Monk: A general-relativistic radiative transfer code

Wenda Zhang (wdzhang@nao.cas.cn)
National Astronomical Observatories, Chinese Academy of Sciences

17-25/07/2021, X-ray Data Modelling of Accreting Black Holes

Outline

- What can Monk do
- How does Monk work
  - For details see Zhang, Dovciak, & Bursa 2019, ApJ, 875, 148
- Where to get Monk and how to run

- Note: all texts in red are bash shell commands!

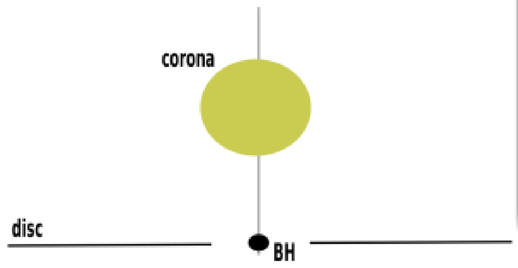
What can Monk do

- Calculate energy spectra and polarization from disc-corona systems in AGNs and BHXRBs
- Post-processing GRMHD simulations

Monk workflow

Procedures

1. sample disc photons: x^μ, k^μ, E_∞, δ, $K_{\rm WP}$
2. propagate x^μ, k^μ along null geodesic in Kerr spacetime; step size $\ll \bar{\lambda}$
3. if photon enters corona:
   - covariant evaluation of optical depth τ, then scattering probability $P = 1 - e^{-\tau}$;
   - if scattering:
     - sample electron four-momentum
     - scattering kernel follows Pozdnyakov+1983; Klein-Nishina cross section
     - update E_∞, k^μ, δ, f^μ, then $K_{\rm WP}$
4. at infinity:
   - E_∞
   - $x^\mu \rightarrow i_{\rm obs}$
   - $k^\mu, \delta, K_{\rm WP} \rightarrow Q, U$



Sampling seed photons

- We are actually sampling superphotons: each photon has a weight that is connected with the generation rate
- We divide the thin disc on the equatorial plane into N_r radial bins
- At each radial bin, we divide the half sky into $N_\vartheta * N_\phi$ pixels
- At each pixel, we
  - evaluate the photon generation rate:

$$\dot{N} = \frac{4\zeta(3)k_{\mathrm{B}}^3}{c^2h^3}\frac{f_{\mathrm{limb}}\cos\theta_{\mathrm{em}}\,d\mathcal{S}\,d\Omega\,T_{\mathrm{eff}}^3}{f_{\mathrm{col}}\,u^t},$$

Limb-darkening

Emission angle

area

Solid angle

Effective tem.

Color correction

Convert time from local frame to observer at infinity

  - Sample photon energy in the fluid rest frame: samples the Planck distribution with temperature of $f_{col}T_{eff}$
  - Calculate the photon wave vector $k^\mu = e^\mu_{(a)}k^{(a)}$
  - The redshift factor: $g = -1/k_\mu U^\mu$ → Four-velocity of the disc fluid

Raytracing photons in the Kerr spacetime

For a massless particle in the Boyer-Lindquist frame, we can separate r and $\mu = \cos\theta$:

$$I = s_r \int \frac{dr}{\sqrt{R(r)}} = s_\mu \int \frac{d\mu}{\sqrt{M(\mu)}},$$

Where: $$s_r,\ s_\mu = \pm 1$$

$$R(r) = r^4 + (a^2 - l^2 - Q)r^2 + 2[Q + (l - a)^2]r - a^2 Q,$$

$$M(\mu) = Q + (a^2 - l^2 - Q)\mu^2 - a^2\mu^4.$$

In monk, for each step we give I a small increment, and solve for r and μ.

Propagating polarization

- The polarization degree is a constant
- The polarization angle is related to a complex constant, the Walker-Penrose constant:

$$\begin{aligned}\kappa_{\rm wp} = (r - ia\cos\theta)\{(k^t f^r - k^r f^t) + a\sin^2\theta(k^r f^\phi - k^\phi f^r) \\ - i[(r^2 + a^2)(k^\phi f^\theta - k^\theta f^\phi) \\ - a(k^t f^\theta - k^\theta f^t)]\sin\theta\}. \qquad (14)\end{aligned}$$

Here f^μ is the polarization vector.

- We just need to evaluate the WP constant when sampling the seed photons, knowing the polarization angle.
- The polarization degree and the WP constant do not change while propagating in vacuum
- We solve the polarization angle upon interacting with material

Scattering

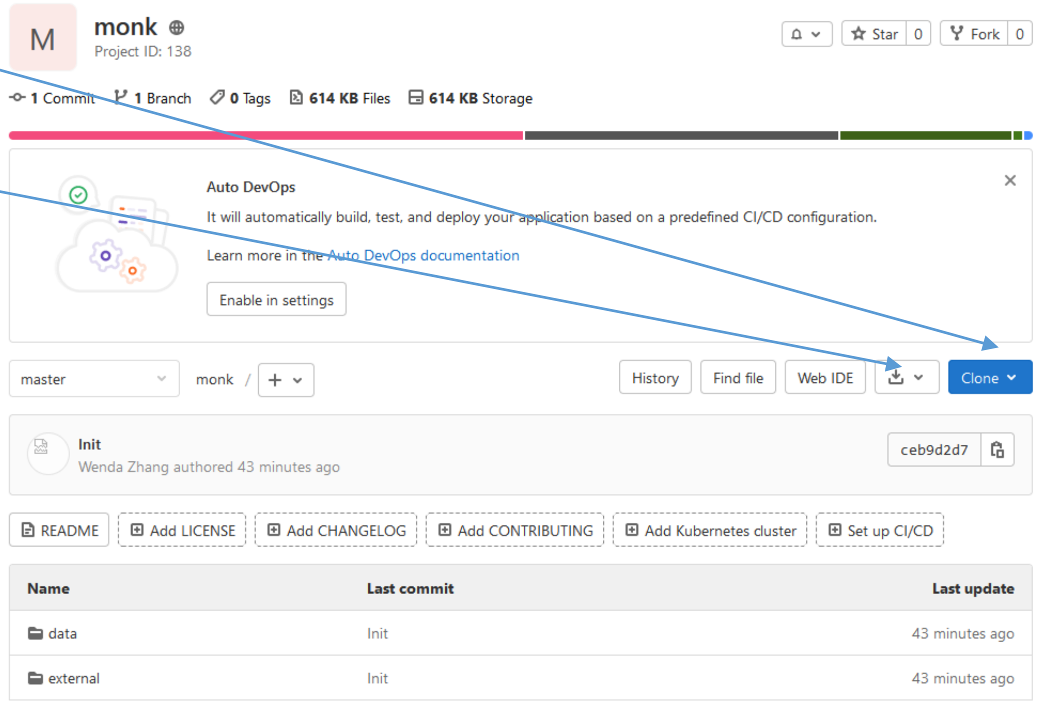
- Sec. 2.6 of Monk paper
- For each step, evaluate the optical depth and subsequently the scattering probability
- If scattered:
  - Sample the momentum of the scattering electron
  - Sample the energy and azimuthal angle of the scattered photon
  - Calculate the Stokes parameters of the scattered photons
  - Calculate the polarization degree and angle of the scattered photon, then the WP constant

Where to get Monk

- https://projects.asu.cas.cz/zhang/monk (probably you need an account to visit; contact Dr. Michal Bursa)
- Either git clone

  git clone https://projects.asu.cas.cz/zhang/monk.git

- Or download a compressed zip/tarball



Monk codes

- Written in C++
  - High efficiency
  - Object-oriented: easy to handle different coronal geometries
- Parallel computation enabled by openmpi
  - Highly scalable
- Prerequisites
  - Platform: linux, OSX
  - A C++ compiler that supports C++14 standard
    - For GCC: GCC 5.0 or later
  - libstdc++fs:
    - For GCC: GCC 5.3 or later
  - openmpi (haven't tested other MPI implementations but you are very welcome to do so)
  - GNU Make

Important files

- sim5 directory: the sim5 library by Dr. Michal Bursa
- scatter.cpp: Compton scattering
- geoinf.cpp: raytracing
- tridgeod.cpp: coronal geometries
- 3dcorona.cpp: calculates geodesics from disc to corona
- 3dcorona_mpi.cpp: does the radiative transfer
- calspec.cpp: calculates energy spectra from products of 3dcorona_mpi.cpp

Compilation & Installation

1. Compile the sim5 library:

   Change directory to $MONKDIR/sim5, then

   make

2. Re-assign the following variables in Makefile:
   - MONKDIR: the directory where you put monk
   - BINDIR: the directory to put temporary binary executable files
   - OBJDIR : the directory to put temporary object files
   - TRASHDIR: the directory to put trash
   - INSTALLDIR: the directory where you put executable files, e.g., ~/.local/bin or /usr/local/bin (the later requires you to have root permission); better to add INSTALLDIR to $PATH

3. Compile object files:

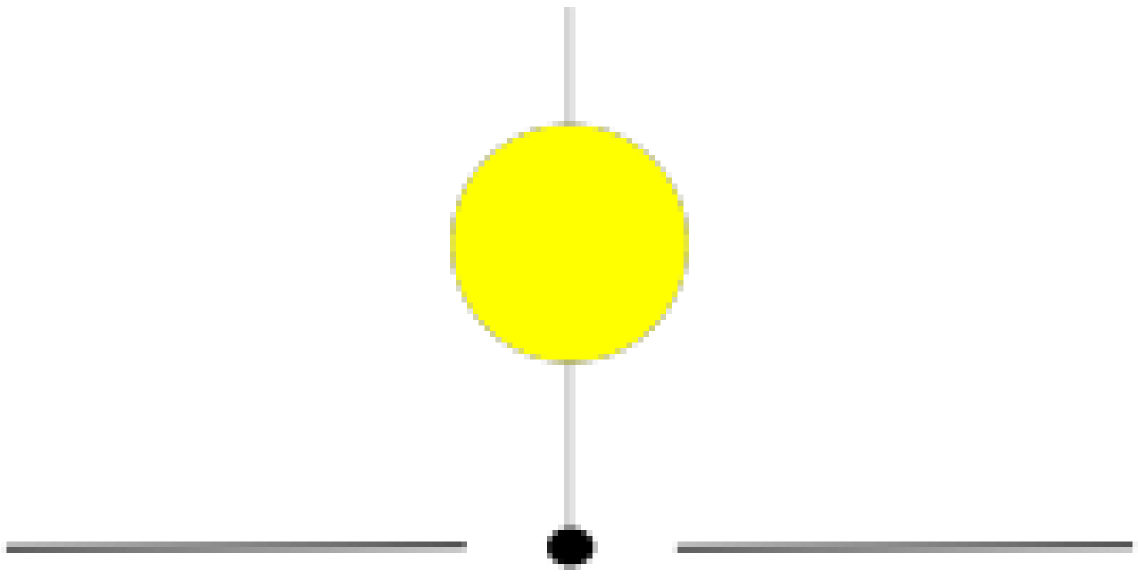
   make objs

4. Compile binaries:

   make 3dcorona 3dcorona_mpi calspec

Using the code

- A simple case: calculates the energy spectra for the following disc-corona system



Step 1: calculating geodesics from disc to corona using 3dcorona

1. Create a parameter file (you can find it in the source code as 3dcorona_params.txt)

3dcorona_params.txt - Notepad

File Edit Format View Help

```
[physical]
a = 0.998 # bh spin
rin = -1 # thin disc rin; if rin < rms: rin = rms
rout = 100. # thin disc rout
gtype = 0 # corona type; 0: spherical above the disc; for more see `3dcorona.cpp`, `tridgeo.cpp`, and `tridgeo.h`
h = 5. # corona height
radius = 1. # corona radius

[gridsize]
nr = 50 # number of radial bins
ntheta = 50 # number of theta-bin when sampling photons emitted by the disc
nphi = 50 # number of phi-bin when sampling photons emitted by the disc

[option]
progress = 1 # whether to show progress
```

2. Run 3dcorona

```
3dcorona # using default parameter file params.txt
3dcorona 3dcorona_params.txt
```

Step 1: calculating geodesics from disc to corona using 3dcorona

Outputs:

- gparams.dat – coronal geometry
- sca_params.dat – geodesics reaching the corona
- disc_params.dat – geodesics reaching infinity
- selfirr_params.dat – geodesics reaching other parts of the disc

Step 2: performing the simulation

1. Create a parameter file for `3corona_mpi` (3dcorona_mpi_params.txt)

```
[physical]
m = 1e7 # BH mass
mdot = 0.1 # Eddington mdot; Mdot = 2.23e18 * m * mdot [g/s]
tau = 0.2 # corona optical depth
te = 100 # corona temperature
rin = -1. # rin of thin disc
fcol = 1.7 # color correction factor

[gridsize]
nphoton = 1 # number of photons per geodesic; increase nphoton to enhance statistics

[option]
progress = 1 # whether to show progress
pol = 1 # whether to do polarization calculations
dr = 1e-4 # raytracing step size;
scafile = ../sca_params.dat # path to sca_params.dat
gparamfile = ../gparams.dat # path to gparams.dat
KN = 1 # whether to use Klein-Nishina cross-section; if not, use Thompson cross-section
chandra = 1 # assumption on the limb-darkening and polarization properties of disc photons; if chandra = 0: we assume the disc
                                        # photons to be unpolarized and isotropic; otherwise the limb-darkening factor and polarization properties
                                        # follows the results for semi-infinite scattering atmosphere, Chandrasekhar 1960 <<Radiative Transfer>>
```

Step 2: performing the simulation

2. Create two directories:
   - inf: photons reaching infinity
   - disc: photons reaching the accretion disc

3. Run `3dcorona_mpi`:

   mpirun –n 4 3corona_mpi # using default parameter file params.txt

   mpirun –n 4 3corona_mpi 3dcorona_mpi_params.txt

Step 2: performing the simulation

4. Products:

- inf: info of photon reaching infinity
  - en0.dat : dimensionless photon energy; E/me c^2, where me is the electron rest mass
  - weight.dat : statistical weight
  - muinf.dat : cosine of photon inclination
  - l.dat, q.dat : two constants of motions
  - ktheta.dat : sign of the theta-component of photon wave vector
  - nsca.dat : number of scattering
  - qweight.dat, uweight.dat : Stokes parameters Q & U; produced if pol option is on
- disc:
  - en0.dat, weight.dat, l.dat, mu.dat, ktheta.dat
  - rhit.dat : radius on the disc where the photon arrives
  - kr.dat : sign of the r-component of the photon wave vector
  - K1.dat, K2.dat : real and imaginary parts of the Walker-Penrose constant; produced if pol option is on

Step 3: calculate the spectrum using calspec

Path to the inf directory

Min energy in keV

calspec ../ -200 0.001 500

Max energy in keV

Number of the energy bins;
minus means log scale in energy

- Energy spectra in an inclination bin:

calspec ../ -200 0.001 500 25 35

Max inclination angle in degree

Min inclination angle in degree

Products of calspec:

- en.dat: energy in keV
- flux.dat: flux density, in photons/s/kev after multiplying by 1.65e42

if polarization option is turned when running 3dcorona_mpi:

- poldeg.dat: polarization degree
- polang.dat: polarization angle, in radians

Read and plot the spectra

- All .dat files produced by calspec contains 64-bit double-precision binary data.
- In python, they can be read with the numpy.fromfile() function

```
#!/usr/local/miniconda/bin/python
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()
en = np.fromfile('en.dat')
flux = np.fromfile('flux.dat')
# 1.60e-9: keV to erg
lum = flux * en * en * 1.65e42 * 1.60e-9

ax.set_ylabel(r'$\nu L_{\nu}\ {\rm[erg\ s^{-1}]}$')
ax.minorticks_on()
ax.set_xlabel('Energy (keV)')
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_ylim(5e35, 2e44)

plt.savefig('flux.pdf', bbox_inches='tight')
```

Different geometries

- There are ~10 built-in geometries in Monk (see tridgeo.h)
- You can define a new geometry by derive a new class based on the abstract base class tridgeo
- You need to implement the new class by defining a few functions

```
class tridgeo {
public:
    //! geometry name
    std::string name;
    //! BH spin
    double a;
    //! Minimum \f$r\f$ of corona
    double rmin;
    //! Maximum \f$r\f$ of corona
    double rmax;
    //! Minimum \f$\mu\f$
    double mumin;
    //! Maximum \f$\mu\f$
    double mumax;
    //! whether if the corona is magnetised
    bool magnetised;
    //! Tell if the photon is inside
    virtual bool inside(const double r, const double mu) const = 0;
    //! Calculate the tetrad attached to the corona fluid
    virtual void caltet(const double, const double, sim5tetrad &) const = 0;
    //! Calculate the four velocity of the corona fluid
    virtual void calumu(const double, const double, std::array<double, 4> &, sim5metric &) const = 0;
    //! Virtual destructor
    virtual ~tridgeo(){};
    //! Return the length scale of the corona
    virtual double length() const = 0;
    virtual void genpos(std::mt19937_64 & gen, double &, double &) const = 0;
    virtual double mean_free_path(const double, const double) const = 0;
    //! returns \f$n_e * \sigma\f$, given r, mu
    virtual double ne_sigmat(const double, const double) const = 0;
    virtual void write_brem_sp(const size_t ndim1, const size_t ndim2, std::ofstream & ofile) const = 0;
    virtual double cal_bfield(const double r, const double mu, const double te_K) const = 0;
};
```

Different electron velocity distribution

- Isotropic thermal and powerlaw distributions available in Monk (isotropic_thermal and isotropic_nonthermal classes in electron_population.h)
- To define a different one, derive a new class based on the electron_population class

```
//! virtual electron velocity distribution class
class electron_population {
public:
    //!
    virtual ~electron_population(){};
    //! Given photon energy, return the mean scattering cross section with respect to a electron population.
    virtual double cross_section(const double x) = 0;
    //! Given photon energy, sample the Lorentz factor of the scattering electron and the angle between the electron and the photon.
    virtual void sample_mu(const double x, double & gamma, double & emu, std::mt19937_64 & gen) const = 0;
};
```

Different seed photon distribution

- Built-in types in Monk: blackbody (bb), powerlaw (pl), exponential cut-off powerlaw (cutoffpl), and monoenergetic (monoenergetic), defined in photon_dist.h
- Define your own photon distribution by create a new class based on the photon_dist class

```
class photon_dist {
public:
    //! Virtual destructor
    virtual ~photon_dist(){};
    //! Constructor
    photon_dist(){};
    //! Sampling method
    virtual double genen(std::mt19937_64 & gen) const = 0;
    virtual double cal_weight() const = 0;
};
```

- The Monk paper: Zhang, Dovicak, Bursa 2019, 875, 148 https://ui.adsabs.harvard.edu/abs/2019ApJ...875..148Z/abstract
- Feel free to contact me: wdzhang@nao.cas.cn