7-24-2015

Simulations of the Angular Dependence of the Dipole-Dipole Interaction

Jacob T. Paul
Ursinus College, japaul@ursinus.edu

Matan Peleg
Ursinus College, mapeleg@ursinus.edu

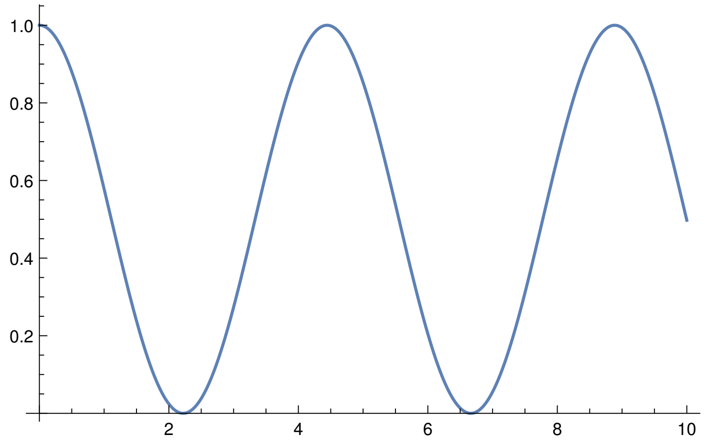
The following code is a simulation of two atoms moving between the d, p, and manifold states.

```
(* define codes for atomic  states *)
dneg = 1;
dpos = 2;
pneg = 3;
ppos = 4;
mneg  = 5;
mpos  = 6;
(* define list of states *)
phi = {{dneg, dneg}, {dneg, dpos}, {dpos, dneg}, {dpos, dpos},
       {pneg, mneg }, {pneg, mpos }, {ppos, mneg }, {ppos, mpos },
       {mneg  , pneg}, {mneg  , ppos}, {mpos  , pneg}, {mpos  , ppos}};
(* define the possible states *)
mjArray  = {0, 0, 0, 0, 0, 0};
mjArray [[dneg]] = -5 / 2;
mjArray [[dpos]] = 5 / 2;
mjArray [[pneg]] = -3 / 2;
mjArray [[ppos]] = 3 / 2;
mjArray [[mneg ]] = -3 / 2;
mjArray [[mpos ]] = 3 / 2;
(* List of constant variables,
H is the Matrix we plan to fill and make  hermatian ,
k is the number  of atoms  we use,
here we are using one and minimize  code by using k and k+1,
theta is the angle for the two atoms ,
mu  and nu are distances between states where R is the distance between atoms . *)
H = ConstantArray[0, {12, 12}];
k = 1;
θ = π / 4;
μ = 4;
ν = 1;
R = 1;
(* Angular is a function that takes in theta and delta
  mj  1 and 2. Here is where we decide that delta MJ is one of
  three options that we derived from  a series of integrals. *)
Angular[θ_, Δmj1_ , Δmj2_ ] := Module[{},
       If[Δmj1  +Δmj2  == 0 , Return[1 - (3 * Cos[θ]^2)]];
       If[Abs[Δmj1  +Δmj2  ] == 1 , Return[(3 / 2) * Sin[θ] * Cos[θ]]];
       If[Abs[Δmj1  +Δmj2  ] == 2 , Return[(1 / 4) * Sin[θ]^2]];
       Return[0];
     ];

(* The major  for loops are below that create and fill
  H. Inside this loof we have four sets of four if statements
  that are the allowed states for the atoms  to move ,
before this though we initiliazeu as an equation that accounts for
  distance between states and the atoms  and calls angular. The first
  set of if statements  is for when dneg or dpos goes to ppos or pneg,
and if dneg or dpos goes to mneg  or mpos . The second set of if statements
  is if dpos or dneg goes to mneg  or mpos  then dneg or dpos goes to
  ppos or pneg. The third case is for when the first atom  starts in
  ppos or pneg and goes to dneg or dpos and the second starts in mneg
  or mpos  and moves  to dpos or dneg. The last case is for when the
  first atom  starts in mneg  or mpos  and moves  to dneg or dpos and
  the second atom  starts in ppos or pneg and moves  to dpos or dneg. *)
```

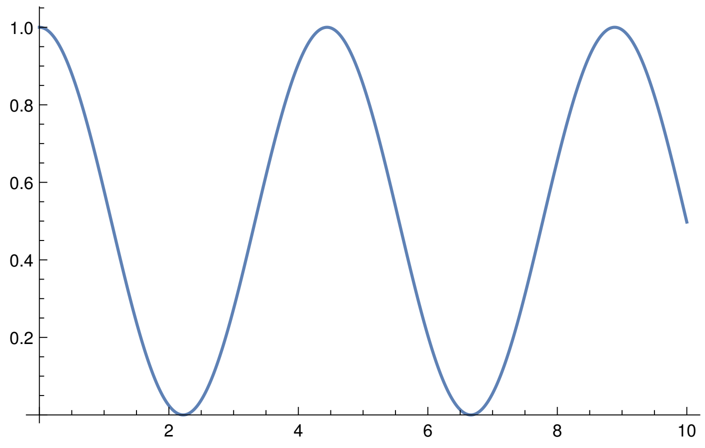
```
For[i = 1, i ≤ 12, i++,
    For[j = i, j ≤ 12, j ++,
        Δmj1  = mjArray  [[phi[[j, k]]]] - mjArray  [[phi[[i, k]]]];
        Δmj2  = mjArray  [[phi[[j, k + 1]]]] - mjArray  [[phi[[i, k + 1]]]];

        u = ((μ * ν) / R^3) Angular[θ, Δmj1  , Δmj2  ];

        If[phi[[i, k]] == dneg || phi[[i, k]] == dpos,
          If[phi[[j, k]] == ppos || phi[[j, k]] == pneg,
              If[phi[[i, k + 1]] == dneg || phi[[i, k + 1]] == dpos,
                  If[phi[[j, k + 1]] == mpos  || phi[[j, k + 1]] == mneg  ,
                      H[[i, j]] = u;
                      H[[j, i]] = u;
                    ];];];];
        If[phi[[i, k]] == dneg || phi[[i, k]] == dpos,
          If[phi[[j, k]] == mpos  || phi[[j, k]] == mneg  ,
              If[phi[[i, k + 1]] == dneg || phi[[i, k + 1]] == dpos,
                  If[phi[[j, k + 1]] == ppos || phi[[j, k + 1]] == pneg,
                      H[[i, j]] = u;
                      H[[j, i]] = u;
                    ];];];];
        If[phi[[i, k]] == ppos || phi[[i, k]] == pneg,
          If[phi[[j, k]] == dpos || phi[[j, k]] == dneg,
              If[phi[[i, k + 1]] == mneg  || phi[[i, k + 1]] == mpos  ,
                  If[phi[[j, k + 1]] == dpos || phi[[j, k + 1]] == dneg,
                      H[[i, j]] = u;
                      H[[j, i]] = u;
                    ];];];];
        If[phi[[i, k]] == mneg  || phi[[i, k]] == mpos  ,
          If[phi[[j, k]] == dpos || phi[[j, k]] == dneg,
              If[phi[[i, k + 1]] == pneg || phi[[i, k + 1]] == ppos,
                  If[phi[[j, k + 1]] == dpos || phi[[j, k + 1]] == dneg,
                      H[[i, j]] = u;
                      H[[j, i]] = u;
                    ];];];];

      ];];
```

```
dataPlot = {};
h = 1;
δ = 10;

(* Here we create the eigenvectors and eigenvalues of H,
than create S and S dagger to create the time  evolution U. The for loop for the
  matrix  Y allows us to pass the eigenvalues of a certain value through U.*)
evec = Normalize /@Eigenvectors[N[H]];
eval = Eigenvalues[N[H]];
S = Transpose[evec];
Sdag = ConjugateTranspose[S];
Y = IdentityMatrix[12];
For[w = 1, w ≤ 12, w++,
    Y[[w, w]] = eval[[w]];
  ];
U[t_] := S.MatrixExp[ⅈ*t*Y/h].Sdag;
initState= {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
probSum  = 0;
count = 0;
For[t = 0, t ≤ 10, t = t + .01,

    currentState = U[t].initState;

    probS = Sum  [currentState[[qq]] * Conjugate[currentState[[qq]]], {qq, 1, 4}];
    AppendTo[dataPlot, {t, probS}];
  ];
```

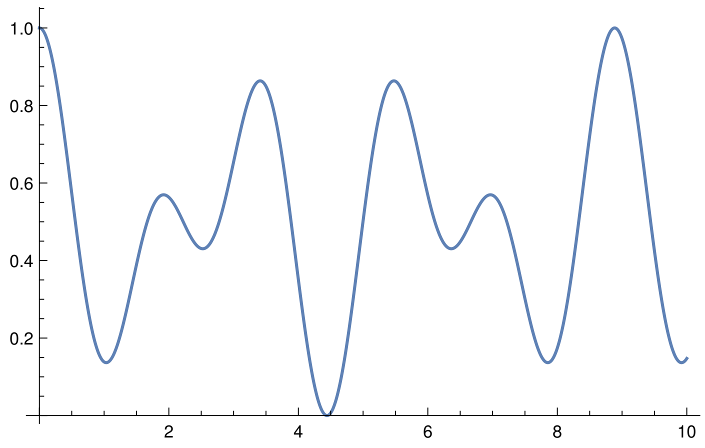
```
(* The plot below is a variation of the rabi
  oscillation of different frequencies using the data from  H*)
plot1 = ListPlot[dataPlot, Joined→ True, PlotRange→All](* 0 *)
```



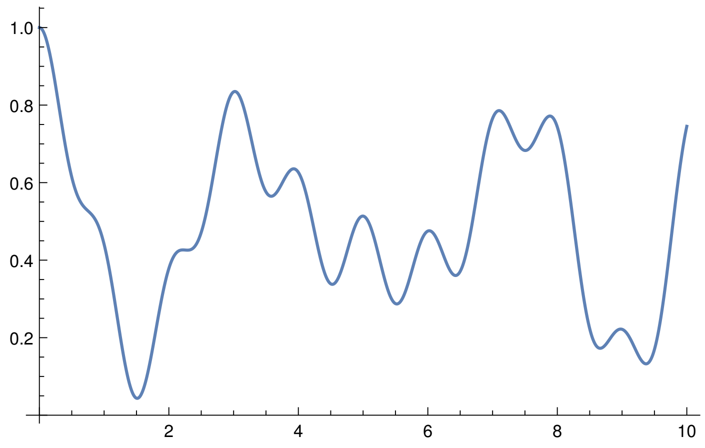
```
plot2 = ListPlot[dataPlot, Joined → True, PlotRange → All] (* pi/4 *)
```



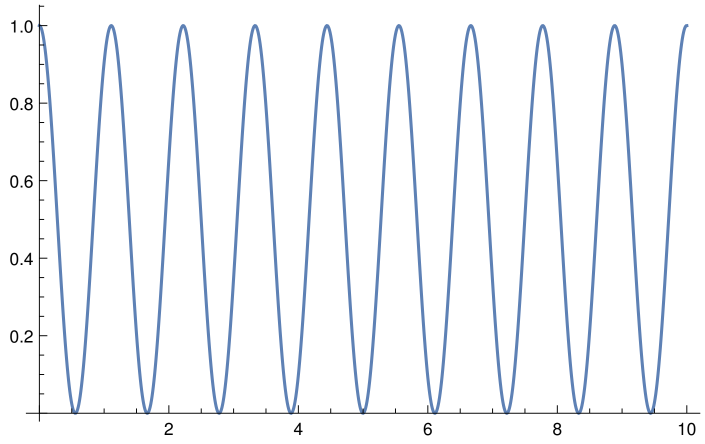
```
plot3 = ListPlot[dataPlot, Joined → True, PlotRange → All] (*pi/2*)
```



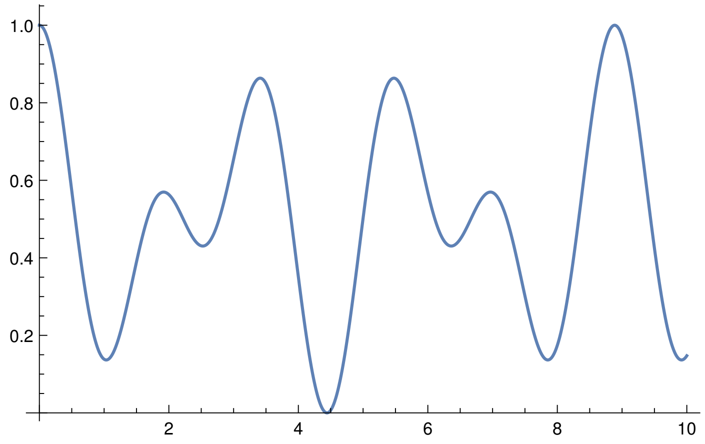
```
plot4 = ListPlot[dataPlot, Joined → True, PlotRange → All] (*5pi/4*)
```



```
plot5 = ListPlot[dataPlot, Joined → True, PlotRange → All] (*pi*)
```

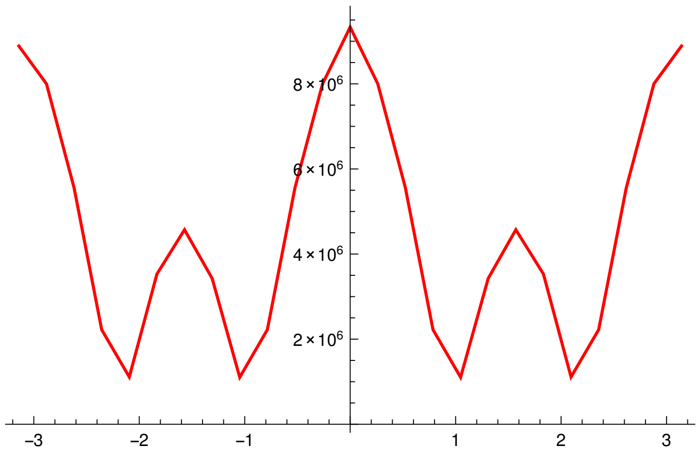


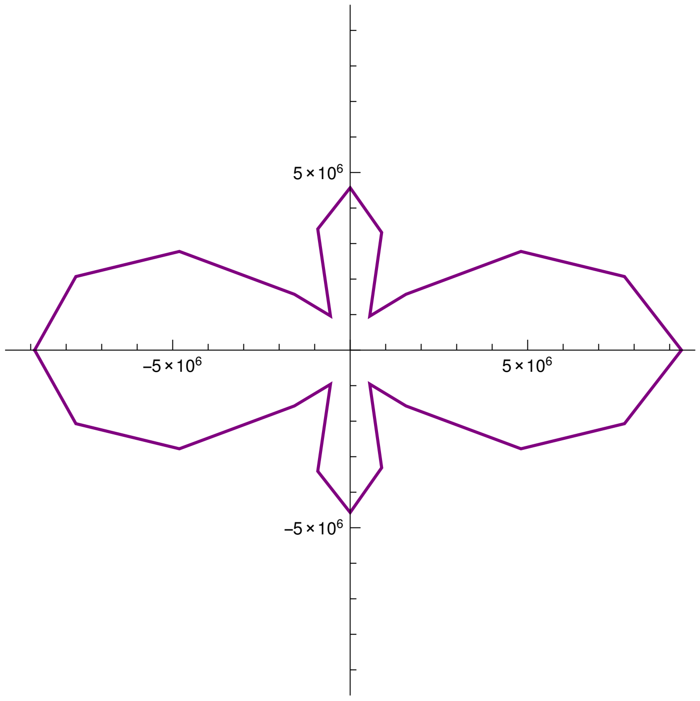
```
plot2 = ListPlot[dataPlot, Joined → True, PlotRange → All] (*3pi/2*)
```



```
(*Below are a list of the frequencies calculated using the above cdoe
  at different angles that allows us to test the angular dependency.*)
```

```
plot0; (* 7 rotations in 7.5*10^-7s *)
freq0 = 7/(7.5*10^-7);
plotπ12; (* 6 rotations in 7.5*10^-7s *)
freqπ12 = 6/(7.5*10^-7);
plotπ6; (* 10 rotations in 1.8*10^-6s *)
freqπ6 = 10/(1.8*10^-6);
plotπ4; (* 10 rotations in 4.5*10^-6s *)
freqπ4 = 10/(4.5*10^-6);
plotπ3 ;(* 8 rotations in 7.25*10^-6s *)
freqπ3 = 8/(7.25*10^-6);
plot5π12; (* 6 rotations in 1.75*10^-6s *)
freq5π12 = 6/(1.75*10^-6);
plotπ2; (* 8 rotations in 1.75*10^-6s *)
freqπ2 = 8/(1.75*10^-6);
plot7π12;(*6 rotations in 1.7*10^-6s *)
freq7π12 = 6/(1.7*10^-6);
plot2π3; (* 8 rotations in 7.2*10^-6s *)
freq2π3 = 8/(7.2*10^-6);
plot3π4; (* 8 rotations in 3.6*10^-6 *)
freq3π4 = 8/(3.6*10^-6);
plot5π6; (* 10 rotations in 1.8*10^-6s *)
freq5π6 = 10/(1.8*10^-6);
plot11π12; (* 7 rotations in 8.75*10^-7s *)
freq11π12 = 7/(8.75*10^-7);
plotπ; (* 8 rotations in 9*10^-67s *)
freqπ = 8/(9*10^-7);
freqs = {{freq0}, {freqπ12}, {freqπ6}, {freqπ4}, {freqπ3}, {freq5π12}, {freqπ2},
      {freq7π12}, {freq2π3}, {freq3π4}, {freq5π6}, {freq11π12}, {freqπ}};


freqplot = {};
angle = π;
For[a = Length[freqs], a > 0, a = a-1,
  AppendTo[freqplot, {-angle, freqs[[a, 1]]}];
  angle = angle-π/12;]

angle = 0;
For[b = 1, b ≤ Length[freqs], b = b+1,
  AppendTo[freqplot, {angle, freqs[[b, 1]]}];
  angle = angle + π/12;
]
(* These plots help us understand the strength of the
  energy exchange at different angles. As you can see on the
  second graph it is a complete  360 degrees and shows us the
  strongest and weakest angles on interaction of the dipoles. *)
plotFreq = ListPlot[freqplot, Joined→ True, PlotRange→All, PlotStyle→Red]
polarListPlot=
  ListPolarPlot[freqplot, Joined→True, PlotRange→All, PlotStyle→Purple]
```


8×10^6
6×10^6
4×10^6
2×10^6
−3
−2
−1
1
2
3

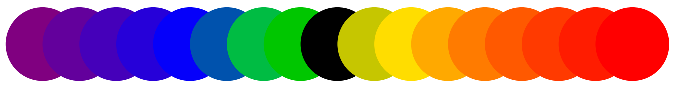

5×10^6
−5×10^6
−5×10^6
5×10^6

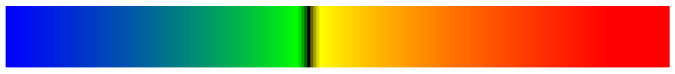
Comparisonof X, Y, and Z Orientationof the energy exchange.

The main goal of this notbeook is to compare two files and how they exchange energy. This is done by running two simulations projected from two different orientations then comparing the data through the time evolution.

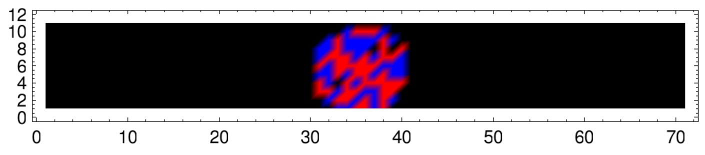
```
simulationName  = "Cylinder705X";
data1 = {};
SetDirectory[
    "/home /japaul/Research/simulationData /" <> simulationName  <> "/"];
(* loop over all lengths for both data 1 and 2*)
For[i = 0, i ≤ Length[FileNames []] - 1, i++,
    filename  = simulationName  <> "_" <> ToString[i] <> ".txt";
    istrm  = OpenRead[filename ];
    a1 = Read[istrm ];
    Close[istrm ];
    data1 = Append[data1, a1];
  ];
simulationName2  = "Cylinder705Z";
data2 = {};
SetDirectory[
    "/home /japaul/Research/simulationData /" <> simulationName2  <> "/"];
For[i = 0, i ≤ Length[FileNames []] - 1, i++,
    filename  = simulationName2  <> "_" <> ToString[i] <> ".txt";
    istrm  = OpenRead[filename ];
    a2 = Read[istrm ];
    Close[istrm ];
    data2 = Append[data2, a2];
  ];


(*To achieve the data we need we use the following
  for loop to iterate through both sets of data achieved
  from  the files compiled  through the super computer *)

Length[data2[[5, 1, 3]]]
```

```
(*data={};
For[j=1,j≤12,j++,
dontCare={};
  For[i=1, i≤ 2000,i++,
    AppendTo[dontCare,data1[[j,1,3,i]]-data2[[j,1,3,i]]];
  ];
  AppendTo[data,dontCare];
];*)

data = {};
For[i = 1, i ≤ 1001, i++,
    AppendTo[data, data1[[1, 1, 3, i]] - data2[[1, 1, 3, i]]];
  ];
(* The list density plot can be set with data[[]] to the
  time evolution we wish to see, as the number increases we
  can see how the energy exchanges differently through time .  *)


DensColor[z_] := RGBColor[If[z < 0, -z, 0], 0, (z + Abs[z]) / 2]
DensColor[z_] :=
  RGBColor[If[z < 0, (-z)^.2, If[z == 0, 1, 0]], If[z == 0, 1, 0], If[z > 0, z^.2, If[z == 0, 1, 0]]]
myblend  = (Blend[{{1, Red}, {.6, Orange}, {.3, Yellow}, {0, Black},
              {-0.3, Green}, {-.6, Blue}, {-1, Purple}}, If[# < 0, -(-#)^.7, #^.7]] &);
Graphics[Table[{myblend  [x], Disk[{8 (x + 1), 0}]}, {x, -1, 1, 1 / 8}]]

myblend  = (Blend[{{1, Red}, {.1, Yellow}, {0, Black}, {-.1, Green}, {-1, Blue}},
            If[# < 0, -(-#)^.7, #^.7]] &);
Graphics[Table[{myblend  [x], Rectangle[{5 (x + 1), 0}]}, {x, -1, 1, 1 / 100}]]
(*BarLegend[{ColorFunction→myblend ,{ -1,1}}]*)
```

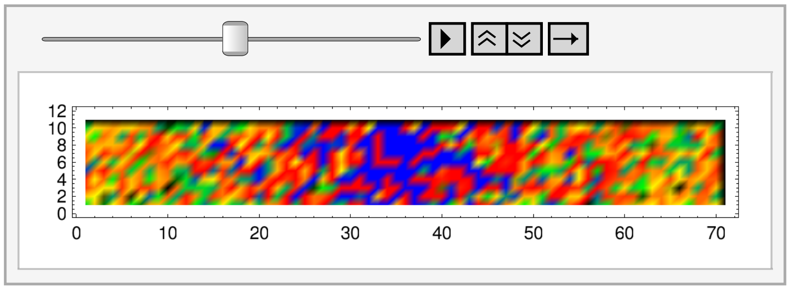




```
ListDensityPlot[data[[1]], PlotRange → All,
  ColorFunction → myblend , ColorFunctionScaling → False, AspectRatio → 1 / 6]
```



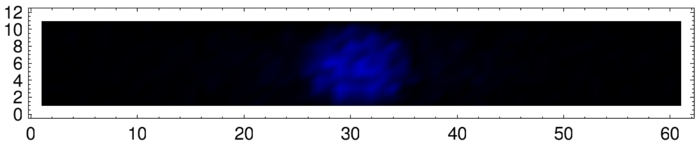
```
movie = ListAnimate[
    Table[ListDensityPlot[data[[t]], PlotRange→All, ColorFunction→myblend ,
        ColorFunctionScaling→False, AspectRatio→1/6], {t, 1, 1001, 1}]]
```



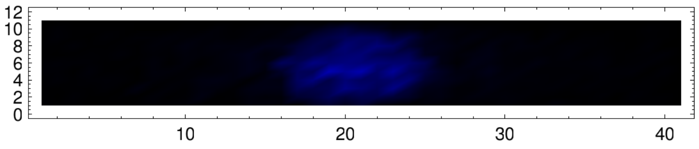
```
(*This small  bit of code takes the above animation and exports it as a .gif file
  that allows us to use it in presentations and for various other things.*)
CylinderEvo3 = Table[ListDensityPlot[data[[t]], PlotRange→All, ColorFunction→
          myblend , ColorFunctionScaling→False, AspectRatio→1/6], {t, 1, 1001, 1}];
Export["/home /japaul/Documents /CylinderEvo3.gif", CylinderEvo3]
```

```
/home /japaul/Documents /CylinderEvo3.gif
```

```
(*The following two plots are just the plots of data1 and data2. Here you
  can see the difficulty of seeing a change in the energy dispersion so we
  want to look at the plots above to see the difference in energy exchange*)

ListDensityPlot[data1[[3, 1, 3, 2000]]/Max[data1[[1, 1, 2, 1]]], PlotRange → All,
  ColorFunction→DensColor, ColorFunctionScaling→ False, AspectRatio→ 1/6]
```



```
ListDensityPlot[data2[[1, 1, 3, 1000]]/Max[data2[[1, 1, 2, 1]]], PlotRange → All,
  ColorFunction→DensColor, ColorFunctionScaling→ False, AspectRatio→ 1/6]
```



```
s
```

```
(*This small  amount  of code is used to show
  the exponentially increasing size of or matricies ,
and thus demonstrate  our need for the super computer . n is the
    total number  of atoms  and ns is the number  of s atoms  n-
  ns is the number  of p atoms  in the simulation . *)
Size[n_, ns_] := Binomial [n, ns] * 2^n
Table[Size[t, t-3], {t, 1, 100, 5}]

{0, 1280, 337920, 36700160, 2789212160, 174483046400, 9652938997760,
  490657063895040, 23441587904184320, 1068197536616939520, 46893731119995289600,
  1997436506731362385920, 82987289901600845332480, 3376492035251796327792640,
  134953428164207286094397440, 5311717819931776936365260800,
  206291101859040322371861872640, 7918173976279113583323523317760,
  300782291345168845392240212705280, 11320119860038088555365559624007680}
```

```
(*The following code is used to creat three cylinders at different sizes angles
  and orientations to model  our series of lasers in the experiment . Where the
  two smaller  cyinders intersect is a overlapping volume  full of p atoms ,
where the larger cylinder intersects that area are the remaining  s atoms . In
  this picture one cylinder is much  larger then the one it is overlapping,
this was done to show there are two different lasers,
in reality the lasers are the same  size.*)
Plot1 = Show[Graphics3D[{Opacity[.3], Red, Cylinder[{{0, 0, 0}, {1, 1, 1}}, 1/16]}],
      Axes → False, Boxed → False];
Plot2 = Show[Graphics3D[{Opacity[.3], Blue, Cylinder[{{1, 1, 0}, {0, 0, 1}}, 1/16]}],
      Axes → False, Boxed → False];
Plot3 = Show[Graphics3D[{Opacity[0.3], Red, Cylinder[{{0, 0, 0}, {1, 1, 1}}, 1/8]}],
      Axes → False, Boxed → False]; (*776*)

Show[Plot1, Plot2, Plot3]
```