generating spectra with grmonty

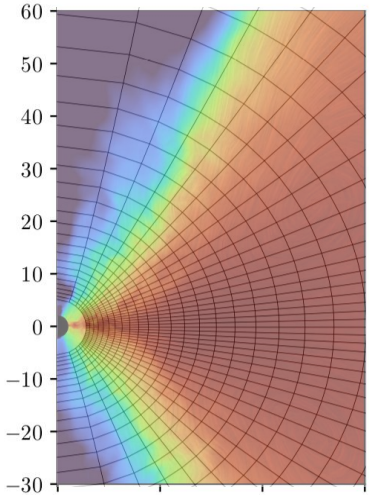
September 7th, 2021

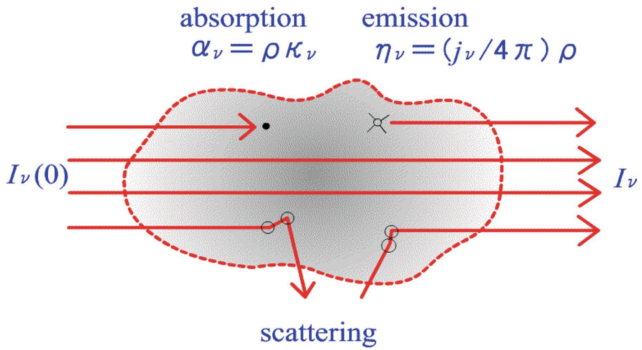
outline

1. overview of grmonty (abstract)
   a. purpose + design
   b. grmonty command flow and important functions
2. using grmonty (practical)
   a. downloading, compiling, and developing
   b. analytic “one-zone” model without Compton
   c. … with Compton
   d. generating spectra from GRMHD fluid simulations
3. some advanced topics (mixed)
   a. bias tuning parameters
   b. modifying the electron distribution function

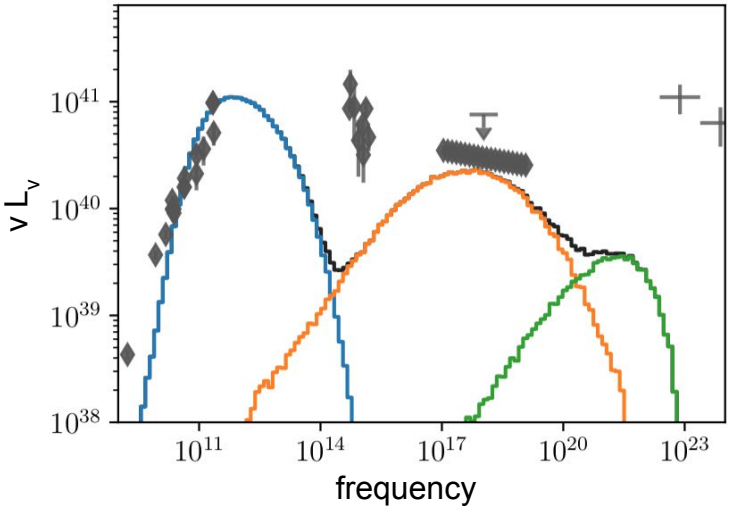
grmonty – at a glance

`grmonty` is an **emitter-to-observer general relativistic ray tracing** code that incorporates the effects of synchrotron* emission and absorption as well as Compton scattering. The base version does not consider polarization.





Kato, Fukue 2020



* also includes limited treatment of electron–electron and electron–ion bremsstrahlung

grmonty – Monte Carlo radiation

To account for **frequency-dependence** and **anisotropy** in the radiation field, `grmonty` uses a **Monte Carlo** approach: it tracks the radiation field with a large number of discrete “packets” of (many!) photons, called **superphotons**

Each superphoton (superph) corresponds to a packet of w (**weight**) photons, each having the **same wavevector**, i.e., the same frequency and direction

Extinction (e.g., due to **absorption** or **scattering** out of the line of sight) **decreases w**

Create (and track) a **new superphoton** with **weight w’** when a large fraction w’/w of a fiducial superphoton undergoes a **scattering event**

Final spectrum is **histogram of all superphotons that make it to infinity**, binned by **inclination**, **energy**, and optionally origin (synchrotron, Compton scattering, etc.)

grmonty – command flow

```
for each zone:  # pre-compute superph-to-be-emitted per zone
    compute total energy emitted across all frequencies/directions
=> sum across zones to produce "zero-absorption/scattering" spectrum
set WEIGHT(frequency) so that same number of superph per frequency bin
set SUPERPH(zone) so that total number of superph = target

for each zone:  # perform full ray tracing
     for each [count] in SUPERPH(zone):
          sample frequency/direction-dependent emissivity; make superph with WEIGHT(frequency)
          EVOLVE(superph)  # track superph across domain and deal with absorption and scattering
```

grmonty – command flow

```
for each zone:  # pre-compute superph-to-be-emitted per zone
    compute total energy emitted across all frequencies/directions
=> sum across zones to produce "zero-[illegible]ctrum
set WEIGHT(frequency) so that same number of superph per frequency bin
set SUPERPH(zone) so that total number of superph = target

for each zone:  # perform
    for each [count] in SUPERPH(zone):
        sample frequency/direction-dependent emissivity; make superph with WEIGHT(frequency)
        EVOLVE(superph)  # track superph across domain and deal with absorption and scattering
```

utils.c:init_weight_table()

utils.c:make_super_photon() + utils.c:sample_zone_photon()

track_super_photon.c:track_super_photon()

grmonty – command flow

```
def EVOLVE(superph):  # track superph across domain and deal with absorption and scattering

    while superph in domain:

        compute scattering likelihood for "1/bias" of superph and "location" of next scattering


        integrate geodesic equation short distance (max = scatter location) and push superph


        if should scatter:  # "scatter" this superph into another

            decrease superph weight w --> w * (1 - 1/bias)

            # now scatter

            create new superphoton superph' with w' = w / bias
            sample from dNe/d(gamma) to find electron for scattering event
            set superph' wavevector according to momentum of electron

            # and track new superphoton

            EVOLVE(superph')

        decrease superph weight according to local absorption
```

grmonty – command flow

```
def EVOLVE(superph):  # tr                                              n and scattering
    while superph in domain:
        compute scattering likelihood for "1/bias" of superph and "location" of next scattering
        integrate geodesic equation sh                                  ion) and push superph
        if should scatter:  # "scatter" this superph into another
            decrease superph weight w --> w * (1 - 1/bias)
            # now scatter
            create new superphoton superph' with w' = w / bias
            sample from dNe/d(gamma) to find e
            set superph' wavevector according
            # and track new superphoton
            EVOLVE(superph')
        decrease superph weight according to local absorption
```

track_super_photon.c:track_super_photon(...)

geodesics.c:push_photon(...)

scatter_super_photon.c:scatter_super_photon(...)

compton.c:sample_scattered_photon(...)
compton.c:sample_electron_distr_p(...)

grmonty – getting set up

- Gammie group maintains the `igrmonty` version of the code on github:

  https://github.com/afd-illinois/igrmonty

- You will need to download the source from github to your computer and then compile it. Unless you know what you're doing, use the primary "master" branch of the repo!

```
$ git clone git@github.com:afd-illinois/igrmonty.git
$ cd igrmonty
# set "model loader" in makefile. default is iharm GRMHD
$ make
```

- We prefer pull requests for code modifications. Let us know if you need help with this.

grmonty – getting set up

- Gamm

  https:

- You wil

  Unless

```
# Problem to compile
MODEL = iharm

# Top directory of HDF5, or blank if using h5pcc
HDF5_DIR =
# Top directory of GSL, or blank if installed to system
GSL_DIR =
# System /lib equivalent (can be /usr/lib, /lib64, /usr/lib64)
# Can leave this blank if it's included automatically by GCC
SYSTEM_LIBDIR = /lib64

# Try pointing this to h5pcc or h5cc on your machine, before hunting down libraries
CC=h5cc
...
```

```
$ c

$ cd igrmonty

# set "model loader" in makefile. default is iharm GRMHD

$ make
```

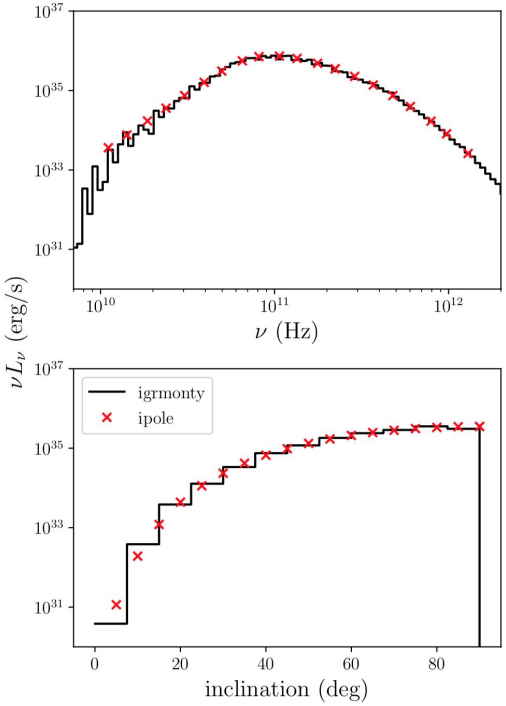
- We prefer pull requests for code modifications. Let us know if you need help with this.

grmonty – your first spectrum

Start with a simple “one-zone” model, i.e., a sphere with constant:

N_e ~ 2.5e5 cm^{-3}, Theta_e = 10, B ~ 3.2 G

```
== makefile ==

# Problem to compile
MODEL = sphere
...
```

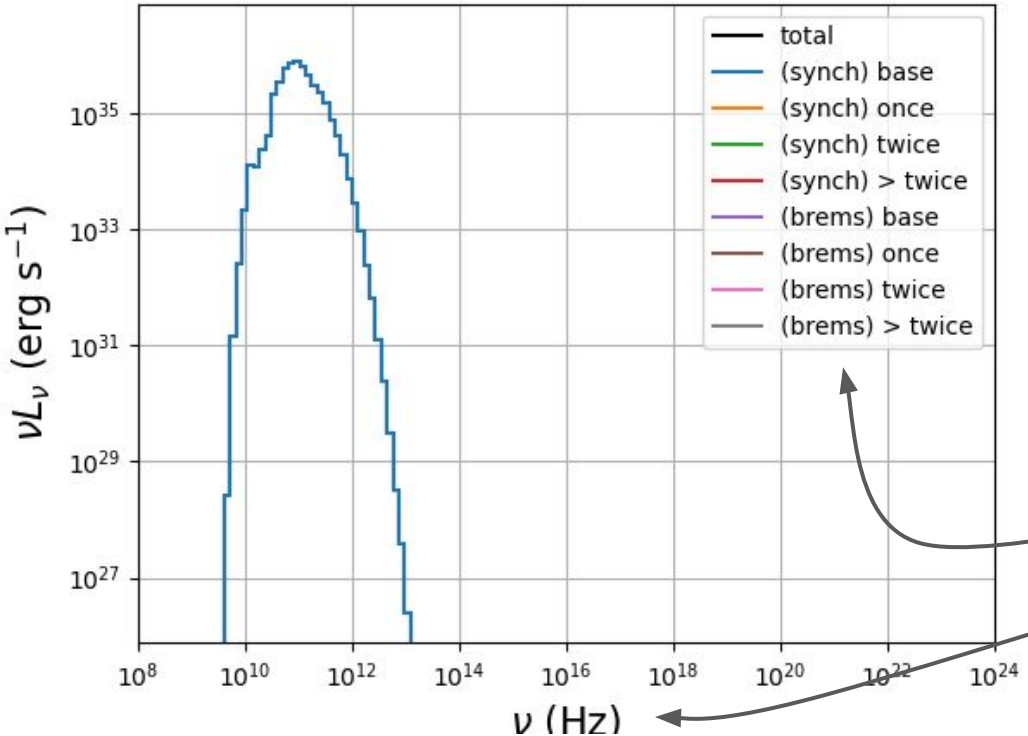



```
$ ./grmonty 10000
...
Running with isothermal sphere model.
MBH, L_unit: 4.1e+06 [Msun], 6.05587e+11
Ne, Thetae, B: 248223 10 3.19697
Sphere radius, Rout: 6.05587e+13 7.26704e+13
...
with synch: 1
with brems: 0
with compt: 0


Entering main loop...
(aiming for Ns=10000)
time 4s, ph made 100k, rate 25k/s, scatter 0k, ratio 0
compute time 3s, ph made 129k, rate 42.7k/s, scatter 0k, ratio 0

...

N_superph_made = 128595
N_superph_scatt = 0
N_superph_recorded = 27708
Total wallclock time: 6.3987 s
```

grmonty – your first spectrum

Plot the output spectrum using the `plspec.py` script.

```
$ python plspec.py spectrum.h5
plotting spectrum for spectrum.png
(8, 200)
plotting spectrum for spectrum-avg.png
(8, 200)
```



- spectrum file is an hdf5 file
- includes `params` and `output` groups with input and spectrum+ data
- frequency reported as `lnu` ~ log(electron temperature)
- `nuLnu` reported in units of Lsun with shape `nuLnu[origin, energy, inclination]`

lowest inclinations “down the pole” ⇒ 0

grmonty – your first spectrum … now with scattering

Turn on COMPTON to track Compton scattering events. We can use **parameter files** to provide extra parameters to grmonty.

```
$ ./grmonty -par my_params.par
...

with synch: 1
with brems: 0
with compt: 1

Entering main loop...
(aiming for Ns=10000)
time 7s, ph made 100k, rate 14.3k/s, scatter 1.89k, ratio 0.0188
compute time 7s, ph made 128k, rate 18.3k/s, scatter 2.19k, ratio 0.0171

...

N_superph_made = 128408
N_superph_scatt = 2191
N_superph_recorded = 29508
Total wallclock time: 23.6732 s
```

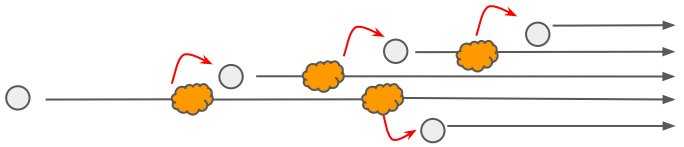
want ratio $\gtrsim 1.0$

```
== model/sphere/model.h ==

...

#define SYNCHROTRON (1)
#define BREMSSTRAHLUNG (0)
#define COMPTON (1)

...
```

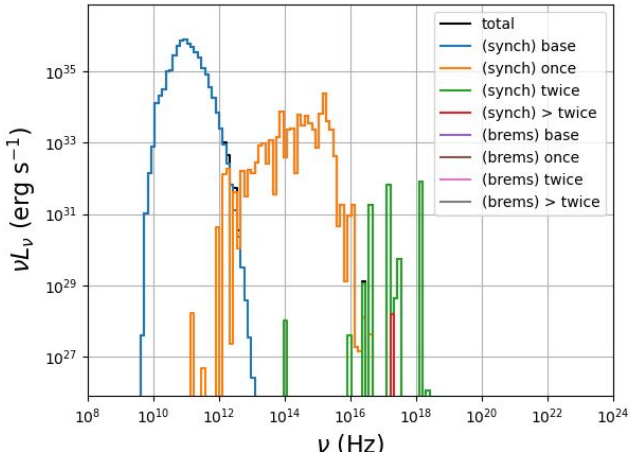
```
== my_params.par ==

Ns 10000
spectrum output.h5

bias 0.01
```

grmonty – your first spectrum … now with scattering

`bias` parameter must be **large enough** that we have scattering events by **not so large** that the scattering goes critical

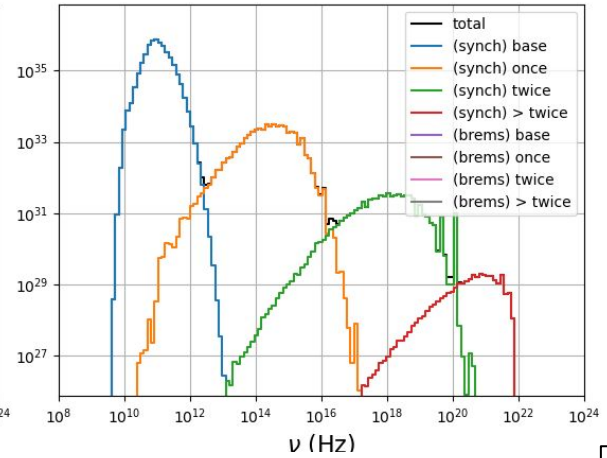


`bias 0.01 ⇒ ratio = 0.018`



too noisy

`bias 0.2 ⇒ ratio = 2.4`



much smoother in general, try increasing Ns to address the spikes in Compton

`bias 1.0 ⇒ ratio > 263`

```
$ ./grmonty -par my_params.par
...

with synch: 1
with brems: 0
with compt: 1

Entering main loop...
(aiming for Ns=10000)
compute time 2s, ph made 0.431k, rate
         0.215k/s, scatter 113k, ratio 263

it seems the bias was too high -- aborting.
Total wallclock time: 13.1141 s
```

code realizes each “seed” photon is scattering too much ⇒ critical state ⇒ halts execution

grmonty – running on a GRMHD simulation

Now try with GRMHD simulation instead of simple analytic model to GRMHD simulation. Recompile the code with alternative "iharm" model. Fluid models have **more parameters**.

```
== makefile ==

# Problem to compile
MODEL = iharm
...
```

final target output resolution

```
== my_params.par ==
Ns    80000

MBH   6.2e9
dump   path/to/GRMHD/snapshot.h5
M_unit     9.50325e+24
trat_small      1
trat_large      40
spectrum   path/to/desired/output/spectrum_for_snapshot_1_40.h5

bias   1.e-3
fit_bias 1
fit_bias_ns     20000
```

model-specific choices (scalings, thermodynamics model, which snapshot to use)

parameters to automatically find good bias

Rather than manually finding the best bias for every snapshot, the code supports a mode to pre-compute spectra in low resolution to try and find a reasonable bias.

grmonty – bias tuning notes

```
$ ./grmonty -par my_params.par
...

with synch: 1
with brems: 1
with compt: 1

Finding bias...
bias 0.001   ratio = 0.171541
bias 0.0015   ratio = 0.338516
bias 0.00225   ratio = 0.722531
in sample_scattered_photon:
kp[0], kpe[0]: -648.905 -0.931558
kpe: -0.931558 -0.931358 0.0192564 -0.00161037
k:  9.09515 5.36953 4.00328 6.19713
ke: -0.931558 0.834332 0.811881 0.230954
p:   449.458 -250.247 -176.098 -329
kp:  -648.905 -361.941 -254.022 -47
K0, K0p, Kp, P[0]: 9.09515 -648.905
bias 0.003375   ratio = 1.67123

Entering main loop...
(aiming for Ns=80000)
time 14, rate 7142.86 ph/s
...
time 1382, rate 2098.41 ph/s
final time 1386, rate 2142.46 ph/s

...

N_superph_made =  2969445
N_superph_scatt =  2555073
N_superph_recorded = 2438126
Total wallclock time: 1638.1 s
```

now compiled with bremsstrahlung!

current implementation does poor man's root-finding with heuristic safety checks

sometimes diagnostic information is printed. you can probably ignore in regular use.

the final ratio may not be the ratio found when running at low resolution!

bias tuning **starts with seed** value and uses a limited secant-method to iteratively rerun (at low resolution) until ratio is reasonable

bias tuning works best when the **seed value is small**

grmonty is written to **fail early** and **fail loudly**, so bias tuning may report errors … or fail entirely.

grmonty – bias tuning notes

```
$ ./grmonty -par my_params.par
...

with synch: 1
with brems: 1
with compt: 1

Finding bias...
bias 0.001  ratio = 0.171541
bias 0.0015  ratio = 0.338516
bias 0.00225  ratio = 0.722531
in sample_scattered_photon:
kp[0], kpe[0]: -648.905 -0.931558
kpe: -0.931558 -0.931358 0.0192564 -0.00161037
k:  9.09515 5.36953 4.00328 6.19713
ke: -0.931558 0.834332 0.811881 0.230954
p:   449.458 -250.247 -176.098 -329.207
kp:  -648.905 -361.941 -254.022 -474.92
K0, K0p, Kp, P[0]: 9.09515 -648.905 -771.157 449.458
bias 0.003375  ratio = 1.67123

Entering main loop...
(aiming for Ns=80000)
time 14, rate 7142.86 ph/s
...
time 1382, rate 2098.41 ph/s
final time 1386, rate 2142.46 ph/s

...

N_superph_made =  2969445
N_superph_scatt =  2555073
N_superph_recorded = 2438126
Total wallclock time: 1638.1 s
```

grmonty jobs are often run unattended in series, so the **code prefers to fail** rather than potentially run in a loop forever

if grmonty does not produce spectrum, then:

1. if ratio is zero, increase seed bias
2. if ratio is consistently too large, decrease seed bias
3. if many **errors** / long runtime, try decreasing bias

if grmonty produces noisy spectrum, then:

1. increase Ns
2. increase bias slightly (compare low/high res ratios)

it may be **hard to run with synch+brems & Compton**. if synch / brems components are well-separated, try **running independently** and **summing** the two outputs

grmonty – modifying the electron distribution function

The electron distribution function (defined in `model_radiation.h` as `MODEL_EDF`) enters grmonty in 3 ways:

1. emission coefficients `jnu_mixed.c:jnu(...); jnu_mixed.c:int_jnu(...);`

2. absorption coefficients `radiation.c:alpha_inv_abs(...)`

3. the scattering computation computes the actual distribution function dN/d(gamma)

```
hotcross.c:dNdgammae(...)
hotcross.c:getnorm_dNdg(...)
compton.c:dfdgam(...)
compton.c:fdist(...)
```

Your eDF may also make use of:

1. user-supplied parameters (e.g., a fixed kappa) `radiation.c:try_set_radiation_parameter(...)`

2. local values computed on the fly (e.g., a position-dependent non-thermal electron fraction)

```
radiation.h:radiation_param_struct
radiation.c:radiation_params get_model_radiation_params(...)
```