KerrGeoOrbit subpackage of KerrGeodesics

Define usage for public functions

```
BeginPackage["KerrGeodesics`KerrGeoOrbit`",
    {"KerrGeodesics`", (*FIXME, this is needed to get the RadialRoots function, m
     "KerrGeodesics`ConstantsOfMotion`",
     "KerrGeodesics`OrbitalFrequencies`"}];

KerrGeoOrbit::usage = "KerrGeoOrbit[a,p,e,x] returns a KerrGeoOrbitFunction[..] wh
KerrGeoOrbitFunction::usage = "KerrGeoOrbitFunction[a,p,e,x,assoc] an object for 

Begin["`Private`"];
```

Schwarzschild

The analytic equations below are taken from Appendix B of "Fast Self-forced Inspirals" by M. van de Meent and N. Warburton, Class. Quant. Grav. 35:144003 (2018), arXiv:1802.05281

```
(*t and ϕ accumulated over one orbit*)
ΦSchwarzDarwin[p_,e_]:=4 Sqrt[p/(p-6+2e)] EllipticK[(4 e)/(p-6+2e)]
TSchwarzDarwin[p_,e_]:=(2p Sqrt[(p-6+2e)((p-2)^2-4e^2)])/((1-e^2)(p-4)) EllipticE
```

```
tSchwarzDarwin[p_,e_,ξ_]:=TSchwarzDarwin[p,e]/2+((p Sqrt[(p-6+2e)((p-2)^2-4e^2)])
rSchwarzDarwin[p_,e_,χ_]:=p/(1 + e Cos[χ])
θSchwarzDarwin[p_,e_,χ_]:= π/2
ϕSchwarzDarwin[p_,e_,ξ_]:=ΦSchwarzDarwin[p,e]/2+2Sqrt[p/(p-6+2e)]EllipticF[ξ/2-π/
```

FIXME: make the below work for inclined orbits and accept initial phases

```
KerrGeoOrbitSchwarzDarwin[p_, e_] := Module[{t, r, θ, ϕ, assoc, consts, En,L,Q},

t[χ_] := tSchwarzDarwin[p,e,χ];
r[χ_] := rSchwarzDarwin[p,e,χ];
θ[χ_] := θSchwarzDarwin[p,e,χ];
ϕ[χ_] := ϕSchwarzDarwin[p,e,χ];

consts = KerrGeoConstantsOfMotion[0,p,e,1];
{En,L,Q} = Values[consts];

assoc = Association[
            "Trajectory" -> {t,r,θ,ϕ},
            "Parametrization" -> "Darwin",
            "ConstantsOfMotion"-> consts,
            "a" -> 0,
            "p" -> p,
            "e" -> e,
            "Inclination" -> 1,
            "Energy" -> En,
            "AngularMomentum" -> L,
            "CarterConstant" -> Q
            ];

KerrGeoOrbitFunction[0, p, e, 1, assoc]

]
```

Kerr

Equatorial (Darwin)

Compute the orbit using Mino time and then convert to Darwin time using $\lambda[r[\chi]]$ where $\lambda[r]$ is found in Fujita and Hikida (2009).

```
KerrGeoOrbitEquatorialDarwin[a_,p_,e_,x_/;x^2==1] := Module[{orbitMino,freqs,r1,r2

orbitMino = KerrGeoOrbit[a,p,e,x];

{r1,r2,r3,r4} = orbitMino["RadialRoots"];
freqs = orbitMino["Frequencies"];
consts = orbitMino["ConstantsOfMotion"];
{En,L,Q} = Values[consts];
```

```
Λr = (2π)/freqs[[1]];

yr[r_]:=Sqrt[(r1-r3)/(r1-r2) (r-r2)/(r-r3)];
kr=(r1-r2)/(r1-r3) (r3-r4)/(r2-r4);
λ0r[r_]:=1/Sqrt[1-En^2]×2/Sqrt[(r1-r3)(r2-r4)] EllipticF[ArcSin[yr[r]],kr];


r[χ_]:=p/(1+e Cos[χ]);

r01=r2;
Λr1=λ0r[r01];


λ[χ_]:=Λr Floor[χ/(2π)]+If[Mod[χ,2π]<=π, λ0r[r[χ]]-Λr1,Λr-λ0r[r[χ]]];
{tMino, rMino, θMino, ϕMino} = orbitMino["Trajectory"];

tDarwin[χ_]:= tMino[λ[χ]];
rDarwin[χ_]:= rMino[λ[χ]];
θDarwin[χ_]:= θMino[λ[χ]];
ϕDarwin[χ_]:= ϕMino[λ[χ]];

assoc = Association[
            "Trajectory" -> {tDarwin,rDarwin,θDarwin,ϕDarwin},
            "Parametrization" -> "Darwin",
            "ConstantsOfMotion"-> consts,
            "RadialRoots"->{r1,r2,r3,r4},
            "a" -> a,
            "p" -> p,
            "e" -> e,
            "Inclination" -> x,
            "Energy" -> En,
            "AngularMomentum" -> L,
            "CarterConstant" -> Q
        ];

KerrGeoOrbitFunction[a, p, e, 0, assoc]

]
```

Equatorial (Fast Spec - Darwin)

```
(* Hopper, Forseth, Osburn, and Evans, PRD 92 (2015)*)
```

Subroutines that checks for the number of samples necessary for spectral integration

```
Clear[DarwinFastSpecIntegrateAndConvergenceCheck];
DarwinFastSpecIntegrateAndConvergenceCheck[func_]:=
Module[{test,compare,res,NInit,iter=1,sampledFunc,phaseList,pg,eps,coeffs,
    coeffsList,coeffsN,△integratedFunc,growthRate,fn,nIter,sampleMax},
    (*DarwinFastSpecIntegrateAndConvergenceCheck takes a function 'func' and inte
    'func' with respect to the Darwin parameter χ using spectral methods:
                -- func: a function that takes an integer 'N' as an argument and
                         function values at N points. These points are sampled at
                         spaced values of the Darwin parameter χ  *)

    (* Memoize function that we are integrating with respect to the Darwin parame
    sampledFunc[NN_]:=sampledFunc[NN]=func[NN];
    (* Determine precision of sampled points.
    Use precision to check for convergence of spectral methods *)
    pg=Precision[sampledFunc[16][[2]]];

    (* Use Mathematica's built in DCT solver to determine DCT coefficients *)
    coeffs[Nr_]:=coeffs[Nr]=FourierDCT[sampledFunc[Nr],1]Sqrt[2/(Nr-1)]/.var_?Num
    (* Find the relative accuracy of the DCT coefficients by comparing the last t
    DCT coefficients to the n=0 coefficient *)
    res[Nr_]:=Min[Abs@RealExponent[coeffs[Nr][[-1]]/coeffs[Nr][[1]]],Abs@RealExpo

    (* Treat machine precision calculations differently from arbitrary precision
    If[pg==$MachinePrecision,
        (* eps sets the precision goal/numerical tolerance for our solutions *)
        eps=15;
        (* Set some initial value of sample points *)
        NInit=2^4;
        (* Find number of sample points necessary to match precision goal 'eps' *
        While[res[NInit]<eps&&iter<20,NInit=2*NInit;iter++]
        ,
        (* Set initial value of sample points at slightly larger value for extend
        NInit=2^5;
        (* Also test for convergence by increasing the sample size until the n=0
        compare=coeffs[NInit/2][[1]]; (*n=0 coefficient *)
        While[((compare =!= (compare = coeffs[NInit][[1]]))||res[NInit]<pg+1)&&it
    ];
    (* After determining the number of sampled points necessary for convergence, 
    coeffsList=coeffs[NInit];
    fn[n_]:=fn[n]=coeffsList[[n+1]];
```

```
    growthRate=coeffsList[[1]]/2;
    (* Evaluate new weighted coefficients due to integrating the interpolated inv
    If[pg==MachinePrecision,
        coeffsN[NInit-1]=fn[NInit-1]/(NInit-1);
        nIter=2^2+2;
        Do[coeffsN[n]=coeffsList[[n+1]]/n,{n,1,nIter-2}];
        eps=15-RealExponent[Sum[Abs[fn[n]],{n,0,nIter-2}]];
        While[(-RealExponent[fn[nIter]]<=eps||-RealExponent[fn[nIter-1]]<=eps)&&n
        coeffsN[nIter]=fn[nIter]/nIter;
        coeffsN[nIter-1]=fn[nIter-1]/(nIter-1);
        sampleMax=Min[nIter,NInit-2],
        Do[coeffsN[n]=coeffsList[[n+1]]/n,{n,1,NInit-1}];
        sampleMax=NInit-2;
    ];
    (* Construct integrated series solution *)
    ΔintegratedFunc[χ_?NumericQ]:=coeffsN[NInit-1]/2 Sin[(NInit-1)*χ]+Sum[coeffsN
    (* Allow function to evaluate lists by threading over them *)
    ΔintegratedFunc[χList_List]:=ΔintegratedFunc[#]&/@χList;
    (* Return the linear rate of growth and the oscillatory function ΔintegratedF
    {growthRate,ΔintegratedFunc}
];
```

Main file that calculates geodesics using spectral integration

```
Clear[KerrGeoOrbitFastSpecDarwin];
KerrGeoOrbitFastSpecDarwin[a_,p_,e_,x_/;x^2==1,initPhases:{_,_,_,_}:{0,0,0,0}]:=
Module[{M=1,consts,En,L,Q,r1,r2,r3,r4,p3,p4,assoc,var,t0, χ0, ϕ0,r0,θ0,t,r,θ,ϕ,χ,
        χr,NrMax,pg,Δtr,Δϕr,ϕC,tC,Pr,r0Sample,PrSample,dtdχ,dϕdχ,TVr,PVr},
    consts = KerrGeoConstantsOfMotion[a,p,e,x];
    {En,L,Q} = Values[consts];
    {r1,r2,r3,r4} = KerrGeodesics`OrbitalFrequencies`Private`KerrGeoRadialRoots[a
    p3=(1-e)r3/M;
    p4=(1+e)r4/M;

    (*Precision of sampling depends on precision of arguments*)
    pg=Min[{Precision[{a,p,e,x}],Precision[initPhases]}];

    (*Parameterize r in terms of Darwin parameter χ*)
    r0[chi_]:=p M/(1+e Cos[chi]);
    θ0[chi_?NumericQ]:=N[Pi/2,pg];
    θ0[chi_List]:=θ0[#]&/@chi;

    (* Expressions for dt/dλ = TVr and dϕ/dλ = PVr *)
```

```
TVr[rp_]:=(En*(a^2 + rp^2)^2)/(a^2 - 2*M*rp + rp^2) + a*L*(1 - (a^2 + rp^2)/(
PVr[rp_]:=-((a^2*L)/(a^2 - 2*M*rp + rp^2)) + a*En*(-1 + (a^2 + rp^2)/(a^2 - 2

(* Sampling of radial position using evenly spaced values of Darwin parameter
If[pg==$MachinePrecision,
    χr[Nr_]:=N[Table[i Pi/(Nr-1),{i,0,Nr-1}]],
    χr[Nr_]:=N[Table[i Pi/(Nr-1),{i,0,Nr-1}],1.5pg]
];
r0Sample[Nr_]:=r0[χr[Nr]];

(* Expression for dλ/dχ*)
Pr[chi_]:=(1-e^2)/Sqrt[(p-p4)+e(p-p4 Cos[chi])]/(M (1-En^2)^(1/2)Sqrt[(p-p3)-
PrSample[Nr_]:=Pr[χr[Nr]];

(* Sampling of expressions for dt/dχ and dϕ/dχ *)
dtdχ[Nr_]:=TVr[r0Sample[Nr]]×PrSample[Nr];
dϕdχ[Nr_]:=PVr[r0Sample[Nr]]×PrSample[Nr];

(*Spectral integration of t and ϕ as functions of χ*)
{growthRateT,Δtr}=DarwinFastSpecIntegrateAndConvergenceCheck[dtdχ];
{growthRatePh,Δϕr}=DarwinFastSpecIntegrateAndConvergenceCheck[dϕdχ];

(*Collect initial phases*)
{t0, χ0, ϕ0} = {initPhases[[1]], initPhases[[2]], initPhases[[4]]};
(* Find integration constants for t and ϕ, so that t(χ=0)=t0 and ϕ(χ=0)=ϕ0 *)
ϕC=Δϕr[χ0];
tC=Δtr[χ0];

t[χ_]:=Δtr[χ+χ0]+growthRateT χ+t0-tC;
r[χ_]:=r0[χ+χ0];
θ[χ_]:=θ0[χ];
ϕ[χ_]:=Δϕr[χ+χ0]+growthRatePh χ+ϕ0-ϕC;

assoc = Association[
    "Parametrization"->"Darwin",
    "Energy" -> En,
    "AngularMomentum" -> L,
    "CarterConstant" -> Q,
    "ConstantsOfMotion" -> consts,
    "Trajectory" -> {t,r,θ,ϕ},
    "RadialRoots" -> {r1,r2,r3,r4},
    "a" -> a,
    "p" -> p,
    "e" -> e,
```

```
        "Inclination" -> x
        ];

    KerrGeoOrbitFunction[a,p,e,x,assoc]
]
```

Circular (Fast Spec - Darwin)

```
(* Hopper, Forseth, Osburn, and Evans, PRD 92 (2015)*)
```

Main file that calculates geodesics using spectral integration

```
KerrGeoOrbitFastSpecDarwin[a_,p_,e_/;e==0,x_,initPhases:{_,_,_,_}:{0,0,0,0}]:=
Module[{M=1,consts,En,L,Q,zp,zm,assoc,var,t0, χ0, ϕ0,r0,θ0,t,r,θ,ϕ,χ,freqT,freqPh
        χθ,pg,Δtθ,Δϕθ,ϕC,tC,Pθ,θ0Sample,PθSample,dtdχ,dϕdχ,TVθ,PVθ,β,α,zRoots},
    consts = KerrGeoConstantsOfMotion[a,p,e,x];
    {En,L,Q} = Values[consts];

    (* Useful constants for θ-dependent calculations *)
    β=a^2(1-En^2);
    α=L^2+Q+β;
    zp=Sqrt[(α+Sqrt[α^2-4 Q β])/(2β)];
    zm=Sqrt[(α-Sqrt[α^2-4 Q β])/(2β)];
    zRoots={ArcCos[zm],Pi-ArcCos[zm]};

    (*Precision of sampling depends on precision of arguments*)
    pg=Min[{Precision[{a,p,e,x}],Precision[initPhases]}];

    (*Parameterize θ in terms of Darwin-like parameter χ*)
    r0[chi_?NumericQ]:=N[p M,pg];
    r0[chi_List]:=r0[#]&/@chi;
    θ0[chi_]:=ArcCos[zm Cos[chi]];

    (* Expressions for dt/dλ = TVθ and dϕ/dλ = PVθ *)
    TVθ[θp_]:=(En*(a^2 + M^2 p^2)^2)/(a^2 - 2*M^2*p + M^2p^2) + a*L*(1 - (a^2 + p
    PVθ[θp_]:=-((a^2*L)/(a^2 - 2*M^2*p + M^2p^2)) + a*En*(-1 + (a^2 + M^2p^2)/(a^

    (* Sampling of radial position using evenly spaced values of Darwin parameter
    If[pg==$MachinePrecision,
        χθ[Nth_]:=N[Table[i Pi/(Nth-1),{i,0,Nth-1}]],
        χθ[Nth_]:=N[Table[i Pi/(Nth-1),{i,0,Nth-1}],1.5pg]
    ];
    θ0Sample[Nth_]:=θ0[χθ[Nth]];
```

```
	(* Expression for dλ/dχ*)
	Pθ[chi_]:=(β(zp^2-zm^2 Cos[chi]^2))^(-1/2);
	PθSample[Nth_]:=Pθ[χθ[Nth]];

	(* Sampling of expressions for dt/dχ and dϕ/dχ *)
	dtdχ[Nr_]:=TVθ[θ0Sample[Nr]]×PθSample[Nr];
	dϕdχ[Nr_]:=PVθ[θ0Sample[Nr]]×PθSample[Nr];

	(*Spectral integration of t and ϕ as functions of χ*)
	{freqT,Δtθ}=DarwinFastSpecIntegrateAndConvergenceCheck[dtdχ];
	{freqPh,Δϕθ}=DarwinFastSpecIntegrateAndConvergenceCheck[dϕdχ];

	(*Collect initial phases*)
	{t0, χ0, ϕ0} = {initPhases[[1]], initPhases[[3]], initPhases[[4]]};
	(* Find integration constants for t and ϕ, so that t(χ=0)=t0 and ϕ(χ=0)=ϕ0 *)
	ϕC=Δϕθ[χ0];
	tC=Δtθ[χ0];

	t[χ_]:=Δtθ[χ+χ0]+freqT χ+t0-tC;
	r[χ_]:=r0[χ];
	θ[χ_]:=θ0[χ+χ0];
	ϕ[χ_]:=Δϕθ[χ+χ0]+freqPh χ+ϕ0-ϕC;

	assoc = Association[
		"Parametrization"->"Darwin",
		"Energy" -> En,
		"AngularMomentum" -> L,
		"CarterConstant" -> Q,
		"ConstantsOfMotion" -> consts,
		"Trajectory" -> {t,r,θ,ϕ},
		"PolarRoots" -> zRoots,
		"a" -> a,
		"p" -> p,
		"e" -> e,
		"Inclination" -> x
		];

	KerrGeoOrbitFunction[a,p,e,x,assoc]
]
```

Generic (Mino)

```
KerrGeoOrbitMino[a_,p_,e_,x_,initPhases:{_,_,_,_}:{0,0,0,0}]:=Module[{M=1,consts,
	consts = KerrGeoConstantsOfMotion[a,p,e,x];
```

```
    {En,L,Q} = Values[consts];
    {ϒr,ϒθ,ϒϕ,ϒt} = Values[KerrGeodesics`OrbitalFrequencies`Private`KerrGeoMinoFr
    {r1,r2,r3,r4} = KerrGeodesics`Private`KerrGeoRadialRoots[a, p, e, x, En, Q];
    {zp,zm} = KerrGeodesics`Private`KerrGeoPolarRoots[a, p, e, x];

    kr = (r1-r2)/(r1-r3) (r3-r4)/(r2-r4);
    kθ = a^2 (1-En^2)(zm/zp)^2;

rp=M+Sqrt[M^2-a^2];
rm=M-Sqrt[M^2-a^2];
hr=(r1-r2)/(r1-r3);
hp=((r1-r2)(r3-rp))/((r1-r3)(r2-rp));
hm=((r1-r2)(r3-rm))/((r1-r3)(r2-rm));

rq = Function[{qr},(r3(r1 - r2)JacobiSN[EllipticK[kr]/π qr,kr]^2-r2(r1-r3))/((r1-

zq = Function[{qz}, zm JacobiSN[EllipticK[kθ] 2/π (qz+π/2),kθ]];

ψr[qr_]:=ψr[qr]= JacobiAmplitude[EllipticK[kr]/π qr,kr];

tr[qr_]:= -En/Sqrt[(1-En^2) (r1-r3) (r2-r4)] (
4(r2-r3) (EllipticPi[hr,kr] qr/π-EllipticPi[hr,ψr[qr],kr])
-4 (r2-r3)/(rp-rm) (
-(1/((-rm+r2) (-rm+r3)))(-2 a^2+rm (4-(a L)/En)) (EllipticPi[hm,kr] qr/π-Elliptic
+1/((-rp+r2) (-rp+r3)) (-2 a^2+rp (4-(a L)/En)) (EllipticPi[hp,kr] qr/π-EllipticP
)
+(r2-r3) (r1+r2+r3+r4) (EllipticPi[hr,kr] qr/π-EllipticPi[hr,ψr[qr],kr] )
+(r1-r3) (r2-r4) (EllipticE[kr] qr/π-EllipticE[ψr[qr],kr]+hr((Sin[ψr[qr]]Cos[ψr[q

ϕr[qr_]:= (2 a En (-1/((-rm+r2) (-rm+r3))(2 rm-(a L)/En) (r2-r3) (EllipticPi[hm,k

ψz[qz_]:= ψz[qz] = JacobiAmplitude[EllipticK[kθ] 2/π (qz+π/2),kθ];
tz[qz_]:= 1/(1-En^2) En zp ( EllipticE[kθ]2((qz+π/2)/π)-EllipticE[ψz[qz],kθ]);
ϕz[qz_]:= -1/zp L ( EllipticPi[zm^2,kθ]2((qz+π/2)/π)-EllipticPi[zm^2,ψz[qz],kθ]);

{qt0, qr0, qz0, qϕ0} = {initPhases[[1]], initPhases[[2]], initPhases[[3]], initPha
(*Calculate normalization constants so that t=0 and ϕ=0 at λ=0 when qt0=0 and qϕ0
Ct=tr[qr0]+tz[qz0]/.i_/;i==0:>0;
Cϕ=ϕr[qr0]+ϕz[qz0]/.i_/;i==0:>0;

t[λ_]:= qt0 + ϒt λ + tr[ϒr λ + qr0] + tz[ϒθ λ + qz0]-Ct;
r[λ_]:= rq[ϒr λ+ qr0];
θ[λ_]:= ArcCos[zq[ϒθ λ + qz0]];
ϕ[λ_]:= qϕ0 + ϒϕ λ + ϕr[ϒr λ+ qr0] + ϕz[ϒθ λ + qz0]-Cϕ;
```

```
    assoc = Association[
    "Parametrization"->"Mino",
    "Energy" -> En,
    "AngularMomentum" -> L,
    "CarterConstant" -> Q,
    "ConstantsOfMotion" -> consts,
    "RadialFrequency" -> ϒr,
    "PolarFrequency" ->  ϒθ,
    "AzimuthalFrequency" -> ϒϕ,
    "Frequencies" -> {ϒr,ϒθ,ϒϕ},
    "Trajectory" -> {t,r,θ,ϕ},
    "RadialRoots" -> {r1,r2,r3,r4},
    "a" -> a,
    "p" -> p,
    "e" -> e,
    "Inclination" -> x
    ];

    KerrGeoOrbitFunction[a,p,e,x,assoc]

]
```

Generic (Fast Spec - Mino)

```
(* Hopper, Forseth, Osburn, and Evans, PRD 92 (2015)*)
```

Subroutines for calculating $\lambda(\psi)$ and $\lambda(\chi)$

```
Clear[MinoRFastSpec];
MinoRFastSpec[a_,p_,e_,x_]:=
Module[{M=1,En,L,Q,ϒr,ϒθ,ϒϕ,ϒt,r1,r2,r3,r4,ψr,r0,p3,p4,Pr,PrSample,NrInit=2^4,
        𝒫rSample,𝒫rn,𝒫rList,Δλr,pg,iter=1,compare,res,rate},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    {r1,r2,r3,r4} = KerrGeodesics`Private`KerrGeoRadialRoots[a, p, e, x, En, Q];
    p3=(1-e)r3/M;
    p4=(1+e)r4/M;
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        ψr[Nr_]:=N[Table[i Pi/(Nr-1),{i,0,Nr-1}]],
        ψr[Nr_]:=N[Table[i Pi/(Nr-1),{i,0,Nr-1}],1.5pg]
    ];
```

```
    Pr[psi_]:=Pr[psi]=(1-e^2)((p-p4)+e(p-p4 Cos[psi]))^(-1/2)/(M (1-En^2)^(1/2)((
    PrSample[Nr_]:=Pr[ψr[Nr]];

    {rate,△λr}=DarwinFastSpecIntegrateAndConvergenceCheck[PrSample];
    △λr
];

Clear[MinoThetaFastSpec];
MinoThetaFastSpec[a_,p_,e_,x_]:=
Module[{M=1,En,L,Q,zp,zm,χθ,β,Pθ,PθSample,NthInit=2^4,𝒫θList,𝒫θSample,
    𝒫θk,△λθ,pg,iter=1,compare,res,α,rate},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    β=a^2(1-En^2);
    α=L^2+Q+β;
    zp=Sqrt[(α+Sqrt[α^2-4 Q β])/(2β)];
    zm=Sqrt[(α-Sqrt[α^2-4 Q β])/(2β)];
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        χθ[Nth_]:=N[Table[i Pi/(Nth-1),{i,0,Nth-1}]],
        χθ[Nth_]:=N[Table[i Pi/(Nth-1),{i,0,Nth-1}],1.5pg]
    ];
    Pθ[chi_]:=Pθ[chi]=(β(zp^2-zm^2 Cos[chi]^2))^(-1/2);
    PθSample[Nth_]:=Pθ[χθ[Nth]];

    {rate,△λθ}=DarwinFastSpecIntegrateAndConvergenceCheck[PθSample];
    △λθ
];
```

Subroutine for calculating $\psi(\lambda)$ and $\chi(\lambda)$

```
PhaseOfMinoFastSpec[ϒ_,sampledMino_]:=
Module[{sampledFunc,NInit,phase},
    NInit=Length[sampledMino];
    sampledFunc[NN_]:=ConstantArray[1,NN];
    phase=DarwinFastSpecMinoIntegrateAndConvergenceCheck[sampledFunc,{ϒ,sampledMi
    phase
];
```

Subroutines for calculating $\Delta\phi r(\lambda)$, $\Delta\phi\theta(\lambda)$, $\Delta tr(\lambda)$, $\Delta t\theta(\lambda)$

```
PhiOfMinoFastSpecR[a_,p_,e_,x_,{ϒr_,minoSampleR_}]:=
Module[{M=1,En,L,Q,sampledFuncR,sampledMinoR,PVr,Δϕr},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    PVr[rp_]:=-((a^2*L)/(a^2 - 2*M*rp + rp^2)) + a*En*(-1 + (a^2 + rp^2)/(a^2 - 2

    sampledFuncR=LambdaToPsiRTransform[a,p,e,x,PVr];
    Δϕr=DarwinFastSpecMinoIntegrateAndConvergenceCheck[sampledFuncR,{ϒr,minoSampl
    Δϕr
];

PhiOfMinoFastSpecR[a_,p_,e_,x_]:=Module[{ϒr,Δλr,λr,λrSample,pg,ψr},
    ϒr = Re[KerrGeodesics`OrbitalFrequencies`Private`KerrGeoMinoFrequencies[a,p,e
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        ψr[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}]],
        ψr[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}],1.5pg]
    ];
    Δλr=MinoRFastSpec[a,p,e,x];
    λr[psi_]:=λr[psi]=Δλr[psi];
    λrSample[Nr_]:=λr[ψr[Nr]];
    PhiOfMinoFastSpecR[a,p,e,x,{ϒr,λrSample}]
];
```

```
PhiOfMinoFastSpecTheta[a_,p_,e_,x_,{ϒθ_,minoSampleTh_}]:=
Module[{M=1,En,L,Q,sampledFuncTheta,sampledMinoTheta,PVθ,Δϕθ},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    PVθ[θp_]:=L*Csc[θp]^2;

    sampledFuncTheta=LambdaToChiThetaTransform[a,p,e,x,PVθ];
    Δϕθ=DarwinFastSpecMinoIntegrateAndConvergenceCheck[sampledFuncTheta,{ϒθ,minoS
    Δϕθ
];

PhiOfMinoFastSpecTheta[a_,p_,e_,x_]:=Module[{ϒθ,Δλθ,λθ,λθSample,pg,χθ},
    ϒθ = Re[KerrGeodesics`OrbitalFrequencies`Private`KerrGeoMinoFrequencies[a,p,e
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        χθ[Nth_]:=N[Table[i 2 Pi/Nth,{i,0,Nth-1}]],
        χθ[Nth_]:=N[Table[i 2 Pi/Nth,{i,0,Nth-1}],1.5pg]
    ];
    Δλθ=MinoRFastSpec[a,p,e,x];
    λθ[psi_]:=λθ[psi]=Δλθ[psi];
    λθSample[Nr_]:=λθ[χθ[Nr]];
    PhiOfMinoFastSpecTheta[a,p,e,x,{ϒθ,λθSample}]
];
```

```
TimeOfMinoFastSpecR[a_,p_,e_,x_,{ϒr_,minoSampleR_}]:=
Module[{M=1,En,L,Q,sampledFuncR,sampledMinoR,TVr,Δtr},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    TVr[rp_]:=(En*(a^2 + rp^2)^2)/(a^2 - 2*M*rp + rp^2) + a*L*(1 - (a^2 + rp^2)/(

    sampledFuncR=LambdaToPsiRTransform[a,p,e,x,TVr];
    Δtr=DarwinFastSpecMinoIntegrateAndConvergenceCheck[sampledFuncR,{ϒr,minoSampl
    Δtr
];

TimeOfMinoFastSpecR[a_,p_,e_,x_]:=Module[{ϒr,Δλr,λr,λrSample,pg,ψr},
    ϒr = Re[KerrGeodesics`OrbitalFrequencies`Private`KerrGeoMinoFrequencies[a,p,e
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        ψr[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}]],
        ψr[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}],1.5pg]
    ];
    Δλr=MinoRFastSpec[a,p,e,x];
    λr[psi_]:=λr[psi]=Δλr[psi];
    λrSample[Nr_]:=λr[ψr[Nr]];
    TimeOfMinoFastSpecR[a,p,e,x,{ϒr,λrSample}]
];
```

```
TimeOfMinoFastSpecTheta[a_,p_,e_,x_,{ϒθ_,minoSampleTh_}]:=
Module[{M=1,En,L,Q,sampledFuncTheta,sampledMinoTheta,TVθ,Δtθ},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    TVθ[θp_]:=-(a^2*En*Sin[θp]^2);

    sampledFuncTheta=LambdaToChiThetaTransform[a,p,e,x,TVθ];
    Δtθ=DarwinFastSpecMinoIntegrateAndConvergenceCheck[sampledFuncTheta,{ϒθ,minoS
    Δtθ
];

TimeOfMinoFastSpecTheta[a_,p_,e_,x_]:=Module[{ϒθ,Δλθ,λθ,λθSample,pg,χθ},
    ϒθ = Re[KerrGeodesics`OrbitalFrequencies`Private`KerrGeoMinoFrequencies[a,p,e
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        χθ[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}]],
        χθ[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}],1.5pg]
    ];
    Δλθ=MinoRFastSpec[a,p,e,x];
    λθ[psi_]:=λθ[psi]=Δλθ[psi];
    λθSample[Nr_]:=λθ[χθ[Nr]];
    TimeOfMinoFastSpecTheta[a,p,e,x,{ϒθ,λθSample}]
];
```

Generic subroutines that transform functions from λ dependence to ψ or χ dependence

```
LambdaToPsiRTransform[a_,p_,e_,x_,rFunc_]:=
Module[{M=1,En,L,Q,r1,r2,r3,r4,p3,p4,ψr,r0,r0Sample,Pr,PrSample,rFuncSample,pg},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    {r1,r2,r3,r4} = KerrGeodesics`Private`KerrGeoRadialRoots[a, p, e, x, En, Q];
    p3=(1-e)r3/M;
    p4=(1+e)r4/M;
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        ψr[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}]],
        ψr[Nr_]:=N[Table[i 2 Pi/Nr,{i,0,Nr-1}],1.5pg]
    ];
    r0[psi_]:=p M/(1+e Cos[psi]);
    r0Sample[Nr_]:=r0[ψr[Nr]];

    Pr[psi_]:=(1-e^2)/Sqrt[(p-p4)+e(p-p4 Cos[psi])]/(M (1-En^2)^(1/2)Sqrt[(p-p3)-
    PrSample[Nr_]:=Pr[ψr[Nr]];

    rFuncSample[Nr_]:=rFunc[r0Sample[Nr]]×PrSample[Nr];
    rFuncSample
];
```

```
LambdaToChiThetaTransform[a_,p_,e_,x_,thFunc_]:=
Module[{M=1,En,L,Q,zp,zm,β,χθ,θ0,θ0Sample,Pθ,PθSample,thFuncSample,pg,α},
    {En,L,Q} = Values[KerrGeoConstantsOfMotion[a,p,e,x]];
    β=a^2(1-En^2);
    α=L^2+Q+β;
    zp=Sqrt[(α+Sqrt[α^2-4 Q β])/(2β)];
    zm=Sqrt[(α-Sqrt[α^2-4 Q β])/(2β)];
    pg=Precision[{a,p,e,x}];

    If[pg==$MachinePrecision,
        χθ[Nth_]:=N[Table[i 2 Pi/Nth,{i,0,Nth-1}]],
        χθ[Nth_]:=N[Table[i 2 Pi/Nth,{i,0,Nth-1}],1.5pg]
    ];
    θ0[chi_]:=ArcCos[zm Cos[chi]];
    θ0Sample[Nth_]:=θ0[χθ[Nth]];

    Pθ[chi_]:=(β(zp^2-zm^2 Cos[chi]^2))^(-1/2);
    PθSample[Nth_]:=Pθ[χθ[Nth]];

    thFuncSample[Nth_]:=thFunc[θ0Sample[Nth]]×PθSample[Nth];
    thFuncSample
];
```

Subroutines that checks for the number of samples necessary for spectral integration of an even function

```
Clear[DarwinFastSpecMinoIntegrateAndConvergenceCheck];
DarwinFastSpecMinoIntegrateAndConvergenceCheck[func_,{freq_,mino_}]:=
Module[{test,compare,res,NInit,iter=1,fn,sampledFunc,phaseList,pg,eps,nTest},
    (*
    DarwinFastSpecMinoIntegrateAndConvergenceCheck takes an even function 'func'
    and determines the number of samples necessary to integrate 'func' with
    respect to Mino time λ using spectral sampling in the Darwin-like parameters
    ψ and χ:
                -- func: a function that includes function values
                that result from sampling the function at evenly spaced values
                of the Darwin-like parameters ψ or χ
                -- freq: Mino time frequency with respect to r or θ (ϒr or ϒθ)
                -- mino: a list of Mino time values that results from
                sampling the function at evenly spaced values of the Darwin-like
                parameters ψ or χ
    *)
```

```
(* Memoize function that we are integrating with respect to the Darwin parame
sampledFunc[NN_]:=sampledFunc[NN]=func[NN];
(* Determine precision of arguments.
Use precision to check for convergence of spectral methods *)
pg=Precision[freq];

(* Treat machine precision calculations differently from arbitrary precision
If[pg==MachinePrecision,
    (* eps sets the precision goal/numerical tolerance for our solutions *)
    eps=15;
    (* Set some initial value of sample points *)
    NInit=2^3;
    (* Sampled points need to also be weighted by the Mino time *)
    phaseList[NN_]:=phaseList[NN]=freq*mino[NN]+N[Table[2Pi i/NN,{i,0,NN-1}]]
    (* Create functions for discrete cosine series coefficient fn *)
    fn[NN_,nn_]:=Total[sampledFunc[NN]Cos[nn phaseList[NN]]]/NN;
    (* Test convergence by comparing coefficients to the n=0 coefficient*)
    nTest=0;
    test[NN_]:=fn[NN,nTest];
    (* If the n=0 coefficient vanishes or is unity, compare against n=1 coeff
    If[test[NInit]==0||test[NInit]==1,nTest=1];
    (* Find the relative accuracy of the DFT coefficients by comparing the la
    DFT coefficient to the test coefficient set above *)
    res[NN_]:=Abs@RealExponent[(fn[NN,NN/2]/.{y_/;y==0:>0})/fn[NN,0]];
    (* Find number of sample points necessary to match precision goal 'eps' *
    While[res[NInit]<eps&&iter<8,NInit=2*NInit; iter++]
    ,
    (* Set some initial value of sample points *)
    NInit=2^6;
    (* Same process as above but with higher precision tolerances *)
    phaseList[NN_]:=phaseList[NN]=freq*mino[NN]+N[Table[2Pi i/NN,{i,0,NN-1}],
    fn[NN_,nn_]:=Total[sampledFunc[NN]Cos[nn phaseList[NN]]]/NN;
    nTest=0;
    test[NN_]:=fn[NN,nTest];
    If[test[NInit]==0||test[NInit]==1,nTest=1];
    res[NN_]:=Abs@RealExponent[(fn[NN,NN/2]/.{y_/;y==0:>0})/fn[NN,0]];
    (* Also test for convergence by increasing the sample size until the test
    compare=test[NInit/2];
    While[((compare =!= (compare = test[NInit]))||res[NInit/2]<pg+1)&&iter<10
    (* Compare residuals by comparing to a different Fourier coefficient to e
    (* This part might be overkill *)
    nTest++;
    compare=test[NInit/2];
    iter=1;
```

```
        While[((compare =!= (compare = test[NInit]))||res[NInit/2]<pg+1)&&iter<10
        (* If the Fourier coefficients were unaffected after doubling the samplin
        then we can sample using the previous number of sample points *)
        If[res[NInit]==0&&res[NInit/2]!=0,NInit,NInit=NInit/2];
    ];
    (* After determining the necessary number of sample points, we sample the fun
    we want to integrate and the Mino time, and pass both lists of sampled points
    spectral integrator *)
    DarwinFastSpecMinoIntegrateEven[sampledFunc[NInit],{freq,mino[NInit]}]
];
```

Subroutine that performs spectral integration on even functions

```
DarwinFastSpecMinoIntegrateEven[sampledFunctionTemp_,{freq_,minoTimeTemp_}]:=
Module[{sampledF,fn,fList,f,sampleN,λ,integratedF,phaseList,pg,nn,samplePhase,nLi
        sampleMax,eps,sampleHalf},
    (*
    DarwinFastSpecMinoIntegrateEven takes an even function 'sampledFunctionTemp'
    and integrates 'sampledFunctionTemp' with respect to Mino time λ using spectr
    sampling in the Darwin-like parameters ψ and χ:
                -- sampledFunctionTemp: a list that includes function values
                that result from sampling the function at evenly spaced values
                of the Darwin-like parameters ψ or χ
                -- freq: Mino time frequency with respect to r or θ (ϒr or ϒθ)
                -- minoTimeTemp: a list of Mino time values that results from
                sampling the function at evenly spaced values of the Darwin-like
                parameters ψ or χ
    *)

    (* Represent values equal to zero with infinite precision *)
    sampledF=sampledFunctionTemp/.x_?NumericQ/;x==0:>0;
    λ=minoTimeTemp/.x_?NumericQ/;x==0:>0;

    (* Determine the number of sampled points and precision of arguments *)
    sampleN=Length[sampledF];
    pg=Precision[freq];

    (* Use precision and number of sampled points to generate list of
    evenly spaced values of ψ or χ *)
    If[pg==$MachinePrecision,
        phaseList=N[Table[2Pi i/sampleN,{i,0,sampleN-1}]],
        phaseList=N[Table[2Pi i/sampleN,{i,0,sampleN-1}],1.5pg]
    ];
```

```
    (* Create functions for discrete cosine series coefficient fn *)
    samplePhase=(freq λ+phaseList);
    fn[n_]:=fn[n]=(sampledF.Cos[nn*samplePhase])/.nn->n;

    (* Calculate series coefficients until they equal 0 (with respect
    to the precision being used) *)
    If[pg==MachinePrecision,
        fList=Block[{halfSample,nIter},
            nIter=2^2+2;
            eps=15-RealExponent[Sum[Abs@fn[n],{n,0,nIter-2}]];
            While[(-RealExponent[fn[nIter]]<=eps||-RealExponent[fn[nIter-1]]<=eps
            sampleMax=Min[nIter,sampleN/2];
            1/sampleN fn[Table[n,{n,0,sampleMax}]]/.x_?NumericQ/;x==0:>0
        ],
        fList=Block[{halfSample,nIter},
            nIter=2^5+2;
            While[(fn[nIter]!=0||fn[nIter-1]!=0)&&nIter<sampleN/2,nIter+=2];
            sampleMax=Min[nIter,sampleN/2];
            1/sampleN fn[Table[n,{n,0,sampleMax}]]/.x_?NumericQ/;x==0:>0
        ]
    ];
    f[n_]:=fList[[n+1]];

    (* Construct integrated series solution *)
    integratedF[mino_?NumericQ]:=2*Sum[f[n]/n Sin[n freq mino],{n,1,sampleMax}];
    (* Allow function to evaluate lists by threading over them *)
    integratedF[minoList_List]:=integratedF[#]&/@minoList;
    integratedF
];
```

Main file that calculates geodesics using spectral integration

```
Clear[KerrGeoOrbitFastSpec];
Options[KerrGeoOrbitFastSpec]={InitialPosition->{0,0,0,0}};
KerrGeoOrbitFastSpec[a_,p_,e_,x_,initPhases:{_,_,_,_}:{0,0,0,0},opts:OptionsPatte
Module[{M=1,consts,En,L,Q,ϒr,ϒθ,ϒϕ,ϒt,r1,r2,r3,r4,p3,p4,α,β,zp,zm,assoc,var,χ0,ψ0
        r0,θ0,qt0,qr0,qθ0,qϕ0,λt0,λr0,λθ0,λϕ0,t,r,θ,ϕ,ψ,χ,Δλr,λr,Δλθ,λθ,rC,θC,Δr,
        ψr,χθ,NrMax,NthMax,pg,λrSample,λθSample,Δtr,Δϕr,Δtθ,Δϕθ,ϕC,tC,zRoots,tIni
    consts = KerrGeoConstantsOfMotion[a,p,e,x];
    {En,L,Q} = Values[consts];

    (* Useful constants for θ-dependent calculations *)
```

```
β=a^2(1-En^2);
α=L^2+Q+β;
zp=Sqrt[(α+Sqrt[α^2-4 Q β])/(2β)];
zm=Sqrt[(α-Sqrt[α^2-4 Q β])/(2β)];
zRoots={ArcCos[zm],Pi-ArcCos[zm]};

(* Useful constants for r-dependent calculations *)
{r1,r2,r3,r4} = KerrGeodesics`Private`KerrGeoRadialRoots[a, p, e, x, En, Q];

(* Mino frequencies of orbit. I call Re, because some frequencies
are given as imaginary for restricted orbits.
Maybe something to fix with KerrGeoMinoFrequencies*)
{ϒr,ϒθ,ϒϕ,ϒt} = Values[KerrGeodesics`OrbitalFrequencies`Private`KerrGeoMinoFr
If[e>0&&(Im[ϒr]!=0||Re[ϒr]==0),Print["Unstable orbit. Aborting."]; Abort[]];
If[r2<1+Sqrt[1-a^2],Print["Unstable orbit. Aborting."]; Abort[]];
{ϒr,ϒθ,ϒϕ,ϒt} = Re[{ϒr,ϒθ,ϒϕ,ϒt}];

(*Parameterize r and θ in terms of Darwin-like parameters ψ and χ*)
r0[psi_]:=p M/(1+e Cos[psi]);
θ0[chi_]:=ArcCos[zm Cos[chi]];

(*Precision of sampling depends on precision of arguments*)
pg=Min[{Precision[{a,p,e,x}],Precision[initPhases]}];
If[pg==$MachinePrecision,
    ψr[Nr_]:=N[Table[2Pi i/Nr,{i,0,Nr-1}]];
    χθ[Nth_]:=N[Table[2Pi i/Nth,{i,0,Nth-1}]],
    ψr[Nr_]:=N[Table[2Pi i/Nr,{i,0,Nr-1}],1.3pg];
    χθ[Nth_]:=N[Table[2Pi i/Nth,{i,0,Nth-1}],1.3pg]
];

(*Solve for Mino time as a function of ψ and χ*)
Δλr=MinoRFastSpec[a,p,e,x];
λr[psi_]:=λr[psi]=Δλr[psi];
λrSample[Nr_]:=λr[ψr[Nr]];
λθ=MinoThetaFastSpec[a,p,e,x];
λθ[chi_]:=λθ[chi]=Δλθ[chi];
λθSample[Nth_]:=λθ[χθ[Nth]];

(*Find the inverse transformation of ψ and χ as functions of λ
using spectral integration*)
If[e>0,
    ψ=PhaseOfMinoFastSpec[ϒr,λrSample],
    ψ[λ_]:=0
];
```

```
If[x^2<1,
    χ=PhaseOfMinoFastSpec[ϒθ,λθSample],
    χ[λ_]:=0
];

(*Spectral integration of t and ϕ as functions of λ*)
If[e>0,
    Δtr=TimeOfMinoFastSpecR[a,p,e,x,{ϒr,λrSample}];
    Δϕr=PhiOfMinoFastSpecR[a,p,e,x,{ϒr,λrSample}],
    Δtr[λ_?NumericQ]:=0;
    Δtr[λ_List]:=Δtr[#]&/@λ;
    Δϕr[λ_?NumericQ]:=0;
    Δϕr[λ_List]:=Δϕr[#]&/@λ;
];
If[x^2<1,
    (* Calculate theta dependence for non-equatorial orbits *)
    Δtθ=TimeOfMinoFastSpecTheta[a,p,e,x,{ϒθ,λθSample}];
    Δϕθ=PhiOfMinoFastSpecTheta[a,p,e,x,{ϒθ,λθSample}],
    (* No theta dependence for equatorial orbits *)
    Δtθ[λ_?NumericQ]:=0;
    Δtθ[λ_List]:=Δtθ[#]&/@λ;
    Δϕθ[λ_?NumericQ]:=0;
    Δϕθ[λ_List]:=Δϕθ [#]&/@λ;
];

(*Collect initial Mino time phases*)
{qt0, qr0, qθ0, qϕ0} = {initPhases[[1]], initPhases[[2]], initPhases[[3]], in

(* If the user specifies a valid set of initial coordinate positions,
find phases that give these initial positions *)
{tInit,rInit,θInit,ϕInit}=OptionValue[InitialPosition];
If[{tInit,rInit,θInit,ϕInit}!={0,0,0,0},
    If[rInit<=r1&&rInit>=r2&&θInit>=zRoots[[1]]&&θInit<=zRoots[[2]],
        If[e==0,ψ0=0,ψ0=ArcCos[p M/(e rInit)-1/e]];
        If[zm==0,χ0=0,χ0=ArcCos[Cos[θInit]/zm]];
        qt0=tInit;
        qr0=ϒr*λr[ψ0]+ψ0;
        qθ0=ϒθ*λθ[χ0]+χ0;
        qϕ0=ϕInit,
        Print["{t,r,θ,ϕ} = "<>ToString[{tInit,rInit,θInit,ϕInit}]<>" is not a
    ]
];
If[ϒr==0,λr0=0,λr0=qr0/ϒr];
If[ϒθ==0,λθ0=0,λθ0=qθ0/ϒθ];
```

```
    (* Find integration constants for t and ϕ, so that t(λ=0)=qt0 and ϕ(λ=0)=qϕ0
    ϕC=Δϕr[λr0]+Δϕθ[λθ0];
    tC=Δtr[λr0]+Δtθ[λθ0];

    t[λ_]:=Δtr[λ+λr0]+Δtθ[λ+λθ0]+Υt λ+qt0-tC;
    r[λ_]:=r0[ψ[λ+λr0]+Υr λ+qr0];
    θ[λ_]:=θ0[χ[λ+λθ0]+Υθ λ+qθ0];
    ϕ[λ_]:=Δϕr[λ+λr0]+Δϕθ[λ+λθ0]+Υϕ λ+qϕ0-ϕC;

    assoc = Association[
        "Parametrization"->"Mino",
        "Energy" -> En,
        "AngularMomentum" -> L,
        "CarterConstant" -> Q,
        "ConstantsOfMotion" -> consts,
        "RadialFrequency" -> Υr,
        "PolarFrequency" ->  Υθ,
        "AzimuthalFrequency" -> Υϕ,
        "Frequencies" -> {Υr,Υθ,Υϕ},
        "Trajectory" -> {t,r,θ,ϕ},
        "RadialRoots" -> {r1,r2,r3,r4},
        "PolarRoots" -> zRoots,
        "a" -> a,
        "p" -> p,
        "e" -> e,
        "Inclination" -> x
        ];

    KerrGeoOrbitFunction[a,p,e,x,assoc]
]
```

KerrGeoOrbit and KerrGeoOrbitFuction

```
Options[KerrGeoOrbit] = {"Parametrization" -> "Mino", "Method" -> "FastSpec"}
SyntaxInformation[KerrGeoOrbit] = {"ArgumentsPattern"->{_,_,OptionsPattern[]}};
```

```
KerrGeoOrbit[a_,p_,e_,x_, initPhases:{_,_,_,_}:{0,0,0,0},OptionsPattern[]]:=Modul
(*FIXME: add stability check but make it possible to turn it off*)

method = OptionValue["Method"];
param = OptionValue["Parametrization"];

If[param == "Darwin" && Abs[x]!=1, Print["Darwin parameterization only valid for 

If[Precision[{a,p,e,x}] > 30, method = "Analytic"];

If[method == "FastSpec",

    If[param == "Mino",  If[PossibleZeroQ[a], Return[KerrGeoOrbitMino[a, p, e, x,
    If[param == "Darwin",
        If[PossibleZeroQ[a], Return[KerrGeoOrbitSchwarzDarwin[p, e]], Return[Kerr
    ];
    Print["Unrecognized parametrization: " <> OptionValue["Parametrization"]];

];

If[method == "Analytic",

    If[param == "Mino", Return[KerrGeoOrbitMino[a, p, e, x, initPhases]]];
    If[param == "Darwin",
        If[PossibleZeroQ[a], Return[KerrGeoOrbitSchwarzDarwin[p, e]], Return[Kerr
    ];
    Print["Unrecognized parametrization: " <> OptionValue["Parametrization"]];

];

Print["Unrecognized method: " <> method];

]
```

```
Format[KerrGeoOrbitFunction[a_, p_, e_, x_, assoc_]] := "KerrGeoOrbitFunction["<>
KerrGeoOrbitFunction[a_, p_, e_, x_, assoc_][λ_/;StringQ[λ] == False] := Through[
KerrGeoOrbitFunction[a_, p_, e_, x_, assoc_][y_?StringQ] := assoc[y]
```

Close the package

```
End[];

EndPackage[];
```