VAST June 2023

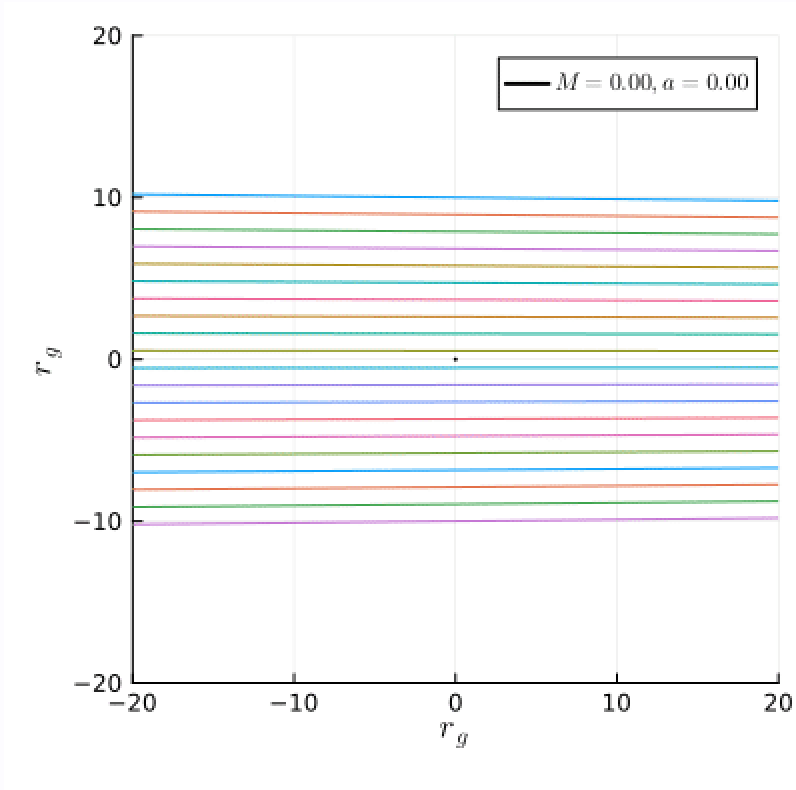
Gradus.jl

Fergus Baker, Andrew Young
University of Bristol

https://github.com/astro-group-bristol/Gradus.jl

General relativistic ray-tracing

- Photon trajectory distorted by **spacetime curvature**
- Curvature encoded in the **metric** $g_{\mu\nu}$



Use cases for GRRT

- Imaging black holes (e.g. Event Horizon Telescope)
- **Spectral modelling**
- **Variability modelling**
  - Measuring metric parameters (M, a, $\ldots$)
  - Measuring disc and coronal parameters (h, r_{disc}, $\ldots$)
  - Testing relativity

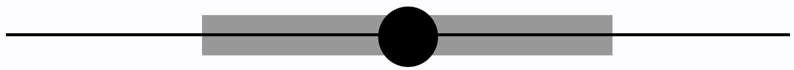
Why Gradus.jl?

- Existing codes are brittle / designed for a single purpose
- Codebase requires familiarly to extend or is tedious
- Ray-tracing can be a difficult or error prone
- *Speed* ⚡ and scalability

Our approach

- Using **automatic differentiation** to calculate the geodesic equation
- Exploiting **symbolic computing** at compile time
- **Multiple-dispatch** for composable abstractions
- Julia's heterogenous parallelism

Example: A user defined metric, disc, and corona



Specifying the metric parameters ...

```
struct Schwarzschild{T} <: AbstractStaticAxisSymmetric{T}
    # if no special symmetries, subtype AbstractMetric
    M::T
end

# event horizon
Gradus.inner_radius(m::Schwarzschild) = 2 * m.M

metric = Schwarzschild(1.0)
```

... specifying the metric components ...

```julia
function Gradus.metric_components(m::Schwarzschild, x)
    r, θ = x

    dt = -(1 - (2m.M / r))
    dr = -1 / dt
    dθ = r^2
    dϕ = r^2 * sin(θ)^2
    dtdϕ = zero(r)

    return SVector(dt, dr, dθ, dϕ, dtdϕ)
end
```

... sanity checks ...

```julia
using Symbolics, Latexify

ds = @variables dt, dr, dθ, dϕ, r, θ, M
comp = Gradus.metric_components(Schwarzschild(M), SVector(r, θ))

sum(ds[i]^2 * comp[i] for i in 1:4) |> latexify
```

$$r^2 d\theta^2 + dt^2 \left(-1 + \frac{2M}{r} \right) + \frac{-dr^2}{-1 + \frac{2M}{r}} + \sin^2\left(\theta \right) r^2 d\phi^2$$

… adding a disc model, composing it …

```julia
struct SlabDisc{T} <: AbstractAccretionDisc{T}
    height::T
    radius::T
    emissivity_coefficient::T
end

Gradus.emissivity_coefficient(m::AbstractMetric, d::SlabDisc, x, ν) =
    d.emissivity_coefficient

# instantiate and compose
slab = SlabDisc(4.0, 20.0, 0.1)
disc = GeometricThinDisc(Gradus.isco(metric), 50.0, π/2) ∘ slab
```

... an intersection criteria ...

```julia
function Gradus.distance_to_disc(d::SlabDisc, x4; gtol)
    if d.radius < x4[2]
        return 1.0
    end

    # current geodesic height along z-axis
    h = abs(x4[2] * cos(x4[3]))

    # if height difference is negative, intersection
    return h - d.height - (gtol * x4[2])
end
```

... adding a coronal model ...

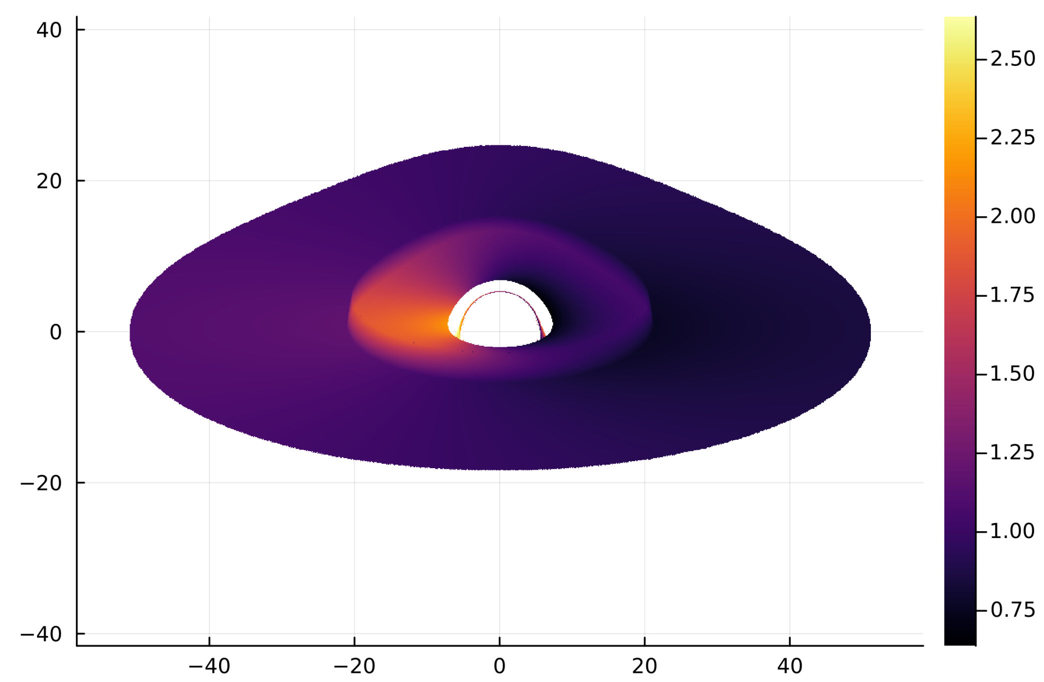
```julia
struct SlabCorona{T} <: AbstractCoronaModel{T}
    height::T
    radius::T
end

# reuse disc parameters in corona
corona = SlabCorona(slab.height, slab.radius)
```

… sampling the source position and velocity …

```julia
function Gradus.sample_position_velocity(
    m::AbstractMetric,
    model::SlabCorona{T},
) where {T}
    # random position on the disc
    ϕ = 2π * rand(T)
    R = sqrt(rand(T) * model.radius^2)
    h = rand(T) * model.height # only upper hemisphere

    # translate from cylindrical to spherical
    r = √(R^2 + h^2) ; θ = atan(R, h) + 1e-2
    x = SVector(0, r, θ, ϕ)

    # use circular orbit velocity as source velocity
    v = if R < r_isco
        CircularOrbits.plunging_fourvelocity(m, R)
    else
        CircularOrbits.fourvelocity(m, R)
    end
    x, v
end
```

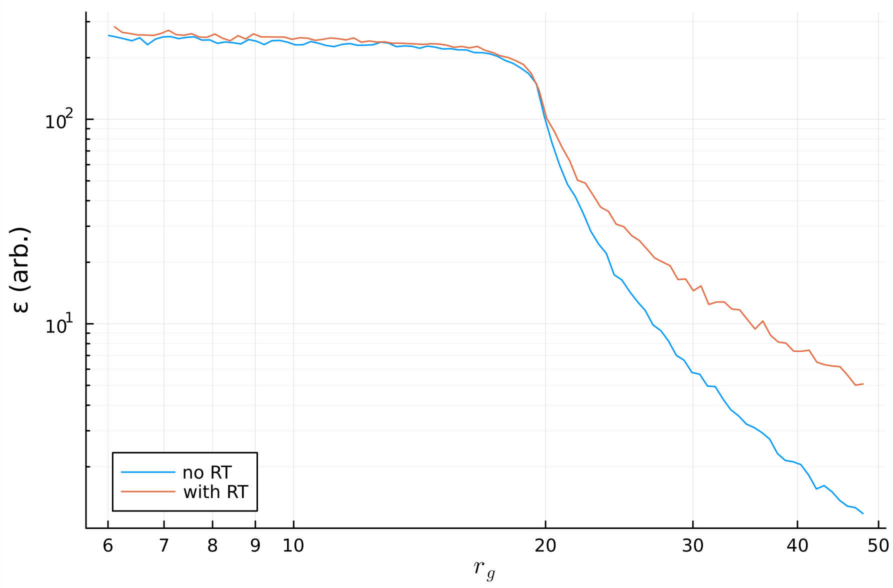
… putting it all together.

```
# observer position
x = SVector(0.0, 10_000.0, deg2rad(70), 0.0)

pf = PointFunction(
        (m, gp, t) -> ConstPointFunctions.redshift(m, gp, t) * gp.aux[1]
    ) ∘ ConstPointFunctions.filter_intersected

a, b, image = @time rendergeodesics(metric, x, disc)
```



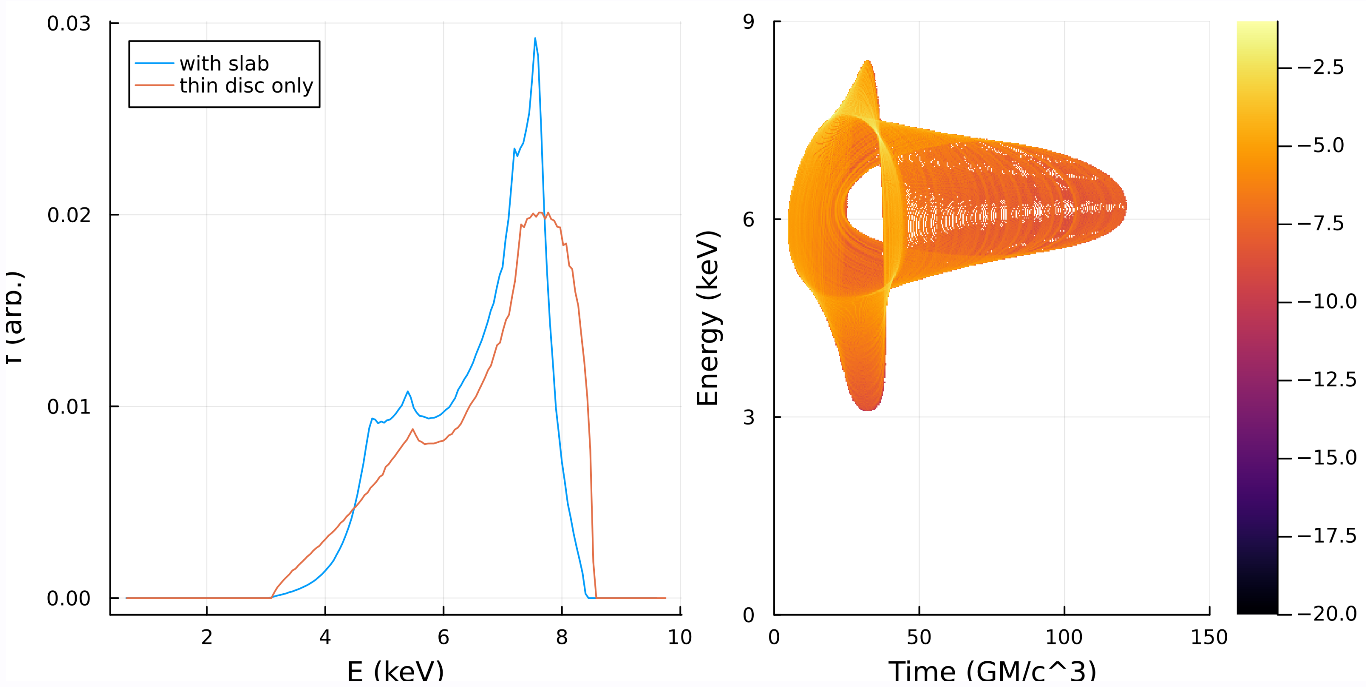
Calculate how the corona illuminates the disc:

```
ep = @time emissivity_profile(metric, disc, corona) |> RadialDiscProfile
```



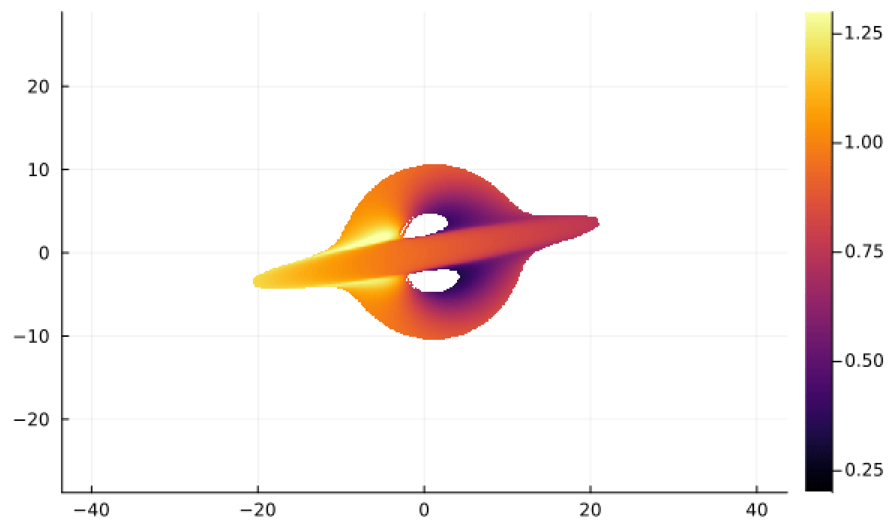
Lineprofile and reverberation transfer functions:

```
E, f = @time lineprofile(m, x, disc, ep)

rtf = @time lagtransfer(m, x, disc, corona)
t, E, f = binflux(rtf, ep)
```



Simple to modify:

```diff
- metric = Schwarzschild(1.0)
+ metric = JohannsenPsaltis(a = 0.6, ϵ3 = 1.0)

- disc = GeometricThinDisc(Gradus.isco(metric), 50.0, π/2) ∘ slab
+ disc = PrecessingDisc(EllipticalDisc(2.0, 20.0))

- corona = SlabCorona(slab.height, slab.radius)
+ corona = LampPost(slab.height)
```



Open-source, with an
open invitation for collaboration ❤️

Thank you :)

Contact: fergus.baker@bristol.ac.uk
GitHub: @fjebaker

- Gradus.jl: https://github.com/astro-group-bristol/Gradus.jl
- The Julia Programming Language: https://julialang.org/
- DifferentialEquations.jl: https://github.com/SciML/DifferentialEquations.jl
- Plots.jl: https://github.com/JuliaPlots/Plots.jl
- ForwardDiff.jl: https://github.com/JuliaDiff/ForwardDiff.jl
- Presentation made with Marp: https://marp.app/