
Reference manual of kerrgeodesic_gw

Release 0.4.1

**Eric Gourgoulhon, Alexandre Le Tiec,
Frederic Vincent, Niels Warburton**

Sep 14, 2021

CONTENTS

1	Kerr spacetime	1
2	Geodesics in Kerr spacetime	17
3	Spherical and spheroidal harmonics	31
3.1	Spin-weighted spherical harmonics	31
3.2	Spin-weighted spheroidal harmonics	33
4	Amplitude factors $Z_{\ell m}^{\infty}(r)$	37
5	Gravitational radiation by a single particle	39
6	Gravitational radiation by a blob of matter	61
7	LISA Detector	69
8	Signal processing	71
9	Astronomical data	79
10	Utilities	81
	Python Module Index	85
	Index	87

KERR SPACETIME

The Kerr black hole is implemented via the class `KerrBH`, which provides the Lorentzian manifold functionalities associated with the Kerr metric, as well as functionalities regarding circular orbits (e.g. computation of the ISCO, Roche limit, etc.).

REFERENCES:

- J. M. Bardeen, W. H. Press and S. A. Teukolsky, *Astrophys. J.* **178**, 347 (1972), doi:10.1086/151796
- L. Dai and R. Blandford, *Mon. Not. Roy. Astron. Soc.* **434**, 2948 (2013), doi:10.1093/mnras/stt1209

```
class kerrgeodesic_gw.kerr_spacetime.KerrBH(a, m=1, manifold_name='M', manifold_latex_name=None,  
metric_name='g', metric_latex_name=None)
```

Bases: `sage.manifolds.differentiable.pseudo_riemannian.PseudoRiemannianManifold`

Kerr black hole spacetime.

The Kerr spacetime is generated as a 4-dimensional Lorentzian manifold, endowed with the Boyer-Lindquist coordinates (default chart). Accordingly the class `KerrBH` inherits from the generic SageMath class `PseudoRiemannianManifold`.

INPUT:

- `a` – reduced angular momentum
- `m` – (default: 1) total mass
- `manifold_name` – (default: 'M') string; name (symbol) given to the spacetime manifold
- `manifold_latex_name` – (default: None) string; LaTeX symbol to denote the spacetime manifold; if none is provided, it is set to `manifold_name`
- `metric_name` – (default: 'g') string; name (symbol) given to the metric tensor
- `metric_latex_name` – (default: None) string; LaTeX symbol to denote the metric tensor; if none is provided, it is set to `metric_name`

EXAMPLES:

Creating a Kerr spacetime with symbolic parameters (a, m):

```
sage: from kerrgeodesic_gw import KerrBH  
sage: a, m = var('a m')  
sage: BH = KerrBH(a, m); BH  
Kerr spacetime M  
sage: dim(BH)  
4
```

The Boyer-Lindquist chart:

```
sage: BH.boyer_lindquist_coordinates()
Chart (M, (t, r, th, ph))
sage: latex(_)
\left(M,(t, r, {\theta}, {\phi})\right)
```

The Kerr metric:

```
sage: g = BH.metric(); g
Lorentzian metric g on the Kerr spacetime M
sage: g.display()
g = -(a^2*cos(th)^2 - 2*m*r + r^2)/(a^2*cos(th)^2 + r^2) dt⊗dt
  - 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dt⊗dph
  + (a^2*cos(th)^2 + r^2)/(a^2 - 2*m*r + r^2) dr⊗dr
  + (a^2*cos(th)^2 + r^2) dth⊗dth
  - 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dph⊗dt
  + (2*a^2*m*r*sin(th)^4 + (a^2*r^2 + r^4 + (a^4 + a^2*r^2)*cos(th)^2)*sin(th)^2)/(a^2*cos(th)^2 + r^2)
↔dph⊗dph
sage: g[0,3]
-2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2)
```

A Kerr spacetime with specific numerical values for (a, m) , namely $m = 1$ and $a = 0.9$:

```
sage: BH = KerrBH(0.9); BH
Kerr spacetime M
sage: g = BH.metric()
sage: g.display() # tol 1.0e-13
g = -(r^2 + 0.81*cos(th)^2 - 2*r)/(r^2 + 0.81*cos(th)^2) dt⊗dt
  - 1.8*r*sin(th)^2/(r^2 + 0.81*cos(th)^2) dt⊗dph
  + (1.0*r^2 + 0.81*cos(th)^2)/(1.0*r^2 - 2.0*r + 0.81) dr⊗dr
  + (r^2 + 0.81*cos(th)^2) dth⊗dth
  - 1.8*r*sin(th)^2/(r^2 + 0.81*cos(th)^2) dph⊗dt
  + (1.62*r*sin(th)^4 + (1.0*r^4 + (0.81*r^2 + 0.6561)*cos(th)^2
  + 0.81*r^2)*sin(th)^2)/(1.0*r^2 + 0.81*cos(th)^2) dph⊗dph
sage: g[0,3]
-1.8*r*sin(th)^2/(r^2 + 0.81*cos(th)^2)
```

The Schwarzschild spacetime as the special case $a = 0$ of Kerr spacetime:

```
sage: BH = KerrBH(0, m)
sage: g = BH.metric()
sage: g.display()
g = (2*m - r)/r dt⊗dt - r/(2*m - r) dr⊗dr + r^2 dth⊗dth + r^2*sin(th)^2 dph⊗dph
```

The object returned by `metric()` belongs to the SageMath class `PseudoRiemannianMetric`, for which many methods are available, like `christoffel_symbols_display()` to get the Christoffel symbols with respect to the Boyer-Lindquist coordinates (by default, only nonzero and non-redundant symbols are displayed):

```
sage: g.christoffel_symbols_display()
Gam^t_t,r = -m/(2*m*r - r^2)
Gam^r_t,t = -(2*m^2 - m*r)/r^3
Gam^r_r,r = m/(2*m*r - r^2)
Gam^r_th,th = 2*m - r
Gam^r_ph,ph = (2*m - r)*sin(th)^2
Gam^th_r,th = 1/r
Gam^th_ph,ph = -cos(th)*sin(th)
Gam^ph_r,ph = 1/r
Gam^ph_th,ph = cos(th)/sin(th)
```

or `ricci()` to compute the Ricci tensor (identically zero here since we are dealing with a solution of Einstein equation in vacuum):

```
sage: g.ricci()
Field of symmetric bilinear forms Ric(g) on the Schwarzschild
```

(continues on next page)

(continued from previous page)

```
spacetime M
sage: g.ricci().display()
Ric(g) = 0
```

Various methods of `KerrBH` class implement the computations of remarkable radii in the Kerr spacetime. Let us use them to reproduce Fig. 1 of the seminal article by Bardeen, Press and Teukolsky, *ApJ* **178**, 347 (1972), doi:10.1086/151796, which displays these radii as functions of the black hole spin parameter a . The radii of event horizon and inner (Cauchy) horizon are obtained by the methods `event_horizon_radius()` and `cauchy_horizon_radius()` respectively:

```
sage: graph = plot(lambda a: KerrBH(a).event_horizon_radius(),
....:              (0., 1.), color='black', thickness=1.5,
....:              legend_label=r"$r_{\rm H}$ (event horizon)",
....:              axes_labels=[r"$a/M$", r"$r/M$"],
....:              gridlines=True, frame=True, axes=False)
sage: graph += plot(lambda a: KerrBH(a).cauchy_horizon_radius(),
....:              (0., 1.), color='black', linestyle=':', thickness=1.5,
....:              legend_label=r"$r_{\rm C}$ (Cauchy horizon)")
```

The ISCO radius is computed by the method `isco_radius()`:

```
sage: graph += plot(lambda a: KerrBH(a).isco_radius(),
....:              (0., 1.), thickness=1.5,
....:              legend_label=r"$r_{\rm ISCO}$ (prograde)")
sage: graph += plot(lambda a: KerrBH(a).isco_radius(retrograde=True),
....:              (0., 1.), linestyle='--', thickness=1.5,
....:              legend_label=r"$r_{\rm ISCO}$ (retrograde)")
```

The radius of the marginally bound circular orbit is computed by the method `marginally_bound_orbit_radius()`:

```
sage: graph += plot(lambda a: KerrBH(a).marginally_bound_orbit_radius(),
....:              (0., 1.), color='purple', linestyle='-', thickness=1.5,
....:              legend_label=r"$r_{\rm mb}$ (prograde)")
sage: graph += plot(lambda a: KerrBH(a).marginally_bound_orbit_radius(retrograde=True),
....:              (0., 1.), color='purple', linestyle='--', thickness=1.5,
....:              legend_label=r"$r_{\rm mb}$ (retrograde)")
```

The radius of the photon circular orbit is computed by the method `photon_orbit_radius()`:

```
sage: graph += plot(lambda a: KerrBH(a).photon_orbit_radius(),
....:              (0., 1.), color='gold', linestyle='-', thickness=1.5,
....:              legend_label=r"$r_{\rm ph}$ (prograde)")
sage: graph += plot(lambda a: KerrBH(a).photon_orbit_radius(retrograde=True),
....:              (0., 1.), color='gold', linestyle='--', thickness=1.5,
....:              legend_label=r"$r_{\rm ph}$ (retrograde)")
```

The final plot:

```
sage: graph
Graphics object consisting of 8 graphics primitives
```

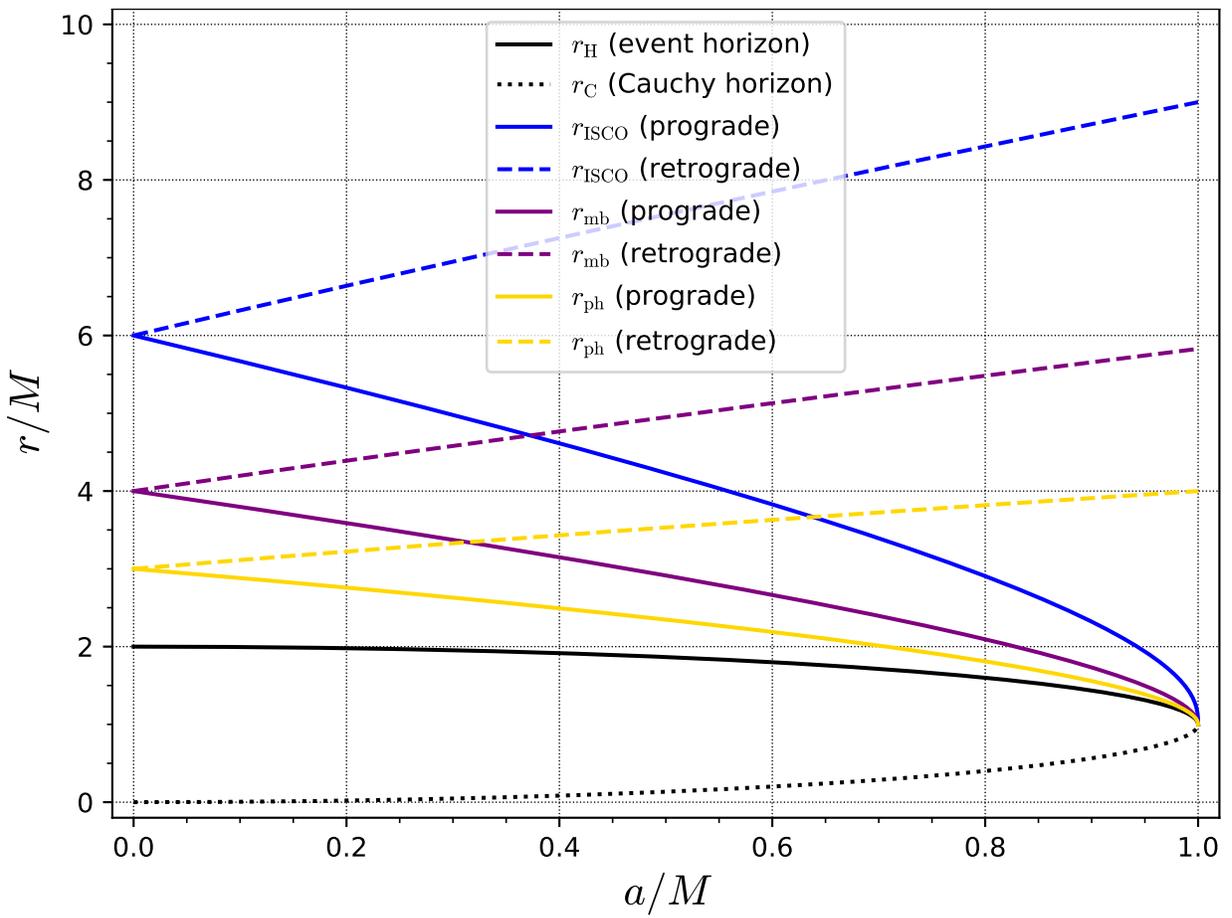
Arbitrary precision computations are possible:

```
sage: a = RealField(200)(0.95) # 0.95 with 200 bits of precision
sage: KerrBH(a).isco_radius() # tol 1e-50
1.9372378781396625744170794927972658947432427390836799716847
```

angular_momentum()

Return the reduced angular momentum of the black hole.

EXAMPLES:



```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.spin()
a
```

An alias is `angular_momentum()`:

```
sage: BH.angular_momentum()
a
```

boyer_lindquist_coordinates(*symbols=None, names=None*)

Return the chart of Boyer-Lindquist coordinates.

INPUT:

- `symbols` – (default: `None`) string defining the coordinate text symbols and LaTeX symbols, with the same conventions as the argument `coordinates` in `RealDiffChart`; this is used only if the Boyer-Lindquist chart has not been already defined; if `None` the symbols are generated as (t, r, θ, ϕ) .
- `names` – (default: `None`) unused argument, except if `symbols` is not provided; it must be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<, >` is used)

OUTPUT:

- the chart of Boyer-Lindquist coordinates, as an instance of `RealDiffChart`

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.boyer_lindquist_coordinates()
Chart (M, (t, r, th, ph))
sage: latex(BH.boyer_lindquist_coordinates())
\left(M,(t, r, {\theta}, {\phi})\right)
```

The coordinate variables are returned by the square bracket operator:

```
sage: BH.boyer_lindquist_coordinates()[0]
t
sage: BH.boyer_lindquist_coordinates()[1]
r
sage: BH.boyer_lindquist_coordinates[:]
(t, r, th, ph)
```

They can also be obtained via the operator `<, >` at the same time as the chart itself:

```
sage: BLchart.<t, r, th, ph> = BH.boyer_lindquist_coordinates()
sage: BLchart
Chart (M, (t, r, th, ph))
sage: type(ph)
<type 'sage.symbolic.expression.Expression'>
```

Actually, `BLchart.<t, r, th, ph> = BH.boyer_lindquist_coordinates()` is a shortcut for:

```
sage: BLchart = BH.boyer_lindquist_coordinates()
sage: t, r, th, ph = BLchart[:]
```

The coordinate symbols can be customized:

```
sage: BH = KerrBH(a)
sage: BH.boyer_lindquist_coordinates(symbols=r"T R Th:\Theta Ph:\Phi")
Chart (M, (T, R, Th, Ph))
sage: latex(BH.boyer_lindquist_coordinates())
\left(M,(T, R, {\Theta}, {\Phi})\right)
```

cauchy_horizon_radius()

Return the value of the Boyer-Lindquist coordinate r at the inner horizon (Cauchy horizon).

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.inner_horizon_radius()
m - sqrt(-a^2 + m^2)
```

An alias is `cauchy_horizon_radius()`:

```
sage: BH.cauchy_horizon_radius()
m - sqrt(-a^2 + m^2)
```

event_horizon_radius()

Return the value of the Boyer-Lindquist coordinate r at the black hole event horizon.

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.outer_horizon_radius()
m + sqrt(-a^2 + m^2)
```

An alias is `event_horizon_radius()`:

```
sage: BH.event_horizon_radius()
m + sqrt(-a^2 + m^2)
```

The horizon radius of the Schwarzschild black hole:

```
sage: assume(m>0)
sage: KerrBH(0, m).event_horizon_radius()
2*m
```

The horizon radius of the extreme Kerr black hole ($a = m$):

```
sage: KerrBH(m, m).event_horizon_radius()
m
```

geodesic(*parameter_range*, *initial_point*, *pt0=None*, *pr0=None*, *pth0=None*, *pph0=None*, *mu=None*, *E=None*, *L=None*, *Q=None*, *r_increase=True*, *th_increase=True*, *chart=None*, *name=None*, *latex_name=None*, *a_num=None*, *m_num=None*, *verbose=False*)

Construct a geodesic on `self`.

INPUT:

- `parameter_range` – range of the affine parameter λ , as a pair (`lambda_min`, `lambda_max`)
- `initial_point` – point of Kerr spacetime from which the geodesic is to be integrated
- `pt0` – (default: `None`) Boyer-Lindquist component p^t of the initial 4-momentum vector
- `pr0` – (default: `None`) Boyer-Lindquist component p^r of the initial 4-momentum vector

- `pth0` – (default: None) Boyer-Lindquist component p^θ of the initial 4-momentum vector
- `pph0` – (default: None) Boyer-Lindquist component p^ϕ of the initial 4-momentum vector
- `mu` – (default: None) mass μ of the particle
- `E` – (default: None) conserved energy E of the particle
- `L` – (default: None) conserved angular momentum L of the particle
- `Q` – (default: None) Carter constant Q of the particle
- `r_increase` – (default: True) boolean; if True, the initial value of $p^r = dr/d\lambda$ determined from the integral of motions is positive or zero, otherwise, p^r is negative
- `th_increase` – (default: True) boolean; if True, the initial value of $p^\theta = d\theta/d\lambda$ determined from the integral of motions is positive or zero, otherwise, p^θ is negative
- `chart` – (default: None) chart on the spacetime manifold in terms of which the geodesic equations are expressed; if None the default chart (Boyer-Lindquist coordinates) is assumed
- `name` – (default: None) string; symbol given to the geodesic
- `latex_name` – (default: None) string; LaTeX symbol to denote the geodesic; if none is provided, `name` will be used
- `a_num` – (default: None) numerical value of the Kerr spin parameter a (required for a numerical integration)
- `m_num` – (default: None) numerical value of the Kerr mass parameter m (required for a numerical integration)
- `verbose` – (default: False) boolean; determines whether some information is printed during the construction of the geodesic

OUTPUT:

- an instance of `KerrGeodesic`

EXAMPLES:

See `KerrGeodesic`.

`inner_horizon_radius()`

Return the value of the Boyer-Lindquist coordinate r at the inner horizon (Cauchy horizon).

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.inner_horizon_radius()
m - sqrt(-a^2 + m^2)
```

An alias is `cauchy_horizon_radius()`:

```
sage: BH.cauchy_horizon_radius()
m - sqrt(-a^2 + m^2)
```

`isco_radius(retrograde=False)`

Return the Boyer-Lindquist radial coordinate of the innermost stable circular orbit (ISCO) in the equatorial plane.

INPUT:

- `retrograde` – (default: `False`) boolean determining whether retrograde or prograde (direct) orbits are considered

OUTPUT:

- Boyer-Lindquist radial coordinate r of the innermost stable circular orbit in the equatorial plane

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.isco_radius()
m*(sqrt(((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3) + 1)^2
+ 3*a^2/m^2) - sqrt(-(((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3)
+ 2*sqrt(((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3) + 1)^2
+ 3*a^2/m^2) + 4)*((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3) - 2)) + 3)
sage: BH.isco_radius(retrograde=True)
m*(sqrt(((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3) + 1)^2
+ 3*a^2/m^2) + sqrt(-(((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3)
+ 2*sqrt(((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3) + 1)^2
+ 3*a^2/m^2) + 4)*((a/m + 1)^(1/3) + (-a/m + 1)^(1/3))*(-a^2/m^2 + 1)^(1/3) - 2)) + 3)
sage: KerrBH(0.5).isco_radius() # tol 1.0e-13
4.23300252953083
sage: KerrBH(0.9).isco_radius() # tol 1.0e-13
2.32088304176189
sage: KerrBH(0.98).isco_radius() # tol 1.0e-13
1.61402966763547
```

ISCO in Schwarzschild spacetime:

```
sage: KerrBH(0, m).isco_radius()
6*m
```

ISCO in extreme Kerr spacetime ($a = m$):

```
sage: KerrBH(m, m).isco_radius()
m
sage: KerrBH(m, m).isco_radius(retrograde=True)
9*m
```

`map_to_Euclidean()`

Map from Kerr spacetime to the Euclidean 4-space, based on Boyer-Lindquist coordinates

`marginally_bound_orbit_radius(retrograde=False)`

Return the Boyer-Lindquist radial coordinate of the marginally bound circular orbit in the equatorial plane.

INPUT:

- `retrograde` – (default: `False`) boolean determining whether retrograde or prograde (direct) orbits are considered

OUTPUT:

- Boyer-Lindquist radial coordinate r of the marginally bound circular orbit in the equatorial plane

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.marginally_bound_orbit_radius()
-a + 2*sqrt(-a + m)*sqrt(m) + 2*m
sage: BH.marginally_bound_orbit_radius(retrograde=True)
a + 2*sqrt(a + m)*sqrt(m) + 2*m
```

Marginally bound orbit in Schwarzschild spacetime:

```
sage: KerrBH(0, m).marginally_bound_orbit_radius()
4*m
```

Marginally bound orbits in extreme Kerr spacetime ($a = m$):

```
sage: KerrBH(m, m).marginally_bound_orbit_radius()
m
sage: KerrBH(m, m).marginally_bound_orbit_radius(retrograde=True)
2*sqrt(2)*m + 3*m
```

mass()

Return the (ADM) mass of the black hole.

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.mass()
m
sage: KerrBH(a).mass()
1
```

metric()

Return the metric tensor.

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.metric()
Lorentzian metric g on the Kerr spacetime M
sage: BH.metric().display()
g = -(a^2*cos(th)^2 - 2*m*r + r^2)/(a^2*cos(th)^2 + r^2) dt⊗dt
- 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dt⊗dph
+ (a^2*cos(th)^2 + r^2)/(a^2 - 2*m*r + r^2) dr⊗dr
+ (a^2*cos(th)^2 + r^2) dth⊗dth
- 2*a*m*r*sin(th)^2/(a^2*cos(th)^2 + r^2) dph⊗dt
+ (2*a^2*m*r*sin(th)^4 + (a^2*r^2 + r^4 + (a^4 + a^2*r^2)*cos(th)^2)*sin(th)^2)/(a^2*cos(th)^2 + r^2) dph⊗dph
```

The Schwarzschild metric:

```
sage: KerrBH(0, m).metric().display()
g = (2*m - r)/r dt⊗dt - r/(2*m - r) dr⊗dr + r^2 dth⊗dth
+ r^2*sin(th)^2 dph⊗dph
```

orbital_angular_velocity(r , *retrograde=False*)

Return the angular velocity on a circular orbit.

The angular velocity Ω on a circular orbit of Boyer-Lindquist radial coordinate r around a Kerr black hole of parameters (m, a) is given by the formula

$$\Omega := \frac{d\phi}{dt} = \pm \frac{m^{1/2}}{r^{3/2} \pm am^{1/2}} \quad (1.1)$$

where (t, ϕ) are the Boyer-Lindquist time and azimuthal coordinates and \pm is $+$ (resp. $-$) for a prograde (resp. retrograde) orbit.

INPUT:

- `r` – Boyer-Lindquist radial coordinate r of the circular orbit
- `retrograde` – (default: `False`) boolean determining whether the orbit is retrograde or prograde

OUTPUT:

- Angular velocity Ω computed according to Eq. (1.1)

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m, r = var('a m r')
sage: BH = KerrBH(a, m)
sage: BH.orbital_angular_velocity(r)
sqrt(m)/(a*sqrt(m) + r^(3/2))
sage: BH.orbital_angular_velocity(r, retrograde=True)
sqrt(m)/(a*sqrt(m) - r^(3/2))
sage: KerrBH(0.9).orbital_angular_velocity(4.) # tol 1.0e-13
0.112359550561798
```

Orbital angular velocity around a Schwarzschild black hole:

```
sage: KerrBH(0, m).orbital_angular_velocity(r)
sqrt(m)/r^(3/2)
```

Orbital angular velocity on the prograde ISCO of an extreme Kerr black hole ($a = m$):

```
sage: EKBH = KerrBH(m, m)
sage: EKBH.orbital_angular_velocity(EKBH.isco_radius())
1/2/m
```

`orbital_frequency(r, retrograde=False)`

Return the orbital frequency of a circular orbit.

The frequency f of a circular orbit of Boyer-Lindquist radial coordinate r around a Kerr black hole of parameters (m, a) is $f = \Omega/(2\pi)$, where Ω is given by Eq. (1.1).

INPUT:

- `r` – Boyer-Lindquist radial coordinate r of the circular orbit
- `retrograde` – (default: `False`) boolean determining whether the orbit is retrograde or prograde

OUTPUT:

- orbital frequency f

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m, r = var('a m r')
sage: BH = KerrBH(a, m)
sage: BH.orbital_frequency(r)
1/2*sqrt(m)/(pi*(a*sqrt(m) + r^(3/2)))
sage: BH.orbital_frequency(r, retrograde=True)
1/2*sqrt(m)/(pi*(a*sqrt(m) - r^(3/2)))
sage: KerrBH(0.9).orbital_frequency(4.) # tol 1.0e-13
0.0178825778754939
sage: KerrBH(0.9).orbital_frequency(float(4)) # tol 1.0e-13
0.0178825778754939
```

Orbital angular velocity around a Schwarzschild black hole:

```
sage: KerrBH(0, m).orbital_frequency(r)
1/2*sqrt(m)/(pi*r^(3/2))
```

Orbital angular velocity on the prograde ISCO of an extreme Kerr black hole ($a = m$):

```
sage: EKBH = KerrBH(m, m)
sage: EKBH.orbital_frequency(EKBH.isco_radius())
1/4/(pi*m)
```

For numerical values, the outcome depends on the type of the entry:

```
sage: KerrBH(0).orbital_frequency(6)
1/72*sqrt(6)/pi
sage: KerrBH(0).orbital_frequency(6.) # tol 1.0e-13
0.0108291222393566
sage: KerrBH(0).orbital_frequency(RealField(200)(6)) # tol 1e-50
0.010829122239356612609803722920461899457548152312961017043180
sage: KerrBH(0.5).orbital_frequency(RealField(200)(6)) # tol 1.0e-13
0.0104728293495021
sage: KerrBH._clear_cache_() # to remove the BH object created with a=0.5
sage: KerrBH(RealField(200)(0.5)).orbital_frequency(RealField(200)(6)) # tol 1.0e-50
0.010472829349502111962146754433990790738415624921109392616237
```

outer_horizon_radius()

Return the value of the Boyer-Lindquist coordinate r at the black hole event horizon.

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.outer_horizon_radius()
m + sqrt(-a^2 + m^2)
```

An alias is `event_horizon_radius()`:

```
sage: BH.event_horizon_radius()
m + sqrt(-a^2 + m^2)
```

The horizon radius of the Schwarzschild black hole:

```
sage: assume(m>0)
sage: KerrBH(0, m).event_horizon_radius()
2*m
```

The horizon radius of the extreme Kerr black hole ($a = m$):

```
sage: KerrBH(m, m).event_horizon_radius()
m
```

photon_orbit_radius(*retrograde=False*)

Return the Boyer-Lindquist radial coordinate of the circular orbit of photons in the equatorial plane.

INPUT:

- `retrograde` – (default: `False`) boolean determining whether retrograde or prograde (direct) orbits are considered

OUTPUT:

- Boyer-Lindquist radial coordinate r of the circular orbit of photons in the equatorial plane

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
```

(continues on next page)

(continued from previous page)

```
sage: BH = KerrBH(a, m)
sage: BH.photon_orbit_radius()
2*m*(cos(2/3*arccos(-a/m)) + 1)
sage: BH.photon_orbit_radius(retrograde=True)
2*m*(cos(2/3*arccos(a/m)) + 1)
```

Photon orbit in Schwarzschild spacetime:

```
sage: KerrBH(0, m).photon_orbit_radius()
3*m
```

Photon orbits in extreme Kerr spacetime ($a = m$):

```
sage: KerrBH(m, m).photon_orbit_radius()
m
sage: KerrBH(m, m).photon_orbit_radius(retrograde=True)
4*m
```

roche_limit_radius(*rho*, *rho_unit*='solar', *mass_bh*=None, *k_rot*=0, *r_min*=None, *r_max*=50)

Evaluate the orbital radius of the Roche limit for a star of given density and rotation state.

The *Roche limit* is defined as the orbit at which the star fills its Roche volume (cf. [roche_volume\(\)](#)).

INPUT:

- **rho** – mean density of the star, in units of **rho_unit**
- **rho_unit** – (default: 'solar') string specifying the unit in which **rho** is provided; allowed values are
 - 'solar': density of the sun ($1.41 \times 10^3 \text{ kg m}^{-3}$)
 - 'SI': SI unit (kg m^{-3})
 - 'M⁻²': inverse square of the black hole mass M (geometrized units)
- **mass_bh** – (default: None) black hole mass M in solar masses; must be set if **rho_unit** = 'solar' or 'SI'
- **k_rot** – (default: 0) rotational parameter $k_\omega := \omega/\Omega$, where ω is the angular velocity of the star (assumed to be a rigid rotator) with respect to some inertial frame and Ω is the orbital angular velocity (cf. [roche_volume\(\)](#))
- **r_min** – (default: None) lower bound for the search of the Roche limit radius, in units of M ; if none is provided, the value of r at the prograde ISCO is used
- **r_max** – (default: 50) upper bound for the search of the Roche limit radius, in units of M

OUTPUT:

- Boyer-Lindquist radial coordinate r of the circular orbit at which the star fills its Roche volume, in units of the black hole mass M

EXAMPLES:

Roche limit of a non-rotating solar type star around a Schwarzschild black hole of mass equal to that of Sgr A* ($4.1 \cdot 10^6 M_\odot$):

```
sage: from kerrgeodesic_gw import KerrBH
sage: BH = KerrBH(0)
sage: BH.roche_limit_radius(1, mass_bh=4.1e6) # tol 1.0e-13
34.23653024850463
```

Instead of providing the density in solar units (the default), we can provide it in SI units (kg m^{-3}):

```
sage: BH.roche_limit_radius(1.41e3, rho_unit='SI', mass_bh=4.1e6) # tol 1.0e-13
34.23517310503541
```

or in geometrized units (M^{-2}), in which case it is not necessary to specify the black hole mass:

```
sage: BH.roche_limit_radius(3.84e-5, rho_unit='M^{-2}') # tol 1.0e-13
34.22977166547967
```

Case of a corotating star:

```
sage: BH.roche_limit_radius(1, mass_bh=4.1e6, k_rot=1) # tol 1.0e-13
37.72150497210367
```

Case of a brown dwarf:

```
sage: BH.roche_limit_radius(131., mass_bh=4.1e6) # tol 1.0e-13
7.3103232747243165
sage: BH.roche_limit_radius(131., mass_bh=4.1e6, k_rot=1) # tol 1.0e-13
7.858389409707688
```

Case of a white dwarf:

```
sage: BH.roche_limit_radius(1.1e6, mass_bh=4.1e6, r_min=0.1) # tol 1.0e-13
0.2848049914201514
sage: BH.roche_limit_radius(1.1e6, mass_bh=4.1e6, k_rot=1, r_min=0.1) # tol 1.0e-13
0.3264724605157346
```

Roche limits around a rapidly rotating black hole:

```
sage: BH = KerrBH(0.999)
sage: BH.roche_limit_radius(1, mass_bh=4.1e6) # tol 1.0e-13
34.250609907563984
sage: BH.roche_limit_radius(1, mass_bh=4.1e6, k_rot=1) # tol 1.0e-13
37.72350054417335
sage: BH.roche_limit_radius(64.2, mass_bh=4.1e6) # tol 1.0e-13
8.74356702311824
sage: BH.roche_limit_radius(64.2, mass_bh=4.1e6, k_rot=1) # tol 1.0e-13
9.575613156232857
```

roche_volume($r0, k_rot$)

Roche volume of a star on a given circular orbit.

The Roche volume depends on the rotational parameter

$$k_\omega := \frac{\omega}{\Omega}$$

where ω is the angular velocity of the star (assumed to be a rigid rotator) with respect to some inertial frame and Ω is the orbital angular velocity (cf. `orbital_angular_velocity()`). The Roche volume is computed according to formulas based on the Kerr metric and established by Dai & Blandford, Mon. Not. Roy. Astron. Soc. **434**, 2948 (2013), doi:10.1093/mnras/stt1209.

INPUT:

- `r0` – Boyer-Lindquist radial coordinate r of the circular orbit, in units of the black hole mass
- `k_rot` – rotational parameter k_ω defined above

OUTPUT:

- the dimensionless quantity $V_R/(\mu M^2)$, where V_R is the Roche volume, μ the mass of the star and M the mass of the central black hole

EXAMPLES:

Roche volume around a Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import KerrBH
sage: BH = KerrBH(0)
sage: BH.roche_volume(6, 0) # tol 1.0e-13
98.49600000000001
```

Comparison with Eq. (26) of Dai & Blandford (2013) doi:10.1093/mnras/stt1209:

```
sage: _ - 0.456*6^3 # tol 1.0e-13
0.000000000000000
```

Case $k_\omega = 1$:

```
sage: BH.roche_volume(6, 1) # tol 1.0e-13
79.145115913556
```

Comparison with Eq. (26) of Dai & Blandford (2013):

```
sage: _ - 0.456/(1+1/4.09)^6^3 # tol 1.0e-13
0.000000000000000
```

Newtonian limit:

```
sage: BH.roche_volume(1000, 0) # tol 1.0e-13
681633403.5759227
```

Comparison with Eq. (10) of Dai & Blandford (2013):

```
sage: _ - 0.683*1000^3 # tol 1.0e-13
-1.36659642407727e6
```

Roche volume around a rapidly rotating Kerr black hole:

```
sage: BH = KerrBH(0.9)
sage: rI = BH.isco_radius()
sage: BH.roche_volume(rI, 0) # tol 1.0e-13
5.70065302837734
```

Comparison with Eq. (26) of Dai & Blandford (2013):

```
sage: _ - 0.456*rI^3 # tol 1.0e-13
0.000000000000000
```

Case $k_\omega = 1$:

```
sage: BH.roche_volume(rI, 1) # tol 1.0e-13
4.58068190295940
```

Comparison with Eq. (26) of Dai & Blandford (2013):

```
sage: _ - 0.456/(1+1/4.09)*rI^3 # tol 1.0e-13
0.000000000000000
```

Newtonian limit:

```
sage: BH.roche_volume(1000, 0) # tol 1.0e-13
6.81881451514361e8
```

Comparison with Eq. (10) of Dai & Blandford (2013):

```
sage: _ - 0.683*1000^3 # tol 1.0e-13
-1.11854848563898e6
```

spin()

Return the reduced angular momentum of the black hole.

EXAMPLES:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a, m = var('a m')
sage: BH = KerrBH(a, m)
sage: BH.spin()
a
```

An alias is `angular_momentum()`:

```
sage: BH.angular_momentum()
a
```


GEODESICS IN KERR SPACETIME

Geodesics of Kerr spacetime are implemented via the class *KerrGeodesic*.

```
class kerrgeodesic_gw.kerr_geodesic.KerrGeodesic(parent, initial_point, pt0=None, pr0=None,
        pth0=None, pph0=None, mu=None, E=None,
        L=None, Q=None, r_increase=True,
        th_increase=True, chart=None, name=None,
        latex_name=None, a_num=None, m_num=None,
        verbose=False)
```

Bases: `sage.manifolds.differentiable.integrated_curve.IntegratedGeodesic`

Geodesic of Kerr spacetime.

Geodesics are computed by solving the geodesic equation via the generic [SageMath geodesic integrator](#).

INPUT:

- `parent` – `IntegratedGeodesicSet`, the set of curves $\text{Hom}_{\text{geodesic}}(I, M)$ to which the geodesic belongs
- `initial_point` – point of Kerr spacetime from which the geodesic is to be integrated
- `pt0` – (default: `None`) Boyer-Lindquist component p^t of the initial 4-momentum vector
- `pr0` – (default: `None`) Boyer-Lindquist component p^r of the initial 4-momentum vector
- `pth0` – (default: `None`) Boyer-Lindquist component p^θ of the initial 4-momentum vector
- `pph0` – (default: `None`) Boyer-Lindquist component p^ϕ of the initial 4-momentum vector
- `mu` – (default: `None`) mass μ of the particle
- `E` – (default: `None`) conserved energy E of the particle
- `L` – (default: `None`) conserved angular momentum L of the particle
- `Q` – (default: `None`) Carter constant Q of the particle
- `r_increase` – (default: `True`) boolean; if `True`, the initial value of $p^r = dr/d\lambda$ determined from the integral of motions is positive or zero, otherwise, p^r is negative
- `th_increase` – (default: `True`) boolean; if `True`, the initial value of $p^\theta = d\theta/d\lambda$ determined from the integral of motions is positive or zero, otherwise, p^θ is negative
- `chart` – (default: `None`) chart on the spacetime manifold in terms of which the geodesic equations are expressed; if `None` the default chart (Boyer-Lindquist coordinates) is assumed
- `name` – (default: `None`) string; symbol given to the geodesic
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the geodesic; if none is provided, `name` will be used
- `a_num` – (default: `None`) numerical value of the Kerr spin parameter a (required for a numerical integration)

- `m_num` – (default: `None`) numerical value of the Kerr mass parameter m (required for a numerical integration)
- `verbose` – (default: `False`) boolean; determines whether some information is printed during the construction of the geodesic

EXAMPLES:

We construct first the Kerr spacetime:

```
sage: from kerrgeodesic_gw import KerrBH
sage: a = var('a')
sage: M = KerrBH(a); M
Kerr spacetime M
sage: BLchart = M.boyer_lindquist_coordinates(); BLchart
Chart (M, (t, r, th, ph))
```

We pick an initial spacetime point for the geodesic:

```
sage: init_point = M((0, 6, pi/2, 0), name='P')
```

A geodesic is constructed by providing the range of the affine parameter, the initial point and either (i) the Boyer-Lindquist components $(p_0^t, p_0^r, p_0^\theta, p_0^\phi)$ of the initial 4-momentum vector $p_0 = \left. \frac{dx}{d\lambda} \right|_{\lambda=0}$, (ii) the four integral of motions (μ, E, L, Q) or (iii) some of the components of p_0 along with with some integrals of motion. We shall also specify some numerical value for the Kerr spin parameter a . Examples of (i) and (iii) are provided below. Here, we choose $\lambda \in [0, 300m]$, the option (ii) and $a = 0.998m$, where m is the black hole mass:

```
sage: geod = M.geodesic([0, 300], init_point, mu=1, E=0.883,
....:                  L=1.982, Q=0.467, a_num=0.998)
sage: geod
Geodesic of the Kerr spacetime M
```

The numerical integration of the geodesic equation is performed via `integrate()`, by providing the step in $\delta\lambda$ in units of m :

```
sage: geod.integrate(step=0.005)
```

We can then plot the geodesic:

```
sage: geod.plot()
Graphics3d Object
```

Actually, many options can be passed to `plot()`. For instance to get a 3D spacetime diagram:

```
sage: geod.plot(coordinates='txy')
Graphics3d Object
```

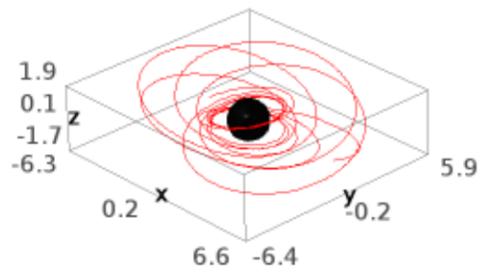
or to get the trace of the geodesic in the (x, y) plane:

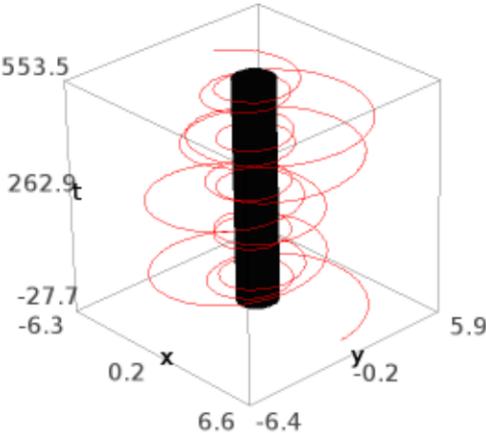
```
sage: geod.plot(coordinates='xy', plot_points=2000)
Graphics object consisting of 2 graphics primitives
```

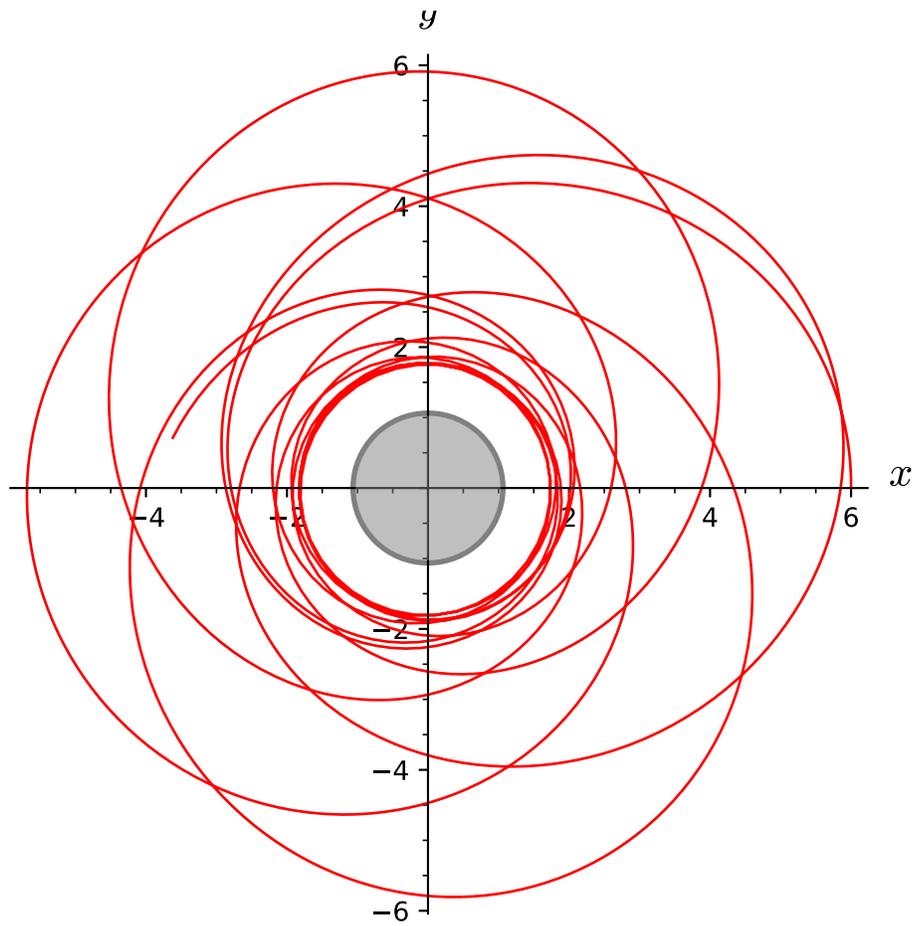
or else to get the trace in the (x, z) plane:

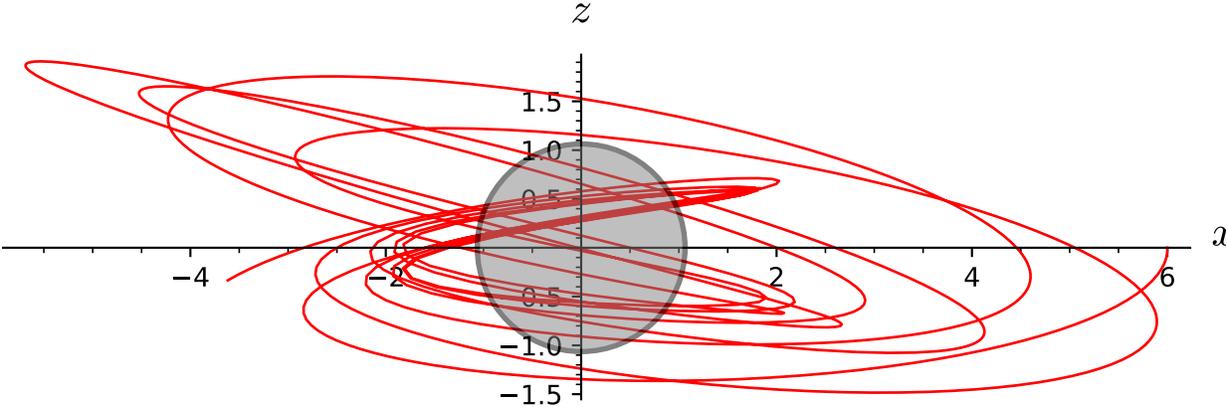
```
sage: geod.plot(coordinates='xz')
Graphics object consisting of 2 graphics primitives
```

As a curve, the geodesic is a map from an interval of \mathbb{R} to the spacetime M :









```
sage: geod.display()
(0, 300) → M
sage: geod.domain()
Real interval (0, 300)
sage: geod.codomain()
Kerr spacetime M
```

It maps values of λ to spacetime points:

```
sage: geod(0)
Point on the Kerr spacetime M
sage: geod(0).coordinates() # coordinates in the default chart # tol 1.0e-13
(0.0, 6.0, 1.5707963267948966, 0.0)
sage: BLchart(geod(0)) # equivalent to above # tol 1.0e-13
(0.0, 6.0, 1.5707963267948966, 0.0)
sage: geod(300).coordinates() # tol 1.0e-13
(553.4637326813786, 3.703552505462962, 1.6613834863942039, 84.62814710987239)
```

The initial 4-momentum vector p_0 is returned by the method `initial_tangent_vector()`:

```
sage: p0 = geod.initial_tangent_vector(); p0
Tangent vector p at Point P on the Kerr spacetime M
sage: p0 in M.tangent_space(init_point)
True
sage: p0.display() # tol 1.0e-13
p = 1.29225788954106 ∂/∂t + 0.00438084990626460 ∂/∂r
+ 0.0189826106258554 ∂/∂th + 0.0646134478134985 ∂/∂ph
sage: p0[:] # tol 1.0e-13
[1.29225788954106, 0.00438084990626460, 0.0189826106258554, 0.0646134478134985]
```

For instance, the components p_0^t and p_0^ϕ are recovered by:

```
sage: p0[0], p0[3] # tol 1.0e-13
(1.29225788954106, 0.0646134478134985)
```

Let us check that the scalar square of p_0 is -1 , i.e. is consistent with the mass parameter $\mu = 1$ used in the construction of the geodesic:

```
sage: g = M.metric()
sage: g.at(init_point)(p0, p0).subs(a=0.998) # tol 1.0e-13
-1.000000000000000
```

The 4-momentum vector p at any value of the affine parameter λ , e.g. $\lambda = 200m$, is obtained by:

```
sage: p = geod.evaluate_tangent_vector(200); p
Tangent vector at Point on the Kerr spacetime M
sage: p in M.tangent_space(geod(200))
True
sage: p.display() # tol 1.0e-13
1.316592599498746 ∂/∂t - 0.07370434215844164 ∂/∂r
- 0.01091195426423706 ∂/∂th + 0.07600209768075264 ∂/∂ph
```

The particle mass μ computed at a given value of λ is returned by the method `evaluate_mu()`:

```
sage: geod.evaluate_mu(0) # tol 1.0e-13
1.000000000000000
```

Of course, it should be conserved along the geodesic; actually it is, up to the numerical accuracy:

```
sage: geod.evaluate_mu(300) # tol 1.0e-13
1.0000117978600134
```

Similarly, the conserved energy E , conserved angular momentum L and Carter constant Q are computed at any value of λ by respectively `evaluate_E()`, `evaluate_L()` and `evaluate_Q()`:

```
sage: geod.evaluate_E(0) # tol 1.0e-13
0.8830000000000000
sage: geod.evaluate_L(0) # tol 1.0e-13
1.9820000000000000
sage: geod.evaluate_Q(0) # tol 1.0e-13
0.4670000000000000
```

Let us check that the values of μ , E , L and Q evaluated at $\lambda = 300m$ are equal to those at $\lambda = 0$ up to the numerical accuracy of the integration scheme:

```
sage: geod.check_integrals_of_motion(300) # tol 1.0e-13
quantity      value      initial value      diff.      relative diff.
 $\mu$           1.0000235958592163  1.0000000000000000  0.00002360  0.00002360
E             0.883067996080701  0.8830000000000000  0.00006800  0.00007701
L             1.98248080818931  1.9820000000000000  0.0004808  0.0002426
Q             0.467214137649741  0.4670000000000000  0.0002141  0.0004585
```

Decreasing the integration step leads to smaller errors:

```
sage: geod.integrate(step=0.001)
sage: geod.check_integrals_of_motion(300) # tol 1.0e-13
quantity      value      initial value      diff.      relative diff.
 $\mu$           1.0000047183936422  1.0000000000000000  4.718e-6    4.718e-6
E             0.883013604456676  0.8830000000000000  0.00001360  0.00001541
L             1.98209626120918  1.9820000000000000  0.00009626  0.00004857
Q             0.467042771975860  0.4670000000000000  0.00004277  0.00009159
```

Various ways to initialize a geodesic

Instead of providing the integral of motions, as for `geod` above, one can initialize a geodesic by providing the Boyer-Lindquist components $(p_0^t, p_0^r, p_0^\theta, p_0^\phi)$ of the initial 4-momentum vector p_0 . For instance:

```
sage: p0
Tangent vector p at Point P on the Kerr spacetime M
sage: p0[:] # tol 1.0e-13
[1.29225788954106, 0.00438084990626460, 0.0189826106258554, 0.0646134478134985]
sage: geod2 = M.geodesic([0, 300], init_point, pt0=p0[0], pr0=p0[1],
....:                    pth0=p0[2], pph0=p0[3], a_num=0.998)
sage: geod2.initial_tangent_vector() == p0
True
```

As a check, we recover the same values of (μ, E, L, Q) as those that were used to initialize `geod`:

```
sage: geod2.evaluate_mu(0)
1.0000000000000000
sage: geod2.evaluate_E(0)
0.8830000000000000
sage: geod2.evaluate_L(0)
1.9820000000000000
sage: geod2.evaluate_Q(0)
0.4670000000000000
```

We may also initialize a geodesic by providing the mass μ and the three spatial components $(p_0^r, p_0^\theta, p_0^\phi)$ of the initial 4-momentum vector:

```
sage: geod3 = M.geodesic([0, 300], init_point, mu=1, pr0=p0[1],
....:                    pth0=p0[2], pph0=p0[3], a_num=0.998)
```

The component p_0^t is then automatically computed:

```
sage: geod3.initial_tangent_vector()[:] # tol 1.0e-13
[1.29225788954106, 0.00438084990626460, 0.0189826106258554, 0.0646134478134985]
```

and we check the identity with the initial vector of `geod`, up to numerical errors:

```
sage: (geod3.initial_tangent_vector() - p0)[:] # tol 1.0e-13
[2.22044604925031e-16, 0, 0, 0]
```

Another way to initialize a geodesic is to provide the conserved energy E , the conserved angular momentum L and the two components (p_0^r, p_0^θ) of the initial 4-momentum vector:

```
sage: geod4 = M.geodesic([0, 300], init_point, E=0.8830, L=1.982,
....:                  pr0=p0[1], pth0=p0[2], a_num=0.998)
sage: geod4.initial_tangent_vector()[:]
[1.29225788954106, 0.00438084990626460, 0.0189826106258554, 0.0646134478134985]
```

Again, we get a geodesic equivalent to `geod`:

```
sage: (geod4.initial_tangent_vector() - p0)[:] # tol 1.0e-13
[0, 0, 0, 0]
```

check_integrals_of_motion(*affine_parameter*, *solution_key=None*)

Check the constancy of the four integrals of motion

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for evaluating the various integrals of motion; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- a `SageMath` table with the absolute and relative differences with respect to the initial values.

evaluate_E(*affine_parameter*, *solution_key=None*)

Compute the conserved energy E at a given value of the affine parameter λ .

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for the evaluation; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- value of E

evaluate_L(*affine_parameter*, *solution_key=None*)

Compute the conserved angular momentum about the rotation axis L at a given value of the affine parameter λ .

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for the evaluation; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- value of L

evaluate_Q(*affine_parameter*, *solution_key=None*)

Compute the Carter constant Q at a given value of the affine parameter λ .

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for the evaluation; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- value of Q

evaluate_mu(*affine_parameter*, *solution_key=None*)

Compute the mass parameter μ at a given value of the affine parameter λ .

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for the evaluation; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- value of μ

evaluate_mu2(*affine_parameter*, *solution_key=None*)

Compute the square of the mass parameter μ^2 at a given value of the affine parameter λ .

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for the evaluation; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- value of μ^2

evaluate_tangent_vector(*affine_parameter*, *solution_key=None*)

Return the tangent vector (4-momentum) at a given value of the affine parameter.

INPUT:

- `affine_parameter` – value of the affine parameter λ
- `solution_key` – (default: `None`) string denoting the numerical solution to use for the evaluation; if `None`, the latest solution computed by `integrate()` is used.

OUTPUT:

- instance of `TangentVector` representing the 4-momentum vector p at λ

initial_tangent_vector()

Return the initial 4-momentum vector.

OUTPUT:

- instance of `TangentVector` representing the initial 4-momentum p_0

EXAMPLES:

Initial 4-momentum vector of a null geodesic in Schwarzschild spacetime:

```

sage: from kerrgeodesic_gw import KerrBH
sage: M = KerrBH(0)
sage: BLchart = M.boyer_lindquist_coordinates()
sage: init_point = M((0, 6, pi/2, 0), name='P')
sage: geod = M.geodesic([0, 100], init_point, mu=0, E=1,
....:                  L=3, Q=0)
sage: p0 = geod.initial_tangent_vector(); p0
Tangent vector p at Point P on the Schwarzschild spacetime M
sage: p0.display()
p = 3/2 ∂/∂t + 1/6*sqrt(30) ∂/∂r + 1/12 ∂/∂ph

```

integrate(*step=None, method='odeint', solution_key=None, parameters_values=None, verbose=False, **control_param*)

Solve numerically the geodesic equation.

INPUT:

- **step** – (default: None) step $\delta\lambda$ for the integration, where λ is the affine parameter along the geodesic; default value is a hundredth of the range of λ declared when constructing the geodesic
- **method** – (default: 'odeint') numerical scheme to use for the integration; available algorithms are:
 - 'odeint' - makes use of `scipy.integrate.odeint` via Sage solver `desolve_odeint()`; `odeint` invokes the LSODA algorithm of the `ODEPACK suite`, which automatically selects between implicit Adams method (for non-stiff problems) and a method based on backward differentiation formulas (BDF) (for stiff problems).
 - 'rk4_maxima' - 4th order classical Runge-Kutta, which makes use of Maxima's dynamics package via Sage solver `desolve_system_rk4()` (quite slow)
 - 'dopri5' - Dormand-Prince Runge-Kutta of order (4)5 provided by `scipy.integrate.ode`
 - 'dop853' - Dormand-Prince Runge-Kutta of order 8(5,3) provided by `scipy.integrate.ode`

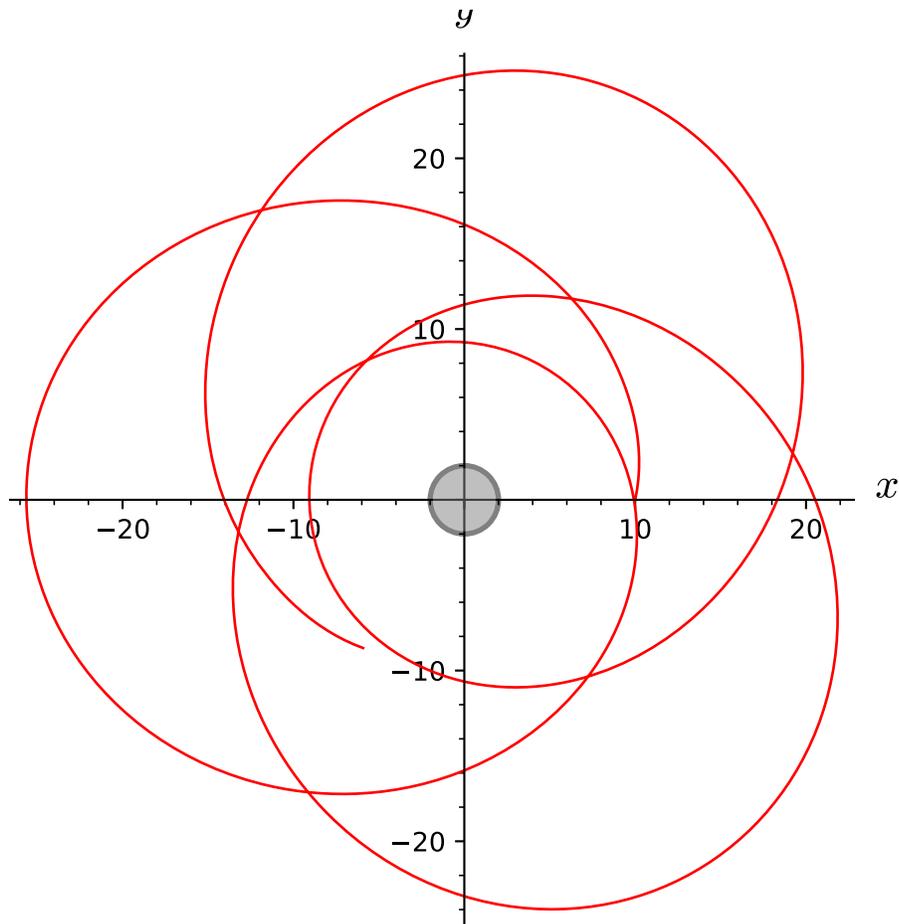
and those provided by GSL via Sage class `ode_solver`:

- 'rk2' - embedded Runge-Kutta (2,3)
- 'rk4' - 4th order classical Runge-Kutta
- 'rkf45' - Runge-Kutta-Fehlberg (4,5)
- 'rkck' - embedded Runge-Kutta-Cash-Karp (4,5)
- 'rk8pd' - Runge-Kutta Prince-Dormand (8,9)
- 'rk2imp' - implicit 2nd order Runge-Kutta at Gaussian points
- 'rk4imp' - implicit 4th order Runge-Kutta at Gaussian points
- 'gear1' - $M = 1$ implicit Gear
- 'gear2' - $M = 2$ implicit Gear
- 'bsimp' - implicit Bulirsch-Stoer (requires Jacobian)
- **solution_key** – (default: None) string to tag the numerical solution; if None, the string method is used.
- **parameters_values** – (default: None) list of numerical values of the parameters present in the system defining the geodesic, to be substituted in the equations before integration
- **verbose** – (default: False) prints information about the computation in progress
- ****control_param** – extra control parameters to be passed to the chosen solver

EXAMPLES:

Bound timelike geodesic in Schwarzschild spacetime:

```
sage: from kerrgeodesic_gw import KerrBH
sage: M = KerrBH(0)
sage: BLchart = M.boyer_lindquist_coordinates()
sage: init_point = M((0, 10, pi/2, 0), name='P')
sage: lmax = 1500.
sage: geod = M.geodesic([0, lmax], init_point, mu=1, E=0.973,
....:                  L=4.2, Q=0)
sage: geod.integrate()
sage: geod.plot(coordinates='xy')
Graphics object consisting of 2 graphics primitives
```



With the default integration step, the accuracy is not very good:

```
sage: geod.check_integrals_of_motion(lmax) # tol 1.0e-13
```

quantity	value	initial value	diff.	relative diff.
μ^2	1.000761704316941	1.000000000000000	0.0007617	0.0007617
E	0.9645485805304451	0.973000000000000	-0.008451	-0.008686
L	3.8897905637080923	4.200000000000000	-0.3102	-0.07386
Q	5.673017835722329e-32	0	5.673e-32	-

Let us improve it by specifying a smaller integration step:

```
sage: geod.integrate(step=0.1)
sage: geod.check_integrals_of_motion(lmax) # tol 1.0e-13
```

(continues on next page)

(continued from previous page)

quantity	value	initial value	diff.	relative diff.
μ^2	1.0000101879128696	1.0000000000000000	0.00001019	0.00001019
E	0.9729448260004574	0.9730000000000000	-0.00005517	-0.00005671
L	4.197973829219027	4.2000000000000000	-0.002026	-0.0004824
Q	6.607560764960032e-32	0	6.608e-32	-

We may set the parameter `solution_key` to keep track of various numerical solutions:

```
sage: geod.integrate(step=0.1, solution_key='step_0.1')
sage: geod.integrate(step=0.02, solution_key='step_0.02')
```

and use it in the various evaluation functions:

```
sage: geod.evaluate_mu(lmax, solution_key='step_0.1') # tol 1.0e-13
1.0000050939434606
sage: geod.evaluate_mu(lmax, solution_key='step_0.02') # tol 1.0e-13
1.0000010212056811
```

plot(*coordinates*='xyz', *prange*=None, *solution_key*=None, *style*='-', *thickness*=1, *plot_points*=1000, *color*='red', *include_end_point*=(True, True), *end_point_offset*=(0.001, 0.001), *verbose*=False, *label_axes*=None, *plot_horizon*=True, *horizon_color*='black', *fill_BH_region*=True, *BH_region_color*='grey', *display_tangent*=False, *color_tangent*='blue', *plot_points_tangent*=10, *width_tangent*=1, *scale*=1, *aspect_ratio*='automatic', ***kws*)

Plot the geodesic in terms of the coordinates (t, x, y, z) deduced from the Boyer-Lindquist coordinates (t, r, θ, ϕ) via the standard polar to Cartesian transformations.

INPUT:

- `coordinates` – (default: 'xyz') string indicating which of the coordinates (t, x, y, z) to use for the plot
- `prange` – (default: None) range of the affine parameter λ for the plot; if None, the entire range declared during the construction of the geodesic is considered
- `solution_key` – (default: None) string denoting the numerical solution to use for the plot; if None, the latest solution computed by `integrate()` is used.
- `verbose` – (default: False) determines whether information is printed about the plotting in progress
- `plot_horizon` – (default: True) determines whether the black hole event horizon is drawn
- `horizon_color` – (default: 'black') color of the event horizon
- `fill_BH_region` – (default: True) determines whether the black hole region is colored (for 2D plots only)
- `BH_region_color` – (default: 'grey') color of the event horizon
- `display_tangent` – (default: False) determines whether some tangent vectors should also be plotted
- `color_tangent` – (default: blue) color of the tangent vectors when these are plotted
- `plot_points_tangent` – (default: 10) number of tangent vectors to display when these are plotted
- `width_tangent` – (default: 1) sets the width of the arrows representing the tangent vectors
- `scale` – (default: 1) scale applied to the tangent vectors before displaying them

See also:

[DifferentiableCurve.plot](#) for the other input parameters

OUTPUT:

- either a 2D graphic object (2 coordinates specified in the parameter coordinates) or a 3D graphic object (3 coordinates in coordinates)

SPHERICAL AND SPHEROIDAL HARMONICS

3.1 Spin-weighted spherical harmonics

Spin-weighted spherical harmonics ${}_s Y_l^m(\theta, \phi)$

`kerrgeodesic_gw.spin_weighted_spherical_harm.spin_weighted_spherical_harmonic`(*s*, *l*, *m*, *theta*,
phi, *con-*
don_shortley=True,
cached=True,
numeri-
cal=None)

Return the spin-weighted spherical harmonic of spin weight *s* and indices (*l*, *m*).

INPUT:

- *s* – integer; the spin weight
- *l* – non-negative integer; the harmonic degree
- *m* – integer within the range $[-1, 1]$; the azimuthal number
- *theta* – colatitude angle
- *phi* – azimuthal angle
- *con_**don_**shortley* – (default: `True`) determines whether the Condon-Shortley phase of $(-1)^m$ is taken into account (see below)
- *cached* – (default: `True`) determines whether the result shall be cached; setting *cached* to `False` forces a new computation, without caching the result
- *numerical* – (default: `None`) determines whether a symbolic or a numerical computation of a given type is performed; allowed values are
 - `None`: the type of computation is deduced from the type of *theta*
 - `RDF`: Sage’s machine double precision floating-point numbers (`RealDoubleField`)
 - `RealField(n)`, where *n* is a number of bits: Sage’s floating-point numbers with an arbitrary precision; note that `RR` is a shortcut for `RealField(53)`.
 - `float`: Python’s floating-point numbers

OUTPUT:

- the value of ${}_s Y_l^m(\theta, \phi)$, either as a symbolic expression or as floating-point complex number of the type determined by *numerical*

ALGORITHM:

The spin-weighted spherical harmonic is evaluated according to Eq. (3.1) of J. N. Golberg et al., J. Math. Phys. **8**, 2155 (1967) [doi:10.1063/1.1705135], with an extra $(-1)^m$ factor (the so-called *Condon-Shortley phase*) if `condon_shortley` is `True`, the actual formula being then the one given in Wikipedia article [Spin-weighted_spherical_harmonics#Calculating](#)

EXAMPLES:

```
sage: from kerrgeodesic_gw import spin_weighted_spherical_harmonic
sage: theta, phi = var('theta phi')
sage: spin_weighted_spherical_harmonic(-2, 2, 1, theta, phi)
1/4*(sqrt(5)*cos(theta) + sqrt(5))*e^(I*phi)*sin(theta)/sqrt(pi)
sage: spin_weighted_spherical_harmonic(-2, 2, 1, theta, phi,
....:                                     condon_shortley=False)
-1/4*(sqrt(5)*cos(theta) + sqrt(5))*e^(I*phi)*sin(theta)/sqrt(pi)
sage: spin_weighted_spherical_harmonic(-2, 2, 1, pi/3, pi/4)
(3/32*I + 3/32)*sqrt(5)*sqrt(3)*sqrt(2)/sqrt(pi)
```

Evaluation as floating-point numbers: the type of the output is deduced from the input:

```
sage: spin_weighted_spherical_harmonic(-2, 2, 1, 1.0, 2.0) # tol 1.0e-13
-0.170114676286891 + 0.371707349012686*I
sage: parent(_)
Complex Field with 53 bits of precision
sage: spin_weighted_spherical_harmonic(-2, 2, 1, RDF(2.0), RDF(3.0)) # tol 1.0e-13
-0.16576451879564585 + 0.023629159118690464*I
sage: parent(_)
Complex Double Field
sage: spin_weighted_spherical_harmonic(-2, 2, 1, float(3.0), float(4.0)) # tol 1.0e-13
(-0.0002911423884400524-0.00033709085352998027j)
sage: parent(_)
<type 'complex'>
```

Computation with arbitrary precision are possible (here 100 bits):

```
sage: R100 = RealField(100); R100
Real Field with 100 bits of precision
sage: spin_weighted_spherical_harmonic(-2, 2, 1, R100(1.5), R100(2.0)) # tol 1.0e-28
-0.14018136537676185317636108802 + 0.30630187143465275236861476906*I
```

Even when the entry is symbolic, numerical evaluation can be enforced via the argument `numerical`. For instance, setting `numerical` to `RDF` (SageMath's Real Double Field):

```
sage: spin_weighted_spherical_harmonic(-2, 2, 1, pi/3, pi/4, numerical=RDF) # tol 1.0e-13
0.2897056515173923 + 0.28970565151739225*I
sage: parent(_)
Complex Double Field
```

One can also use `numerical=RR` (SageMath's Real Field with precision set to 53 bits):

```
sage: spin_weighted_spherical_harmonic(-2, 2, 1, pi/3, pi/4, numerical=RR) # tol 1.0e-13
0.289705651517392 + 0.289705651517392*I
sage: parent(_)
Complex Field with 53 bits of precision
```

Another option is to use Python floats:

```
sage: spin_weighted_spherical_harmonic(-2, 2, 1, pi/3, pi/4, numerical=float) # tol 1.0e-13
(0.28970565151739225+0.2897056515173922j)
sage: parent(_)
<type 'complex'>
```

One can go beyond double precision, for instance using 100 bits of precision:

```
sage: spin_weighted_spherical_harmonic(-2, 2, 1, pi/3, pi/4,
....:                                     numerical=RealField(100)) # tol 1.0e-28
0.28970565151739218525664455148 + 0.28970565151739218525664455148*I
sage: parent(_)
Complex Field with 100 bits of precision
```

3.2 Spin-weighted spheroidal harmonics

Spin-weighted spheroidal harmonics

`kerrgeodesic_gw.spin_weighted_spheroidal_harm.spin_weighted_spheroidal_eigenvalue`(*s*, *l*, *m*, *gamma*, *verbose*, *bose=False*, *cached=True*, *min_nmax=8*)

Return the spin-weighted oblate spheroidal eigenvalue of spin weight *s*, degree *l*, azimuthal order *m* and spheroidicity *gamma*.

INPUT:

- *s* – integer; the spin weight
- *l* – non-negative integer; the harmonic degree
- *m* – integer within the range $[-1, 1]$; the azimuthal number
- *gamma* – spheroidicity parameter γ
- *verbose* – (default: `False`) determines whether some details of the computation are printed out
- *cached* – (default: `True`) determines whether the eigenvalue and the eigenvectors shall be cached; setting *cached* to `False` forces a new computation, without caching the result
- *min_nmax* – (default: 8) integer; floor for the evaluation of the parameter *nmax*, which sets the highest degree of the spherical harmonic expansion as $l+nmax$.

OUTPUT:

- eigenvalue λ related to the eigenvalue $\mathcal{E}_{\ell m}$ of the spheroidal harmonic by

$$\lambda = \mathcal{E}_{\ell m} - 2m\gamma + \gamma^2 - s(s+1)$$

ALGORITHM:

The method is adapted from that exposed in Appendix A of S.A. Hughes, Phys. Rev. D **61**, 084004 (2000) [doi:10.1103/PhysRevD.61.084004].

EXAMPLES:

```
sage: from kerrgeodesic_gw import spin_weighted_spheroidal_eigenvalue
sage: spin_weighted_spheroidal_eigenvalue(-2, 2, 1, 1.2) # tol 1.0e-13
0.5167945263162421
sage: spin_weighted_spheroidal_eigenvalue(-2, 2, 1, 1.2, cached=False) # tol 1.0e-13
0.5167945263162421
sage: spin_weighted_spheroidal_eigenvalue(-2, 2, 1, 0)
4.0
```

`kerrgeodesic_gw.spin_weighted_spheroidal_harm.spin_weighted_spheroidal_harmonic`(*s*, *l*, *m*,
gamma,
theta, *phi*,
ver-
bose=False,
cached=True,
min_nmax=8)

Return the spin-weighted oblate spheroidal harmonic of spin weight *s*, degree *l*, azimuthal order *m* and spheroidicity *gamma*.

INPUT:

- *s* – integer; the spin weight
- *l* – non-negative integer; the harmonic degree
- *m* – integer within the range $[-l, l]$; the azimuthal number
- *gamma* – spheroidicity parameter
- *theta* – colatitude angle
- *phi* – azimuthal angle
- *verbose* – (default: `False`) determines whether some details of the computation are printed out
- *cached* – (default: `True`) determines whether the eigenvectors shall be cached; setting *cached* to `False` forces a new computation, without caching the result
- *min_nmax* – (default: 8) integer; floor for the evaluation of the parameter *nmax*, which sets the highest degree of the spherical harmonic expansion as $l+nmax$.

OUTPUT:

- value of ${}_s S_{lm}^\gamma(\theta, \phi)$

ALGORITHM:

The spin-weighted oblate spheroidal harmonics are computed by an expansion over spin-weighted *spherical* harmonics, the coefficients of the expansion being obtained by solving an eigenvalue problem, as exposed in Appendix A of S.A. Hughes, Phys. Rev. D **61**, 084004 (2000) [doi:10.1103/PhysRevD.61.084004].

EXAMPLES:

```
sage: from kerrgeodesic_gw import spin_weighted_spheroidal_harmonic
sage: spin_weighted_spheroidal_harmonic(-2, 2, 2, 1.1, pi/2, 0) # tol 1.0e-13
0.08702532727529422
sage: spin_weighted_spheroidal_harmonic(-2, 2, 2, 1.1, pi/3, pi/3) # tol 1.0e-13
-0.14707166027821453 + 0.25473558795537715*I
sage: spin_weighted_spheroidal_harmonic(-2, 2, 2, 1.1, pi/3, pi/3, cached=False) # tol 1.0e-13
-0.14707166027821453 + 0.25473558795537715*I
sage: spin_weighted_spheroidal_harmonic(-2, 2, 2, 1.1, pi/3, pi/4) # tol 1.0e-13
1.801108380050024e-17 + 0.2941433205564291*I
sage: spin_weighted_spheroidal_harmonic(-2, 2, -1, 1.1, pi/3, pi/3, cached=False) # tol 1.0e-13
0.11612826056899399 - 0.20114004750009495*I
```

Test that the relation ${}_s S_{lm}^\gamma(\theta, \phi) = (-1)^{l+s} {}_s S_{l,-m}^{-\gamma}(\pi - \theta, -\phi)$ [cf. Eq. (2.3) of Arxiv 1810.00432], which is used to evaluate ${}_s S_{lm}^\gamma(\theta, \phi)$ when $m < 0$ and *cached* is `True`, is correctly implemented:

```
sage: spin_weighted_spheroidal_harmonic(-2, 2, -2, 1.1, pi/3, pi/3) # tol 1.0e-13
-0.04097260436590737 - 0.07096663248016997*I
sage: abs(_ - spin_weighted_spheroidal_harmonic(-2, 2, -2, 1.1, pi/3, pi/3,
.....:                                     cached=False)) < 1.e-13
True
```

(continues on next page)

(continued from previous page)

```
sage: spin_weighted_spheroidal_harmonic(-2, 3, -1, 1.1, pi/3, pi/3) # tol 1.0e-13
0.1781880511506843 - 0.3086307578946672*I
sage: abs(_ - spin_weighted_spheroidal_harmonic(-2, 3, -1, 1.1, pi/3, pi/3,
.....:                                     cached=False)) < 1.e-13
True
```


AMPLITUDE FACTORS $Z_{\ell M}^{\infty}(R)$

Functions $Z_{\ell m}^{\infty}(r)$ giving the amplitude of the gravitational radiation emitted by a particle on a circular orbit at radius r about a Kerr black hole.

REFERENCES:

- S. A. Teukolsky, *Astrophys. J.* **185**, 635 (1973)
- S. Detweiler, *Astrophys. J.* **225**, 687 (1978)
- M. Shibata, *Phys. Rev. D* **50**, 6297 (1994)
- D. Kennefick, *Phys. Rev. D* **58**, 064012 (1998)
- S. A. Hughes, *Phys. Rev. D* **61**, 084004 (2000) [doi:10.1103/PhysRevD.61.084004]
- E. Gourgoulhon, A. Le Tiec, F. Vincent, N. Warburton, *Astron. Astrophys.* **627**, A92 (2019) [doi:10.1051/0004-6361/201935406]; [Arxiv 1903.02049](https://arxiv.org/abs/1903.02049)

`kerrgeodesic_gw.zinf.Zinf(a, l, m, r, algorithm='spline')`

Amplitude factor of the mode (ℓ, m) .

The factor $Z_{\ell m}^{\infty}(r)$ is obtained by spline interpolation of tabulated numerical solutions of the radial component of the Teukolsky equation.

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `l` – integer ≥ 2 ; the harmonic degree ℓ
- `m` – integer within the range $[-1, 1]$; the azimuthal number m
- `r` – areal radius of the orbit (in units of M)
- `algorithm` – (default: 'spline') string describing the computational method; allowed values are
 - 'spline': spline interpolation of tabulated data
 - '1.5PN' (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, *Phys. Rev. D* **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497], with a minus one factor accounting for a different convention for the metric signature.

OUTPUT:

- coefficient $Z_{\ell m}^{\infty}(r)$ (in units of M^{-2})

EXAMPLES:

```
sage: from kerrgeodesic_gw import Zinf
sage: Zinf(0.98, 2, 2, 1.7) # tol 1.0e-13
-0.04302234478778856 + 0.28535368610053824*I
sage: Zinf(0., 2, 2, 10.) # tol 1.0e-13
0.0011206407919254163 - 0.0003057608384581628*I
sage: Zinf(0., 2, 2, 10., algorithm='1.5PN') # tol 1.0e-13
0.0011971529546749354 - 0.0003551610880408921*I
```

`kerrgeodesic_gw.zinf.Zinf_Schwarzschild_PN(l, m, r)`

Amplitude factor of the mode (ℓ, m) for a Schwarzschild BH at the 1.5PN level.

The 1.5PN formulas are taken from E. Poisson, Phys. Rev. D **47**, 1497 (1993), doi:10.1103/PhysRevD.47.1497.

INPUT:

- l – integer ≥ 2 ; the harmonic degree ℓ
- m – integer within the range $[-1, 1]$; the azimuthal number m
- r – areal radius of the orbit (in units of M , the BH mass)

OUTPUT:

- coefficient $Z_{\ell m}^{\infty}(r)$ (in units of M^{-2})

EXAMPLES:

```
sage: from kerrgeodesic_gw import Zinf_Schwarzschild_PN
sage: Zinf_Schwarzschild_PN(2, 2, 6.) # tol 1.0e-13
-0.00981450418730346 + 0.003855681972781947*I
sage: Zinf_Schwarzschild_PN(5, 3, 6.) # tol 1.0e-13
-6.958527913913504e-05*I
```

GRAVITATIONAL RADIATION BY A SINGLE PARTICLE

Gravitational radiation by a particle on a circular orbit around a Kerr black hole

The gravitational wave emitted by a particle of mass μ on a circular orbit in the equatorial plane of a Kerr black hole of mass M and angular momentum parameter a is given by the formula:

$$h_+ - ih_\times = \frac{2\mu}{r} \sum_{\ell=2}^{\infty} \sum_{\substack{m=-\ell \\ m \neq 0}}^{\ell} \frac{Z_{\ell m}^{\infty}(r_0)}{(m\omega_0)^2} {}_{-2}S_{\ell m}^{am\omega_0}(\theta, \phi) e^{-im\phi_0} e^{-im\omega_0(t-r_*)} \quad (5.1)$$

where

- $h_+ = h_+(t, r, \theta, \phi)$ and $h_\times = h_\times(t, r, \theta, \phi)$, (t, r, θ, ϕ) being the Boyer-Lindquist coordinates of the observer
- r_* is the tortoise coordinate corresponding to r
- r_0 is the Boyer-Lindquist radius of the particle's orbit
- ϕ_0 is some constant phase factor
- ω_0 is the orbital angular velocity
- $Z_{\ell m}^{\infty}(r_0)$ is a solution of the radial component of the Teukolsky equation (cf. `Zinf()`)
- ${}_{-2}S_{\ell m}^{am\omega_0}(\theta, \phi)$ is the spin-weighted spheroidal harmonic of weight -2 (cf. `spin_weighted_spheroidal_harmonic()`)

According to Eq. (5.1), the Fourier-series expansion of the waveform (h_+, h_\times) received at the location (t, r, θ, ϕ) is

$$h_{+,\times}(t, r, \theta, \phi) = \sum_{m=1}^{+\infty} [A_m^{+,\times}(r, \theta) \cos(m\psi) + B_m^{+,\times}(r, \theta) \sin(m\psi)], \quad (5.2)$$

where

$$\psi := \omega_0(t - r_*) - \phi + \phi_0,$$

ω_0 being the orbital angular velocity of the particle and r_* the tortoise coordinate corresponding to r .

Note that the dependence of the Fourier coefficients $A_m^{+,\times}(r, \theta)$ and $B_m^{+,\times}(r, \theta)$ with respect to r is simply μ/r , where μ is the particle's mass, i.e. we may consider the following rescaled Fourier coefficients, which depend on θ only:

$$\bar{A}_m^{+,\times}(\theta) := \frac{r}{\mu} A_m^{+,\times}(r, \theta) \quad \text{and} \quad \bar{B}_m^{+,\times}(\theta) := \frac{r}{\mu} B_m^{+,\times}(r, \theta)$$

According to Eqs. (5.1) and (5.2), we have

$$\bar{A}_m^+(\theta) = \frac{2}{(m\omega_0)^2} \sum_{\ell=2}^{\infty} \text{Re}(Z_{\ell m}^{\infty}(r_0)) \left[(-1)^\ell {}_{-2}S_{\ell, -m}^{-am\omega_0}(\theta, 0) + {}_{-2}S_{\ell m}^{am\omega_0}(\theta, 0) \right] \quad (5.3)$$

$$\bar{B}_m^+(\theta) = \frac{2}{(m\omega_0)^2} \sum_{\ell=2}^{\infty} \text{Im}(Z_{\ell m}^{\infty}(r_0)) \left[(-1)^{\ell} {}_{-2}S_{\ell, -m}^{-am\omega_0}(\theta, 0) + {}_{-2}S_{\ell m}^{am\omega_0}(\theta, 0) \right] \quad (5.4)$$

$$\bar{A}_m^{\times}(\theta) = \frac{2}{(m\omega_0)^2} \sum_{\ell=2}^{\infty} \text{Im}(Z_{\ell m}^{\infty}(r_0)) \left[(-1)^{\ell} {}_{-2}S_{\ell, -m}^{-am\omega_0}(\theta, 0) - {}_{-2}S_{\ell m}^{am\omega_0}(\theta, 0) \right] \quad (5.5)$$

$$\bar{B}_m^{\times}(\theta) = \frac{2}{(m\omega_0)^2} \sum_{\ell=2}^{\infty} \text{Re}(Z_{\ell m}^{\infty}(r_0)) \left[(-1)^{\ell+1} {}_{-2}S_{\ell, -m}^{-am\omega_0}(\theta, 0) + {}_{-2}S_{\ell m}^{am\omega_0}(\theta, 0) \right] \quad (5.6)$$

This module implements the following functions:

- `h_plus_particle()`: evaluates rh_{+}/μ via Eq. (5.2)
- `h_cross_particle()`: evaluates rh_{\times}/μ via Eq. (5.2)
- `h_particle_quadrupole()`: evaluates rh_{+}/μ or rh_{\times}/μ at the quadrupole approximation
- `h_plus_particle_fourier()`: evaluates rA_m^+/μ and rB_m^+/μ via Eqs. (5.3)-(5.4)
- `h_cross_particle_fourier()`: evaluates rA_m^{\times}/μ and rB_m^{\times}/μ via Eqs. (5.5)-(5.6)
- `h_amplitude_particle_fourier()`: evaluates $(r/\mu)\sqrt{(A_m^+)^2 + (B_m^+)^2}$ and $(r/\mu)\sqrt{(A_m^{\times})^2 + (B_m^{\times})^2}$
- `h_particle_signal()`: time sequence of rh_{+}/μ or rh_{\times}/μ
- `plot_h_particle()`: plot rh_{+}/μ and/or rh_{\times}/μ in terms of the retarded time
- `plot_spectrum_particle()`: plot $(r/\mu)\sqrt{(A_m^{+, \times})^2 + (B_m^{+, \times})^2}$ in terms of m
- `radiated_power_particle()`: total radiated power (gravitational luminosity)
- `secular_frequency_change()`: change in orbital frequency $\dot{\omega}_0/\omega_0$ due to the gravitational radiation reaction
- `decay_time()`: time for the orbit to shrink to an orbit of given radius

REFERENCES:

- S. A. Teukolsky, *Astrophys. J.* **185**, 635 (1973)
- S. Detweiler, *Astrophys. J.* **225**, 687 (1978)
- M. Shibata, *Phys. Rev. D* **50**, 6297 (1994)
- D. Kennefick, *Phys. Rev. D* **58**, 064012 (1998)
- S. A. Hughes, *Phys. Rev. D* **61**, 084004 (2000) [doi:10.1103/PhysRevD.61.084004]
- E. Gourgoulhon, A. Le Tiec, F. Vincent, N. Warburton, *Astron. Astrophys.* **627**, A92 (2019) [doi:10.1051/0004-6361/201935406]; Arxiv 1903.02049

`kerrgeodesic_gw.gw_particle.decay_time(a, r_init, r_final, l_max=None, m_min=1, approximation=None, quad_epsrel=1e-06, quad_limit=500)`

Return the time spent in the migration from a circular orbit of radius `r_init` to that of radius `r_final`, induced by gravitational radiation reaction.

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `r_init` – Boyer-Lindquist radius of the initial orbit (in units of M)
- `r_final` – Boyer-Lindquist radius of the final orbit (in units of M)
- `l_max` – (default: `None`) upper bound in the summation over the harmonic degree ℓ ; if `None`, `l_max` is determined automatically from the available tabulated data
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m

- `approximation` – (default: `None`) string describing the computational method; allowed values are
 - `None`: exact computation
 - `'1.5PN'` (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]
 - `'quadrupole'` (only for `a=0`): quadrupole approximation (0-post-Newtonian); see `h_particle_quadrupole()`
- `quad_epsrel` – (default: `1.e-6`) relative error tolerance in the computation of the integral giving the decay time
- `quad_limit` – (default: `500`) upper bound on the number of subintervals used in the adaptive algorithm to compute the integral (this corresponds to the argument `limit` of SciPy's function `quad`)

OUTPUT:

- rescaled decay time $T(\mu/M^2)$, where M is the BH mass and μ the mass of the orbiting particle.

EXAMPLES:

Time to migrate from $r_0 = 10M$ to $r_0 = 6M$ around a Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import decay_time
sage: decay_time(0, 10, 6) # tol 1.0e-13
90.80876605028857
```

Let us check that at large radius, there is a good agreement with the quadrupole formula:

```
sage: decay_time(0, 50, 6) # tol 1.0e-13
121893.29664651724
sage: decay_time(0, 50, 6, approximation='quadrupole') # tol 1.0e-13
122045.0
```

`kerrgeodesic_gw.gw_particle.h_amplitude_particle_fourier`($m, a, r0, theta, l_max=10,$
algorithm_Zinf='spline')

Return the amplitude Fourier mode of a given order m of the rescaled gravitational wave emitted by a particle in circular orbit around a Kerr black hole.

The rescaled Fourier mode of order m received at the location (t, r, θ, ϕ) is

$$\frac{r}{\mu} h_m^{+, \times} = \bar{A}_m^{+, \times} \cos(m\psi) + \bar{B}_m^{+, \times} \sin(m\psi)$$

where μ is the particle mass and $\psi := \omega_0(t - r_*) - \phi$, ω_0 being the orbital frequency of the particle and r_* the tortoise coordinate corresponding to r and $\bar{A}_m^{+, \times}$ and $\bar{B}_m^{+, \times}$ are given by Eqs. (5.3)-(5.6) above.

The $+$ and \times amplitudes of the Fourier mode m are defined respectively by

$$\frac{r}{\mu} |h_m^+| := \sqrt{(\bar{A}_m^+)^2 + (\bar{B}_m^+)^2} \quad \text{and} \quad \frac{r}{\mu} |h_m^\times| := \sqrt{(\bar{A}_m^\times)^2 + (\bar{B}_m^\times)^2}$$

INPUT:

- m – positive integer defining the Fourier mode
- a – BH angular momentum parameter (in units of M , the BH mass)
- $r0$ – Boyer-Lindquist radius of the orbit (in units of M)
- $theta$ – Boyer-Lindquist colatitude θ of the observer
- l_max – (default: 10) upper bound in the summation over the harmonic degree ℓ in Eqs. (5.3)-(5.6)

- `algorithm_Zinf` – (default: 'spline') string describing the computational method for $Z_{\ell m}^{\infty}(r_0)$; allowed values are
 - 'spline': spline interpolation of tabulated data
 - '1.5PN' (only for $a=0$): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

OUTPUT:

- tuple $((r/\mu)|h_m^+|, (r/\mu)|h_m^\times|)$ (cf. the above expression)

EXAMPLE:

For a Schwarzschild black hole ($a = 0$):

```
sage: from kerrgeodesic_gw import h_amplitude_particle_fourier
sage: a = 0
sage: h_amplitude_particle_fourier(2, a, 6., pi/2) # tol 1.0e-13
(0.27875846152963557, 1.5860176188287866e-16)
sage: h_amplitude_particle_fourier(2, a, 6., pi/4) # tol 1.0e-13
(0.47180033963220214, 0.45008696580919527)
sage: h_amplitude_particle_fourier(2, a, 6., 0) # tol 1.0e-13
(0.6724377101568336, 0.6724377101568336)
sage: h_amplitude_particle_fourier(2, a, 6., pi/4, l_max=5) # tol 1.0e-13
(0.47179830286565255, 0.4500948389153302)
sage: h_amplitude_particle_fourier(2, a, 6., pi/4, l_max=5, # tol 1.0e-13
...: algorithm_Zinf='1.5PN')
(0.5381495951380861, 0.5114366815383188)
```

For a rapidly rotating Kerr black hole ($a = 0.95M$):

```
sage: a = 0.95
sage: h_amplitude_particle_fourier(2, a, 6., pi/4) # tol 1.0e-13
(0.39402068296301823, 0.37534143024659444)
sage: h_amplitude_particle_fourier(2, a, 2., pi/4) # tol 1.0e-13
(0.7358730645589858, 0.7115113031184368)
```

`kerrgeodesic_gw.gw_particle.h_cross_particle(a, r0, u, theta, phi, phi0=0, l_max=10, m_min=1, algorithm_Zinf='spline')`

Return the rescaled h_\times -part of the gravitational radiation emitted by a particle in circular orbit around a Kerr black hole.

The computation is based on Eq. (5.2) above.

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `r0` – Boyer-Lindquist radius of the orbit (in units of M)
- `u` – retarded time coordinate of the observer (in units of M): $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `phi` – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- `phi0` – (default: 0) phase factor
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `algorithm_Zinf` – (default: 'spline') string describing the computational method for $Z_{\ell m}^{\infty}(r_0)$; allowed values are

- 'spline': spline interpolation of tabulated data
- '1.5PN' (only for $a=0$): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

OUTPUT:

- the rescaled waveform $(r/\mu)h_{\times}$, where μ is the particle's mass and r is the Boyer-Lindquist radial coordinate of the observer

EXAMPLES:

Let us consider the case $a = 0$ (Schwarzschild black hole) and $r_0 = 6M$ (emission from the ISCO). For $\theta = \pi/2$, we have $h_{\times} = 0$:

```
sage: from kerrgeodesic_gw import h_cross_particle
sage: a = 0
sage: h_cross_particle(a, 6., 0., pi/2, 0.) # tol 1.0e-13
1.0041370414185673e-16
```

while for $\theta = \pi/4$, we have:

```
sage: h_cross_particle(a, 6., 0., pi/4, 0.) # tol 1.0e-13
0.275027796440582
sage: h_cross_particle(a, 6., 0., pi/4, 0., l_max=5) # tol 1.0e-13
0.2706516303570341
sage: h_cross_particle(a, 6., 0., pi/4, 0., l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
0.2625307460899205
```

For an orbit of larger radius ($r_0 = 12M$), the 1.5-post-Newtonian approximation is in better agreement with the exact computation:

```
sage: h_cross_particle(a, 12., 0., pi/4, 0.)
0.1050751824554463
sage: h_cross_particle(a, 12., 0., pi/4, 0., l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
0.10244926162224487
```

A plot of the waveform generated by a particle orbiting at the ISCO:

```
sage: hc = lambda u: h_cross_particle(a, 6., u, pi/4, 0.)
sage: plot(hc, (0, 200.), axes_labels=[r'$(t-r_*)/M$', r'$r h_{\times}/\mu$'],
....:      gridlines=True, frame=True, axes=False)
Graphics object consisting of 1 graphics primitive
```

Case $a/M = 0.95$ (rapidly rotating Kerr black hole):

```
sage: a = 0.95
sage: h_cross_particle(a, 2., 0., pi/4, 0.) # tol 1.0e-13
-0.2681353673743396
```

Assessing the importance of the mode $m = 1$:

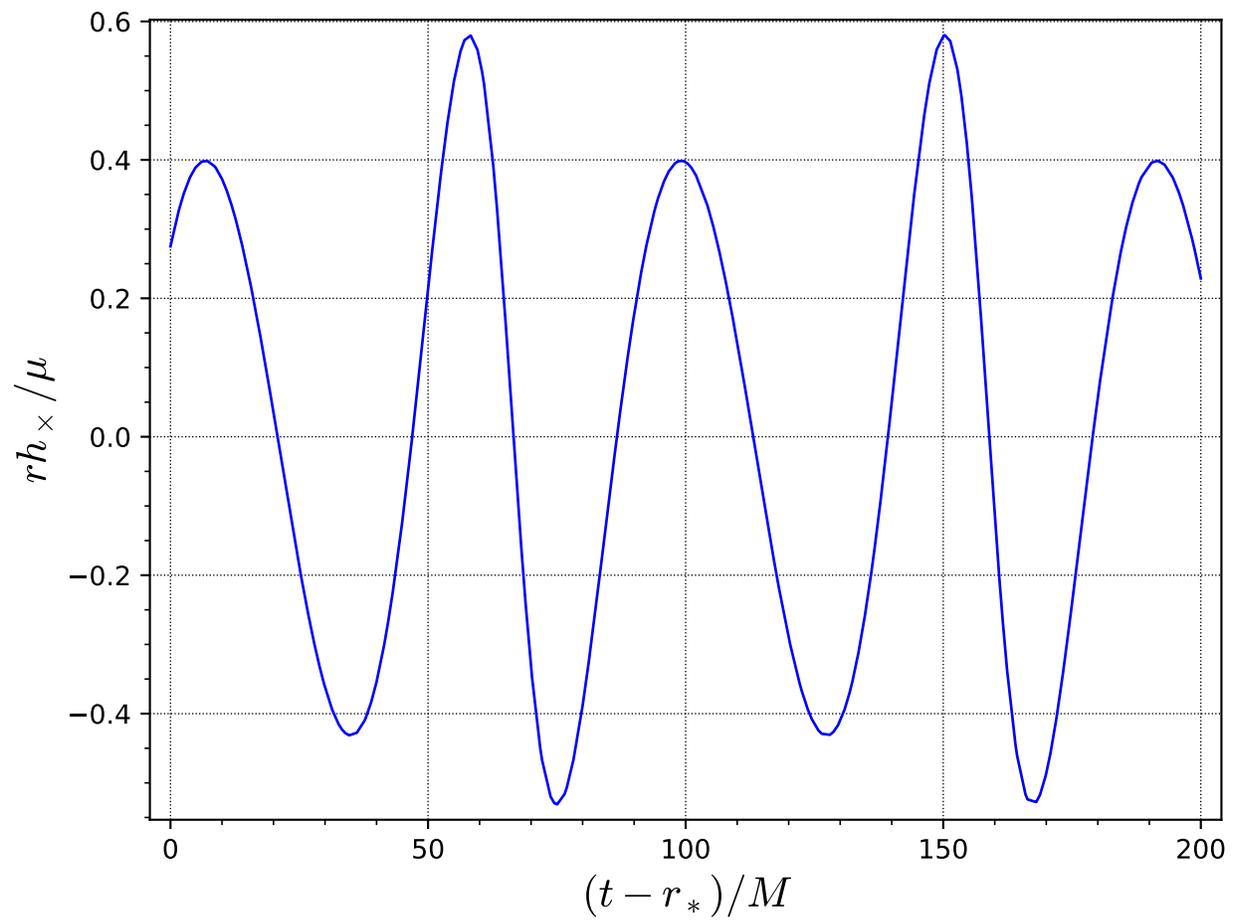
```
sage: h_cross_particle(a, 2., 0., pi/4, 0., m_min=2) # tol 1.0e-13
-0.3010579420748449
```

`kerrgeodesic_gw.gw_particle.h_cross_particle_fourier`($m, a, r0, theta, l_max=10,$
algorithm_Zinf='spline')

Return the Fourier mode of a given order m of the rescaled h_{\times} -part of the gravitational wave emitted by a particle in circular orbit around a Kerr black hole.

The rescaled Fourier mode of order m received at the location (t, r, θ, ϕ) is

$$\frac{r}{\mu} h_m^{\times} = \bar{A}_m^{\times} \cos(m\psi) + \bar{B}_m^{\times} \sin(m\psi)$$



where μ is the particle mass and $\psi := \omega_0(t - r_*) - \phi$, ω_0 being the orbital frequency of the particle and r_* the tortoise coordinate corresponding to r and \bar{A}_m^\times and \bar{B}_m^\times are given by Eqs. (5.5)-(5.6) above.

INPUT:

- `m` – positive integer defining the Fourier mode
- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `r0` – Boyer-Lindquist radius of the orbit (in units of M)
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ in Eqs. (5.5)-(5.6)
- `algorithm_Zinf` – (default: 'spline') string describing the computational method for $Z_{\ell m}^\infty(r_0)$; allowed values are
 - 'spline': spline interpolation of tabulated data
 - '1.5PN' (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

OUTPUT:

- tuple $(\bar{A}_m^\times, \bar{B}_m^\times)$

EXAMPLES:

Let us consider the case $a = 0$ first (Schwarzschild black hole), with $m = 2$:

```
sage: from kerrgeodesic_gw import h_cross_particle_fourier
sage: a = 0
```

h_m^\times is always zero in the direction $\theta = \pi/2$:

```
sage: h_cross_particle_fourier(2, a, 8., pi/2) # tol 1.0e-13
(3.444996575846961e-17, 1.118234985040581e-16)
```

Let us then evaluate h_m^\times in the direction $\theta = \pi/4$:

```
sage: h_cross_particle_fourier(2, a, 8., pi/4) # tol 1.0e-13
(0.09841144532628172, 0.31201728756415015)
sage: h_cross_particle_fourier(2, a, 8., pi/4, l_max=5) # tol 1.0e-13
(0.09841373523119075, 0.31202073689061305)
sage: h_cross_particle_fourier(2, a, 8., pi/4, l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
(0.11744823578781578, 0.34124272645755677)
```

Values of m different from 2:

```
sage: h_cross_particle_fourier(3, a, 20., pi/4) # tol 1.0e-13
(0.022251439699635174, -0.005354134279052387)
sage: h_cross_particle_fourier(3, a, 20., pi/4, l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
(0.017782177999686274, 0.0)
sage: h_cross_particle_fourier(1, a, 8., pi/4) # tol 1.0e-13
(0.03362589948237155, -0.008465651545641889)
sage: h_cross_particle_fourier(0, a, 8., pi/4)
(0, 0)
```

The case $a/M = 0.95$ (rapidly rotating Kerr black hole):

```
sage: a = 0.95
sage: h_cross_particle_fourier(2, a, 8., pi/4) # tol 1.0e-13
(0.08843838892991202, 0.28159329265206867)
sage: h_cross_particle_fourier(8, a, 8., pi/4) # tol 1.0e-13
```

(continues on next page)

(continued from previous page)

```
(-0.00014588821622195107, -8.179557811364057e-06)
sage: h_cross_particle_fourier(2, a, 2., pi/4) # tol 1.0e-13
(-0.6021994882885746, 0.3789513303450391)
sage: h_cross_particle_fourier(8, a, 2., pi/4) # tol 1.0e-13
(0.01045760329050054, -0.004986913120370192)
```

`kerrgeodesic_gw.gw_particle.h_particle_quadrupole(r0, u, theta, phi, phi0=0, mode='+')`

Return the rescaled h_+ or h_\times part of the gravitational radiation emitted by a particle in quasi-Newtonian circular orbit around a massive body, computed at the quadrupole approximation.

The computation is performed according to the following formulas:

$$h_+ = 2 \frac{\mu}{r} \frac{M}{r_0} (1 + \cos^2 \theta) \cos [2\omega_0(t - r) + 2(\phi_0 - \phi)]$$

$$h_\times = 4 \frac{\mu}{r} \frac{M}{r_0} \cos \theta \sin [2\omega_0(t - r) + 2(\phi_0 - \phi)]$$

where M is the mass of the central body, $\mu \ll M$ the mass of the orbiting particle, r_0 the orbital radius, $\omega_0 = \sqrt{M/r_0^3}$ the corresponding orbital angular velocity and (t, r, θ, ϕ) the coordinates of the observer.

INPUT:

- `r0` – radius of the orbit (in units of M , the mass of the central body)
- `u` – retarded time coordinate $u = t - r$ of the observer (in units of M)
- `theta` – colatitude θ of the observer
- `phi` – azimuthal coordinate ϕ of the observer
- `phi0` – (default: 0) phase factor
- `mode` – (default: '+') string determining which GW polarization mode is considered; allowed values are '+' and 'x', for respectively h_+ and h_\times

OUTPUT:

- the rescaled waveform $(r/\mu)h$, where μ is the particle's mass, r is the radial coordinate of the observer and h is either h_+ or h_\times (depending on the value of `mode`)

EXAMPLES:

Values of h_+ for $r_0 = 12M$:

```
sage: from kerrgeodesic_gw import (h_particle_quadrupole,
....:                               h_plus_particle, h_cross_particle)
sage: theta, phi = pi/3, 0
sage: r0 = 12.
sage: porb = n(2*pi*r0^1.5) # orbital period
sage: u = porb/16
sage: h_particle_quadrupole(r0, u, theta, phi) # tol 1.0e-13
0.1473139127471974
sage: h_plus_particle(., r0, u, theta, phi) # exact value, tol 1.0e-13
0.041745779012809306
sage: h_plus_particle(., r0, u, theta, phi, l_max=5,
....:                 algorithm_Zinf='1.5PN') # 1.5 PN approx, tol 1.0e-13
0.06396788402755646
```

Values of h_\times for $r_0 = 12M$:

```
sage: h_particle_quadrupole(r0, u, theta, phi, mode='x') # tol 1.0e-13
0.1178511301977579
sage: h_cross_particle(., r0, u, theta, phi) # exact value, tol 1.0e-13
0.1351232731503482
```

(continues on next page)

(continued from previous page)

```
sage: h_cross_particle(0., r0, u, theta, phi, l_max=5,
....:                 algorithm_Zinf='1.5PN') # 1.5 PN approx, tol 1.0e-13
0.14924631043762673
```

Values of h_+ for $r_0 = 50M$:

```
sage: r0 = 50.
sage: porb = n(2*pi*r0^1.5) # orbital period
sage: u = porb/16
sage: h_particle_quadrupole(r0, u, theta, phi) # tol 1.0e-13
0.03535533905932738
sage: h_plus_particle(0., r0, u, theta, phi, l_max=5) # exact value, tol 1.0e-13
0.024850367000986223
sage: h_plus_particle(0., r0, u, theta, phi, l_max=5,
....:                 algorithm_Zinf='1.5PN') # 1.5 PN approx, tol 1.0e-13
0.026007035506911764
```

The difference between the exact value and the one resulting from the quadrupole approximation looks large, but actually this results mostly from some dephasing, as one can see on a plot:

```
sage: hp_quad = lambda t: h_particle_quadrupole(r0, t, theta, phi)
sage: hc_quad = lambda t: h_particle_quadrupole(r0, t, theta, phi, mode='x')
sage: hp = lambda t: h_plus_particle(0., r0, t, theta, phi, l_max=5)
sage: hc = lambda t: h_cross_particle(0., r0, t, theta, phi, l_max=5)
sage: umax = 2*porb
sage: plot(hp_quad, (0, umax), legend_label=r'$h_+$ quadrupole',
....:      axes_labels=[r'$(t-r_*)/M$', r'$r h/\mu$'],
....:      title=r'$r_0=50 M$, $\theta=\pi/3$',
....:      gridlines=True, frame=True, axes=False) + \
....: plot(hp, (0, umax), color='red', legend_label=r'$h_+$ exact') + \
....: plot(hc_quad, (0, umax), linestyle='--',
....:      legend_label=r'$h_+\times$ quadrupole') + \
....: plot(hc, (0, umax), color='red', linestyle='--',
....:      legend_label=r'$h_+\times$ exact')
Graphics object consisting of 4 graphics primitives
```

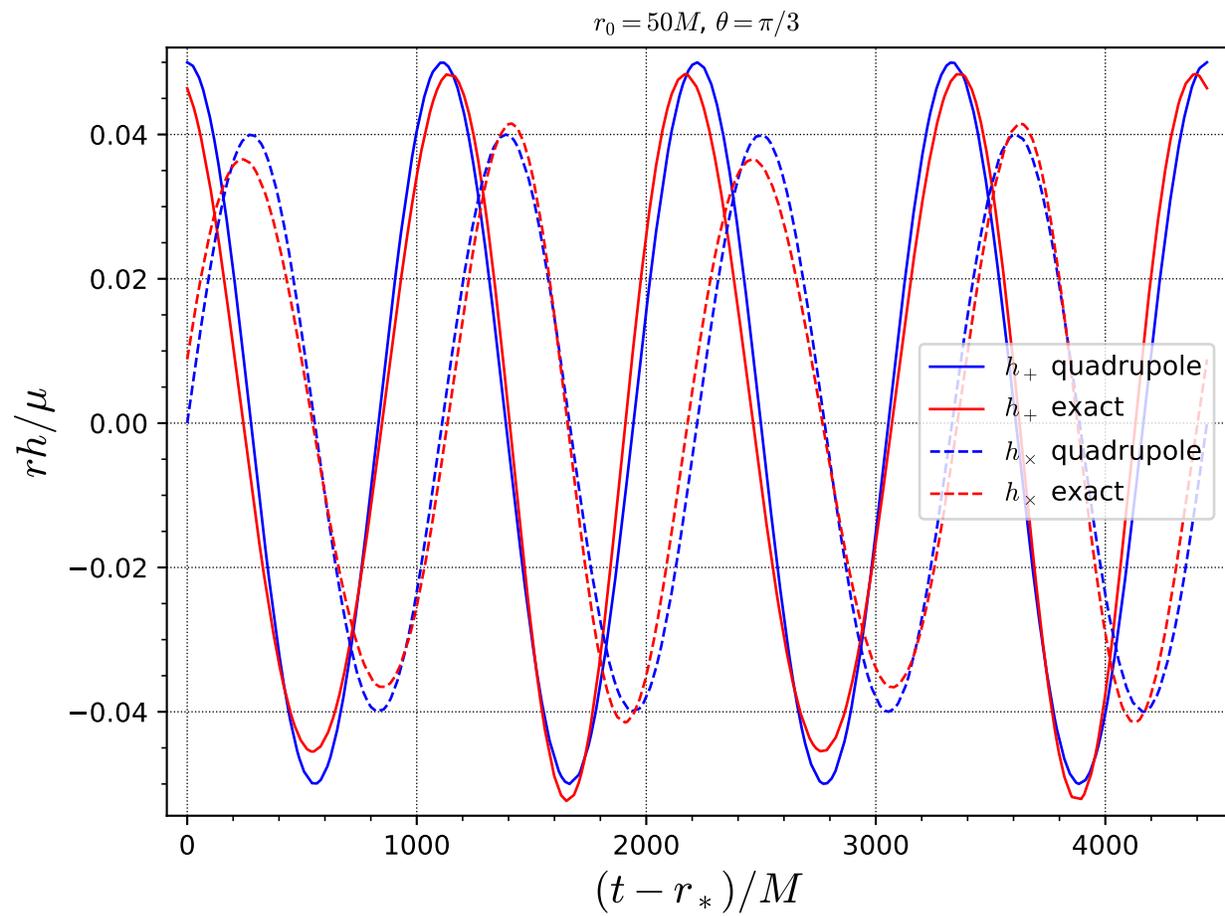
`kerrgeodesic_gw.gw_particle.h_particle_signal`(a , r_0 , θ , ϕ , u_{\min} , u_{\max} , $mode='+'$,
 $nb_points=100$, $\phi_0=0$, $l_{\max}=10$, $m_{\min}=1$,
 $approximation=None$, $store=None$)

Return a time sequence of the h_+ or the h_x part of the gravitational radiation from a particle in circular orbit around a Kerr black hole.

Note: It is more efficient to use this function than to perform a loop over `h_plus_particle()` or `h_cross_particle()`. Indeed, the Fourier modes, which involve the computation of spin-weighted spheroidal harmonics and of the functions $Z_{\ell m}^\infty(r_0)$, are evaluated once for all, prior to the loop on the retarded time u .

INPUT:

- a – BH angular momentum parameter (in units of M)
- r_0 – Boyer-Lindquist radius of the orbit (in units of M)
- θ – Boyer-Lindquist colatitude θ of the observer
- ϕ – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- u_{\min} – lower bound of the retarded time coordinate of the observer (in units of the black hole mass M):
 $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- u_{\max} – upper bound of the retarded time coordinate of the observer (in units of the black hole mass M)



- `mode` – (default: '+') string determining which GW polarization mode is considered; allowed values are '+' and 'x', for respectively h_+ and h_x
- `nb_points` – (default: 100) number of points in the interval (`u_min`, `u_max`)
- `phi0` – (default: 0) phase factor
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `approximation` – (default: None) string describing the computational method; allowed values are
 - None: exact computation
 - '1.5PN' (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [[doi:10.1103/PhysRevD.47.1497](https://doi.org/10.1103/PhysRevD.47.1497)]
 - 'quadrupole' (only for `a=0`): quadrupole approximation (0-post-Newtonian); see `h_particle_quadrupole()`
- `store` – (default: None) string containing a file name for storing the time sequence; if None, no storage is attempted

OUTPUT:

- a list of `nb_points` pairs $(u, rh/\mu)$, where u is the retarded time, h is either h_+ or h_x depending on the parameter `mode`, μ is the particle mass and r is the Boyer-Lindquist radial coordinate of the observer

EXAMPLES:

h_+ signal at $\theta = \pi/2$ from a particle at the ISCO of a Schwarzschild black hole ($a = 0$, $r_0 = 6M$):

```
sage: from kerrgeodesic_gw import h_particle_signal
sage: h_particle_signal(0., 6., pi/2, 0., 0., 200., nb_points=9) # tol 1.0e-13
[(0.0000000000000000, 0.1536656546005028),
 (25.0000000000000000, -0.2725878162016865),
 (50.0000000000000000, 0.3525164756465054),
 (75.0000000000000000, 0.047367530900643974),
 (100.0000000000000000, -0.06816472285771447),
 (125.0000000000000000, -0.10904082076122341),
 (150.0000000000000000, 0.11251491162759894),
 (175.0000000000000000, 0.2819301792449237),
 (200.0000000000000000, -0.24646401049292863)]
```

Storing the sequence in a file:

```
sage: h = h_particle_signal(0., 6., pi/2, 0., 0., 200.,
.....:                      nb_points=9, store='h_plus.d')
```

The h_x signal, for $\theta = \pi/4$:

```
sage: h_particle_signal(0., 6., pi/4, 0., 0., 200., nb_points=9, mode='x') # tol 1.0e-13
[(0.0000000000000000, 0.275027796440582),
 (25.0000000000000000, -0.18713017721920192),
 (50.0000000000000000, 0.2133141583155321),
 (75.0000000000000000, -0.531073507307601),
 (100.0000000000000000, 0.3968872953624949),
 (125.0000000000000000, -0.4154274307718398),
 (150.0000000000000000, 0.5790969355083798),
 (175.0000000000000000, -0.24074783639714234),
 (200.0000000000000000, 0.22869838143661578)]
```

`kerrgeodesic_gw.gw_particle.h_plus_particle(a, r0, u, theta, phi, phi0=0, l_max=10, m_min=1, algorithm_Zinf='spline')`

Return the rescaled h_+ -part of the gravitational radiation emitted by a particle in circular orbit around a Kerr black hole.

The computation is based on Eq. (5.2) above.

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `r0` – Boyer-Lindquist radius of the orbit (in units of M)
- `u` – retarded time coordinate of the observer (in units of M): $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `phi` – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- `phi0` – (default: 0) phase factor
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `algorithm_Zinf` – (default: 'spline') string describing the computational method for $Z_{\ell m}^\infty(r_0)$; allowed values are
 - 'spline': spline interpolation of tabulated data
 - '1.5PN' (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

OUTPUT:

- the rescaled waveform $(r/\mu)h_+$, where μ is the particle's mass and r is the Boyer-Lindquist radial coordinate of the observer

EXAMPLES:

Let us consider the case $a = 0$ (Schwarzschild black hole) and $r_0 = 6M$ (emission from the ISCO):

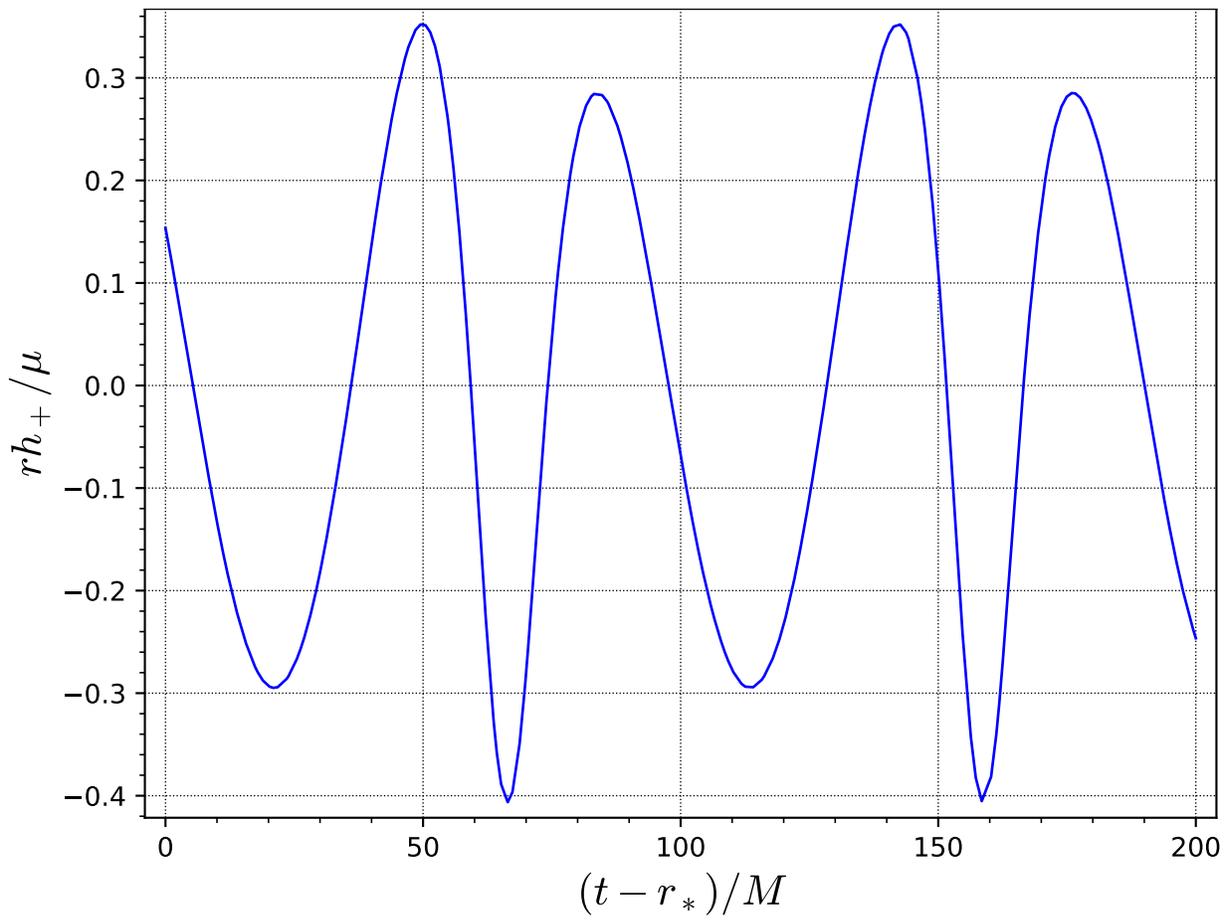
```
sage: from kerrgeodesic_gw import h_plus_particle
sage: a = 0
sage: h_plus_particle(a, 6., 0., pi/2, 0.) # tol 1.0e-13
0.1536656546005028
sage: h_plus_particle(a, 6., 0., pi/2, 0., l_max=5) # tol 1.0e-13
0.157759938177291
sage: h_plus_particle(a, 6., 0., pi/2, 0., l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
0.22583887001798497
```

For an orbit of larger radius ($r_0 = 12M$), the 1.5-post-Newtonian approximation is in better agreement with the exact computation:

```
sage: h_plus_particle(a, 12., 0., pi/2, 0.) # tol 1.0e-13
0.11031251832047866
sage: h_plus_particle(a, 12., 0., pi/2, 0., l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
0.12935832450325302
```

A plot of the waveform generated by a particle orbiting at the ISCO:

```
sage: hp = lambda u: h_plus_particle(a, 6., u, pi/2, 0.)
sage: plot(hp, (0, 200.), axes_labels=[r'$(t-r_*)/M$', r'$r h_+/\mu$'],
...:      gridlines=True, frame=True, axes=False)
Graphics object consisting of 1 graphics primitive
```



Case $a/M = 0.95$ (rapidly rotating Kerr black hole):

```
sage: a = 0.95
sage: h_plus_particle(a, 2., 0., pi/2, 0.) # tol 1.0e-13
0.20326150400852214
```

Assessing the importance of the mode $m = 1$:

```
sage: h_plus_particle(a, 2., 0., pi/2, 0., m_min=2) # tol 1.0e-13
0.21845811047370495
```

`kerrgeodesic_gw.gw_particle.h_plus_particle_fourier`($m, a, r0, theta, l_max=10,$
 $algorithm_Zinf='spline'$)

Return the Fourier mode of a given order m of the rescaled h_+ -part of the gravitational wave emitted by a particle in circular orbit around a Kerr black hole.

The rescaled Fourier mode of order m received at the location (t, r, θ, ϕ) is

$$\frac{r}{\mu} h_m^+ = \bar{A}_m^+ \cos(m\psi) + \bar{B}_m^+ \sin(m\psi)$$

where μ is the particle mass and $\psi := \omega_0(t - r_*) - \phi$, ω_0 being the orbital frequency of the particle and r_* the tortoise coordinate corresponding to r and \bar{A}_m^+ and \bar{B}_m^+ are given by Eqs. (5.3)-(5.4) above.

INPUT:

- m – positive integer defining the Fourier mode
- a – BH angular momentum parameter (in units of M , the BH mass)
- $r0$ – Boyer-Lindquist radius of the orbit (in units of M)
- $theta$ – Boyer-Lindquist colatitude θ of the observer
- l_max – (default: 10) upper bound in the summation over the harmonic degree ℓ in Eqs. (5.3)-(5.4)
- $algorithm_Zinf$ – (default: 'spline') string describing the computational method for $Z_{\ell m}^\infty(r_0)$; allowed values are
 - 'spline': spline interpolation of tabulated data
 - '1.5PN' (only for $a=0$): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

OUTPUT:

- tuple $(\bar{A}_m^+, \bar{B}_m^+)$

EXAMPLES:

Let us consider the case $a = 0$ first (Schwarzschild black hole), with $m = 2$:

```
sage: from kerrgeodesic_gw import h_plus_particle_fourier
sage: a = 0
sage: h_plus_particle_fourier(2, a, 8., pi/2) # tol 1.0e-13
(0.2014580652208302, -0.06049343736886148)
sage: h_plus_particle_fourier(2, a, 8., pi/2, l_max=5) # tol 1.0e-13
(0.20146097329552273, -0.060495372034569186)
sage: h_plus_particle_fourier(2, a, 8., pi/2, l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
(0.2204617125753912, -0.0830484439639611)
```

Values of m different from 2:

```
sage: h_plus_particle_fourier(3, a, 20., pi/2) # tol 1.0e-13
(-0.005101595598729037, -0.021302121442654077)
sage: h_plus_particle_fourier(3, a, 20., pi/2, l_max=5, algorithm_Zinf='1.5PN') # tol 1.0e-13
(0.0, -0.016720919174427588)
sage: h_plus_particle_fourier(1, a, 8., pi/2) # tol 1.0e-13
(-0.014348477201223874, -0.05844679575244101)
sage: h_plus_particle_fourier(0, a, 8., pi/2)
(0, 0)
```

The case $a/M = 0.95$ (rapidly rotating Kerr black hole):

```
sage: a = 0.95
sage: h_plus_particle_fourier(2, a, 8., pi/2) # tol 1.0e-13
(0.182748773431646, -0.05615306925896938)
sage: h_plus_particle_fourier(8, a, 8., pi/2) # tol 1.0e-13
(-4.724709221209198e-05, 0.0006867183495228116)
sage: h_plus_particle_fourier(2, a, 2., pi/2) # tol 1.0e-13
(0.1700402877617014, 0.33693580916655747)
sage: h_plus_particle_fourier(8, a, 2., pi/2) # tol 1.0e-13
(-0.009367442995129153, -0.03555092085651877)
```

`kerrgeodesic_gw.gw_particle.plot_h_particle`(a , $r0$, $theta$, phi , u_min , u_max , $plot_points=200$, $phi0=0$, $l_max=10$, $m_min=1$, $approximation=None$, $mode=('+', 'x')$, $color=None$, $linestyle=None$, $legend_label=('h_+' + '$', '$h_\\times$')$, $xlabel='$(t - r_*)/M$'$, $ylabel=None$, $title=None$, $gridlines=True$)

Plot the gravitational waveform emitted by a particle in circular orbit around a Kerr black hole.

INPUT:

- a – BH angular momentum parameter (in units of M , the black hole mass)
- $r0$ – Boyer-Lindquist radius of the orbit (in units of M)
- $theta$ – Boyer-Lindquist colatitude θ of the observer
- phi – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- u_min – lower bound of the retarded time coordinate of the observer (in units of M): $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- u_max – upper bound of the retarded time coordinate of the observer (in units of M)
- $plot_points$ – (default: 200) number of points involved in the sampling of the interval (u_min , u_max)
- $phi0$ – (default: 0) phase factor
- l_max – (default: 10) upper bound in the summation over the harmonic degree ℓ
- m_min – (default: 1) lower bound in the summation over the Fourier mode m
- $approximation$ – (default: None) string describing the computational method; allowed values are
 - None: exact computation
 - '1.5PN' (only for $a=0$): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]
 - 'quadrupole' (only for $a=0$): quadrupole approximation (0-post-Newtonian); see `h_particle_quadrupole()`
- $mode$ – (default: ('+', 'x')) string determining the plotted quantities: allowed values are '+' and 'x', for respectively h_+ and h_x , as well as ('+', 'x') for plotting both polarization modes
- $color$ – (default: None) a color (if $mode = '+'$ or 'x') or a pair of colors (if $mode = ('+', 'x')$) for the plot(s); if None, the default colors are 'blue' for h_+ and 'red' for h_x

- `linestyle` – (default: None) a line style (if `mode = '+'` or `'x'`) or a pair of line styles (if `mode = ('+', 'x')`) for the plot(s); if None, the default style is a solid line
- `legend_label` – (default: `(r'h_+', r'h_{\times}')`) labels for the plots of h_+ and h_{\times} ; used only if `mode` is `('+', 'x')`
- `xlabel` – (default: `r'$(t - r_*)/M$'`) label of the x -axis
- `ylabel` – (default: None) label of the y -axis; if None, `r'$(r/\mu)\, , h_+$'` is used for `mode = '+'`, `r'$(r/\mu)\, , h_{\times}$'` for `mode = 'x'` and `r'$(r/\mu)\, , h_{+, \times}$'` for `mode = ('+', 'x')`
- `title` – (default: None) plot title; if None, the title is generated from `a`, `r0` and `theta` (see the example below)
- `gridlines` – (default: True) indicates whether the gridlines are to be plotted

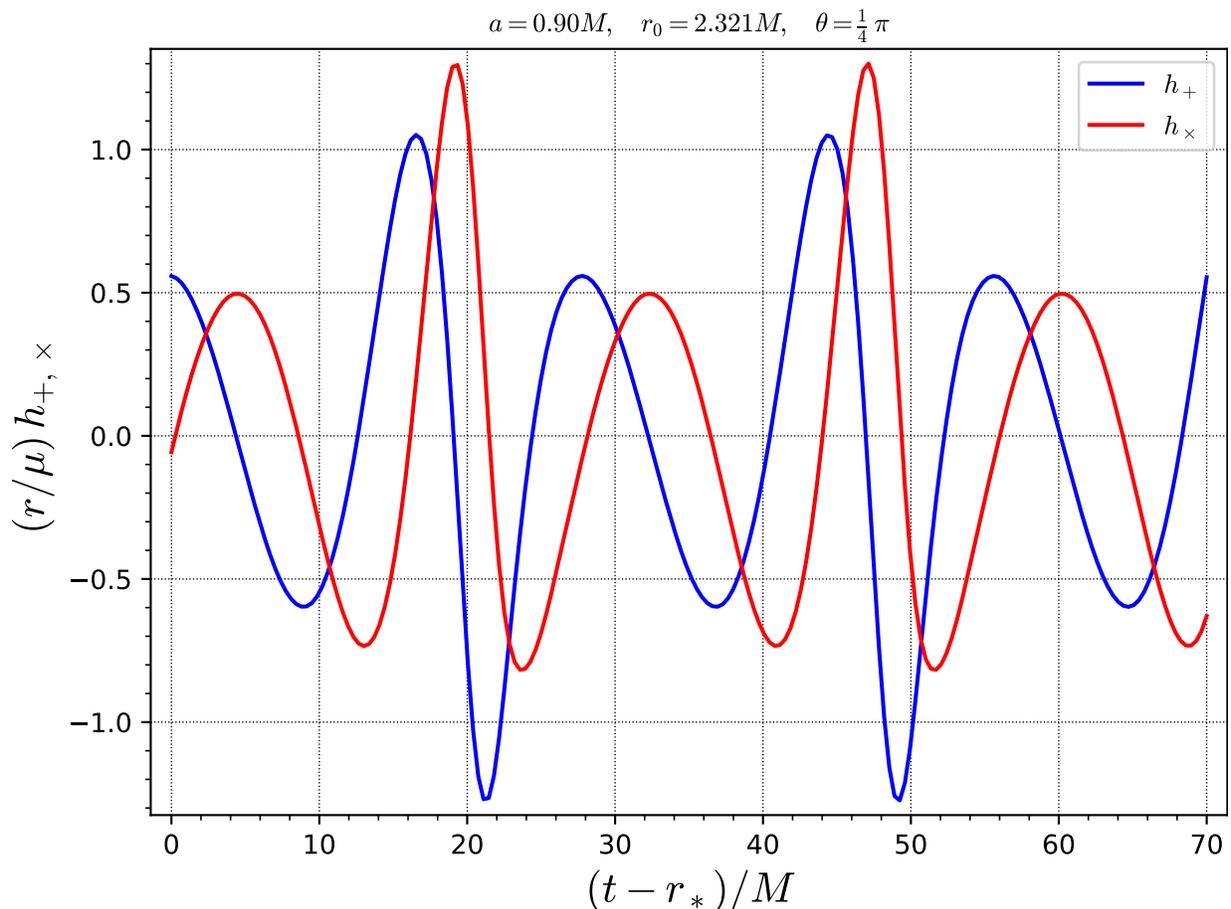
OUTPUT:

- a graphics object

EXAMPLES:

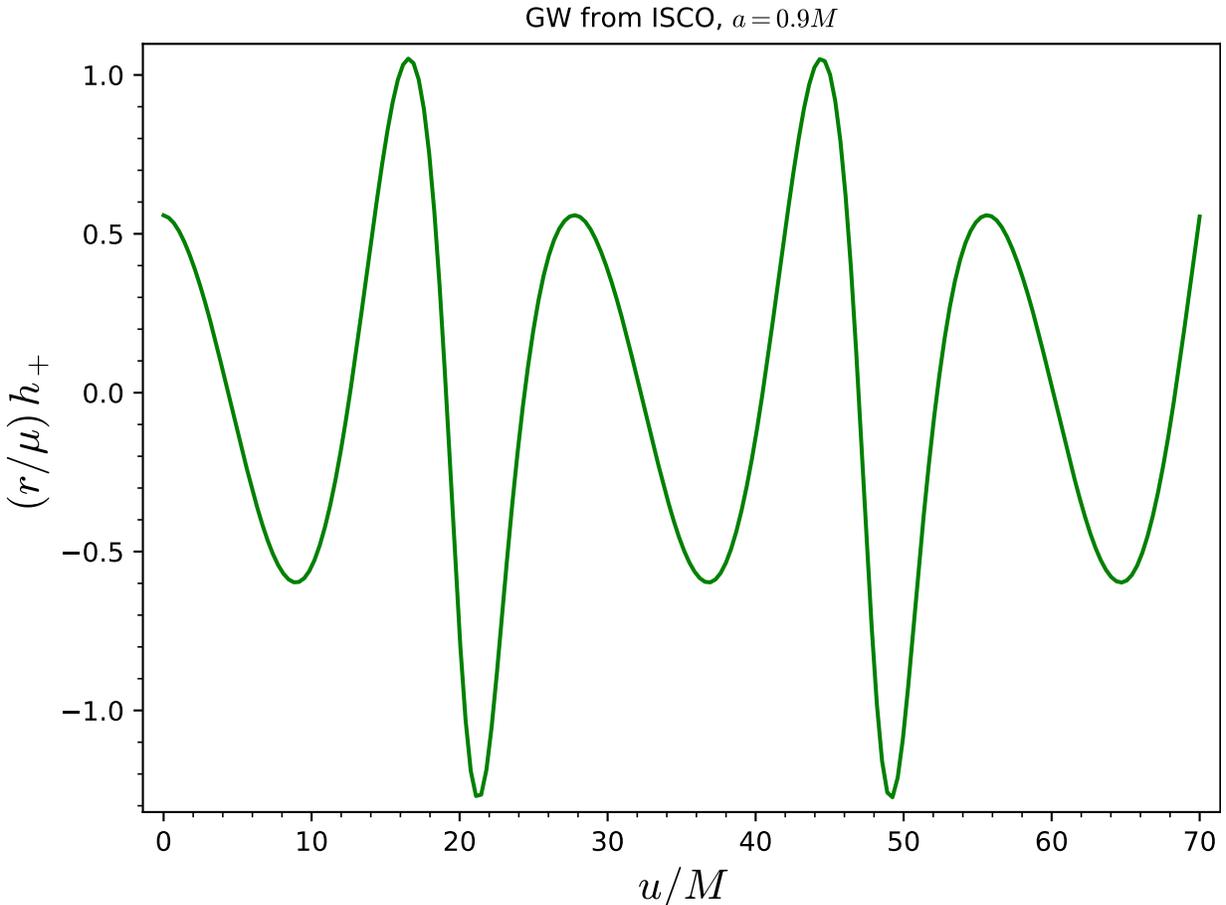
Plot of the gravitational waveform generated by a particle orbiting at the ISCO of a Kerr black hole with $a = 0.9M$:

```
sage: from kerrgeodesic_gw import plot_h_particle
sage: plot_h_particle(0.9, 2.321, pi/4, 0., 0., 70.)
Graphics object consisting of 2 graphics primitives
```



Plot of h_+ only, with some non-default options:

```
sage: plot_h_particle(0.9, 2.321, pi/4, 0., 0., 70., mode='+',
.....:                color='green', xlabel=r'$u/M$', gridlines=False,
.....:                title='GW from ISCO, $a=0.9M$')
Graphics object consisting of 1 graphics primitive
```



```
kerrgeodesic_gw.gw_particle.plot_spectrum_particle(a, r0, theta, mode='+', m_max=10, l_max=10,
.....: algorithm_Zinf='spline', color='blue',
.....: linestyle='-', thickness=2, legend_label=None,
.....: offset=0, xlabel=None, ylabel=None, title=None,
.....: gridlines=True)
```

Plot the Fourier spectrum of the gravitational radiation emitted by a particle in equatorial circular orbit around a Kerr black hole.

The *Fourier spectrum* is defined by the following sequence indexed by m :

$$H_m^{+, \times}(\theta) := \frac{r}{\mu} \sqrt{(A_m^{+, \times}(\theta))^2 + (B_m^{+, \times}(\theta))^2}$$

where the functions A_m^+ , B_m^+ , A_m^\times , B_m^\times are defined by (5.2).

INPUT:

- a – BH angular momentum parameter (in units of M , the BH mass)
- $r0$ – Boyer-Lindquist radius of the orbit (in units of M)

- `theta` – Boyer-Lindquist colatitude θ of the observer
- `mode` – (default: '+') string determining which GW polarization mode is considered; allowed values are '+' and 'x', for respectively h_+ and h_x
- `m_max` – (default: 10) maximal value of m
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ
- `algorithm_Zinf` – (default: 'spline') string describing the computational method for $Z_{\ell m}^{\infty}(r_0)$; allowed values are
 - 'spline': spline interpolation of tabulated data
 - '1.5PN' (only for $a=0$): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [[doi:10.1103/PhysRevD.47.1497](https://doi.org/10.1103/PhysRevD.47.1497)]
- `color` – (default: 'blue') color of vertical lines
- `linestyle` – (default: '-') style of vertical lines
- `legend_label` – (default: None) legend label for this spectrum
- `offset` – (default: 0) horizontal offset for the position of the vertical lines
- `xlabel` – (default: None) label of the x-axis; if none is provided, the label is set to m
- `ylabel` – (default: None) label of the y-axis; if none is provided, the label is set to $H_m^{+,x}$
- `title` – (default: None) plot title; if None, the title is generated from a , r_0 and `theta`
- `gridlines` – (default: True) indicates whether the gridlines are to be plotted

OUTPUT:

- a graphics object

EXAMPLES:

Spectrum of gravitational radiation generated by a particle orbiting at the ISCO of a Schwarzschild black hole ($a = 0$, $r_0 = 6M$):

```
sage: from kerrgeodesic_gw import plot_spectrum_particle
sage: plot_spectrum_particle(0, 6., pi/2)
Graphics object consisting of 10 graphics primitives
```

`kerrgeodesic_gw.gw_particle.radiated_power_particle(a, r0, l_max=None, m_min=1, approximation=None)`

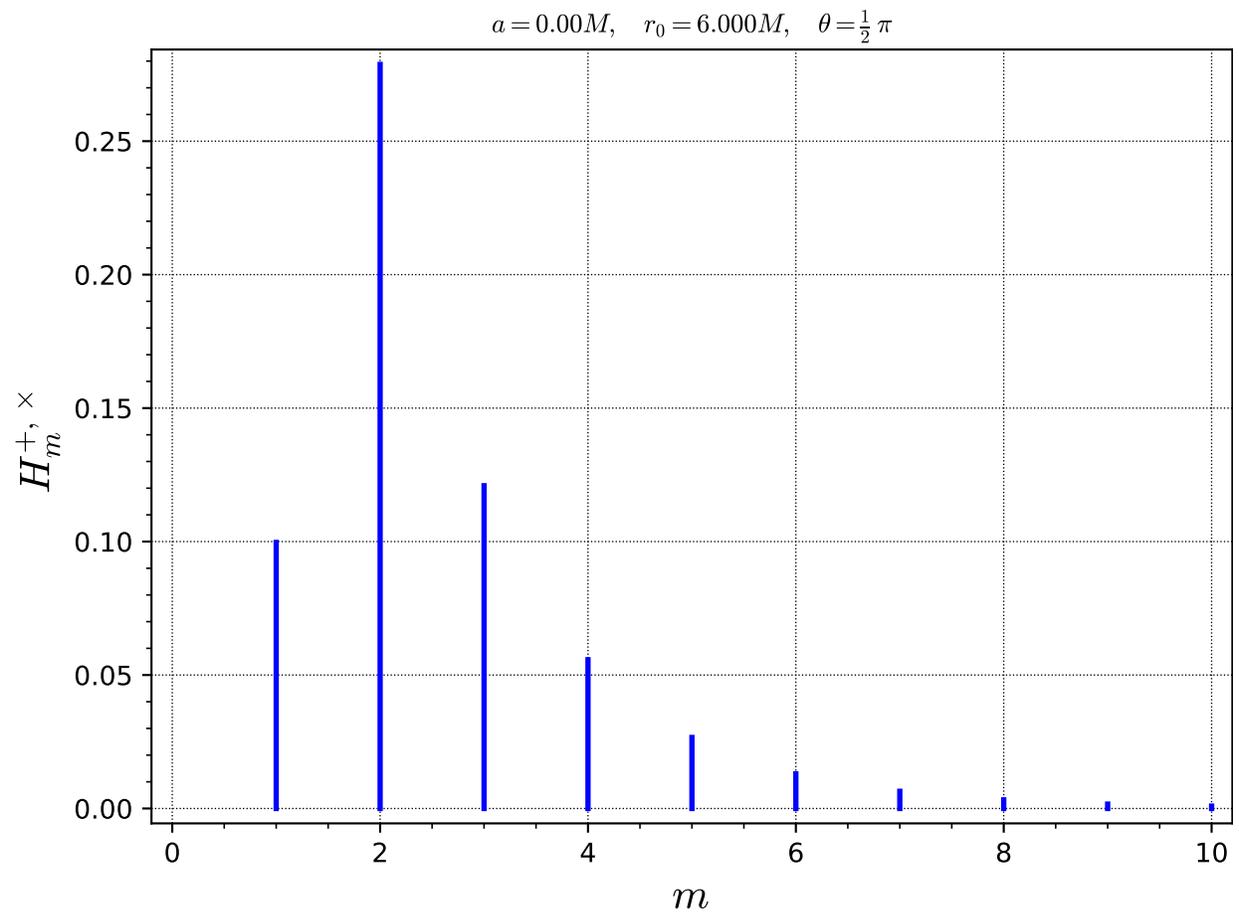
Return the total (i.e. summed over all directions) power of the gravitational radiation emitted by a particle in circular orbit around a Kerr black hole.

The total radiated power (or *luminosity*) L is computed according to the formula:

$$L = \frac{\mu^2}{4\pi} \sum_{\ell=2}^{\infty} \sum_{\substack{m=-\ell \\ m \neq 0}}^{\ell} \frac{|Z_{\ell m}^{\infty}(r_0)|^2}{(m\omega_0)^2}$$

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `r0` – Boyer-Lindquist radius of the orbit (in units of M)
- `l_max` – (default: None) upper bound in the summation over the harmonic degree ℓ ; if None, `l_max` is determined automatically from the available tabulated data
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m



- `approximation` – (default: `None`) string describing the computational method; allowed values are
 - `None`: exact computation
 - `'1.5PN'` (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]
 - `'quadrupole'` (only for `a=0`): quadrupole approximation (0-post-Newtonian); see `h_particle_quadrupole()`

OUTPUT:

- rescaled radiated power $L (M/\mu)^2$, where M is the BH mass and μ the mass of the orbiting particle.

EXAMPLES:

Power radiated by a particle at the ISCO of a Schwarzschild BH:

```
sage: from kerrgeodesic_gw import radiated_power_particle
sage: radiated_power_particle(0, 6.)
0.000937262782177525
```

Power radiated by a particle at the ISCO of a rapidly rotating Kerr BH ($a = 0.98M$):

```
sage: radiated_power_particle(0.98, 1.61403) # tol 1.0e-13
0.08629927053494096
```

Power computed according to various approximations:

```
sage: radiated_power_particle(0, 6., approximation='1.5PN') # tol 1.0e-13
0.0010435864384751143
sage: radiated_power_particle(0, 6., approximation='quadrupole') # tol 1.0e-13
0.000823045267489712
```

Let us check that at large radius, these approximations get closer to the actual result:

```
sage: radiated_power_particle(0, 50.) # tol 1.0e-13
1.9624555021692417e-08
sage: radiated_power_particle(0, 50., approximation='1.5PN') # tol 1.0e-13
1.958318200436188e-08
sage: radiated_power_particle(0, 50., approximation='quadrupole') # tol 1.0e-13
2.0480000000000002e-08
```

`kerrgeodesic_gw.gw_particle.secular_frequency_change`(a , $r0$, $l_max=None$, $m_min=1$, $approximation=None$)

Return the gravitational-radiation induced change of the orbital frequency of a particle in circular orbit around a Kerr black hole.

INPUT:

- a – BH angular momentum parameter (in units of M , the BH mass)
- $r0$ – Boyer-Lindquist radius of the orbit (in units of M)
- l_max – (default: `None`) upper bound in the summation over the harmonic degree ℓ ; if `None`, l_max is determined automatically from the available tabulated data
- m_min – (default: 1) lower bound in the summation over the Fourier mode m
- `approximation` – (default: `None`) string describing the computational method; allowed values are
 - `None`: exact computation
 - `'1.5PN'` (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

- 'quadrupole' (only for $a=0$): quadrupole approximation (0-post-Newtonian); see `h_particle_quadrupole()`

OUTPUT:

- rescaled fractional change in orbital frequency $\dot{\omega}_0/\omega_0 (M^2/\mu)$, where M is the BH mass and μ the mass of the orbiting particle.

EXAMPLES:

Relative change in orbital frequency change at $r_0 = 10M$ around a Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import secular_frequency_change
sage: secular_frequency_change(0, 10) # tol 1.0e-13
0.002701529506901975
sage: secular_frequency_change(0, 10, approximation='quadrupole') # tol 1.0e-13
0.00192
```

At larger distance, the quadrupole approximation works better:

```
sage: secular_frequency_change(0, 50) # tol 1.0e-13
3.048598175592674e-06
sage: secular_frequency_change(0, 50, approximation='quadrupole') # tol 1.0e-13
3.072e-06
```

At the ISCO, $\dot{\omega}_0$ diverges:

```
sage: secular_frequency_change(0, 6)
+infinity
```

while the quadrupole approximation would have predict a finite value there:

```
sage: secular_frequency_change(0, 6, approximation='quadrupole') # tol 1.0e-13
0.014814814814814817
```

Case of a Kerr black hole:

```
sage: secular_frequency_change(0.5, 6) # tol 1.0e-13
0.020393023677356764
```

GRAVITATIONAL RADIATION BY A BLOB OF MATTER

Gravitational radiation by a blob of matter on a circular orbit around a Kerr black hole

This module implements the following functions:

- `h_blob()`: evaluates rh_+ or rh_\times
- `h_blob_signal()`: time sequence of rh_+/μ or rh_\times/μ
- `h_toy_model_semi_analytic()`: rh_+/μ and rh_\times/μ from a semi-analytic approximation based on a constant density blob
- `surface_density_toy_model()`: $\Sigma(\bar{r}, \bar{\phi})$ as an indicator function
- `surface_density_gaussian()`: $\Sigma(\bar{r}, \bar{\phi})$ as a Gaussian profile
- `blob_mass()`: mass of the blob of matter, by integration of $\Sigma(\bar{r}, \bar{\phi})$

`kerrgeodesic_gw.gw_blob.blob_mass(a, surf_dens, param_surf_dens, integ_range, epsabs=1e-08, epsrel=1e-08)`

Compute the mass of a blob of matter by integrating its surface density.

INPUT:

- `a` – BH angular momentum parameter (in units of M)
- `surf_dens` – surface density function Σ ; must take three arguments: (`r_bar`, `phi_bar`, `param_surf_dens`), where
 - `r_bar` is the Boyer-Lindquist radial coordinate \bar{r} in the matter blob
 - `phi_bar` is the Boyer-Lindquist azimuthal coordinate $\bar{\phi}$ in the matter blob
 - `param_surf_dens` are parameters defining the function $\Sigma(\bar{r}, \bar{\phi})$
- `param_surf_dens` – parameters to be passed as the third argument to the function `surf_dens` (see above)
- `integ_range` – tuple $(\bar{r}_1, \bar{r}_2, \bar{\phi}_1, \bar{\phi}_2)$ defining the integration range
- `epsabs` – (default: 1e-8) absolute tolerance of the inner 1-D quadrature integration
- `epsrel` – (default: 1e-8) relative tolerance of the inner 1-D integrals

OUTPUT:

- a pair `(mu, err)`, where `mu` is the mass of the matter blob and `err` is an estimate of the absolute error in the computation of the integral

EXAMPLES:

```
sage: from kerrgeodesic_gw import blob_mass, surface_density_gaussian
sage: param = [6.5, 0., 0.3]
sage: integ_range = [6, 7, -0.1, 0.1]
sage: blob_mass(0., surface_density_gaussian, param, integ_range) # tol 1.0e-13
(0.3328779622200767, 5.671806593829388e-10)
sage: blob_mass(0., surface_density_gaussian, param, integ_range, # tol 1.0e-13
...:                epsabs=1.e-3)
(0.33287796222007715, 1.109366597806814e-06)
```

`kerrgeodesic_gw.gw_blob.h_blob(u, theta, phi, a, surf_dens, param_surf_dens, integ_range, mode='+', l_max=10, m_min=1, epsabs=1e-06, epsrel=1e-06)`

Return the rescaled value of h_+ or h_\times (depending on the parameter `mode`) for the gravitational radiation emitted by a matter blob in circular orbit around a Kerr black hole.

INPUT:

- `u` – retarded time coordinate of the observer (in units of M , the BH mass): $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `phi` – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- `a` – BH angular momentum parameter (in units of M)
- `surf_dens` – surface density function Σ ; must take three arguments: (`r_bar`, `phi_bar`, `param_surf_dens`), where
 - `r_bar` is the Boyer-Lindquist radial coordinate \bar{r} in the matter blob
 - `phi_bar` is the Boyer-Lindquist azimuthal coordinate $\bar{\phi}$ in the matter blob
 - `param_surf_dens` are parameters defining the function $\Sigma(\bar{r}, \bar{\phi})$
- `param_surf_dens` – parameters to be passed as the third argument to the function `surf_dens` (see above)
- `integ_range` – tuple $(\bar{r}_1, \bar{r}_2, \bar{\phi}_1, \bar{\phi}_2)$ defining the integration range
- `mode` – (default: '+') string determining which GW polarization mode is considered; allowed values are '+' and 'x', for respectively h_+ and h_\times
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `epsabs` – (default: 1e-6) absolute tolerance passed directly to the inner 1-D quadrature integration
- `epsrel` – (default: 1e-6) relative tolerance of the inner 1-D integrals

OUTPUT:

- a pair (`rh`, `err`), where `rh` is the rescaled waveform rh_+ (resp. rh_\times) if `mode` = '+' (resp. 'x'), r being the Boyer-Lindquist radial coordinate of the observer, and `err` is an estimate of the absolute error in the computation of the integral

EXAMPLES:

Gravitational emission h_+ from a constant density blob close to the ISCO of a Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import h_blob, surface_density_toy_model
sage: a = 0
sage: param_surf_dens = [6.5, 0, 0.6, 0.1]
sage: integ_range = [6.3, 6.7, -0.04, 0.04]
sage: h_blob(0., pi/2, 0., a, surface_density_toy_model, # tol 1.0e-13
...:                param_surf_dens, integ_range)
(0.03688373245628765, 9.872900827148663e-10)
```

(continues on next page)

(continued from previous page)

```
sage: h_blob(0., pi/2, 0., a, surface_density_toy_model, # tol 1.0e-13
....:      param_surf_dens, integ_range, l_max=5)
(0.037532424224875585, 1.3788099591183387e-09)
```

The h_{\times} part at $\theta = \pi/4$:

```
sage: h_blob(0., pi/4, 0., a, surface_density_toy_model, # tol 1.0e-13
....:      param_surf_dens, integ_range, mode='x')
(0.06203815455135455, 1.972915290337975e-09)
sage: h_blob(0., pi/4, 0., a, surface_density_toy_model, # tol 1.0e-13
....:      param_surf_dens, integ_range, mode='x', l_max=5)
(0.06121422594295032, 1.924590678064715e-09)
```

`kerrgeodesic_gw.gw_blob.h_blob_signal(u_min, u_max, theta, phi, a, surf_dens, param_surf_dens, integ_range, mode='+', nb_points=100, l_max=10, m_min=1, epsabs=1e-06, epsrel=1e-06, store=None, verbose=True)`

Return a time sequence of h_{+} or h_{\times} (depending on the parameter mode) describing the gravitational radiation from a matter blob orbiting a Kerr black hole.

INPUT:

- `u_min` – lower bound of the retarded time coordinate of the observer (in units of M , the BH mass): $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- `u_max` – upper bound of the retarded time coordinate of the observer (in units of M)
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `phi` – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- `a` – BH angular momentum parameter (in units of M)
- `surf_dens` – surface density function Σ ; must take three arguments: (`r_bar`, `phi_bar`, `param_surf_dens`), where
 - `r_bar` is the Boyer-Lindquist radial coordinate \bar{r} in the matter blob
 - `phi_bar` is the Boyer-Lindquist azimuthal coordinate $\bar{\phi}$ in the matter blob
 - `param_surf_dens` are parameters defining the function $\Sigma(\bar{r}, \bar{\phi})$
- `param_surf_dens` – parameters to be passed as the third argument to the function `surf_dens` (see above)
- `integ_range` – tuple $(\bar{r}_1, \bar{r}_2, \bar{\phi}_1, \bar{\phi}_2)$ defining the integration range
- `mode` – (default: '+') string determining which GW polarization mode is considered; allowed values are '+' and 'x', for respectively h_{+} and h_{\times}
- `nb_points` – (default: 100) number of points in the interval (`u_min`, `u_max`)
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `epsabs` – (default: 1e-6) absolute tolerance passed directly to the inner 1-D quadrature integration
- `epsrel` – (default: 1e-6) relative tolerance of the inner 1-D integrals
- `store` – (default: None) string containing a file name for storing the time sequence; if None, no storage is attempted
- `verbose` – (default: True) boolean determining whether to monitor the progress of the sequence computation

OUTPUT:

- a list of `nb_points` pairs $(u, rh_+/\mu)$ or $(u, rh_-/\mu)$ (depending on mode), where μ is the blob's mass and r is the Boyer-Lindquist radial coordinate of the observer

EXAMPLES:

h_+ sequence for a Gaussian-density matter blob around a Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import h_blob_signal, surface_density_gaussian
sage: param_surf_dens = [6.5, 0., 0.3]
sage: integ_range = [6, 7, -0.1, 0.1]
sage: a = 0
sage: h_blob_signal(0., 100., pi/4, 0., a, surface_density_gaussian, # tol 1.0e-13
....:               param_surf_dens, integ_range, nb_points=5,
....:               epsrel=1e-4, verbose=False)
[(0.0000000000000000, 0.2988585811681569),
 (25.00000000000000, -0.3902717353505495),
 (50.00000000000000, 0.4196374371978159),
 (75.00000000000000, -0.4729306367862637),
 (100.000000000000, 0.3464193312741741)]
```

The corresponding h_\times sequence:

```
sage: h_blob_signal(0., 100., pi/4, 0., a, surface_density_gaussian, # tol 1.0e-13
....:               param_surf_dens, integ_range, mode='x',
....:               nb_points=5, epsrel=1e-4, verbose=False)
[(0.0000000000000000, 0.24810446643240472),
 (25.00000000000000, -0.060862007486422516),
 (50.00000000000000, -0.12309577666126706),
 (75.00000000000000, -0.0017707729883368948),
 (100.000000000000, 0.08100026787377135)]
```

`kerrgeodesic_gw.gw_blob.h_toy_model_semi_analytic`(*u, theta, phi, a, r0, phi0, lam, Dphi, l_max=10*)

Return the gravitational wave emitted by a matter blob orbiting a Kerr black hole (semi-analytic computation based on a toy model surface density).

The surface density of the matter blob is that given by `surface_density_toy_model()`.

The gravitational wave is computed according to the formula

$$h = \frac{2\mu}{r} \sum_{\ell=2}^{\infty} \sum_{m=-\ell}^{\ell} \frac{Z_{\ell m}^{\infty}(r_0)}{(m\omega_0)^2} \operatorname{sinc}\left(\frac{m}{2}\Delta\varphi\right) \operatorname{sinc}\left(\frac{3}{4}\varepsilon m\omega_0(1 - a\omega_0)u\right) e^{-im(\omega_0 u + \phi_0)} {}_{-2}S_{\ell m}^{am\omega_0}(\theta, \varphi)$$

INPUT:

- `u` – retarded time coordinate of the observer (in units of M , the BH mass): $u = t - r_*$, where t is the Boyer-Lindquist time coordinate and r_* is the tortoise coordinate
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `phi` – Boyer-Lindquist azimuthal coordinate ϕ of the observer
- `a` – BH angular momentum parameter (in units of M)
- `r0` – mean radius r_0 of the matter blob (Boyer-Lindquist coordinate)
- `phi0` – mean azimuthal angle ϕ_0 of the matter blob (Boyer-Lindquist coordinate)
- `lam` – radial extent λ of the matter blob
- `Dphi` – opening angle $\Delta\phi$ of the matter blob
- `l_max` – (default: 10) upper bound in the summation over the harmonic degree ℓ

OUTPUT:

- a pair (hp, hc), where hp (resp. hc) is $(r/\mu)h_+$ (resp. $(r/\mu)h_\times$), μ being the blob's mass and r is the Boyer-Lindquist radial coordinate of the observer

EXAMPLES:

Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import h_toy_model_semi_analytic
sage: a = 0
sage: r0, phi0, lam, Dphi = 6.5, 0, 0.6, 0.1
sage: u = 60.
sage: h_toy_model_semi_analytic(u, pi/4, 0., a, r0, phi0, lam, Dphi) # tol 1.0e-13
(0.2999183296797872, 0.36916647790743246)
sage: hp, hc = _
```

Comparison with the exact value:

```
sage: from kerrgeodesic_gw import (h_blob, blob_mass,
....:                               surface_density_toy_model)
sage: param_surf_dens = [r0, phi0, lam, Dphi]
sage: integ_range = [6.2, 6.8, -0.05, 0.05]
sage: mu = blob_mass(a, surface_density_toy_model, param_surf_dens,
....:               integ_range)[0]
sage: hp0 = h_blob(u, pi/4, 0., a, surface_density_toy_model,
....:               param_surf_dens, integ_range)[0] / mu
sage: hc0 = h_blob(u, pi/4, 0., a, surface_density_toy_model,
....:               param_surf_dens, integ_range, mode='x')[0] / mu
sage: hp0, hc0 # tol 1.0e-13
(0.2951163078053617, 0.3743683023327848)
sage: (hp - hp0) / hp0 # tol 1.0e-13
0.01627162494047128
sage: (hc - hc0) / hc0 # tol 1.0e-13
-0.013894938201066784
```

kerrgeodesic_gw.gw_blob.**surface_density_gaussian**($r, \phi, param$)

Surface density of a matter blob with a Gaussian profile

INPUT:

- r – Boyer-Lindquist radial coordinate \bar{r} in the matter blob
- ϕ – Boyer-Lindquist azimuthal coordinate $\bar{\phi}$ in the matter blob
- $param$ – list of parameters defining the position and width of matter blob:
 - $param[0]$: mean radius r_0 (Boyer-Lindquist coordinate)
 - $param[1]$: mean azimuthal angle ϕ_0 (Boyer-Lindquist coordinate)
 - $param[2]$: width λ of the Gaussian profile
 - $param[3]$ (optional): amplitude Σ_0 ; if not provided, then $\Sigma_0 = 1$ is used

OUTPUT:

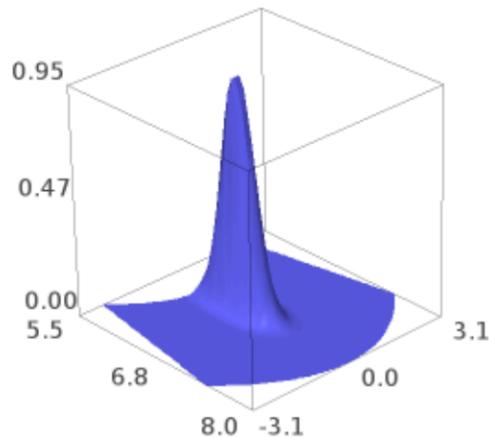
- surface density $\Sigma(\bar{r}, \bar{\phi})$

EXAMPLES:

```
sage: from kerrgeodesic_gw import surface_density_gaussian
sage: param = [6.5, 0., 0.3]
sage: surface_density_gaussian(6.5, 0, param)
1.0
sage: surface_density_gaussian(8., 0, param) # tol 1.0e-13
1.3887943864964021e-11
sage: surface_density_gaussian(6.5, pi/16, param) # tol 1.0e-13
1.4901161193847656e-08
```

3D representation: $z = \Sigma(\bar{r}, \bar{\phi})$ in terms of $x := \bar{r} \cos \bar{\phi}$ and $y := \bar{r} \sin \bar{\phi}$:

```
sage: s_plot = lambda r, phi: surface_density_gaussian(r, phi, param)
sage: r, phi, z = var('r phi z')
sage: plot3d(s_plot, (r, 6, 8), (phi, -0.4, 0.4),
....:         transformation=(r*cos(phi), r*sin(phi), z))
Graphics3d Object
```



Use with a non-default amplitude ($\Sigma_0 = 10^{-5}$):

```
sage: sigma0 = 1.e-5
sage: param = [6.5, 0., 0.3, sigma0]
sage: surface_density_gaussian(6.5, 0, param)
1e-05
```

`kerrgeodesic_gw.gw_blob.surface_density_toy_model(r, phi, param)`

Surface density of the toy model matter blob.

INPUT:

- *r* – Boyer-Lindquist radial coordinate \bar{r} in the matter blob
- *phi* – Boyer-Lindquist azimuthal coordinate $\bar{\phi}$ in the matter blob
- *param* – list of parameters defining the position and extent of the matter blob:
 - `param[0]`: mean radius r_0 (Boyer-Lindquist coordinate)
 - `param[1]`: mean azimuthal angle ϕ_0 (Boyer-Lindquist coordinate)

- param[2]: radial extent λ
- param[3]: opening angle $\Delta\phi$
- param[4] (optional): amplitude Σ_0 ; if not provided, then $\Sigma_0 = 1$ is used

OUTPUT:

- surface density $\Sigma(\bar{r}, \bar{\phi})$

EXAMPLES:

Use with the default amplitude ($\Sigma_0 = 1$):

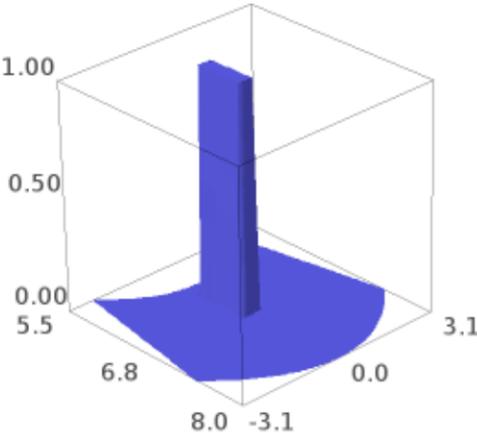
```
sage: from kerrgeodesic_gw import surface_density_toy_model
sage: param = [6.5, 0, 0.6, 0.1]
sage: surface_density_toy_model(6.5, 0, param)
1.0
sage: surface_density_toy_model(6.7, -0.04, param)
1.0
sage: surface_density_toy_model(6.3, 0.04, param)
1.0
sage: surface_density_toy_model(7, -0.06, param)
0.0
sage: surface_density_toy_model(5., 0.06, param)
0.0
```

3D representation: $z = \Sigma(\bar{r}, \bar{\phi})$ in terms of $x := \bar{r} \cos \bar{\phi}$ and $y := \bar{r} \sin \bar{\phi}$:

```
sage: s_plot = lambda r, phi: surface_density_toy_model(r, phi, param)
sage: r, phi, z = var('r phi z')
sage: plot3d(s_plot, (r, 6, 8), (phi, -0.4, 0.4),
...:         transformation=(r*cos(phi), r*sin(phi), z))
Graphics3d Object
```

Use with a non-default amplitude ($\Sigma_0 = 2$):

```
sage: sigma0 = 2.
sage: param = [6.5, 0, 0.6, 0.1, sigma0]
sage: surface_density_toy_model(6.5, 0, param)
2.0
```



LISA DETECTOR

Functions relative to the LISA detector.

`kerrgeodesic_gw.lisa_detector.power_spectral_density(freq)`

Return the effective power spectral density (PSD) of the detector noise at a given frequency.

INPUT:

- `freq` – frequency f (in Hz)

OUTPUT:

- effective power spectral density $S(f)$ (in Hz^{-1})

EXAMPLES:

```
sage: from kerrgeodesic_gw import lisa_detector
sage: Sn = lisa_detector.power_spectral_density
sage: Sn(1.e-1) # tol 1.0e-13
3.3944027493062926e-39
sage: Sn(1.e-2) # tol 1.0e-13
2.738383947022306e-40
sage: Sn(1.e-3) # tol 1.0e-13
3.269807574220045e-38
```

`kerrgeodesic_gw.lisa_detector.power_spectral_density_RCLfit(freq)`

Return the effective power spectral density (PSD) of the detector noise at a given frequency, according to the analytical fit by Robson, Cornish and Liu, [Arxiv 1803.01944](https://arxiv.org/abs/1803.01944)

INPUT:

- `freq` – frequency f (in Hz)

OUTPUT:

- effective power spectral density $S(f)$ (in Hz^{-1})

EXAMPLES:

```
sage: from kerrgeodesic_gw import lisa_detector
sage: Sn = lisa_detector.power_spectral_density_RCLfit
sage: Sn(1.e-1) # tol 1.0e-13
2.12858262120861e-39
sage: Sn(1.e-2) # tol 1.0e-13
1.44307343517977e-40
sage: Sn(1.e-3) # tol 1.0e-13
1.63410027259543e-38
```

`kerrgeodesic_gw.lisa_detector.strain_sensitivity(freq)`

Return LISA strain spectral sensitivity at a given frequency.

The strain spectral sensitivity is the square root of the effective noise power spectral density (cf. `power_spectral_density()`).

INPUT:

- `freq` – frequency f (in Hz)

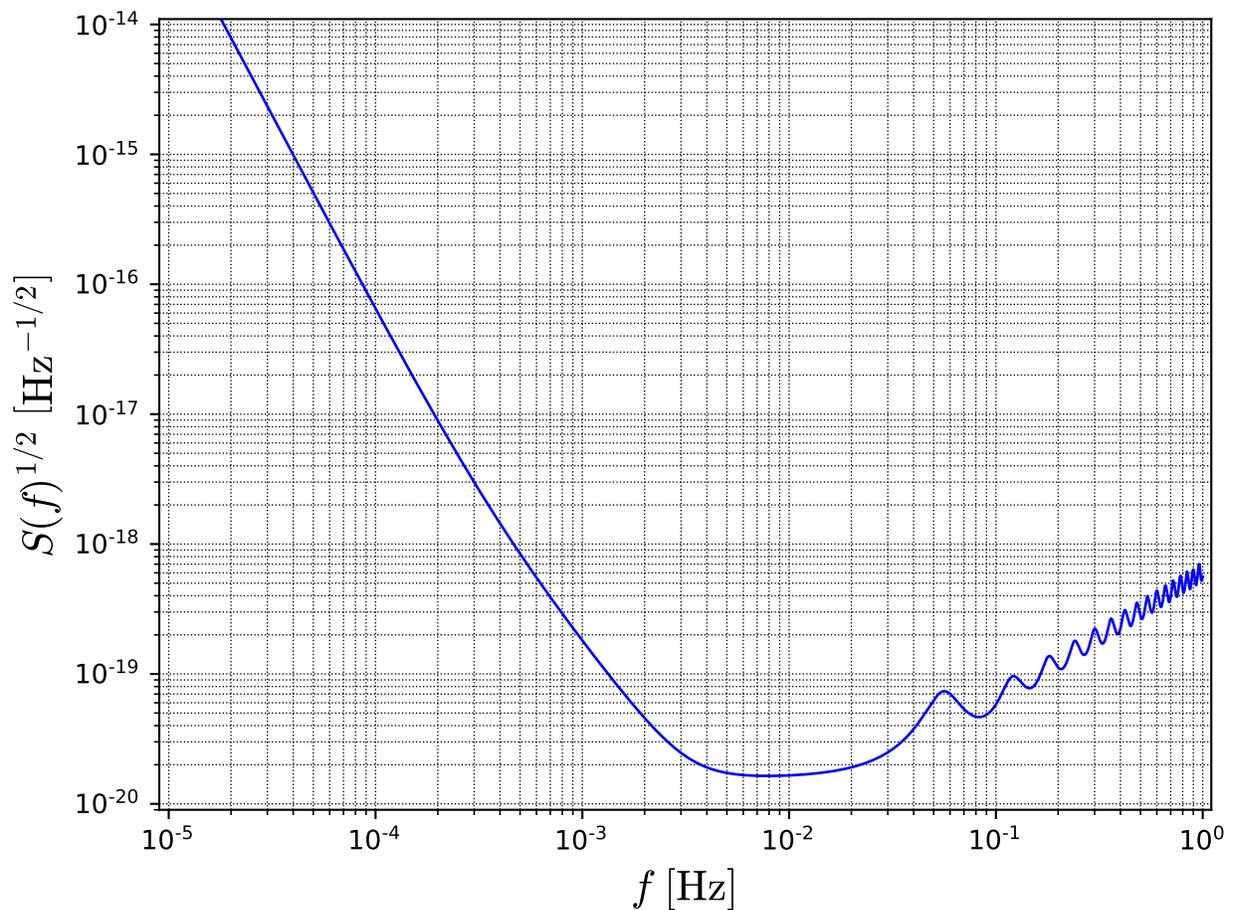
OUTPUT:

- strain sensitivity $S(f)^{1/2}$ (in $\text{Hz}^{-1/2}$)

EXAMPLES:

```
sage: from kerrgeodesic_gw import lisa_detector
sage: hn = lisa_detector.strain_sensitivity
sage: hn(1.e-1) # tol 1.0e-13
5.82615031500758e-20
sage: hn(1.e-2) # tol 1.0e-13
1.654806317072275e-20
sage: hn(1.e-3) # tol 1.0e-13
1.8082609253700212e-19
```

```
sage: plot_loglog(hn, (1e-5, 1), plot_points=2000, ymin=1e-20, ymax=1e-14,
....:             axes_labels=[r"$f$ \ [\mathrm{Hz}]$",
....:                        r"$S(f)^{1/2}$ \ \left[\mathrm{Hz}^{-1/2}\right]"),
....:             gridlines='minor', frame=True, axes=False)
Graphics object consisting of 1 graphics primitive
```



SIGNAL PROCESSING

Signal processing

`kerrgeodesic_gw.signal_processing.fourier`(*signal*)

Compute the Fourier transform of a time signal.

The *Fourier transform* of the time signal $h(t)$ is defined by

$$\tilde{h}(f) = \int_{-\infty}^{+\infty} h(t) e^{-2\pi i f t} dt$$

INPUT:

- `signal` – list of pairs $(t, h(t))$, where t is the time and $h(t)$ is the signal at t . *NB*: the sampling in t must be uniform

OUTPUT:

- a numerical approximation (FFT) of the Fourier transform, as a list of pairs $(f, \tilde{h}(f))$

EXAMPLES:

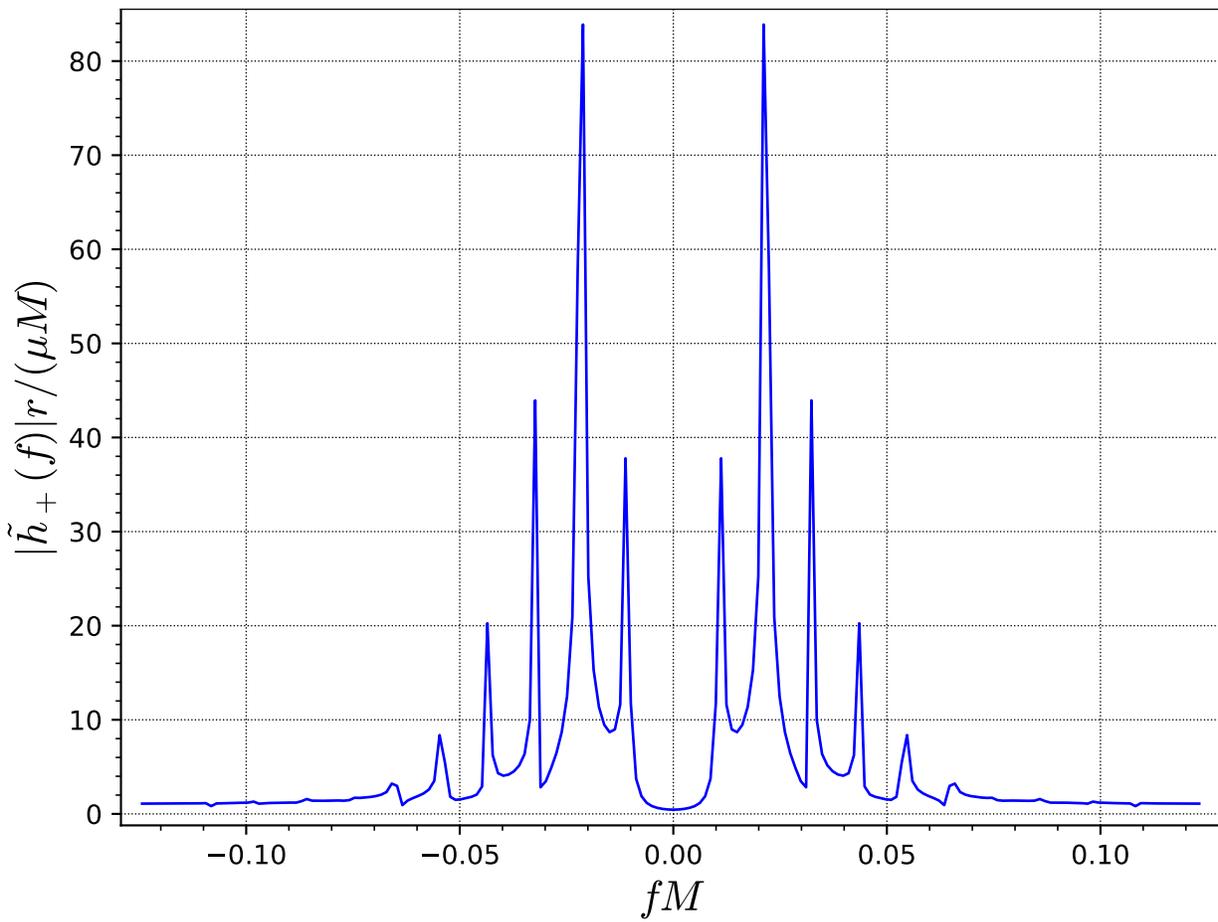
Fourier transform of the h_+ signal from a particle orbiting at the ISCO of a Schwarzschild black hole:

```
sage: from kerrgeodesic_gw import h_particle_signal, fourier
sage: a, r0 = 0., 6.
sage: theta, phi = pi/2, 0
sage: sig = h_particle_signal(a, r0, theta, phi, 0., 800., nb_points=200)
sage: sig[:5] # tol 1.0e-13
[(0.0000000000000000, 0.1536656546005028),
 (4.02010050251256, 0.03882960191740132),
 (8.04020100502512, -0.0791773803745108),
 (12.0603015075377, -0.18368354016995483),
 (16.0804020100502, -0.25796165580196795)]
sage: ft = fourier(sig)
sage: ft[:5] # tol 1.0e-13
[(-0.12437500000000001, (1.0901773159466113+0j)),
 (-0.12313125000000001, (1.0901679813096925+0.021913447210468399j)),
 (-0.12188750000000001, (1.0901408521337845+0.043865034665617614j)),
 (-0.12064375000000001, (1.0900987733011678+0.065894415355341171j)),
 (-0.11940000000000001, (1.0900473072498438+0.088044567109304195j))]
```

Plot of the norm of the Fourier transform:

```
sage: ft_norm = [(f, abs(hf)) for (f, hf) in ft]
sage: line(ft_norm, axes_labels=[r'$f$ M$', r'$|\tilde{h}_+(f)| r/(\mu M)$'],
....:      gridlines=True, frame=True, axes=False)
Graphics object consisting of 1 graphics primitive
```

The first peak is at the orbital frequency of the particle:



```
sage: f0 = n(1/(2*pi*r0^(3/2))); f0
0.0108291222393566
```

while the highest peak is at twice this frequency ($m = 2$ mode).

`kerrgeodesic_gw.signal_processing.max_detectable_radius`($a, mu, theta, psd, BH_time_scale, distance, t_obs_yr=1, snr_threshold=10, r_min=None, r_max=200, m_min=1, m_max=None, approximation=None$)

Compute the maximum orbital radius $r_{0,max}$ at which a particle of given mass is detectable.

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `mu` – mass of the orbiting object, in solar masses
- `theta` – Boyer-Lindquist colatitude θ of the observer
- `psd` – function with a single argument (f) representing the detector’s one-sided noise power spectral density $S_n(f)$ (see e.g. `lisa_detector.power_spectral_density()`)
- `BH_time_scale` – value of M in the same time unit as $S_n(f)$; if $S_n(f)$ is provided in Hz^{-1} , then `BH_time_scale` must be M expressed in seconds.
- `distance` – distance r to the detector, in parsecs
- `t_obs_yr` – (default: 1) observation period, in years
- `snr_threshold` – (default: 10) signal-to-noise value above which a detection is claimed
- `r_min` – (default: None) lower bound of the search interval for $r_{0,max}$ (in units of M); if None then the ISCO value is used
- `r_max` – (default: 200) upper bound of the search interval for $r_{0,max}$ (in units of M)
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `m_max` – (default: None) upper bound in the summation over the Fourier mode m ; if None, `m_max` is set to 10 for $r_0 \leq 20M$ and to 5 for $r_0 > 20M$
- `approximation` – (default: None) string describing the computational method for the signal; allowed values are
 - None: exact computation
 - 'quadrupole': quadrupole approximation; see `gw_particle.h_particle_quadrupole()`
 - '1.5PN' (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [doi:10.1103/PhysRevD.47.1497]

OUTPUT:

- Boyer-Lindquist radius (in units of M) of the most remote orbit for which the signal-to-noise ratio is larger than `snr_threshold` during the observation time `t_obs_yr`

EXAMPLES:

Maximum orbital radius for LISA detection of a 1 solar-mass object orbiting Sgr A* observed by LISA, assuming a BH spin $a = 0.9M$ and a vanishing inclination angle:

```
sage: from kerrgeodesic_gw import (max_detectable_radius, lisa_detector,
....:                               astro_data)
sage: a = 0.9
sage: mu = 1
```

(continues on next page)

(continued from previous page)

```
sage: theta = 0
sage: psd = lisa_detector.power_spectral_density_RCLfit
sage: BH_time_scale = astro_data.SgrA_mass_s
sage: distance = astro_data.SgrA_distance_pc
sage: max_detectable_radius(a, mu, theta, psd, BH_time_scale, distance) # tol 1.0e-13
46.98292057022975
```

Lowering the SNR threshold to 5:

```
sage: max_detectable_radius(a, mu, theta, psd, BH_time_scale, distance, # tol 1.0e-13
....:                          snr_threshold=5)
53.503386416644645
```

Lowering the data acquisition time to 1 day:

```
sage: max_detectable_radius(a, mu, theta, psd, BH_time_scale, distance, # tol 1.0e-13
....:                          t_obs_yr=1./365.25)
27.158725683511463
```

Assuming an inclination angle of $\pi/2$:

```
sage: theta = pi/2
sage: max_detectable_radius(a, mu, theta, psd, BH_time_scale, distance) # tol 1.0e-13
39.81826500968452
```

Using the 1.5-PN approximation (a has to be zero and `m_max` has to be at most 5):

```
sage: a = 0
sage: max_detectable_radius(a, mu, theta, psd, BH_time_scale, # tol 1.0e-13
....:                          distance, approximation='1.5PN', m_max=5)
39.74273792957649
```

`kerrgeodesic_gw.signal_processing.read_signal(filename, dirname=None)`

Read a signal from a data file.

INPUT:

- `filename` – string; name of the file
- `dirname` – (default: None) string; name of directory where the file is located

OUTPUT:

- signal as a list of pairs $(t, h(t))$

EXAMPLES:

```
sage: from kerrgeodesic_gw import save_signal, read_signal
sage: sig0 = [(RDF(i/10), RDF(sin(i/5))) for i in range(5)]
sage: from sage.misc.tmpfile import tmp_filename
sage: filename = tmp_filename(ext='.dat')
sage: save_signal(sig0, filename)
sage: sig = read_signal(filename)
sage: sig # tol 1.0e-13
[(0.0, 0.0),
 (0.1, 0.19866933079506122),
 (0.2, 0.3894183423086505),
 (0.3, 0.5646424733950354),
 (0.4, 0.7173560908995228)]
```

A test:

```
sage: sig == sig0
True
```

`kerrgeodesic_gw.signal_processing.save_signal(sig, filename, dirname=None)`

Write a signal in a data file.

INPUT:

- `sig` – signal as a list of pairs $(t, h(t))$
- `filename` – string; name of the file
- `dirname` – (default: None) string; name of directory where the file is located

EXAMPLES:

```
sage: from kerrgeodesic_gw import save_signal, read_signal
sage: sig = [(RDF(i/10), RDF(sin(i/5))) for i in range(5)]
sage: sig # tol 1.0e-13
[(0.0, 0.0),
 (0.1, 0.19866933079506122),
 (0.2, 0.3894183423086505),
 (0.3, 0.5646424733950354),
 (0.4, 0.7173560908995228)]
sage: from sage.misc.tmpfile import tmp_filename
sage: filename = tmp_filename(ext='.dat')
sage: save_signal(sig, filename)
```

A test:

```
sage: sig1 = read_signal(filename)
sage: sig1 == sig
True
```

`kerrgeodesic_gw.signal_processing.signal_to_noise(signal, time_scale, psd, fmin, fmax, scale=1, interpolation='linear', quad_epsrel=1e-05, quad_limit=500)`

Evaluate the signal-to-noise ratio of a signal observed in a detector of a given power spectral density.

The *signal-to-noise ratio* ρ of the time signal $h(t)$ is computed according to the formula

$$\rho^2 = 4 \int_0^{+\infty} \frac{|\tilde{h}(f)|^2}{S_n(f)} df \quad (8.1)$$

where $\tilde{h}(f)$ is the Fourier transform of $h(t)$ (see `fourier()`) and $S_n(f)$ is the detector's one-sided noise power spectral density (see e.g. `lisa_detector.power_spectral_density()`).

INPUT:

- `signal` – list of pairs $(t, h(t))$, where t is the time and $h(t)$ is the signal at t . *NB*: the sampling in t must be uniform
- `time_scale` – value of t unit in terms of $S_n(f)$ unit; if $S_n(f)$ is provided in Hz^{-1} , then `time_scale` must be the unit of t in `signal` expressed in seconds.
- `psd` – function with a single argument (f) representing the detector's one-sided noise power spectral density $S_n(f)$
- `fmin` – lower bound used instead of 0 in the integral (8.1)
- `fmax` – upper bound used instead of $+\infty$ in the integral (8.1)
- `scale` – (default: 1) scale factor by which $h(t)$ must be multiplied to get the actual signal

- `interpolation` – (default: `'linear'`) string describing the type of interpolation used to evaluate $|h(f)|^2$ from the list resulting from the FFT of `signal`; allowed values are
 - `'linear'`: linear interpolation between two data points
 - `'spline'`: cubic spline interpolation
- `quad_epsrel` – (default: `1.e-6`) relative error tolerance in the computation of the integral (8.1)
- `quad_limit` – (default: `500`) upper bound on the number of subintervals used in the adaptive algorithm to compute the integral (this corresponds to the argument `limit` of SciPy’s function `quad`)

OUTPUT:

- the signal-to-noise ratio ρ computed via the integral (8.1), with the boundaries 0 and $+\infty$ replaced by respectively `fmin` and `fmax`.

EXAMPLES:

Let us evaluate the SNR of the gravitational signal generated by a 1-solar mass object orbiting at the ISCO of Sgr A* observed by LISA during 1 day. We need the following functions:

```
sage: from kerrgeodesic_gw import (h_particle_signal, signal_to_noise,
....:                             lisa_detector, astro_data)
```

We model Sgr A* as a Schwarzschild black hole and we consider that the signal is constituted by the mode $h_+(t)$ observed in the orbital plane:

```
sage: a, r0 = 0., 6.
sage: theta, phi = pi/2, 0
sage: tmax = 24*3600/astro_data.MSgrA_s # 1 day in units of Sgr A* mass
sage: h = h_particle_signal(a, r0, theta, phi, 0., tmax, mode='+',
....:                       nb_points=4000)
```

The signal-to-noise ratio is then computed as:

```
sage: time_scale = astro_data.MSgrA_s # Sgr A* mass in seconds
sage: psd = lisa_detector.power_spectral_density_RCLfit
sage: fmin, fmax = 1e-5, 5e-3
sage: mu_ov_r = astro_data.Msol_m / astro_data.dSgrA # mu/r
sage: signal_to_noise(h, time_scale, psd, fmin, fmax, # tol 1.0e-13
....:                  interpolation='spline', scale=mu_ov_r)
7583.104124621172
```

Signal-to-noise for a signal computed at the quadrupole approximation:

```
sage: h = h_particle_signal(a, r0, theta, phi, 0., tmax, mode='+',
....:                       nb_points=4000, approximation='quadrupole')
sage: signal_to_noise(h, time_scale, psd, fmin, fmax, # tol 1.0e-13
....:                  interpolation='spline', scale=mu_ov_r)
5383.264004811493
```

`kerrgeodesic_gw.signal_processing.signal_to_noise_particle(a, r0, theta, psd, t_obs, BH_time_scale, m_min=1, m_max=None, scale=1, approximation=None)`

Evaluate the signal-to-noise ratio of gravitational radiation emitted by a single orbiting particle observed in a detector of a given power spectral density.

INPUT:

- `a` – BH angular momentum parameter (in units of M , the BH mass)
- `r0` – Boyer-Lindquist radius of the orbit (in units of M)
- `theta` – Boyer-Lindquist colatitude θ of the observer

- `psd` – function with a single argument (f) representing the detector’s one-sided noise power spectral density $S_n(f)$ (see e.g. `lisa_detector.power_spectral_density()`)
- `t_obs` – observation period, in the same time unit as $S_n(f)$
- `BH_time_scale` – value of M in the same time unit as $S_n(f)$; if $S_n(f)$ is provided in Hz^{-1} , then `BH_time_scale` must be M expressed in seconds.
- `m_min` – (default: 1) lower bound in the summation over the Fourier mode m
- `m_max` – (default: None) upper bound in the summation over the Fourier mode m ; if None, `m_max` is set to 10 for $r_0 \leq 20M$ and to 5 for $r_0 > 20M$
- `scale` – (default: 1) scale factor by which $h(t)$ must be multiplied to get the actual signal; this should be μ/r , where μ is the particle mass and r the radial coordinate of the detector
- `approximation` – (default: None) string describing the computational method for the signal; allowed values are
 - None: exact computation
 - 'quadrupole': quadrupole approximation; see `gw_particle.h_particle_quadrupole()`
 - '1.5PN' (only for `a=0`): 1.5-post-Newtonian expansion following E. Poisson, Phys. Rev. D **47**, 1497 (1993) [[doi:10.1103/PhysRevD.47.1497](https://doi.org/10.1103/PhysRevD.47.1497)]

OUTPUT:

- the signal-to-noise ratio ρ

EXAMPLES:

Let us evaluate the SNR of the gravitational signal generated by a 1-solar mass object orbiting at the ISCO of Sgr A* observed by LISA during 1 day:

```
sage: from kerrgeodesic_gw import (signal_to_noise_particle,
....:                             lisa_detector, astro_data)
sage: a, r0 = 0., 6.
sage: theta = pi/2
sage: t_obs = 24*3600 # 1 day in seconds
sage: BH_time_scale = astro_data.SgrA_mass_s # Sgr A* mass in seconds
sage: psd = lisa_detector.power_spectral_density_RCLfit
sage: mu_ov_r = astro_data.Msol_m / astro_data.dSgrA # mu/r
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                          BH_time_scale, scale=mu_ov_r)
7565.450402023329
```

Using the quadrupole approximation:

```
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                          BH_time_scale, scale=mu_ov_r,
....:                          approximation='quadrupole')
5230.216539094391
```

Using the 1.5-PN approximation (`m_max` has to be at most 5):

```
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                          BH_time_scale, scale=mu_ov_r,
....:                          approximation='1.5PN', m_max=5)
7601.135299165022
```

For large values of r_0 , the 1.5-PN approximation and the quadrupole one converge:

```
sage: r0 = 100
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                               BH_time_scale, scale=mu_ov_r,
....:                               approximation='quadrupole')
0.003053021617751869
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                               BH_time_scale, scale=mu_ov_r,
....:                               approximation='1.5PN')
0.003144006483338098
```

```
sage: r0 = 1000
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                               BH_time_scale, scale=mu_ov_r,
....:                               approximation='quadrupole')
9.663153184393872e-09
sage: signal_to_noise_particle(a, r0, theta, psd, t_obs, # tol 1.0e-13
....:                               BH_time_scale, scale=mu_ov_r,
....:                               approximation='1.5PN')
9.686830661779979e-09
```

ASTRONOMICAL DATA

Astronomical data.

Having imported the `astro_data` module, type

- `astro_data.<TAB>`, where `<TAB>` stands for the tabulation key, to get the list of available data
- `astro_data.??` to see the sources of the numerical values

EXAMPLES:

Fundamental constants (in SI units):

```
sage: from kerrgeodesic_gw import astro_data
sage: astro_data.G
6.6743e-11
sage: astro_data.c
299792458.0
```

Solar mass in kg:

```
sage: astro_data.solar_mass_kg
1.98848e+30
```

Solar mass in meters (geometrized units):

```
sage: astro_data.solar_mass_m # tol 1.0e-13
1476.6771171937082
```

Solar mass in seconds (geometrized units):

```
sage: astro_data.solar_mass_s # tol 1.0e-13
4.925664664965348e-06
```

Mass of Sgr A* in solar masses:

```
sage: astro_data.SgrA_mass_sol
4100000.0
```

Mass of Sgr A* in kg:

```
sage: astro_data.SgrA_mass_kg # tol 1.0e-13
8.152768e+36
```

Mass of Sgr A* in meters (geometrized units):

```
sage: astro_data.SgrA_mass_m # tol 1.0e-13
6054376180.4942045
```

Mass of Sgr A* in seconds (geometrized units):

```
sage: astro_data.SgrA_mass_s # tol 1.0e-13
20.19522512635793
```

Distance to Sgr A* in parsecs and meters, respectively:

```
sage: astro_data.SgrA_distance_pc
8120.0
sage: astro_data.SgrA_distance_m # tol 1.0e-13
2.5055701961709902e+20
```

UTILITIES

Utilities

This module contains some utility functions. For the moment, there is only

- `graphics_inset()`: generates plots with insets

`kerrgeodesic_gw.utilities.graphics_inset(main, inset, position, **kws)`

Insert a graphics object as an inset into another graphics object.

This function compensates for the (current) lack of the inset functionality in SageMath graphics.

The output is not a SageMath graphics object but a Matplotlib object, of type `Figure`.

INPUT:

- `main` – a SageMath graphics object
- `inset` – a SageMath graphics object to be inserted into `main`
- `position` – a 4-uple (`left`, `bottom`, `width`, `height`) specifying the position and size of the inset (all quantities are in fractions of figure width and height)
- `kws` – options passed to method `matplotlib.figure.Figure.add_axes`

OUTPUT:

- a `matplotlib.figure.Figure` object

EXAMPLES:

```
sage: from kerrgeodesic_gw.utilities import graphics_inset
sage: g1 = plot(x*sin(1/x), (x, -2, 2), axes_labels=[r"$x$", r"$y$"])
sage: g2 = plot(x*sin(1/x), (x, -0.05, 0.05),
...:         axes_labels=[r"$x$", r"$y$"], fontsize=8, frame=True)
sage: figure = graphics_inset(g1, g2, (0.72, 0.4, 0.25, 0.25))
sage: figure
<Figure size 640x480 with 2 Axes>
```

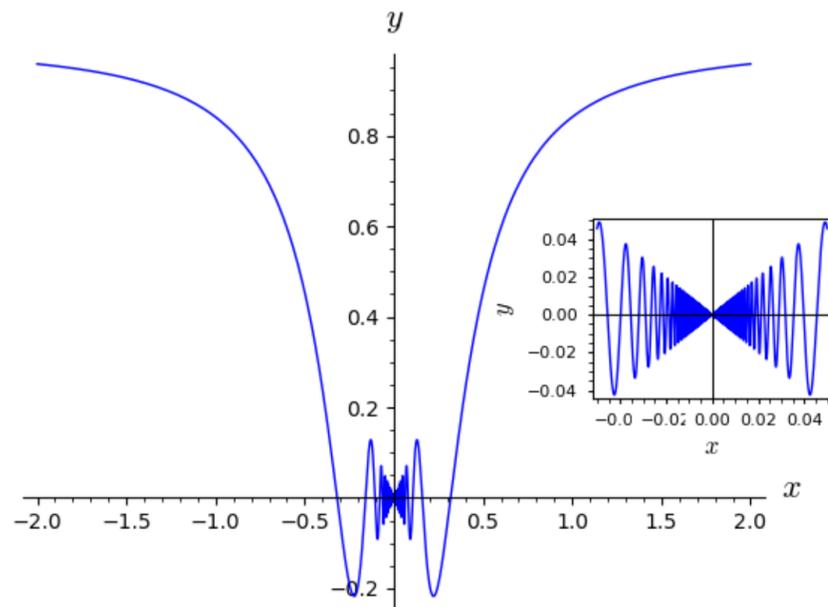
The output is a Matplotlib object, of type `Figure`:

```
sage: type(figure)
<class 'matplotlib.figure.Figure'>
```

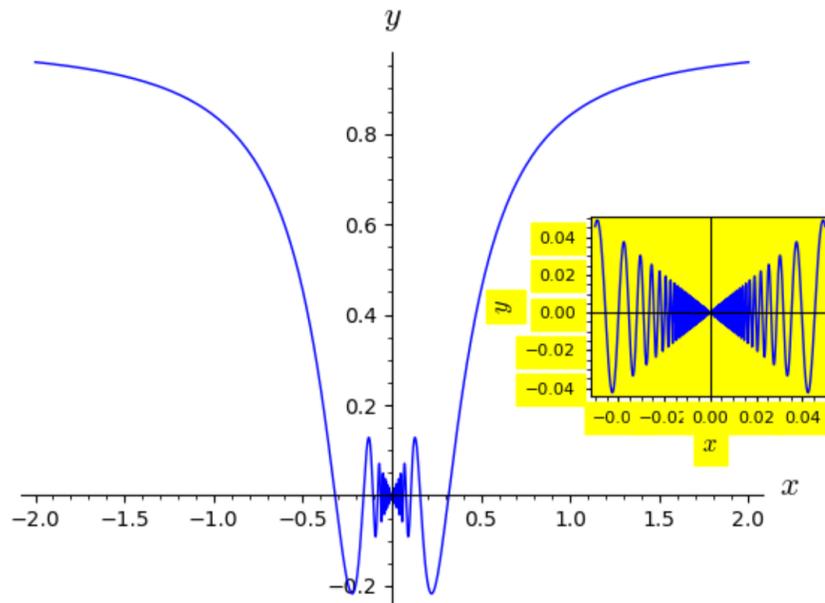
It can be saved into a pdf file via the method `savefig`:

```
sage: figure.savefig("figure.pdf")
```

An example with a Matplotlib keyword:



```
sage: figure = graphics_inset(g1, g2, (0.72, 0.4, 0.25, 0.25),
....:                          facecolor='yellow')
sage: figure
<Figure size 640x480 with 2 Axes>
```



PYTHON MODULE INDEX

k

kerrgeodesic_gw.astro_data, 79
kerrgeodesic_gw.gw_blob, 61
kerrgeodesic_gw.gw_particle, 39
kerrgeodesic_gw.kerr_geodesic, 17
kerrgeodesic_gw.kerr_spacetime, 1
kerrgeodesic_gw.lisa_detector, 69
kerrgeodesic_gw.signal_processing, 71
kerrgeodesic_gw.spin_weighted_spherical_harm, 31
kerrgeodesic_gw.spin_weighted_spheroidal_harm, 33
kerrgeodesic_gw.utilities, 81
kerrgeodesic_gw.zinf, 37

A

angular_momentum() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 3

B

blob_mass() (*in module kerrgeodesic_gw.gw_blob*), 61

boyer_lindquist_coordinates() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 5

C

cauchy_horizon_radius() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 6

check_integrals_of_motion() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 25

D

decay_time() (*in module kerrgeodesic_gw.gw_particle*), 40

E

evaluate_EC() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 25

evaluate_LC() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 25

evaluate_mu() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 26

evaluate_mu2() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 26

evaluate_QC() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 25

evaluate_tangent_vector() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 26

event_horizon_radius() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 6

F

fourier() (*in module kerrgeodesic_gw.signal_processing*), 71

G

geodesic() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 6

graphics_inset() (*in module kerrgeodesic_gw.utilities*), 81

H

h_amplitude_particle_fourier() (*in module kerrgeodesic_gw.gw_particle*), 41

h_blob() (*in module kerrgeodesic_gw.gw_blob*), 62

h_blob_signal() (*in module kerrgeodesic_gw.gw_blob*), 63

h_cross_particle() (*in module kerrgeodesic_gw.gw_particle*), 42

h_cross_particle_fourier() (*in module kerrgeodesic_gw.gw_particle*), 43

`h_particle_quadrupole()` (in module `kerrgeodesic_gw.gw_particle`), 46
`h_particle_signal()` (in module `kerrgeodesic_gw.gw_particle`), 47
`h_plus_particle()` (in module `kerrgeodesic_gw.gw_particle`), 49
`h_plus_particle_fourier()` (in module `kerrgeodesic_gw.gw_particle`), 52
`h_toy_model_semi_analytic()` (in module `kerrgeodesic_gw.gw_blob`), 64

I

`initial_tangent_vector()` (`kerrgeodesic_gw.kerr_geodesic.KerrGeodesic` method), 26
`inner_horizon_radius()` (`kerrgeodesic_gw.kerr_spacetime.KerrBH` method), 7
`integrate()` (`kerrgeodesic_gw.kerr_geodesic.KerrGeodesic` method), 27
`isco_radius()` (`kerrgeodesic_gw.kerr_spacetime.KerrBH` method), 7

K

`KerrBH` (class in `kerrgeodesic_gw.kerr_spacetime`), 1
`KerrGeodesic` (class in `kerrgeodesic_gw.kerr_geodesic`), 17
`kerrgeodesic_gw.astro_data`
 module, 79
`kerrgeodesic_gw.gw_blob`
 module, 61
`kerrgeodesic_gw.gw_particle`
 module, 39
`kerrgeodesic_gw.kerr_geodesic`
 module, 17
`kerrgeodesic_gw.kerr_spacetime`
 module, 1
`kerrgeodesic_gw.lisa_detector`
 module, 69
`kerrgeodesic_gw.signal_processing`
 module, 71
`kerrgeodesic_gw.spin_weighted_spherical_harm`
 module, 31
`kerrgeodesic_gw.spin_weighted_spheroidal_harm`
 module, 33
`kerrgeodesic_gw.utilities`
 module, 81
`kerrgeodesic_gw.zinf`
 module, 37

M

`map_to_Euclidean()` (`kerrgeodesic_gw.kerr_spacetime.KerrBH` method), 8
`marginally_bound_orbit_radius()` (`kerrgeodesic_gw.kerr_spacetime.KerrBH` method), 8
`mass()` (`kerrgeodesic_gw.kerr_spacetime.KerrBH` method), 9
`max_detectable_radius()` (in module `kerrgeodesic_gw.signal_processing`), 73
`metric()` (`kerrgeodesic_gw.kerr_spacetime.KerrBH` method), 9
module
 `kerrgeodesic_gw.astro_data`, 79
 `kerrgeodesic_gw.gw_blob`, 61
 `kerrgeodesic_gw.gw_particle`, 39
 `kerrgeodesic_gw.kerr_geodesic`, 17
 `kerrgeodesic_gw.kerr_spacetime`, 1

kerrgeodesic_gw.lisa_detector, 69
 kerrgeodesic_gw.signal_processing, 71
 kerrgeodesic_gw.spin_weighted_spherical_harm, 31
 kerrgeodesic_gw.spin_weighted_spheroidal_harm, 33
 kerrgeodesic_gw.utilities, 81
 kerrgeodesic_gw.zinf, 37

O

orbital_angular_velocity() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 9
 orbital_frequency() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 10
 outer_horizon_radius() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 11

P

photon_orbit_radius() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 11
 plot() (*kerrgeodesic_gw.kerr_geodesic.KerrGeodesic method*), 29
 plot_h_particle() (*in module kerrgeodesic_gw.gw_particle*), 53
 plot_spectrum_particle() (*in module kerrgeodesic_gw.gw_particle*), 55
 power_spectral_density() (*in module kerrgeodesic_gw.lisa_detector*), 69
 power_spectral_density_RCLfit() (*in module kerrgeodesic_gw.lisa_detector*), 69

R

radiated_power_particle() (*in module kerrgeodesic_gw.gw_particle*), 56
 read_signal() (*in module kerrgeodesic_gw.signal_processing*), 74
 roche_limit_radius() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 12
 roche_volume() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 13

S

save_signal() (*in module kerrgeodesic_gw.signal_processing*), 75
 secular_frequency_change() (*in module kerrgeodesic_gw.gw_particle*), 58
 signal_to_noise() (*in module kerrgeodesic_gw.signal_processing*), 75
 signal_to_noise_particle() (*in module kerrgeodesic_gw.signal_processing*), 76
 spin() (*kerrgeodesic_gw.kerr_spacetime.KerrBH method*), 15
 spin_weighted_spherical_harmonic() (*in module kerrgeodesic_gw.spin_weighted_spherical_harm*), 31
 spin_weighted_spheroidal_eigenvalue() (*in module kerrgeodesic_gw.spin_weighted_spheroidal_harm*), 33
 spin_weighted_spheroidal_harmonic() (*in module kerrgeodesic_gw.spin_weighted_spheroidal_harm*), 33
 strain_sensitivity() (*in module kerrgeodesic_gw.lisa_detector*), 69
 surface_density_gaussian() (*in module kerrgeodesic_gw.gw_blob*), 65
 surface_density_toy_model() (*in module kerrgeodesic_gw.gw_blob*), 66

Z

Zinf() (*in module kerrgeodesic_gw.zinf*), 37
 Zinf_Schwarzschild_PN() (*in module kerrgeodesic_gw.zinf*), 38