In[ ]:=

Monte Carlo simulations of coherent backscattering (CBS) in anisotropic media

This Mathematica code has been used to generate the results that can be found in the paper “Coherent backscattering of light by an anisotropic biological network” by Jacucci et al.

In[ ]:=

Definition of the simulation parameters

```
ClearAll["Global`*"];
nWalkers = 1; (*N° of photons used*)
lowerCut = 1;
thickness = 7;(*Sample thickness*)
γ = 1.4; (*XY components of the TMFP*)
a = 0.725; (*Anisotropy parameter*)
γz = γ * a; (*Z component of the TMFP*)
scatteringorder = (tagliocamminosup/γz)^2;
(*Parameter that you can tune to understand the
 influence of the scattering order on the interference*)
indexWalker = 0;
indexReflected = 0;
indexTransmitted = 0;
indexOut = 0;
indexSingle = 0;
indexBouncedR = 0;
indexBouncedT = 0;
listα = Table[0, {nWalkers}];
listα2 = Table[0, {nWalkers}];
listr = Table[0, {nWalkers}];
listz = Table[0, {nWalkers}];
values = Join[Range[-10, -10, 1], Range[1, 1, 1]];
```

In[•]:= Calculation of R for internal reflections. More details can be found here

Out[•]= Calculation of R for internal reflections. More details can be found here

```
In[•]:= nc = 1.55; (*Chitin refractive index*)
α = (nc^2 - 1)/(nc^2 + 2);
n[f_] = Sqrt[(1 + 2 f * α)/(1 - f * α)]; (*Effective refractive index,
MG's theory, from Eq (10) of Soukolis et al.*)
n1 = n[0.45]; (*Chitin filling fraction in the white beetle,
from Wilts et al.*)
n2 = 1;

Rs[θ_] = Abs[(n1 * Cos[θ] - n2 * Sqrt[(1 - ((n1/n2) * Sin[θ])^2)])/
    (n1 * Cos[θ] + n2 * Sqrt[(1 - ((n1/n2) * Sin[θ])^2)])]^2; (*Fresnel's coefficients*)
Rp[θ_] = Abs[(-n2 * Cos[θ] + n1 * Sqrt[(1 - ((n1/n2) * Sin[θ])^2)])/
    (n2 * Cos[θ] + n1 * Sqrt[(1 - ((n1/n2) * Sin[θ])^2)])]^2;
R[θ_] = (Rs[θ] + Rp[θ])/2;
Ru[θ_] = (Rs[θ] + Rp[θ])/2;
c1 = NIntegrate[Ru[θ] * Sin[θ] * Cos[θ] , {θ, 0, π/2}, MaxRecursion → 100];
c2 = NIntegrate[Ru[θ] * Sin[θ] * (Cos[θ])^2, {θ, 0, π/2}, MaxRecursion → 100];
Ravg = (3 c2 + 2 c1)/(3 c2 - 2 c1 + 2);
```

In[•]:= Functions used in the main loop to discriminate the photons

In[•]:=
```
doS1[] :=
  ((*Single scattered photons. They do not contribute to the CBS effect*)
              indexSingle++;
              Break[];
          );

doS2[] := ( (*Photons scattered more than the scattering order decided*)
              indexWalker++;
              indexOut++;
              Break[];
          );

doRef[] :=
  (    (*Photons that contributes to the CBS: we want to calculate the
      distance between the first and the last scattering positions*)
              lastPoint = oldPoint;
              outPoint = currentPoint;
              delta = firstPoint - lastPoint;


              If[
                   firstPoint == lastPoint, Break[],
                   listα[[indexWalker]] =
     ArcSin[delta[[3]] / (√ (delta[[1]]^2 + delta[[2]]^2 + delta[[3]]^2))];
                   listr[[indexWalker]] =
     √ ((firstPoint[[1]] - lastPoint[[1]])^2 + (firstPoint[[2]] - lastPoint[[2]])^2 +
        (firstPoint[[3]] - lastPoint[[3]])^2);
                   listz[[indexWalker]] = outPoint[[3]];
                   indexWalker++;
                   indexReflected ++;
                   Break[];
                ]
          );

doTra[] := ((*Photons transmitted*)
              lastPoint = oldPoint;
              indexWalker++;
              indexTransmitted ++;
              Break[];
          );
```

```
doSpecR[] := ((*Photons reflected at the entrance
    surface of the material (z=0) back in the material*)
             indexBouncedR++;
             lastPoint = oldPoint;
             outPoint = currentPoint;
             delta2 = lastPoint - outPoint;

             intercept =
    RegionCentroid[RegionIntersection[InfiniteLine[{lastPoint, outPoint}],
      InfinitePlane[{{0, 0, 0}, {0, 1, 0}, {1, 0, 0}}]]];
             rt = ReflectionTransform[{delta2[[1]], delta2[[2]], 0},
     intercept];
             specular = rt[lastPoint];
             specular2 = - (γz) * Log[RandomReal[]];
   specularPoint =
    RegionCentroid[RegionIntersection[InfiniteLine[{intercept, specular}],
      InfinitePlane[{{0, 0, specular2}, {0, 1, specular2}, {1, 0, specular2}}]]];
   currentPoint = specularPoint

  );

doSpecT[] := ((*Photons reflected at the exit
    surface of the material (z=thickness) back in the material*)
             indexBouncedT++;
             lastPoint = oldPoint;
             outPoint = currentPoint;
             delta2 = lastPoint - outPoint;
                        intercept =
    RegionCentroid[RegionIntersection[InfiniteLine[{lastPoint, outPoint}],
      InfinitePlane[{{0, 0, thickness}, {0, 1, thickness}, {1, 0, thickness}}]]];
             rt = ReflectionTransform[{delta2[[1]], delta2[[2]], 0},
     intercept];
             specular = rt[lastPoint];
             specular2 = thickness + (γz) * Log[RandomReal[]];
   specularPoint =
    RegionCentroid[RegionIntersection[InfiniteLine[{intercept, specular}],
      InfinitePlane[{{0, 0, specular2}, {0, 1, specular2}, {1, 0, specular2}}]]];
   currentPoint = specularPoint;
  );
```

In[∘]:=

Main loop where the random walk is performed

```
Timing[
    While[
        indexWalker <= nWalkers,
        firstPoint = {0, 0, - (γz) * Log[RandomReal[]]};
        (*Depth of 1st scatter: probability sampled
    using an inverse sampling techinique of exp(-r/γ )/γ*)
        oldPoint = firstPoint;
        currentPoint = {0, 0, 0};
        currentOrder = 1;
        lastPoint = {0, 0, 0};
        Ravg = 0;

     While[True, (*While loop,
   the conditions to stop it are in the following Ifs*)
         ϕ = Random[Real, {0, π}];
         ψ = Random[Real, {0, 2 π}];
         δxy = -γ * Log[RandomReal[]];
         δz = a * δxy;

         randomStep =
   {δxy * √(1 - Cos[ϕ]^2) * Cos[ψ], δxy * √(1 - Cos[ϕ]^2) * Sin[ψ], δz * Cos[ϕ]};
         currentPoint = {oldPoint[[1]] + randomStep[[1]],
    oldPoint[[2]] + randomStep[[2]], oldPoint[[3]] + randomStep[[3]]};

         r = RandomReal[];
         Which[
             currentPoint[[3]] < 0 && currentOrder ≤ lowerCut , doS1[],
             currentPoint[[3]] < 0 && currentOrder > scatteringOrder, doS2[],
             currentPoint[[3]] < 0 && currentOrder > lowerCut &&
    currentOrder ≤ scatteringOrder && r > Ravg, doRef[],
             currentPoint[[3]] < 0 && currentOrder > lowerCut &&
    currentOrder ≤ scatteringOrder && r ≤ Ravg, doSpecR[],
             currentPoint[[3]] > thickness && r ≤ Ravg, doSpecT[],
             currentPoint[[3]] > thickness, doTra[]
             ];
   currentOrder++;
   oldPoint = currentPoint;
         ]
     ]
]
```

In[•]:=

Calculation of the coherent backscattering (CBS) line shape

```
In[•]:= λ = 0.635;
(*Clearing the zeros that correspond to transmitted photons*)
a = DeleteCases[listr, 0];
b = DeleteCases[listα, 0];
c = DeleteCases[-listz, 0];
(*Coherent term*)
γc[θ_] := Sum[Exp[-c[[i]] / γz] *
    (1 - Ravg) * Cos[2 π/λ a[[i]] * (Sin[b[[i]]] - Sin[b[[i]] + θ])], {i, 1, indexReflected}];
(*To normalise the CBS it is necessary to perform a simulation for semi-
 infinite media (we used OT=1000).
   Simulations with different R requires different normalisatios *)
CBS[θ_] := γc[θ];
```

Exporting the data for further analysis

```
In[•]:=
θmin = 0.0;
θmax = 0.4;
θstep = 0.2 / 57;

θstepnumber = Ceiling[(θmax - θmin)/θstep];
simulationData = Table[0, {θstepnumber}, {2}];
For[i = 1, i ≤ θstepnumber, i++,
 simulationData[[i, 1]] = N[θmin + (i - 1) * θstep];
 simulationData[[i, 2]] = CBS[θmin + (i - 1) * θstep];
   ]
(*Export["",simulationData,"Table"];*)
```

In the notebook, the sum in γc is shown as $\sum_{i=1}^{indexReflected}$, the Cos argument as $\frac{2\pi}{\lambda}$ a[[i]] * (…), and the Ceiling argument as $\frac{\theta max - \theta min}{\theta step}$.