RELAGN: Documentation

Scott Hagen
Email: scott.hagen@durham.ac.uk

Contents

1 RELAGN

We will start by describing the main model RELAGN. Throughout we assume you have either downloaded or cloned the GitHub repository, and we will work on the assumption that you have not changed the directory structure within the repository since downloading it.

This documentation is only meant as a guide on how to use the code. For a description of the model, please see the main paper (Hagen & Done, in prep). If this code us useful in your work, please cite: Input bibtex for paper here!!

1.1 Input Parameters

Here we give an overview of the parameters that define the model. We include a brief description, the units, and the defualt values (i.e what the code will use if you do not pass this parameter). For some parameters we also include limits. These are **not based off any physical argument** - but actual limits that will break the code if exceeded (for a variety of reasons). For an idea of sensible **physical** limits, see Hagen & Done, insert here for a description of the physics that go into the model.

M

Mass of central black hole
- *Units*: $M_\odot$
- *Default*: 10^8

D

Distance from the observer to the black hole
- *Units*: Mpc
- *Defualt*: 100
- *Limits*: $D > 0$ - Must be greater than 0 distance...

log_mdot

$\log \dot{m}$, Mass-accretion rate - Scaled by the Eddington mass accretion rate, such that $\dot{m} = \dot{M}/\dot{M}_{\rm Edd}$, where $\dot{M}$ is the physical mass accretion rate of the system (i.e unit mass per unit time) and $\dot{M}_{\rm Edd}$ is the Eddigton mass accretion rate. This is related to the Eddington luminosity through $L_{\rm Edd} = \eta \dot{M}_{\rm Edd} c^2$, where η is a black hole spin dependent efficiency factor, and c is the speed of light.
- *Units*: Dimensionless
- *Default*: -1

a

Black hole spin parameter. 0 Implies non-spinning, while 1 is maximally spinning with prograde rotation (i.e in the same direction as the accretion disc). Note that the code enforces an upper limit of 0.998, which is the theoretical maximum assuming the presence of a disc insert Thorne 1974 citation!
- *Units*: Dimensionless
- *Default*: 0
- *Limits* $0 \leq a \leq 0.998$ (Retrograde rotation not currently supported by the GR transfer functions we use)

cos_inc

$\cos(i)$, Cosine of the inclination of the observer with respect to the disc. This is measured from the z-axis, with the disc in the x-y plane (i.e $\cos(i) = 1$ would imply an observer located on the z-axis looking straight down onto the disc, while $\cos(i) = 0$ will imply an edge on view of the disc).
- *Units*: Dimensionless
- *Default*: 0.5
- *Limits*: $0.09 \leq \cos(i) \leq 1$ (Exactly edge on will give you a disc that is not visible...)

kTe_hot

$kT_{e,h}$, Electron temperature for the hot Comptonisation region (i.e the X-ray corona). This sets the high-energy roll-over of hot Comptonisation component in the spectrum.
- *Units*: keV
- *Default*: 100
- *Limits*: $0 < kT_{e,h}$ (Apart from being wildly unrealistic, 0 electron temperature will lead to segmentation faults)

kTe_warm

$kT_{e,w}$, Electron temperature for the warm Componisation region
- *Units*: keV

- *Default*: 0.2
- *Limits*: $0 < kT_{e,w}$ (Same reasoning as above!)

gamma_hot Γ_h, Spectral index for the hot Comptonisation component
- *Units*: Dimensionless
- *Default*: 1.7
- *Limits*: $1.1 \leq \Gamma_h$ (NTHCOMP will break for unrealistically steep spectra)

gamma_warm Γ_w, Spectral index for the warm Comptonisation component
- *Units*: Dimensionless
- *Default*: 2.7
- *Limits*: $1.1 \leq \Gamma_w$ (Same reasoning as above)

r_hot r_h, Outer radius of the hot Comptonisation region (X-ray corona). If this is negative, then the code will use the innermost stable circular orbit, $r_{\rm isco}$
- *Units*: Dimensionless gravitational units, $r = R/R_G$ where $R_G = GM/c^2$
- *Default*: 10

r_warm r_w, Outer radius of the warm Comptonisation region. If this is negative, then the code will use $r_{\rm isco}$.
- *Units*: R_G
- *Default*: 10

log_rout : $\log r_{\rm out}$, Outer disc radius. If this is negative, then the code will use the self-gravity radius from Insert Laor and Netzer ref here.
- *Units*: R_G
- *Default*: -1 (i.e self-gravity)

fcol $f_{\rm col}$, Colour-temperature correction. Note, that this is only applied to the standard disc region. If negative, then the code will use the relation given in insert Done et al 2012 reference here!!. Otherwise it is treated as a constant correction across the entire standard disc region, such that the black-body emission from each annulus is given by $B_\nu(f_{\rm col}T(r))/f_{\rm col}^4$, where $T(r)$ is the temperature at that the annulus and B_ν denotes the black-body emission.
- *Units*: Dimensionless
- *Default*: 1

h_max $h_{\rm max}$, Maximal scale-height of the hot X-ray corona. This is a tuning parameter, and only affects the seed photons from the disc intercepted by the corona. If $h_{\rm max} > r_h$, then the code will automatically switch to r_h as the maximal scale-height.
- *Units*: R_G
- *Default*: 10

z Redshift of the source (i.e the black hole) as seen by the observer. As all calculations are initially done in the frame of the black hole, this correction is only applied when transforming to an observed spectrum.

- *Units*: Dimensionless
- *Default*: 0

1.2 Running through XSPEC

If you wish to fit the model to observational data the easiest way is through XSPEC, as this will take into account telescope effective areas and responses. To this extent, we have written a bespoke XSPEC version of the model in FORTRAN. Before getting started though, there are a couple of steps to required by you in order to compile the model. Once the model has been compiled and loaded, you should refer to the XSPEC documentation (https://heasarc.gsfc.nasa.gov/xanadu/xspec/) for instructions (e.g fitting models to data, initiating MCMC, etc.).

1.2.1 Installation and Compilation

As the code is written for XSPEC it should also be compiled within XSPEC. To make this simple we have included a shell script, COMPILE_TO_XSPEC.SH, which executes the required commands for compilation. This is found within the main RELAGN directory and is excecuted by typing:

```
> sh compile_to_xspec.sh
```

while within the RELAGN main directory. Note that this will compile both RELAGN and RELQSO. What this does is execute the following commands (which you can type manually if you wish, instead of using the shell script):

```
> xspec
> initpackage relagn lmod_relagn.dat /Path/To/RELAGN/src/fortran_version/relagn_dir
```

where /Path/To is a place-holder for the directory path to RELAGN. The code should now be compiled (you should check the terminal for any big errors!). If the compilation was successful, then you do not need to repeat this step - ever! (unless you happen to delete or move the source code).

The next step is to load the compiled code as a local model. This is done from within XSPEC. So within your terminal, type:

```
> xspec
> lmod relagn /Path/To/RELAGN/src/fortran_version/relagn_dir
```

The model is now loaded, and you are good to go! Enjoy! (Note, that you will need to load it into XSPEC **every** time, unless you append to your XSPEC.RC file. More on that below. If you want more information regarding compiling and loading local models in XSPEC, taken a look at the XSPEC documentation (https://heasarc.gsfc.nasa.gov/xanadu/xspec/manual/XSappendixLocal.html)

Automatically loading RELAGN upon starting XSPEC :

If, like me, you do not want to have to type LMOD ETC... every time you wish to use the model in a new XSPEC session, you can modify your XSPEC.RC file. This file contains any commands you wish XSPEC to execute upon startup, and is located in the ∼/.XSPEC directory (I'm assuming you compiled HEASOFT using the source code, and following the instructions, and so this directory **should** exist within you home directory.).

Now, cd into the ∼/.XSPEC directory, and open the xspec.rc file. If this file doesn't exist, create one. Within the file, add the line:

```
lmod relagn /Path/To/RELAGN/src/fortran_version/relagn_dir
```

XSPEC will now automatically execute that command upon start-up. Don't want to have to type all that out yourself? No problemo, we've also included a shell script that will modify your xspec.rc file accordingly - so no typos! From within the RELAGN directory, simply type:

```
> sh init_autoLoad_xspec.sh
```

This will **append** the lmod line into your xspec.rc file (Note it will append for **both** RELAGN and RELQSO). Only run this if you want XSPEC to automatically load both models **every** time you start a new session!!! For more info on modifying XSPEC, take a look at their documentation (https://heasarc.gsfc.nasa.gov/xanadu/xspec/manual/node33.html)

1.3 Running through PYTHON

Occasionally you might not wish to run the model through XSPEC. For example, you could be using the model as a part of one of your own codes/models, or you simply don't enjoy using XSPEC for your data analysis. To this extent using the PYTHON version makes more sense (incidentally this was the original version of the model. The FROTRAN version only came into existence when I wanted an easier way of making it work with XSPEC...)

In PYTHON the RELAGN model exists as a class that you can initiate (by passing your desired input parameters), and then you choose what parts of the model you want to calculate/extract (e.g you can choose to skip the GR ray tracing calculations, or only calculate specific components of the SED, etc.). Below we give details of the RELAGN class attributes and methods. For examples on using the sc python version, see the Jupyter Notebooks in the /examples directory within the RELAGN repository.

1.3.1 RELAGN class attributes

risco Innermost stable circular orbit
- *Type*: float
- *Units*: R_G

r_sg Self-gravity radius
- *Type* float
- *Units*: R_G

eta η, efficiency
- *Type*: float
- *Units*: Dimensionless

Egrid Energy grid used for calculations (this is in the frame of the black hole)
- *Type*: numpy array
- *Units*: keV

nu_grid Frequency grid corresponding to Egrid
- *Type*: numpy array
- *Units*: Hz

wave_grid Wavelength grid corresponding to Egrid
- *Type*: numpy array
- *Units*: Angstrom, Å

E_obs Energy grid converted to the observers frame (i.e redshift corrected)
- *Type*: numpy array
- *Units*: keV

nu_obs Frequency grid converted to the observers frame
- *Type*: numpy array
- *Units*: Hz

wave_obs Wavelength grid converted to the observers frame
- *Type*: numpy array
- *Units*: Angstrom, Å

1.3.2 RELAGN class methods

get_totSED(*rel=True*)

Extracts (and calculates if necessary) the total SED (i.e disc component + warm component + hot component)

Parameters

rel : Bool, optional
Flag for whether to include full GR ray tracing
- *True* : Full GR is used
- *False* : GR ray tracing ignored (i.e SED in black hole frame)

The default is *True*

Returns

spec_tot : array
Total output spectrum in whatever units are set

(see **set_units()** method for a description on unit options)
Generally though, this will be some flavour of spectral density, in luminosity or flux (e.g erg/s/Hz, or W/m^2/Å, or ph/s/cm^2/keV)
If you are using the default, then output is in erg/s/Hz

get_discComponent(*rel=True*)

Extracts (and calculates if necessary) the disc component of the SED (i.e ONLY contribution from the standard disc region)

Parameters

---

rel : Bool, optional
Flag for whether to include full GR ray tracing
- *True* : Full GR is used
- *False* : GR ray tracing ignored (i.e SED in black hole frame)

The default is *True*

Returns

---

spec_disc : array
disc spectral components in whatever units are set
(see **set_units()** method for a description on unit options)
Generally though, this will be some flavour of spectral density, in luminosity or flux (e.g erg/s/Hz, or W/m^2/Å, or ph/s/cm^2/keV)
If you are using the default, then output is in erg/s/Hz

get_WarmComponent(*rel=True*)

Extracts (and calculates if necessary) the warm Comptonised component of the SED (i.e ONLY contribution from the warm Comptonisation region)

Parameters

---

rel : Bool, optional
Flag for whether to include full GR ray tracing
- *True* : Full GR is used
- *False* : GR ray tracing ignored (i.e SED in black hole frame)

The default is *True*

Returns

---

spec_warm : array

warm Compton spectral component in whatever units are set
(see **set_units()** method for a description on unit options)
Generally though, this will be some flavour of spectral density, in luminosity or flux
(e.g erg/s/Hz, or W/m^2/Å, or ph/s/cm^2/keV)
If you are using the default, then output is in erg/s/Hz

get_HotComponent(*rel=True*)

Extracts (and calculates if necessary) the hot Comptonised component of the SED (i.e ONLY contribution from the hot Comptonisation disc region)

Parameters

---

rel : Bool, optional
Flag for whether to include full GR ray tracing
- *True* : Full GR is used
- *False* : GR ray tracing ignored (i.e SED in black hole frame)

The default is *True*

Returns

---

spec_hot : array
Hot Compton spectral component in whatever units are set
(see **set_units()** method for a description on unit options)
Generally though, this will be some flavour of spectral density, in luminosity or flux
(e.g erg/s/Hz, or W/m^2/Å, or ph/s/cm^2/keV)
If you are using the default, then output is in erg/s/Hz

set_units(*new_unit='cgs'*)

Changes the output units when using the get methods

Parameters

---

new_unit : {'cgs', 'cgs_wave', 'SI', 'SI_wave', 'counts'}
String indicating what unit system to use. These will give the following outputs:
- **cgs**: Spectra: *ergs/s/Hz*, Luminosities: *ergs/s*, Accretion rate: *g/s*, Distances: *cm*
- **cgs_wave**: Spectra: *ergs/s/Å*, otherwise same as **cgs**.
- **SI**: Spectra: *W/Hz*, Luminosities: *W*, Accretion rate: *kg/s*, Distances: *m*
- **SI_wave**: Spectra *W/Å*, otherwise same as **SI**.
- **counts**: Spectra: *Photons/s/keV*, otherwise same as **cgs**

The default is **cgs**

Returns

None

Note that is you have set the output to be in flux instead of luminosities, then **cgs**, **cgs_wave**, and **counts** will give spectra and luminosities in cm^{-2}, while **SI** and **SI_wave** will give them in m^{-2}. E.g if flux and cgs set, the spectra will be given in *ergs/s/cm^2/Hz*
Also, you do not need to call this if you don't wish to change the output from the default units. (i.e if you're perfectly happy in cgs)

set_flux()

Sets all output to be in flux rather than luminosities. (e.g a spectrum in cgs units would be given in *ergs/s/cm^2/Hz*

Parameters

None

Returns

None

set_lum()

Sets all output to be in luminosoity (e.g a spectrum in cgs units would be given in *ergs/s/Hz*)
Note: This is the default output! So no need to call this **unless** you have already set flux output previously in your code and you now wish to change back

Parameters

None

Returns

None

get_Ledd()

Extracts the Eddington luminosity of the system in whatever units are set (e.g output in *ergs/s* or *W*).
Note, this is **always** given as a luminosity. So the code will not care whether or not **set_flux()** has been called.

Parameters

None

Returns

Ledd : float
The systems Eddington luminosity

get_Rg()

Extracts the length scale of a gravitational unit in whatever units are set (e.g output in *cm* or *m*)

Parameters

None

Returns

R_G : float
The size scale of a gravitational unit

get_Mdot()

Extracts the **physical** mass accretion rate of the system in whatever units are set (e.g output in *g/s* or *kg/s*)

Parameters

None

Returns

Mdot : float
Physical mass accretion rate of the system

2 RELQSO

2.1 Input Parameters

2.2 Running through XSPEC

2.2.1 Installation and Compilation

2.3 Running through PYTHON

2.3.1 Class methods