```
BeginPackage["VectorFieldNorm`"]

AupNorm::usage = "A^\mu[t,r,θ,ϕ] :> Normalized contravariant vector field solution for
AdownNorm::usage = "A_\mu[t,r,θ,ϕ] :> Normalized covariant vector field solution for g
VectorField::usage = "VectorField[Rsol(interpolating func.),Anglsol(explicit in terms o
edensityNorm::usage = "ρ[t,r,θ,ϕ] :> Normalized energy density of field mode."
rtst
rstop
θstart
θstop
Anorm
massNorm
massNormint
dataout
functionplotsR
functionplotsS
```

```
BBLtemp1
BBLtemp2

Rad
```

```
dRad
ddRad
Sangl
dSangl
ddSangl

Begin["Private`"]
Needs["xAct`xCoba`"]

(*xAct initializations*)
$DefInfoQ=False;
$CVVerbose=False;
$PrePrint=ScreenDollarIndices;
$ShowTimeThreshold=0.5
DefManifold[M,4,IndexRange[b,l]];
DefMetric[-1,met[-b,-c],CD,PrintAs→"g"];
DefChart[ch,M,{0,1,2,3},{t[],r[],θ[],ϕ[]}];
ch/:CIndexForm[0,ch]:="t";
ch/:CIndexForm[1,ch]:="r";
ch/:CIndexForm[2,ch]:="θ";
ch/:CIndexForm[3,ch]:="ϕ";
```

```
DefConstantSymbol[χ]

(*Defining the Kerr metric*)
Σ=r[]^2+χ^2 Cos[θ[]]^2;
Δ=r[]^2-2 r[]+χ^2;
KerrBL={{-(1-(2r[])/Σ),0,0,(-2 r[] χ Sin[θ[]]^2)/Σ},{0,Σ/Δ,0,0},{0,0,Σ,0},{(-2 r[] χ Sin[θ[]]^2)/Σ,0,0,
ComponentValue[met[-b,-c]//ToBasis[ch]//ComponentArray,KerrBL]
MetricCompute[met,ch,All]
MetricCompute[met,ch,"Christoffel"[1,-1,-1],CVSimplify→Simplify]
MetricCompute[met,ch,"Ricci"[-1,-1],CVSimplify→Simplify]
eval[A_]:=A//ToBasis[ch]//ComponentArray//ToBasis[ch]//TraceBasisDummy//ToValues//Simplify
cd=CovDOfMetric[met];

(*Emulation of coordinate transformation from Dolan's coordinates to Boyer-Lindquist
DefTensor[Λ[b,-c],M]
ΛupdownFKKStoBL={{1,0,0,s^2},{0,1,0,0},{0,0,-1/(s Sin[θ[]]),0},{0,0,0,s}};
ComponentValue[Λ[b,-c]//ToBasis[ch]//ComponentArray,ΛupdownFKKStoBL];

(*Polarization tensor in Dolan's coordinates (differ from FKKS by a complex unit in fr
DefTensor[BDolan[b,c],M]
BupupFKKS={{-((-2 m r y^4+s^2 y^4-2 m r^3 y^4 ν^2+r^2 y^4 (1+s^2 ν^2)+r^4 (y^2+s^2 (-1+y^2
```

```
ComponentValue[BDolan[b,c]//ToBasis[ch]//ComponentArray,BupupFKKS];

(*Polarization tensor in Boyer-Lindquist coordinates*)
DefTensor[BBL[b,c],M]
BBLtemp1=(Λ[f,-g]Λ[d,-e]BDolan[g,e]//eval)/.{y→χ Cos[θ[]],s→χ}//Transpose;
BBLtemp2=(ΛupdownFKKStoBL.BupupFKKS).Transpose[ΛupdownFKKStoBL]/.{y→χ Cos[θ[]],s→χ};
BBLtest=(BBLtemp1-BBLtemp2)//Simplify;
Print["BBL test = "<>ToString[BBLtest]];
Print["Should be 0! If not, then BBL is not correct!"];
ComponentValue[BBL[b,c]//ToBasis[ch]//ComponentArray,BBLtemp2];

(*The Dolan eq. (8)*)
DefScalarFunction[R]
DefScalarFunction[S]
DefConstantSymbol[ω]
DefConstantSymbol[m]
DefConstantSymbol[νr]
DefConstantSymbol[νi]
DefConstantSymbol[ωr]
DefConstantSymbol[ωi]
Z[t[],r[],θ[],ϕ[]]=R[r[]]*S[θ[]]Exp[I m ϕ[]]Exp[-I ω t[]];
```

```
DefTensor[A[-b],M]
A/:A[c_]:=Module[{b},BBL[c,b]*CD[-b]@Z[t[],r[],θ[],ϕ[]]];
A/:A[-c_]:=Module[{b,e},met[-e,-c]*BBL[e,b]*CD[-b]@Z[t[],r[],θ[],ϕ[]]];

(*Analyitc form of the real part of the field A^\mu. This part takes about 50 min to
(*sin=Sin[m ϕ-ωr t];
cos=Cos[m ϕ-ωr t];
complextoreal={ν→rν+I iν,ω→ωr+I ωi,R[r[]]:→ Rr[r[]]+I Ri[r[]],R'[r[]]:→ Rr'[r[]]+I Ri'[r[]],S[θ[]]→Sr[

AupSimptemp=(A[b]//eval)//.complextoreal//.{t[]→t,r[]→r,θ[]→θ,ϕ[]→ϕ};
AupSimptemp1=CoefficientList[AupSimptemp[[1]]//Re//ComplexExpand,{sin,cos}]//Simplify; Print[
AupSimptemp2=CoefficientList[AupSimptemp[[2]]//Re//ComplexExpand,{sin,cos}]//Simplify; Print[
AupSimptemp3=CoefficientList[AupSimptemp[[3]]//Re//ComplexExpand,{sin,cos}]//Simplify; Print[
AupSimptemp4=CoefficientList[AupSimptemp[[4]]//Re//ComplexExpand,{sin,cos}]//Simplify; Print[

AupSimp={sin AupSimptemp1[[2,1]]+cos AupSimptemp1[[1,2]],sin AupSimptemp2[[2,1]]+cos AupSimpt
Export["/home/nils/uni/projects/superrad/mathematica_scripts/Teukolsky_code/AupSimp.mx",Au
(*-----------------------------------------------------------------------------------
(*Analyitc form of the real part of the field A_\mu. This part takes about 50 min to
(*sin=Sin[m ϕ-ωr t];
cos=Cos[m ϕ-ωr t];
```

```
complextoreal={v→rv+I iv,ω→ωr+I ωi,R[r[]]:→ Rr[r[]]+I Ri[r[]],R'[r[]]:→ Rr'[r[]]+I Ri'[r[]],S[θ[]]→Sr[(

AdownSimptemp=(A[-b]//eval)//.complextoreal//.{t[]→t,r[]→r,θ[]→θ,ϕ[]→ϕ};
AdownSimptemp1=CoefficientList[AdownSimptemp[[1]]//Re//ComplexExpand,{sin,cos}]//Simplify; Pr
AdownSimptemp2=CoefficientList[AdownSimptemp[[2]]//Re//ComplexExpand,{sin,cos}]//Simplify; Pr
AdownSimptemp3=CoefficientList[AdownSimptemp[[3]]//Re//ComplexExpand,{sin,cos}]//Simplify; Pr
AdownSimptemp4=CoefficientList[AdownSimptemp[[4]]//Re//ComplexExpand,{sin,cos}]//Simplify; Pr

AdownSimp={sin AdownSimptemp1[[2,1]]+cos AdownSimptemp1[[1,2]],sin AdownSimptemp2[[2,1]]+cos
Export["/home/nils/uni/projects/superrad/mathematica_scripts/Teukolsky_code/AdownSimp.mx",

(*Importing the result of the commented code above*)
coordinates={t[]→t,r[]→r,θ[]→θ,ϕ[]→ϕ};
coordconv={Global`t→t,Global`r→r,Global`θ→θ,Global`ϕ→ϕ};
coordrev={t→Private`t,r→Private`r,θ→Private`θ,ϕ→Private`ϕ};
AupSimp=Import["./AupSimp.mx"]//.coordinates//.{Global`m→Private`m,Global`ωr→Private`ωr,
(*AdownSimp=(met[-b,-c]//eval).AupSimp//.coordinates//.coordconv/.{t→Private`t,r→Private`r,(
AdownSimp=Import["./AdownSimp.mx"]//.coordinates//.{Global`m→Private`m,Global`ωr→Private
(*AupSimp=(met[b,c]//eval).AdownSimp//.coordinates//.coordconv/.{t→Private`t,r→Private`r,θ→
Print["Imported analytic field modes."]
```

```
(*Generating the energy momentum tensor for this minimally coupled massive vector*)
DefTensor[Aplace[-b],M]
DArule=Table[PDch[{i,-ch}]@Aplace[{k,-ch}]→ToExpression["D"<>ToString[i]<>ToString["A"]<>ToS
AtoAi=Table[{Aplace[{i,-ch}]→ToExpression["A"<>ToString[i]]},{i,0,3}]//Flatten;

DefConstantSymbol[μ]
DefTensor[FS[-b,-c],M]
DefTensor[T[-b,-c],M]
FS/:FS[-b_,-c_]:=CD[-b]@Aplace[-c]-CD[-c]@Aplace[-b]
(*Tdd=(ChangeCovD[(μ^2Aplace[-h]Aplace[-c]+met[d,e]FS[-h,-d]FS[-c,-e]-1/4met[-h,-c](FS[-d,-e]FS[-f
Export["/home/nils/uni/projects/superrad/mathematica_scripts/Teukolsky_code/Tdd.mx",Tdd];*)

(*The energy density is the contraction of the timelike Killing field into the T_\mu\
Tdd=Import["./Tdd.mx"]//.coordinates//.{Global`m→Private`m,Global`ωr→Private`ωr,Global`ω
edenrule={Table[ToExpression["Global`D"<>ToString[i]<>ToString["A"]<>ToString[k]]→ToExpress
edenrule1={Table[ToExpression["Private`D"<>ToString[i]<>ToString["A"]<>ToString[k]]→ToExpre
Tud=(met[d,e]//eval).Tdd;
Export["./Tud.mx",Tud];
Tud=Import["./Tud.mx"]//.coordinates//.{Global`m→Private`m,Global`ωr→Private`ωr,Global`ω
Tudtemp=Tud[[1,1]]//.edenrule/.edenrule1/.{t→Private`t,r→Private`r,θ→Private`θ,ϕ→Private`
Print["Imported analytic Tud, Tdd and energy density."];
```

```
(*Interpolation code*)
```

```
VectorField[Rsol0_InterpolatingFunction,Anglsol0_,χ0_,m0_,nh0_,rv0_,iv0_,ωr0_,μ0_,MBH1_]
prec=SetPrecision[#,20]&;
parameters=prec@{χ→χ0,m→m0,rv→rv0,iv→iv0,ωi→0,ωr→ωr0,μ→μ0,nh→nh0};

(*The radial range depending on the input parameters of the system*)
Mtilde=1; (*1/(MBH1+1);*) (*we introduce Mtilde here ONLY to fix rstop to be same (or rou
rstop=(3/2)If[m0<3,prec@(4(10(m0+nh0)Mtilde)/(μ0^2)-2),prec@((400(m0+nh0)Mtilde)/(7μ0^2)-1)]; (*
rplus=1+Sqrt[1-χ0^2];
ϵr=10^-2;
rtst=prec@(ϵr+rplus);
rglue=If[nh0==0,0.12*rstop,0.2*rstop];
Print["rplus = "<>ToString[rplus]];
Print["rstart = "<>ToString[rtst]];
Print["epsilonr = "<>ToString[(rtst-rplus)/rplus]];
Print["rstop = "<>ToString[rstop]];
Print["r glue = "<>ToString[rglue]];

(*Generating the interpolations for a grid of 40x40 points over (r,θ) for each of the
Print["Interpolation: Start."];
θmesh=If[m0==1,49,If[m0==2,59,59]];
rmesh=If[m0==1,199,If[m0==2,699,699]];
```

```
p=If[rstop<500,4,5];
Print["rmesh = "<>ToString[rmesh]];
Print["Thetamesh = "<>ToString[θmesh]];
rdatapoint[x_]:=(rstop-rtst)/((rmesh+1)^p-1) x^p+rtst-(rstop-rtst)/((rmesh+1)^p-1);
(*rrange=Range[rtst,rstop,(rstop-rtst)/rmesh];*)
rrange=rdatapoint[Range[2,rmesh+2]];

slp=0.25; (*slope of the step function*)
step[x_]:=π/(θmesh+1) ((θmesh+1)^2/2 Exp[slp(x-(θmesh+1)/2)])/((θmesh+1)+(θmesh+1)/2(Exp[slp(x-(θme
θrange=prec@step[Range[1,θmesh+1]];
θstart=prec@(θrange//First);
θstop=prec@(θrange//Last);
Print["ThetaStart = "<>ToString[θstart]];
Print["ThetaStop = "<>ToString[θstop]];

Rθranges=prec@{{rtst,rstop},{θstart,θstop}};

Print["Exponential extension: Start."];
(*Exponential extension of the Radial function*)
Rslope[f1_,x1_,f2_,x2_]:=(f1-f2)/(x1-x2);
(*First matching point*)
```

```
r1=2rglue/3;
R1re=Rsol[r1]//Re//Log;
R1im=Rsol[r1]//Im//Log;
dR1re=D[Rsol[r],r]/.{r→r1}//Re//Log;
dR1im=D[Rsol[r],r]/.{r→r1}//Im//Log;
ddR1re=D[Rsol[r],{r,2}]/.{r→r1}//Re//Log;
ddR1im=D[Rsol[r],{r,2}]/.{r→r1}//Im//Log;
(*Second matching point*)
r2=rglue;
R2re=Rsol[r2]//Re//Log;
R2im=Rsol[r2]//Im//Log;
dR2re=D[Rsol[r],r]/.{r→r2}//Re//Log;
dR2im=D[Rsol[r],r]/.{r→r2}//Im//Log;
ddR2re=D[Rsol[r],{r,2}]/.{r→r2}//Re//Log;
ddR2im=D[Rsol[r],{r,2}]/.{r→r2}//Im//Log;
(*Slopes between matching points*)
RslopeRe=Rslope[R1re,r1,R2re,r2];
RslopeIm=Rslope[R1im,r1,R2im,r2];
dRslopeRe=Rslope[dR1re,r1,dR2re,r2];
dRslopeIm=Rslope[dR1im,r1,dR2im,r2];
ddRslopeRe=Rslope[ddR1re,r1,ddR2re,r2];
```

```
ddRslopeIm=Rslope[ddR1im,r1,ddR2im,r2];
(*Final solution*)
Rad[rr_]:=Piecewise[{{Rsol[y]//.{y→rr},rr<rglue},{Exp[RslopeRe rr]/Exp[RslopeRe rglue]Re[Rsol[r
dRad[rr_]:=Piecewise[{{D[Rsol[y],y]//.{y→rr},rr<rglue},{Exp[dRslopeRe rr]/Exp[dRslopeRe rglue]R
ddRad[rr_]:=Piecewise[{{D[Rsol[y],{y,2}]//.{y→rr},rr<rglue},{Exp[ddRslopeRe rr]/Exp[ddRslopeRe r
(*Angular solution*)
Sangl[θθ_]:=Anglsol/.{Global`θ→Private`θ,TeuInterEnv`θ→Private`θ}//.coordinates//.{θ→θθ};
dSangltemp=D[Sangl[θ],θ];
dSangl[θθ1_]:=dSangltemp//.{θ→θθ1};
ddSangltemp=D[dSangl[θ],θ];
ddSangl[θθ2_]:=ddSangltemp//.{θ→θθ2};
solidentify={Ri[r_]:→Im[Rad[r]],Rr[r_]:→Re[Rad[r]],Ri'[r_]:→Im[dRad[r]],Rr'[r_]:→Re[dRad[r]],Rr''[r_]:→Re
Print["Solution at horizon: Test:"];
Print["Rad[rtst] = "<>ToString[Rad[rtst]]];
Print["Exponential extension: Done!"];


AdownSimpExpl={0,0,0,0};
AupSimpExpl={0,0,0,0};
dataout={0,0,0,0};
```

```
sin=Sin[m ϕ-t ωr];
cos=Cos[m ϕ-t ωr];

Do[
Clear[coefflistdown,coefflistup,coefflistInterdown,coefflistInterup];
coefflistdownExpl=ConstantArray[0,{rmesh+1,θmesh+1,2}];
coefflistupExpl=ConstantArray[0,{rmesh+1,θmesh+1,2}];
coefflistdown=DeleteCases[CoefficientList[AdownSimp⟦i⟧/.{ωi→0},{sin,cos}]/.{vi→iv,vr→rv}/.{
coefflistup=DeleteCases[CoefficientList[AupSimp⟦i⟧/.{ωi→0},{sin,cos}]/.{vi→iv,vr→rv}/.{χ→χ
    Do[
    Clear[rinput,θinput];
    rinput=prec@rrange⟦ir⟧;
    θinput=prec@θrange⟦iθ⟧;
    coefflistdownExpl⟦ir,iθ,1⟧=prec@(coefflistdown⟦1⟧//.solidentify//.{r→rinput}//.{θ→θinp
    coefflistupExpl⟦ir,iθ,1⟧=prec@(coefflistup⟦1⟧//.solidentify//.{r→rinput}//.{θ→θinput});
    coefflistdownExpl⟦ir,iθ,2⟧=prec@(coefflistdown⟦2⟧//.solidentify//.{r→rinput}//.{θ→θinp
    coefflistupExpl⟦ir,iθ,2⟧=prec@(coefflistup⟦2⟧//.solidentify//.{r→rinput}//.{θ→θinput});
    ,{ir,1,rmesh+1},{iθ,1,θmesh+1}];
Print["Data produced for: "<>ToString[i]];
dataout⟦i⟧=Table[{{rrange⟦k⟧,θrange⟦j⟧},coefflistdownExpl⟦k,j,1⟧},{k,1,rmesh+1},{j,1,θmesh+
coefflistInterdown=Table[Interpolation[Flatten[Table[{{rrange⟦k⟧,θrange⟦j⟧},coefflistdownE
```

```
coefflistInterup=Table[Interpolation[Flatten[Table[{{rrange[[k]],θrange[[j]]},coefflistupExpl[
AdownSimpExpl[[i]]=(cos  coefflistInterdown[[1]][r,θ]+sin coefflistInterdown[[2]][r,θ]);
AupSimpExpl[[i]]=(cos  coefflistInterup[[1]][r,θ]+sin coefflistInterup[[2]][r,θ]);
,{i,1,4}];
Print["Interpolation: Done."];

(*Final output of the the interpolating of the fields*)
Adownvector[tt1_,rr1_,θθ1_,ϕϕ1_]:=AdownSimpExpl//.parameters//.{t→tt1,r→rr1,θ→θθ1,ϕ→ϕϕ1};
Aupvector[tt2_,rr2_,θθ2_,ϕϕ2_]:=AupSimpExpl//.parameters//.{t→tt2,r→rr2,θ→θθ2,ϕ→ϕϕ2};

(*Testing the vector result*)
atest=Adownvector[0,2,2,2];
Print["Adownvector[0,2,2,2] = "<>ToString[atest]];

(*Integrating over the spatial 3-slice outside of the horizon for the complete mass*)
Aexpl=Table[{ToExpression["Private`A"<>ToString[i]]→Adownvector[t,r,θ,ϕ][[i+1]]},{i,0,3}]//Flat
dAexpl=Table[{ToExpression["Private`D"<>ToString[i]<>ToString["A"]<>ToString[k]]→D[Adownvec

Print["Massnormalization: Start."];
(*Mass integral*)
prec=SetPrecision[#,30]&;
```

```
TuptdowntTEMP=prec@(Tudtemp/.{ωi→0}//.parameters)//.Aexpl//.dAexpl//.coordinates;
Tuptdownt[tt_,rr_,θθ_,ϕϕ_]:=TuptdowntTEMP//.{t→tt,r→rr,θ→θθ,ϕ→ϕϕ};
Print["Tuptdownt test = "<>ToString[prec@Tuptdownt[0,2,2,2]]];
sqrtmg[r_,θ_]:=prec@(Sin[θ](r^2+χ^2 Cos[θ]^2)//.parameters);
Print["sqrtmg test = "<>ToString[sqrtmg[2,2]]];
Print["rplus = "<>ToString[rplus]];
rintstart=prec@(rplus+10^-1.9);
rintstop=prec@Last[rrange];
Print["rintstart = "<>ToString[rintstart]];
Print["epsilonrint = "<>ToString[(rintstart-rplus)/rplus]];
Print["rintstop = "<>ToString[rintstop]];
Print["thetastart = "<>ToString[θstart]];
Print["thetastop = "<>ToString[θstop]];
massNorm=NIntegrate[-Tuptdownt[0,r,θ,ϕ]*sqrtmg[r,θ],{r,rintstart,rintstop},{θ,θstart,θsto
massNormint[rr_,θθ_,ϕϕ_]:=-Tuptdownt[0,r,θ,ϕ]*sqrtmg[r,θ]/.{r→rr,θ→θθ,ϕ→ϕϕ};
Print["massNorm obtained: "<>ToString[massNorm]];

(*Normalization to the mass in terms of the inital BH mass*)
Anorm=SetPrecision[Sqrt[MBH1/massNorm],20];
Print["Anorm obtained: "<>ToString[Anorm]];
Print["Massnormalization: Done."];
```

```
(*Final output of the the interpolating of the fields and the final output of the pac
edensityNorm[tt_,rr_,θθ_,ϕϕ_]:=-Anorm^2 Tuptdownt[t,r,θ,ϕ] Sqrt[(-met[{0,-ch},{0,-ch}]//eval)/
(*Note the sign and the additional lapse factor in front of the meassure of the inter
AdownNorm[tt_,rr_,θθ_,ϕϕ_]:=Anorm AdownSimpExpl//.coordinates/.parameters//.{t→tt,r→rr,θ→
AupNorm[tt_,rr_,θθ_,ϕϕ_]:=Anorm AupSimpExpl//.coordinates/.parameters//.{t→tt,r→rr,θ→θθ,ϕ
]

Print["VectorFieldNorm Functions:"];
Print["VectorField[Rsol,Angsol,spin,Field-m-mode,vr,vi,Re(ω),Procamass,overtone,Normali
Print["FieldEnergyMomentum[AdownNorm[t,r,θ,ϕ],Procamass,spin]"];
Print["------------------------------------------------------------"];
End[]

EndPackage[]
```