind speed, and the power-law index β defines the *shape* of the velocity law (Figure 2). If we denote the sound speed and escape velocity at R_* by c_s and v_{esc}, respectively,the winds of early-type stars are reasonably well described by $v_0 \simeq c_s$, $v_\infty \simeq (1-3) \times v_{esc}$, and $\beta \simeq 0.5-4$. In physical units, $v_0 \sim \mathrm{afew} \times 10\mathrm{kms}^{-1}$ and $v_\infty \sim \mathrm{afew} \times 1000\mathrm{kms}^{-1}$ for such stars. Note that $\beta = 0.5$ approximates the classic "CAK" model for line-driven stellar winds (Castor, Abbott & Klein 1975).

The only other parameter required to fully specify a spherical wind model is the mass-loss rate into the wind, $\dot{M}_w$. The density at any point in the outflow, $\rho(r)$ can then be calculated from the continuity equation, $\dot{M}_w = 4\pi r^2 \rho(r) v(r)$.

The correspondence between physical model parameters and code input variables is:

$\dot{M}_w$	$\hat{=}$	`stellar_wind_mdot(msol/yr);`
R_{min}	$\hat{=}$	`stellar_wind_radmin(cm);`
v_0	$\hat{=}$	`stellar_wind_vbase(cm/s);`
v_∞	$\hat{=}$	`stellar_wind_v_infinity(cm/s);`
β	$\hat{=}$	`stellar_wind_acceleration_exponent`.

7.2. A simple split monopole disk wind (Knigge, Woods & Drew 1995)

The second type of outflow geometry built into PYTHON is the "split monopole" disk wind model introduced by Knigge, Woods & Drew (1995, KWD hereafter). This model describes a rotating, biconical outflow driven from the surface of an accretion disk. It should therefore only be used in models that actually include such a disk (`disk.type=1 or 2`).

The KWD wind geometry is illustrated in Figure **??**. The outflow is assumed to emanate from the entire surface of the disk, with all poloidal (non-rotational) streamlines above/below the disk plane converging onto a single point located at a distance d below/above the center of the disk. The parameter d controls the overall geometry of the outflow. More specifically, the inner and outer opening angles of the wind are given by $\tan theta_{min} = R_*/d$ and $\tan theta_{max} = R_d/d$, where R_* and R_d are the inner and outer radius of the disk. It can be shown that the range of opening angles, $\Delta\theta = \theta_{max} - \theta_{min}$ is maximized for $d_0 = \sqrt{R_* R_d}$ (KWD). For $d >> d_0$, the outflow becomes a highly collimated jet, while for $d << d_0$, it approaches an equatorial flow along the disk plane. In the more common regime $d \sim d_0$, the model describes a moderately collimated bipolar disk wind.

The poloidal velocity field in the outflow is assumed to follow a velocity law of the form

$$v_p(l) = c_s(R_0) + [f v_{esc}(R_0) - c_s(R_0) \left(1 - \frac{R_s}{l + R_s},\right)^\alpha \qquad (2)$$

where l is the distance measured along a poloidal streamline, $c_s(R) \simeq 10\sqrt{T_{eff,d}(R))/10^4\mathrm{K}}$ is the sound-speed in the disk photosphere at a cylindrical distance $R_0 = R(l = 0)$ from the center, and $v_{esc}(R_0)$ is the local escape velocity from the same location in the disk. The parameter f sets the terminal poloidal velocity of the outflow along a given streamline, in units of the escape velocity at the launch point of the streamline on the disk surface. The parameters R_s and β define the shape of the velocity law. Roughly speaking, R_s sets the length scale over which the outflow achieves a significant fraction of its terminal velocity, while β controls the way in which the poloidal velocity increases towards R_s. For $\beta \lesssim 1$, the wind speed increases more gradually from $l = 0$ to $l = R_s$, while for $\beta \gtrsim 1$, the wind speed stays low until closer to$l \sim R_s$ and then increases sharply (Figure **??**. However, note that it is really the *combination* of the two parameters that defines the shape of the velocity law. For example, v/v_∞ at $l = R_s$ is still a function of β in this prescription.

The velocity field of the outflow also includes a rotational component. More specifically, wind material is assumed to emerge from the disk plane with the local Keplerian rotational velocity, i.e. $v_\phi(R_0) = v_{Kep}(R_0)$, where R is the cylindrical radius from the disk center. As the outflow rises above the disk, it is assumed to conserve specific angular momentum, so that the rotational velocity field at a distance l along a particular streamline is defined by $R(l)v_{\phi}(l) = R_0 v_{Kep}(l = 0)$.

In the KWD model, the wind always emerges from the entire area of the disk. However, the model does allow for a radial dependence of the *specific* mass-loss rate, $\dot{m}_w(R)$, i.e. the mass-loss rate per unit area across the disk. Specifically, $\dot{m}_w$ is assumed to track the local effective temperature of the disk, so that

$$\dot{m}_w \propto T_{eff}(R)^{4\lambda}. \quad (3)$$

The constant of proportionality in this relation is fixed by integrating $\dot{w}$ across both sides of the disk and demanding that the result matches the user-specified total mass-loss rate, $\dot{M}_w$.

The correspondence between physical model parameters and code input variables is:

$\dot{M}_w$	$\hat{=}$	`wind_mdot(msol/yr)`;
d d_{min}	$\hat{=}$	`kn_dratio`;
f	$\hat{=}$	`kn_v_infinity`;
R_s	$\hat{=}$	`kn_r_scale`;
α	$\hat{=}$	`kn_alpha`.
λ	$\hat{=}$	`kn_lambda`.

7.3. A flexible biconical disk wind (Shlosman & Vitello 1993)

The third kinematic model built into PYTHON is biconical disk wind model described by Shlosman & Vitello (1993, SV hereafter). This outflow parameterization is more flexible than the KWD one, at the cost of extra free parameters.

In the SV prescription, the outflow emanates from the disk between R_{min} and R_{max}. The poloidal streamlines at these radii emerge at angles θ_{min} and θ_{max} relative to the vertical axis. Inbetween R_{min} and R_{max}, the ejection angle θ is assumed to vary linearly with radius R.

7.4. Externally generated wind models

8. Continuum Sources

ck: we rather forgot about this when we discussed this, Knox.... we definitely need a section on continuum sources – i.e. disk, star, BL, X-ray source. Not 100% clear if this should come before or after the wind descriptions. Also not sure if "continuum source" is still the best term for that – it was when all that we were looking at were BBs, but for SAs, perhaps that's confusing? Maybe "external radiation sources"? "Photon sources"? Sth else?

9. Physics

9.1. Ionization Modes

9.2. Radiative (Line) Transfer Modes

jm: I've added an atomic data section below. This should describe file name formats, actual data formats, sources, and possibly also any relevant physics.

10. Atomic Data

10.1. Lines

10.2. Levels

10.3. Photoionization Cross-sections

10.3.1. TopBase data

File format: topbase_xx_phot.py.

We use Topbase photoionization cross-sections where possible, with the following format for simple-ions:

```
PhotTopS  1  1 200    1    13.605698  100
PhotTop    13.605698 6.304e-18
PhotTop    16.627193 3.679e-18
```

The PhotTopS entry describes the ion and excitation state that the cross-section is associated with. The columns after the identifier are:

- Atomic charge, z
- Ionization stage
- ISLP number, describing the spin, angular momentum and parity of the state
- Level using TopBase convention
- Ionization potential, eV
- Number of entries in the cross-section.

The PhotTop entries then contain energy in eV and cross secition in cm^{-1} For macro-atoms, the format is similar:

```
PhotMacS        1       1       1       1       13.598430     100
PhotMac        13.598430   6.3039999e-18
PhotMac        13.942675   5.8969998e-18
```

Only this time the PhotMacS entries now have the following columns:

- Atomic charge, z
- Ionization stage
- Lower level, matched with level data
- Upper level, matched with level data
- Ionization potential, eV
- Number of entries in the cross-section.

Note that with macro atoms one requires the lower and upper level so that the different bound-free jumps are available in an excited macro-atom.

10.3.2. Verner / VFKY data

10.4. Collisional Data

10.5. Gaunt Factors

10.6. Dielectronic Recombination

11. 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

Sim, S. A., Drew, J. E., & Long, K. S. 2005, MNRAS, 363, 615

Visualizing wind properties: PYWIND

A. Notes on style, etc.

A.1. How to Leave Comments

Comments can be added to the margins of the document using the todo command, as shown in the example on the right. You can also add inline comments:

Here's a comment in the margin!

This is an inline comment.

This preprint was prepared with the AAS LaTeX macros v5.2.

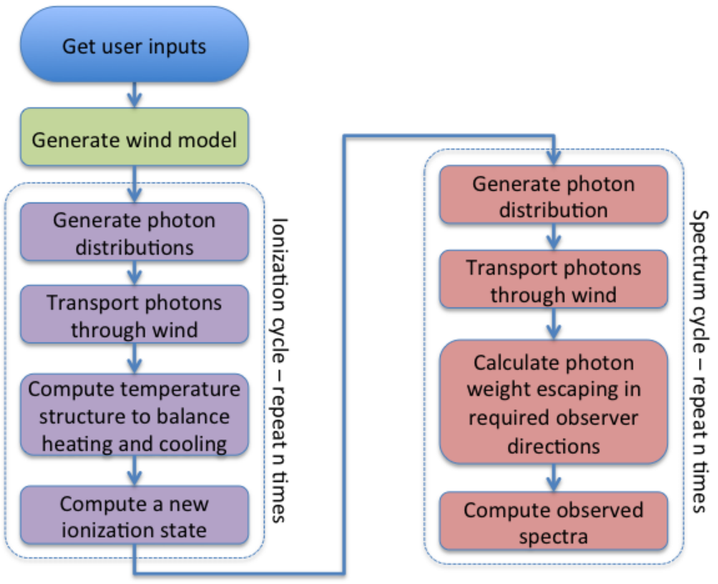
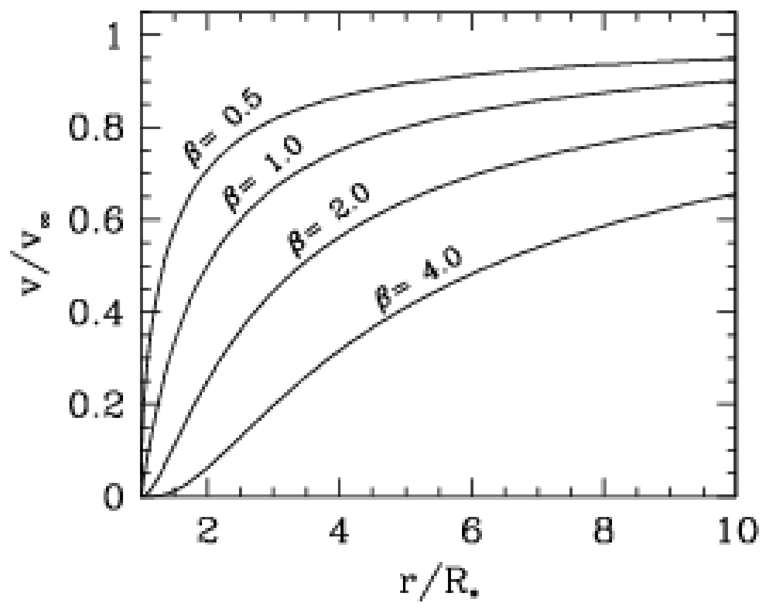
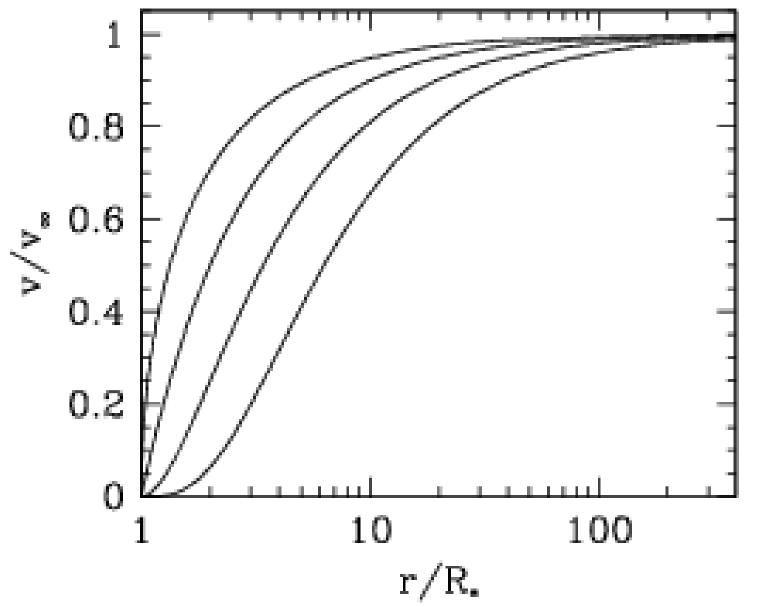
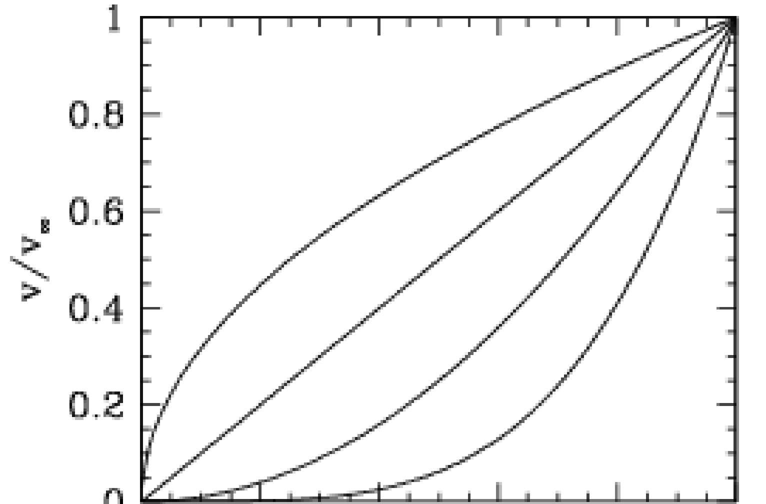

Get user inputs
Generate wind model
Generate photon distributions
Transport photons through wind
Compute temperature structure to balance heating and cooling
Compute a new ionization state
Ionization cycle – repeat n times
Generate photon distribution
Transport photons through wind
Calculate photon weight escaping in required observer directions
Compute observed spectra
Spectrum cycle – repeat n times

Fig. 1.— Code flow


1
0.8
0.6
0.4
0.2
0
v/v∞
β= 0.5
β= 1.0
β= 2.0
β= 4.0
2
4
6
8
10
r/R*


1
0.8
0.6
0.4
0.2
0
v/v∞
1
10
100
r/R*


1
0.8
0.6
0.4
0.2
0
v/v∞