```
QNMnonRel[min_, nin_, spin_, μsta_, μsto_, branch1_, ωImIn_, Sin_, direc_] :=
  Module[{min1 = min, nin1 = nin, spin1 = spin, μsta1 = μsta,
    μsto1 = μsto, branch2 = branch1, wijk = ωImIn, Sin1 = Sin},

destinationPATH = ToString[
     "/home/shaunf/Documents/Computer/Code/projects/Massive_Vector_Field_Dynamical
       _Friction/ProcaAroundKerr/NilsSiemonsenCode/surrogate"];
$Assumptions = M > 0 && x != 0;
kmax = 5;
xmax = 2;
ϵ = 10^-6;
xstart = (ϵ M) / (rp[a] - rm[a]) // Simplify // Rationalize[#, 0] &;
ωnmax = (m χ) / (2 (1 + Sqrt[1 - χ^2])) // Simplify;
setprec = 24;
maxstep = 10^8;
modeprec = 20;

m = min;
η = If[Sin == 0, 1, 0];
nh = nin;
S = Sin;
n = Abs[m] + nh + S + 1;
χ = spin // Rationalize[#, 0] &;

Print["We consider the following parameters:"];
Print["m = " <> ToString[m]];
Print["n = " <> ToString[nh]];
Print["χ = " <> ToString[χ // N]];
Print["S = " <> ToString[Sin // N]];
Print["branch = " <> ToString[branch1]];
Print["Log of precision cutoff = 10^-" <> ToString[modeprec]];
Print["-------------------------------------"];

a = χ M; μ = μn / M; ω = (ωnr + I ωni) / M; ν = νn / M;
rp[a_] := M + Sqrt[M^2 - a^2];
rm[a_] := M - Sqrt[M^2 - a^2];
rstop[μ_] := 2 (10 (m + nh)) / μ^2 // Rationalize[#, 0] &;
rtstart[a_] := (xstart (rp[a] - rm[a]) + rp[a]) / M // Simplify // Rationalize[#, 0] &;
ΩH[a_] := a / (2 M rp[a]);
Δ[r_, a_] := r^2 - 2 M r + a^2;
Kr[r_, a_, ω_, m_] := (a^2 + r^2) ω - a m;
qr[r_, ν_] := 1 + ν^2 r^2;
```

```
Λ[ν_, μ_, a_, m_, ω_] := μ^2/ν^2 - (ω + a ν^2 (m - a ω))/ν + 2 a ω m - a^2 ω^2;
γ[ω_, μ_] := Sqrt[ω^2 - μ^2];
σ[ω_, a_, ν_, m_] := ω + a ν^2 (m - a ω);
Three[l1_, l2_, l3_, m1_] := If[l3 > l1 + l2 || Abs[l1 - l2] > l3 || Abs[m1] > l1 || Abs[m1] > l3,
    0, ThreeJSymbol[{l1, -m1}, {l2, 0}, {l3, m1}]];
Bracket[l1_, l2_, l3_, m_] := (-1)^m Sqrt[((2 l1 + 1) (2 l2 + 1) (2 l3 + 1))/(4 π)]
    Three[l1, l2, l3, 0] × Three[l1, l2, l3, m];
c2[l_, m_, lp_] := (2 Sqrt[π])/3 Bracket[l, 0, lp, m] + 4/3 Sqrt[π/5] Bracket[l, 2, lp, m];
c4[l_, m_, lp_] := (2 Sqrt[π])/5 Bracket[l, 0, lp, m] +
    8/7 Sqrt[π/5] Bracket[l, 2, lp, m] + (16 Sqrt[π])/105 Bracket[l, 4, lp, m];
d2[l_, m_, lp_] := Sqrt[(4 π)/3] (lp Sqrt[((lp + 1)^2 - m^2)/((2 lp + 1) (2 lp + 3))] Bracket[l, 1,
        lp + 1, m] - (lp + 1) Sqrt[(lp^2 - m^2)/((2 lp + 1) (2 lp - 1))] Bracket[l, 1, lp - 1, m]);
Mtemp[l_, m_, lp_, ν_, μ_, a_, ω_] := (Λ[ν, μ, a, m, ω] - lp (lp + 1)) KroneckerDelta[l, lp] +
     (-ν^2 Λ[ν, μ, a, m, ω] + ν^2 lp (lp + 1) - 2 σ[ω, a, ν, m] ν + γ[ω, μ]^2) a^2 c2[l, m, lp] -
     2 a^2 ν^2 d2[l, m, lp] - γ[ω, μ]^2 ν^2 a^4 c4[l, m, lp] // Refine;
Mat[m_, η_, ν_, μ_, a_, ω_] := Table[Mtemp[Abs[m] + 2 k + η, m, Abs[m] + 2 kp + η, ν, μ, a, ω],
     {k, 0, kmax}, {kp, 0, kmax}] // Simplify;
κ[a_, ω_, m_] := (2 M rp[a] (ω - m ΩH[a]))/(rp[a] - rm[a]) // Simplify;
(*Clear[χ,ωni,ωnr,νn,μn,m,η,a,ω,μ,ν]
DiffRadr :=
   D[Δ[r,a]D[#,r],r]+(Kr[r,a,ω,m]^2/Δ[r,a]-Λ[ν,μ,a,m,ω]+2 a ω m-a^2ω^2-μ^2r^2)#-
     (2r ν^2)/qr[r,ν](Δ[r,a]D[#,r]+r σ[ω,a,ν,m]/ν#)&;
Collect[DiffRadr@Y[r]/.{Y→(Y[#/M]&)}/.{r→rt M}//Simplify,{Y[rt],Y'[rt],Y''[rt]}]
Collect[(DiffRadr@f[r]/.{f→(f[(#-rp[a])/(rp[a]-rm[a])]&)}/.{r→x(rp[a]-rm[a])+rp[a]})/(x(1+x))//
     Simplify,{f[x],f'[x],f''[x]}]*)
DiffRadrt = (-((2 rt^2 νn (m νn^2 χ - I (-1 + νn^2 χ^2) (ωni - I ωnr)))/
           ((1 + rt^2 νn^2) (-2 rt + rt^2 + χ^2))) + (-rt^2 μn^2 - μn^2/νn^2 +
           (m χ - I (rt^2 + χ^2) (ωni - I ωnr))^2/(-2 rt + rt^2 + χ^2) +
           (I ωni + ωnr)/νn + νn χ (m - χ (I ωni + ωnr)))/(-2 rt + rt^2 + χ^2)) # +
     (-((2 rt νn^2)/(1 + rt^2 νn^2)) + (2 (-1 + rt))/(-2 rt + rt^2 + χ^2)) D[#, rt] +
     D[#, {rt, 2}] &;
DiffRadx = 1/(x (1 + x)) (-(μn^2/νn^2) - μn^2 (1 + Sqrt[1 - χ^2] + 2 x Sqrt[1 - χ^2])^2 -
        (2 M νn (1 + Sqrt[1 - χ^2] + 2 x Sqrt[1 - χ^2]) (1 - χ^2 + Sqrt[1 - χ^2] - 2 x (-1 + χ^2))
           (m νn^2 χ - I (-1 + νn^2 χ^2) (ωni - I ωnr)))/(Sqrt[1 - χ^2] (M + M νn^2
              (2 - χ^2 + 2 Sqrt[1 - χ^2] - 4 x^2 (-1 + χ^2) + 4 x (1 - χ^2 + Sqrt[1 - χ^2])))) +
        (I m χ + 2 (1 + Sqrt[1 - χ^2] - 2 x^2 (-1 + χ^2) + 2 x (1 - χ^2 + Sqrt[1 - χ^2])) (ωni -
               I ωnr))^2/(4 x (1 + x) (-1 + χ^2)) + (I ωni + ωnr)/νn + νn χ (m - χ (I ωni + ωnr)))
      # + (1 + 2 x + (4 M x (1 + x) νn^2 (-1 + χ^2) (1 + Sqrt[1 - χ^2] + 2 x Sqrt[1 - χ^2]))/
          (Sqrt[1 - χ^2] (M + M νn^2 (2 - χ^2 + 2 Sqrt[1 - χ^2] - 4 x^2 (-1 + χ^2) +
                4 x (1 - χ^2 + Sqrt[1 - χ^2]))))) /(x (1 + x)) D[#, x] + D[#, {x, 2}] &;

prec = SetPrecision[#, setprec] &;
(*The angular equation in matrix form*)
```

```
mat = prec@(Mat[m, η, v, μ, a, ω] vn^2);
matdettemp = prec@Det[mat]; (*With and without the vn^12 in front,
   which is analytically equivalent to not having it there,
   changes the resulting root starting from the sixth decimal*)
matdet[rv_, iv_] := prec@(matdettemp //. {vn -> rv + I iv});
absdetmat[rv_, iv_] := prec@Log[Abs[matdet[rv, iv]]];

Rmaxfunc[wIn_, vIn_, μIn_, branch3_] := Module[{wIn1 = wIn, vIn1 = vIn, μIn1 = μIn},
    prec = SetPrecision[#, setprec] &;
    μi = prec@μIn;

    wrpoint = Re[wIn] // Rationalize[#, 0] &;
    wipoint = Im[wIn] // Rationalize[#, 0] &;

    vrpoint = Re[vIn] // Rationalize[#, 0] &;
    vipoint = Im[vIn] // Rationalize[#, 0] &;

    vrootSollist = prec@
       NSolve[SetPrecision[(matdettemp /. {μn -> μi, ωnr -> wrpoint, ωni -> wipoint}),
          setprec] == 0, vn, WorkingPrecision -> setprec];
    vrootVallist = prec@Table[{vn /. vrootSollist[[l]] // Re, vn /. vrootSollist[[l]] // Im},
       {l, 1, Length[vrootSollist]}];
    vroottemp = prec@Nearest[vrootVallist, {vrpoint, vipoint},
        WorkingPrecision -> setprec] // First;
    vroot = prec@{rv -> vroottemp[[1]], iv -> vroottemp[[2]]};

    Frob[x_] := x^(-I λ) (1 + Sum[Symbol["QNMcode`c" <> ToString[n]] x^n, {n, 1, 2}]);
    exptemp = Series[
       SetPrecision[x^(I λ + 2) (DiffRadx@Frob[x]) // Simplify, setprec], {x, 0, 2}] == 0;
    K = (-m χ + 2 (1 + Sqrt[1 - χ^2]) (I ωni + ωnr)) / (2 Sqrt[1 - χ^2]) /.
       {ωnr -> wrpoint, ωni -> wipoint};
    exptemp1 = SetPrecision[exptemp /. {μn -> μi} /. {ωnr -> wrpoint, ωni -> wipoint} /.
         {vn -> rv + I iv} /. vroot // LogicalExpand, setprec];
    (*setprecNSolve[ju_]:=Piecewise[{{setprec-1,ju≤ 22},{setprec-2,ju>22}}];*)
    Coeff = NSolve[exptemp1, {λ, c1, c2},
       WorkingPrecision -> setprec - 2(*setprecNSolve[i]*)][[branch3]];
    λtest = Abs[(λ /. Coeff) + K];
    (*If[Log[λtest]<-15,{},Print[λtest]Print[
        "Error: Possible problem with value for λ or κ!"]Print[Precision[K]]];*)
    FrobOfr[rt_] := SetPrecision[Frob[x] //. Coeff //. {x -> (rt M - rp[a]) / (rp[a] - rm[a])}
        (*/.{λ→K}*) /. {μn -> μi, ωnr -> wrpoint, ωni -> wipoint} // Simplify, setprec];
    R0 = SetPrecision[Limit[FrobOfr[rt], rt -> rtstart[a]], setprec];
    dR0 = SetPrecision[Limit[D[FrobOfr[rt], rt], rt -> rtstart[a]], setprec];
```

```
    solR = NDSolve[{(DiffRadrt @R[rt] /. {νn -> rν + I iν} /. νroot /. {μn -> μi, ωnr -> wrpoint ,
            ωni -> wipoint}) == 0, R[rtstart[a]] == R0, R '[rtstart[a]] == dR0},
        R, {rt, rtstart[a], rstop[μi]}, Method -> {"StiffnessSwitching "},
        WorkingPrecision -> setprec - 2, MaxSteps -> maxstep] // First;

    Rmax = SetPrecision [Log[Abs[R[rt] /. solR /. {rt -> rstop[μi]}]], setprec];
    νrootout = prec @(νroottemp[[1]] + I νroottemp[[2]]);
    ωout = prec @wIn;
];

(*Regime of consideration *)
μstart = μsta;
μstop = μsto;
ωmesh = 20;
ωprecstep = 2;
ωrstepsize = 400;
μmesh = 1;
branch = branch1;

(*Non-relativistic limit and its values as first guesses*)
ωnonRel[μn_, na_] := (μn (1 - μn^2 / (2 na^2))) // Rationalize [#, 0] &;
  (*Provides only a guess for the real part.*)
If[Sin == -1,
(*This is for S=-1*)
Print["Picked S=-1 νnonrel"];
νnonRel[ωa_] := - ωa / (1 - χ ωa) // Rationalize [#, 0] &;
,
If[Sin == 0,
(*This is for S=0*)
Print["Picked S=0 νnonrel"];
νnonRel[ωa_] := 1 / (2 χ) (2 - χ ωa + Sqrt[(- 2 + χ ωa) ^ 2 + 4 χ ωa]);
,
If[Sin == 1, (*This is for S=+1*)
Print["Picked S=+1 νnonrel"];
νnonRel[ωb_] :=
       v //. (NSolve[a v ^ 3 (1 - a ωa) - (6 - a ωa (2 - a ωa)) v ^ 2 + ωa v + ωa ^ 2, v][[2]] // Simplify) /.
         {M -> 1} //. {ωa -> ωb};
,
Print["Error: S not in (-1,0,+1)!"];
Abort[];
]]];
```

```
(*For a given complex (!) ω,
   this provides an initial guess for real and imaginary parts*)
μrange = {0, Range[μstart, μstop, (μstop - μstart) / μmesh]} // Flatten // Rationalize[#, 0] &;
μall = Take[μrange, -Length[μrange] + 1];
μrangenumber = Length[μrange] - 1;

(*Initial ω guesses*)
ωintR = ωnonRel[μstart, n];
ωintI = ωImIn;
νint = SetPrecision[
     {rν -> Re[νnonRel[ωintR + I ωintI]], iν -> Im[νnonRel[ωintR + I ωintI]]}, setprec];

(*The function to be minimized*)
funcmin[win_, νin_, μin_, branch_] := Module[{win1 = win, νin1 = νin, μin1 = μin},
Rmaxfunc[win, νin, μin, branch];
Rmax
];

(*Isomorphism from R^2 to C (and inverse)*)
R2toC[vec_] := vec[[1]] + I vec[[2]];
CtoR2[num_] := {Re[num], Im[num]};

νrootminall = ConstantArray[0, Length[μrange]];
solRminall = ConstantArray[0, Length[μrange]];
ωminall = ConstantArray[0, Length[μrange]];
μminall = ConstantArray[0, Length[μrange]];
plotlist = ConstantArray[0, {Length[μrange], ωprecstep}];

νrootminout = ConstantArray[0, Length[μrange]];
solRminout = ConstantArray[0, Length[μrange]];
ωminout = ConstantArray[0, Length[μrange]];
μminout = ConstantArray[0, Length[μrange]];
lminout = ConstantArray[0, Length[μrange]];

(*The only reliable value for ω and ν is starting at [[2]],
   since in the first step,
   we use the non-rel. result without (!) optimization and minimization to our
     equations!!! Always remove the first value from the list*)
νrootminall[[1]] = νint;
ωminall[[1]] = SetPrecision[ωintR + I ωintI, setprec];

(*The minimum search mesh boundaries for the real part of ω*)
rrinitial = 1 / 2 (ωnonRel[μstart, n + 1] - ωnonRel[μstart, n]);
```

```
rrdelta[u_] :=
    If[u == 1, rrinitial , 1 / 2 (ωnonRel [μrange[[u + 1]], n + 1] - ωnonRel [μrange[[u + 1]], n])];

(*The minimum search mesh boundaries for the imaginary part of ω*)
iiThreshhold = 10 ^ -1;
iiinitial = 1;
iidelta[u_] := SetPrecision [If[u < 4, iiinitial ,
       If[Abs[Log10[Im[ωminall [[u]]]] - Log10[Im[ωminall [[u - 1]]]]] > iiThreshhold ,
        Abs[Log10[Im[ωminall [[u]]]] - Log10[Im[ωminall [[u - 1]]]]], 0.5]], setprec];

(*The code:*)
offset = 0;
Clear[y];
For[i = 1, i <= μrangenumber , i++,
branchtemp = branch;
If[i <= 2, {},
If[Abs[Log10[Im[ωminall [[i]]]] - Log10[Im[ωminall [[i - 1]]]]] > 2,
branch = Select[{1, 2}, # != branchtemp &] // First;
ωminall [[i]] = ωminalltemp ;
i = i - 1;
Print["flip!"];
,
branch = branchtemp ]];

Print[ToString [i] <> ToString ["/"] <> ToString [μmesh]];
μi = μrange[[i + 1]];
Print[μi // N];

(*We always take the previous ω as the new guess,
    since these are much closer to the actual value than the non-
     rel limit. In the very first step, we use the non-rel limit as guess*)
wintr = SetPrecision [ωnonRel [μi, n], setprec];
winti = SetPrecision [Im[ωminall [[i]]], setprec];
νroot = νrootminall [[i]];

(*The initial iteration boaundries *)
rrboundary = rrdelta[i];
iiboundary = iidelta[i];

(*For the very first iteration,
    we choose this asymmetric search interval, since the real part of the
     frequency is monotonically increasing with μ (if not rescaled by μ^-1)*)
(*ωrrange =SetPrecision [If[wintr+rrboundary <ωnmax ,Range[wintr-rrboundary ,
```

```
        wintr+rrboundary ,(2rrboundary )/ωmesh],Range[wintr-rrboundary ,
        ωnmax- 10^-6,Abs[wintr-rrboundary -(ωnmax- 10^-6)]/ωmesh]],setprec];*)
ωirange = SetPrecision [10 ^ Range[Log10[winti] - iiboundary ,
       Log10[winti] + iiboundary , (2 iiboundary ) / ωmesh], setprec];
(*ωrrangeplot [i]=Show[ListPlot [Table[{i ,ωrrange [[k]]},{k ,1,Length[ωrrange ]}]],
      ListPlot [{{i ,ωminall [[i]]//Re}},PlotStyle →Red]];*)
ωirangeplot [i] = Show[ListLogPlot [Table[{i , ωirange [[k]]}, {k, 1, Length[ωirange ]}]],
     ListLogPlot [{{i, ωminall [[i]] // Im}}, PlotStyle -> Red]];

   δωr = 0;
   aa = 1;
   aatest = -1;
   wrpoint = wintr ;
      While[aatest < 0,
         AAtest = maxR[2] - maxR[1];
         wrpoint = SetPrecision [If[aa <= 2, wintr - aa δωr + offset ,
      If[AAtest < 0, wintr - aa δωr + offset , wintr + aa δωr + offset]], setprec];
         wipoint = winti ;

         (*To pick the correct ν solution form the set of roots of the EVP,
    we compare it to either the non-rel. limit using the current ω,
    or the previous (previous mass) minimal result for ν*)
         νrpoint = prec @(rν /. νroot);
         νipoint = prec @(iν /. νroot);

         νrootSollist =
     NSolve[SetPrecision [(matdettemp /. {μn -> μi, ωnr -> wrpoint , ωni -> wipoint}),
        setprec] == 0, νn, WorkingPrecision -> setprec];
         νrootVallist = SetPrecision [Table[{νn /. νrootSollist [[l]] // Re,
        νn /. νrootSollist [[l]] // Im}, {l, 1, Length[νrootSollist ]}], setprec];
         νroottemp = Nearest[νrootVallist , {νrpoint, νipoint},
      WorkingPrecision -> setprec] // First;
         νroot = SetPrecision [{rν -> νroottemp[[1]], iν -> νroottemp[[2]]}, setprec];

         Frob[x_] := x ^ (- I λ) (1 + Sum[Symbol["QNMcode`c " <> ToString [n]] x ^ n, {n, 1, 2}]);
         exptemp = Series[
      SetPrecision [x ^ (I λ + 2) (DiffRadx @ Frob[x]) // Simplify , setprec], {x, 0, 2}] == 0;
         K = (- m χ + 2 (1 + Sqrt[1 - χ ^ 2]) (I ωni + ωnr)) / (2 Sqrt[1 - χ ^ 2]) /.
     {ωnr -> wrpoint , ωni -> wipoint};
         exptemp1 = SetPrecision [exptemp /. {μn -> μi} /. {ωnr -> wrpoint ,
          ωni -> wipoint} /. {νn -> rν + I iν} /. νroot // LogicalExpand , setprec];
         (*setprecNSolve [ju_]:=Piecewise [{{setprec -1,ju≤ 22},
      {setprec -2,ju>22}}];*)
```

```
      Coeff = NSolve[exptemp1 , {λ, c1, c2}, WorkingPrecision -> setprec - 2
    (*setprecNSolve [i]*)][[branch]];
      λtest = Abs[(λ /. Coeff) + K];
      (*If[Log[λtest]<-15,{},Print[λtest]Print[
     "Error: Possible problem with value for λ or κ!"]Print[Precision[K]]];*)
      FrobOfr[rt_] := SetPrecision [Frob[x] //. Coeff //.
     {x -> (rt M - rp[a]) / (rp[a] - rm[a])}(*/.{λ→K}*) /.
     {μn -> μi, ωnr -> wrpoint , ωni -> wipoint} // Simplify , setprec];
      R0 = SetPrecision [Limit[FrobOfr[rt], rt -> rtstart[a]], setprec];
      dR0 = SetPrecision [Limit[D[FrobOfr[rt], rt], rt -> rtstart[a]], setprec];

      solR =
  NDSolve[{(DiffRadrt @R[rt] /. {νn -> rν + I iν} /. νroot /. {μn -> μi, ωnr -> wrpoint ,
        ωni -> wipoint}) == 0, R[rtstart[a]] == R0, R'[rtstart[a]] == dR0},
    R, {rt, rtstart[a], rstop[μi]}, Method -> {"StiffnessSwitching "},
    WorkingPrecision -> setprec - 2, MaxSteps -> maxstep] // First;

     solRIterationR [aa] = solR;
      νrootIterationR [aa] = νroot;
      ωIterationR [aa] = SetPrecision [wrpoint , 20];

      maxR[aa] =
  SetPrecision [Log[Abs[R[rt] /. solRIterationR [aa] /. {rt -> rstop[μi]}]], setprec];
      aatest = If[aa <= 2, -1, maxR[aa] - maxR[aa - 1]];
      δωr = SetPrecision [rrdelta[i] / ωrstepsize , 20];
      If[aa > 500, Print["aa>500: Terminated !"]; Break[], {}];
    ; aa++];
    If[AAtest < 0, Print["-1"], Print["+1"]];
    Print[aa - 1];
    ωminR = ωIterationR [aa - 1];
    If[aa > 50 && AAtest < 0, offset = 1 (ωminR - ωnonRel [μi, n]), {}];
    If[aa > 100 && AAtest < 0, offset = 2 (ωminR - ωnonRel [μi, n]), {}];
    νtransfer = νroottemp[[1]] + I νroottemp[[2]];
    ωtransfer = ωminR + I winti ;
(*-------------------------------------------------------------------------------
                                           -------------------------------------------
                                           ----------------------------------*)
(*Initial simplex dimensions *)
prec = SetPrecision [#, 20] &;

Print["Simplex code..."];
rrinitial2 = δωr / 10;
iiinitial2 = 10 ^(Log10[Im[ωminall [[i]]]] - 1);
```

```
νint = prec @ νtransfer ;
wint = prec @ ωtransfer ;
wvec0 = prec @{Re[wint], Im[wint]};
wvec1 = prec @{Re[wint] - rrinitial2 , Im[wint]};
wvec2 = prec @{Re[wint], Im[wint] - iiinitial2 };
newsimplex = prec @{wvec0 , wvec1 , wvec2};

Print[SetPrecision [wint , 20]];
testnorm = 1;
count = 1;
temprint = "Count: " <> ToString [count];
While[
(*termination test:*)
testnorm > 10 ^(Log10[Im[wint]] - modeprec ),


(*Simplex *)
funcofsimplex = Table[funcmin[R2toC[newsimplex [[j]]], νint, μi, branch], {j, 1, 3}];
fh = prec @ Max[funcofsimplex ];
posfh = Position[funcofsimplex , fh][[1]];
xh = prec @ newsimplex [[posfh]] // First ;
fl = Min[funcofsimplex ];
posfl = Position[funcofsimplex , fl][[1]];
xl = prec @ newsimplex [[posfl]] // First ;
fs = prec @ Delete[funcofsimplex , {posfh , posfl}] // First ;
posfs = Position[funcofsimplex , fs][[1]];
xs = prec @ newsimplex [[posfs]] // First ;
centroid = prec @ 1 / 2 (xl + xs);

(*Transformation of simplex *)
αSimplex = 1;
βSimplex = 1 / 2;
γSimplex = 2;
δSimplex = 1 / 2;
(*Reflection point*)
xr = prec @(centroid + αSimplex (centroid - xh));
fr = prec @ funcmin[R2toC[xr], νint, μi, branch];

(*Body of the method *)
If[fl <= fr && fr < fs,
(*Print["Reflected 1"];*)
xnew = xr;
x1 = xs;
```

```
x2 = xl;
,
If[fr < fl,
(*expansion*)
xe = prec@(centroid + γSimplex (xr - centroid));
fe = prec@funcmin[R2toC[xe], νint, μi, branch];
If[fe < fr,
(*Print["Expanded"];*)
xnew = xe;
x1 = xs;
x2 = xl;
,
(*Print["Reflected 2"];*)
xnew = xr;
x1 = xs;
x2 = xl;
];
,
(*contraction*)
If[fr >= fs,
(*outside*)
If[fr < fh,
xc = prec@(centroid + βSimplex (xr - centroid));
fc = prec@funcmin[R2toC[xc], νint, μi, branch];
If[fc <= fr,
(*Print["Contracted 1"];*)
xnew = xc;
x1 = xs;
x2 = xl;
,
(*Print["Shrunk 1"];*)
xnew = prec@(xl + δSimplex (xs - xl));
x1 = prec@(xl + δSimplex (xh - xl));
x2 = xl;
];
,
(*inside*)
xc = prec@(centroid + βSimplex (xh - centroid));
fc = prec@funcmin[R2toC[xc], νint, μi, branch];
If[fc < fh,
(*Print["Contracted 2"]*)
xnew = xc;
x1 = xs;
```

```
x2 = xl;
,
(*Print["Shrunk 2"];*)
xnew = prec@(xl + δSimplex (xs - xl));
x1 = prec@(xl + δSimplex (xh - xl));
x2 = xl;
];
, Print["Error1!"]];

, Print["Error2!"]];
];
];

(*new simplex*)
newsimplex = prec@({xnew, x1, x2} // Abs);
xnewprint = R2toC[xnew];
simplexset[i, count] = newsimplex;
vint = prec@vrootout;
testnorm = prec@Max[{Norm[xnew - x1], Norm[xnew - x2], Norm[x1 - x2]}];
testnormset[i, count] = testnorm;
count = count + 1;
temprint = "Count: " <> ToString[count];
Print[temprint];
If[count > 5000, Print["Terminated loop 3"]; Break[], {}];
];
    ωminalltemp = ωminall[[i]];
    funcmin[prec@R2toC[1 / 3 (xnew + x1 + x2)], vint, μi, branch];
    μminall[[i + 1]] = μi;
    solRminall[[i + 1]] = solR;
    vrootminall[[i + 1]] = prec@{rv -> Re[vrootout], iv -> Im[vrootout]};
    ωminall[[i + 1]] = ωout;
    lminout[[i]] = λ /. Coeff;
    printw = prec@ωminall[[i + 1]];
    Print[printw];
    Print["---------------------------------------------"];
(*If[count>5000,Print["Terminated 3!"];Break[],{}];*)
If[aa > 500, Print["Terminated 2!"]; Break[], {}];
Print[count];
Print[testnorm];

solRminout[[i]] = solRminall[[i + 1]];
ωminout[[i]] = ωminall[[i + 1]];
vrootminout[[i]] = vrootminall[[i + 1]];
```

```
μminout[[i]] = μi;

Print["Solving angular equation..."];
b = Table[Symbol["QNMcode`b" <> ToString[i]], {i, 0, kmax}];
AnglCoeffList = ConstantArray[0, {Length[μminout]}];
AnglFuncList = ConstantArray[0, {Length[μminout]}];

(*The b0 will parameterize all the solutions for \vec{b} in the
    kernel of mat. With pick a b0, such that the resulting angular
    solution has a global maximum of O(1); for numerical convenience.*)
Y[l_, m_, θ_] := SphericalHarmonicY[l, m, θ, ϕ] Exp[-I m ϕ] // Simplify;
Sfunc[m_, θ_, η_] :=
    Sum[Symbol["QNMcode`b" <> ToString[kp]] × Y[Abs[m] + 2 kp + η, m, θ], {kp, 0, kmax}];

Do[(*Note the multiplication by some power of νn in order to get the
     components of the matrix to be of order 1, rather than e-10*)
b0norm = 10;
matplug = prec@(mat // Simplify) //. {νn -> rν + I iν} /. νrootminout[[n]] /.
       {μn -> μminout[[n]], ωnr -> Re[ωminout[[n]]], ωni -> Im[ωminout[[n]]]} // Simplify;
mattemp = prec@matplug . b;
linsys = Table[mattemp[[l]] == 0, {l, 1, kmax + 1}] //. {b0 -> b0norm};
solvar = Table[Symbol["QNMcode`b" <> ToString[i]], {i, 0, kmax}];
AnglCoeff = Solve[linsys, solvar] //. {b0 -> b0norm} // Flatten;
AnglCoeffList[[n]] = AnglCoeff;
Splot[θ_] := Sfunc[m, θ, η] //. AnglCoeff //. {b0 -> b0norm} // Simplify;
AnglFuncList[[n]] = Splot[θ];
, {n, 1, i}];
Print["Done"];

modedataoutput = {AnglCoeffList, AnglFuncList,
     μminout, νrootminout, ωminout, solRminout, lminout};
spinstring = NumberForm[spin * 10^6 // Round, 6, DigitBlock -> 5,
     ExponentStep -> 6, NumberSeparator -> ""];

Export[ToString[ToString[destinationPATH] <> "/m" <> ToString[m] <> ToString["n"] <>
      ToString[nh] <> ToString["_a"] <> ToString[spinstring] <> ToString["_S"] <>
      ToString[If[Sin < 0, "m", "p"]] <> ToString[Abs[Sin]] <> ToString["_prec_"] <>
      ToString[If[direc < 1, "m", "p"]] <> ToString["_HPee.mx"]], modedataoutput];

ClearSystemCache["Numerical"];

];
Print["Minimization: Done!"];
```