A Classical explanation for the correlation of entangled Quantum particles via the detection loophole

Abstract

Quantum Mechanics claim that particles can become entangled such that there is a correlation in the detected results from EPR type experiments that cannot be explained by Classical Physics. This paper shows one way, via the detection loophole, that the result can be fully explained by Classical Physics, and that the correlation curve for different angles between the two detectors can by reproduced when modelled this way.

The Explanation

The following URL is for a Javascript website that allows for different models to be evaluated to try and match the result as measured by Quantum Mechanics:

http://fmoldove.blogspot.com.au/2013_08_01_archive.html

The following screenshot is from that website, showing the Classical prediction in Blue, and the Quantum Mechanical result in Green:

The goal is to try and match the QM (Green) line.

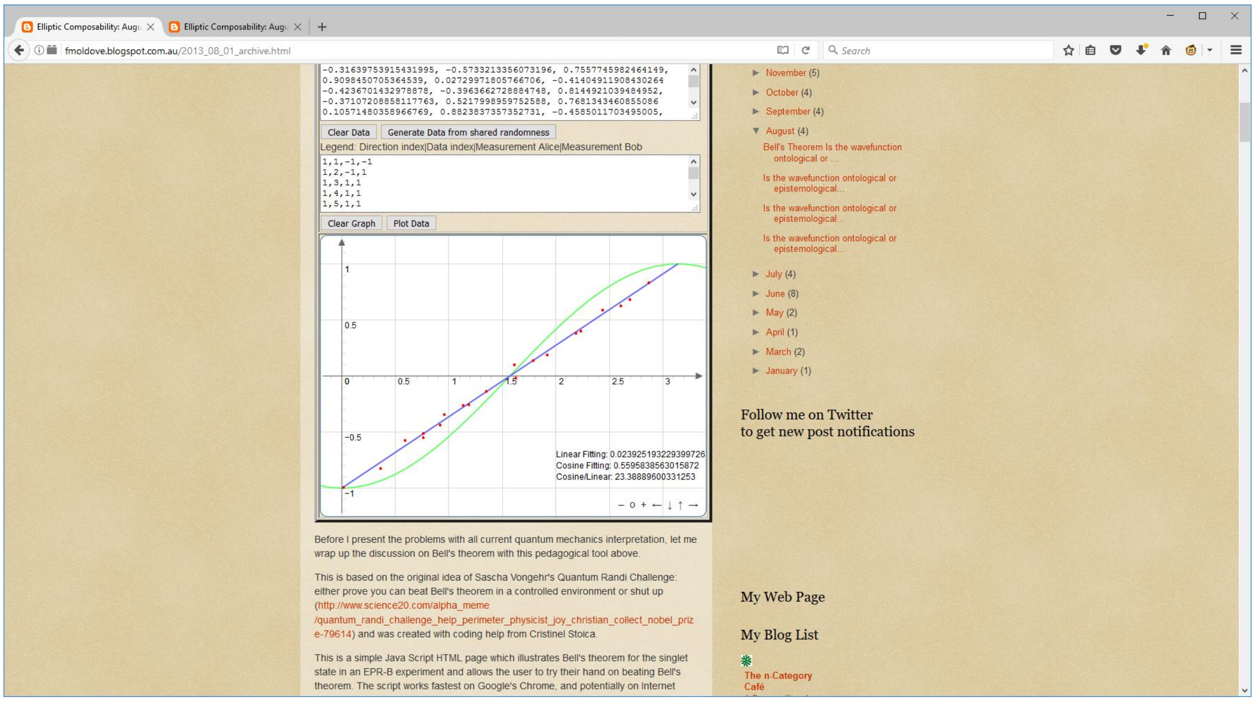


Figure (1).

I can fully explain the QM result using only Classical Physics. The plot below, Figure (2), is generated using my source code (show at the end of this paper in Appendix B) and as you can see, the plotted curve is an exact match of the QM (Green) curve in Figure (1).

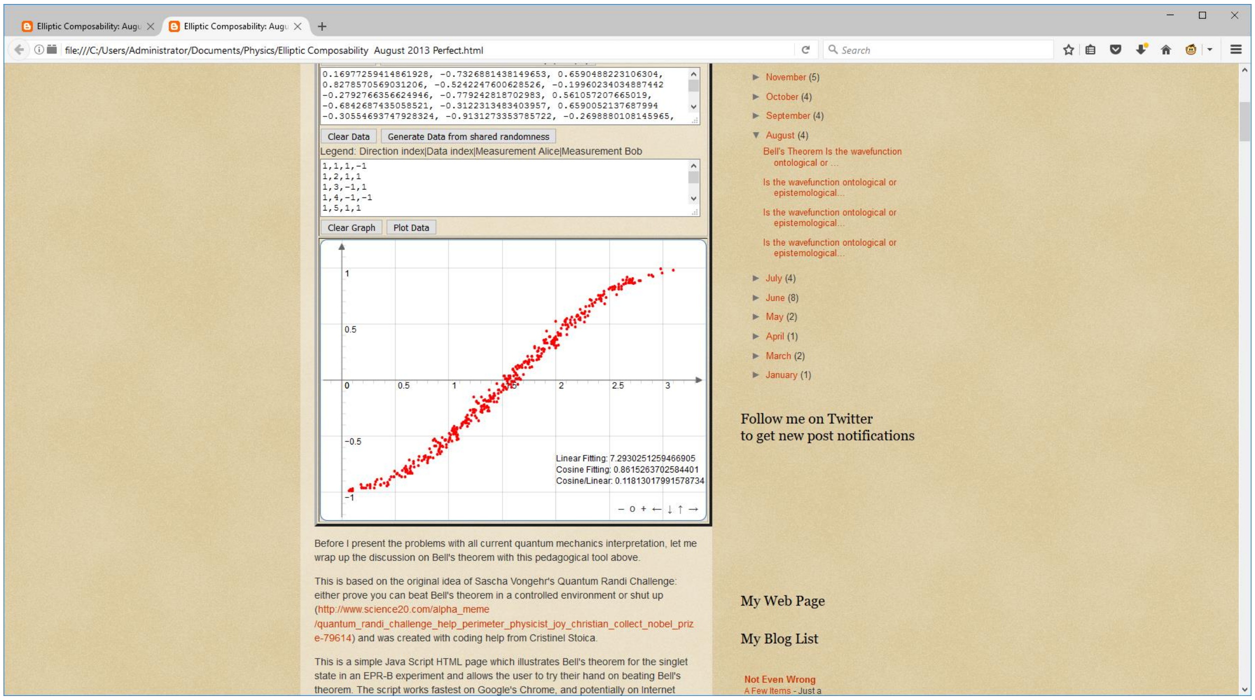


Figure (2).

The explanation for the result is that when the QM particles are incident on the detectors at a very shallow angle (Pi/8 or less), on average only half of the occurrences at each detector result in a detection, so the incidence of a double-detection (that is both detectors registering a result) is reduced even further. The other half of the results from each detector are discarded. The resulting plot is a graph of the recorded measurements from the successful double-detection measurements. In a real experiment, the shallow angles may be more difficult to register detections, and even if the experiment is run for longer at these angles to build up the same number of samples, half of them would have been rejected, leading to the result shown in Figure (2).

Only three sections of code have been modified from the original website code:

(1) The function: GenerateAliceOutputFromSharedRandomness(), to reject half of the very shallow angle incident photons for Alice's detector.
(2) The function: GenerateBobOutputFromSharedRandomness (), to reject half of the very shallow angle incident photons for Bob's detector.
(3) A retry of another random photon angle (in function generateData()) if either Alice or Bob do not record a detection.

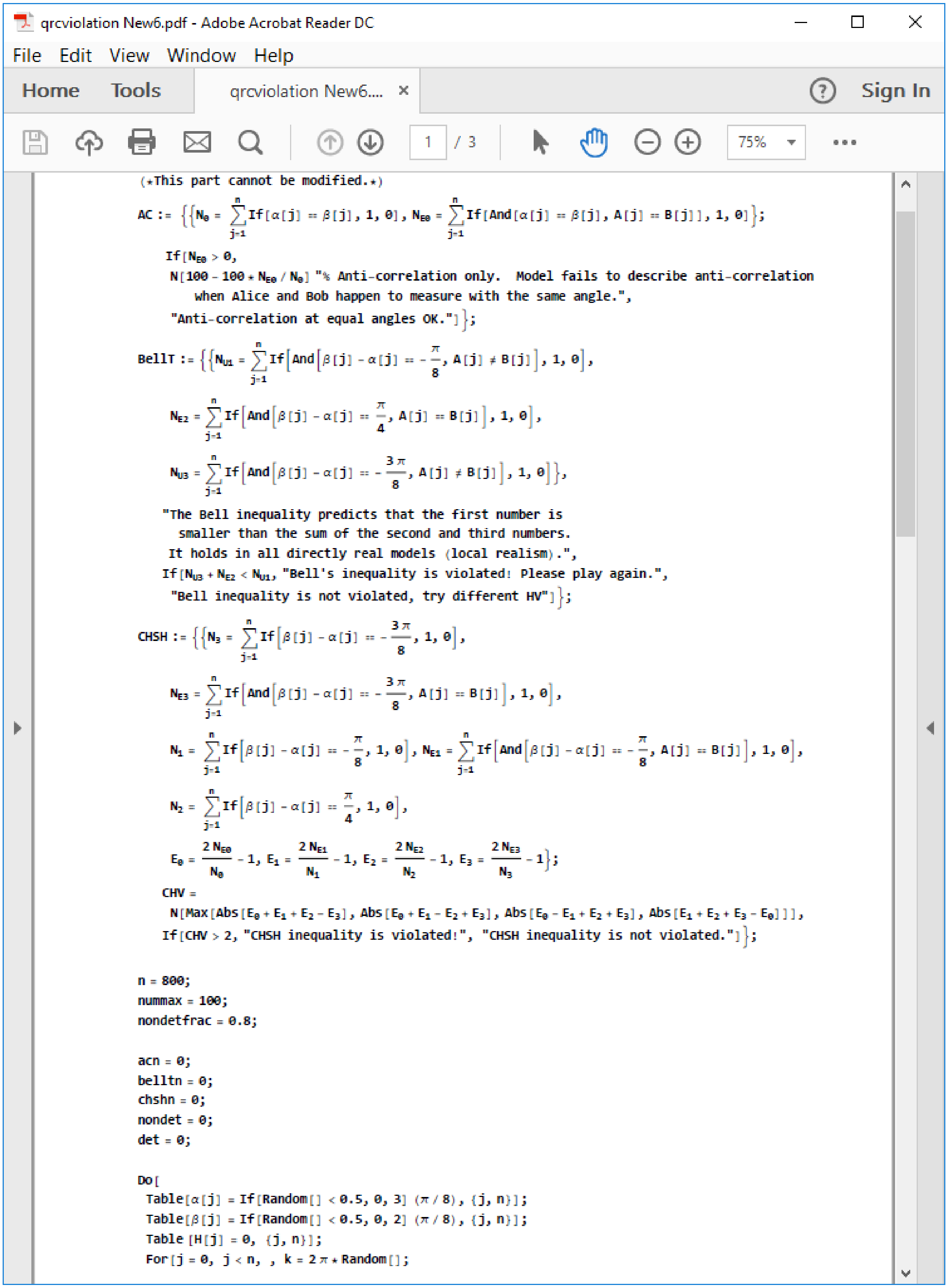
The changed code is highlighted in Cyan to show the changes I have made to the Javascript.

The solution in Mathematica Wolfram language, based on the Quantum Randi Challenge code is shown in Appendix A.

Conclusion

This result indicates that QM entanglement is possible to be explained via the detection loophole not involving "spooky action at a distance" or other mystical communication between the two QM particles. However, recent experiments claim to be "loophole free" and still detect the correlation.

Appendix A



```
(*This part cannot be modified.*)
AC := {{N0 = Sum[If[α[j] == β[j], 1, 0], {j, 1, n}], NE0 = Sum[If[And[α[j] == β[j], A[j] == B[j]], 1, 0], {j, 1, n}]};
   If[NE0 > 0,
    N[100 - 100 * NE0 / N0] "% Anti-correlation only.  Model fails to describe anti-correlation
       when Alice and Bob happen to measure with the same angle.",
    "Anti-correlation at equal angles OK."]};

BellT := {{NU1 = Sum[If[And[β[j] - α[j] == -π/8, A[j] ≠ B[j]], 1, 0], {j, 1, n}],

    NE2 = Sum[If[And[β[j] - α[j] == π/4, A[j] == B[j]], 1, 0], {j, 1, n}],

    NU3 = Sum[If[And[β[j] - α[j] == -3π/8, A[j] ≠ B[j]], 1, 0], {j, 1, n}]},

   "The Bell inequality predicts that the first number is
     smaller than the sum of the second and third numbers.
    It holds in all directly real models (local realism).",
   If[NU3 + NE2 < NU1, "Bell's inequality is violated! Please play again.",
    "Bell inequality is not violated, try different HV"]};

CHSH := {{N3 = Sum[If[β[j] - α[j] == -3π/8, 1, 0], {j, 1, n}],

    NE3 = Sum[If[And[β[j] - α[j] == -3π/8, A[j] == B[j]], 1, 0], {j, 1, n}],

    N1 = Sum[If[β[j] - α[j] == -π/8, 1, 0], {j, 1, n}], NE1 = Sum[If[And[β[j] - α[j] == -π/8, A[j] == B[j]], 1, 0], {j, 1, n}],

    N2 = Sum[If[β[j] - α[j] == π/4, 1, 0], {j, 1, n}],

    E0 = 2 NE0/N0 - 1, E1 = 2 NE1/N1 - 1, E2 = 2 NE2/N2 - 1, E3 = 2 NE3/N3 - 1};
   CHV =
    N[Max[Abs[E0 + E1 + E2 - E3], Abs[E0 + E1 - E2 + E3], Abs[E0 - E1 + E2 + E3], Abs[E1 + E2 + E3 - E0]]],
   If[CHV > 2, "CHSH inequality is violated!", "CHSH inequality is not violated."]};

n = 800;
nummax = 100;
nondetfrac = 0.8;

acn = 0;
belltn = 0;
chshn = 0;
nondet = 0;
det = 0;

Do[
 Table[α[j] = If[Random[] < 0.5, 0, 3] (π / 8), {j, n}];
 Table[β[j] = If[Random[] < 0.5, 0, 2] (π / 8), {j, n}];
 Table [H[j] = 0, {j, n}];
 For[j = 0, j < n, , k = 2 π * Random[];
```

qrcviolation New6.pdf - Adobe Acrobat Reader DC

File Edit View Window Help

Home Tools qrcviolation New6.... ×

Sign In

2 / 3 75%


```
If[( ( ((Dot[AngleVector[α[j]], AngleVector[k]] > 0) &&
        (ArcCos[Dot[AngleVector[α[j]], AngleVector[k]]] > π * (1 / 2 - 1 / 8)) &&
        (Random[] <= nondetfrac)) || ((Dot[AngleVector[α[j]], AngleVector[k]] <= 0) &&
        (ArcCos[Dot[AngleVector[α[j]], AngleVector[k]]] < π * (1 / 2 + 1 / 8)) &&
        (Random[] <= nondetfrac)) ) || ( ((Dot[AngleVector[β[j]], AngleVector[k]] > 0) &&
        (ArcCos[Dot[AngleVector[β[j]], AngleVector[k]]] > π * (1 / 2 - 1 / 8)) &&
        (Random[] <= nondetfrac)) || ((Dot[AngleVector[β[j]], AngleVector[k]] <= 0) &&
        (ArcCos[Dot[AngleVector[β[j]], AngleVector[k]]] < π * (1 / 2 + 1 / 8)) &&
        (Random[] <= nondetfrac)) ) ), nondet++, det++;
 H[j] = k;
 j++]];

(*Random angles and random outcomes.*)
Table[A[j] = If[(Dot[AngleVector[α[j]], AngleVector[H[j]]] > 0), 1, 0], {j, n}];

(*Bob's outcomes are correlated with Alice's angles.*)
Table[B[j] = If[(Dot[AngleVector[β[j]], AngleVector[H[j]]] > 0), 0, 1], {j, n}];

AC;
MatrixForm[BellT];
CHSH;

If[N_E0 > 0, , acn++];
If[(N_U3 + N_E2) < N_U1, belltn++,];
If[CHV > 2, chshn++,];

Clear[α];
Clear[β];
Clear[A];
Clear[B];
Clear[H];
, {num, nummax}]

Print["% AC" acn];
Print["% Bell" belltn];
Print["% CHSH" chshn];
Print[" Detects" det];
Print[" Non Detects" nondet];

ClearAll["Global'*"]
```

```
42 278 Non Detects

80 000 Detects

100 % CHSH

99 % Bell

100 % AC
```

Appendix B

```
//Dot is the scalar product of 2 3D vectors

function Dot(a, b)

{

 return a[0]*b[0] + a[1]*b[1] + a[2]*b[2];

};


//Norm computes the norm of a 3D vector

function GetNorm(vect)

{

 return Math.sqrt(Dot(vect, vect));

};


//Normalize generates a unit vector out of a vector

function Normalize(vect)

{

 //declares the variable

 var ret = new Array(3);


 //computes the norm

 var norm = GetNorm(vect);


 //scales the vector

 ret[0] = vect[0]/norm;

 ret[1] = vect[1]/norm;

 ret[2] = vect[2]/norm;

 return ret;

};
```

```
//RandomDirection create a 3D unit vector of random direction
function RandomDirection()
{
 //declares the variable
 var ret = new Array(3);

 //fills a 3D cube with coordinates from -1 to 1 on each direction
 ret[0] = 2*(Math.random()-0.5);
 ret[1] = 2*(Math.random()-0.5);
 ret[2] = 2*(Math.random()-0.5);

 //excludes the points outside of a unit sphere (tries again)
 if(GetNorm(ret) > 1)
  return RandomDirection();

 return Normalize(ret);
};

var generateData = function()
{
   clearBoard();
 clearOutput();
 //gets the data
 var angMom = new Array();
 var t = document.getElementById('in_data').value;
 var data = t.split('\n');
 for (var i=0;i<data.length;i++)
 {
    var vect = data[i].split(',');
    if(vect.length == 3)
```

```
  angMom[i] = data[i].split(',');
  }

  var newTotAngMom = angMom.length;
  clearBoard();

var varianceLinear = 0;
var varianceCosine = 0;
var totTestDirs = document.getElementById('totTestDir').value;


var abDirections = new Array();
var AliceDirections = new Array();
var BobDirections = new Array();
var t2 = document.getElementById('in_test').value;
var data2 = t2.split('\n');
for (var k = 0; k < data2.length; k++)
{
      var vect2 = data2[k].split(',');
      if (vect2.length == 6)
 {
          abDirections[k] = data2[k].split(',');
          AliceDirections[k] = data2[k].split(',');
  BobDirections[k] = data2[k].split(',');

     AliceDirections[k][0] = abDirections[k][0];
     AliceDirections[k][1] = abDirections[k][1];
     AliceDirections[k][2] = abDirections[k][2];
     BobDirections[k][0]   = abDirections[k][3];
     BobDirections[k][1]   = abDirections[k][4];
```

```
      BobDirections[k][2]   = abDirections[k][5];
 }
}

var TempOutput = "";
var alice_result = 0;
var bob_result = 0;

//computes the output
for(var j=0; j<totTestDirs; j++)
{
 var a = AliceDirections[j];
 var b = BobDirections[j];

 for(var i=0; i<newTotAngMom; i++)
 {
                angMom[i] = RandomDirection();

      alice_result = GenerateAliceOutputFromSharedRandomness(a, angMom[i]);
                bob_result = GenerateBobOutputFromSharedRandomness(b, angMom[i]);

                if ( (alice_result == 0) || (bob_result == 0) ) {
                 i--;
                        continue;
                }

      TempOutput = TempOutput + (j+1);
      TempOutput = TempOutput + ",";
      TempOutput = TempOutput + (i+1);
      TempOutput = TempOutput + ",";
```

```
      TempOutput = TempOutput +
      (alice_result);
      TempOutput = TempOutput + ",";
      TempOutput = TempOutput +
      (bob_result);
   if(i != newTotAngMom-1 || j != totTestDirs-1)
       TempOutput = TempOutput + " \n";
  }
 }

 apendResults(TempOutput);
};

var plotData = function()
{
   clearBoard();
  boardCorrelations.suspendUpdate();
 //gets the data
 var angMom = new Array();
 var t = document.getElementById('in_data').value;
 var data = t.split('\n');
 for (var i=0;i<data.length;i++)
 {
    var vect = data[i].split(',');
    if(vect.length == 3)
   angMom[i] = data[i].split(',');
   }

   var newTotAngMom = angMom.length;
```

```
var varianceLinear = 0;

var varianceCosine = 0;

var totTestDirs = document.getElementById('totTestDir').value;


//extract directions

var abDirections = new Array();

var AliceDirections = new Array();

var BobDirections = new Array();

var t2 = document.getElementById('in_test').value;

var data2 = t2.split('\n');

for (var k = 0; k < data2.length; k++)

{

    var vect2 = data2[k].split(',');

    if (vect2.length == 6)

 {

      abDirections[k] = data2[k].split(',');

      AliceDirections[k] = data2[k].split(',');

  BobDirections[k] = data2[k].split(',');


   AliceDirections[k][0] = abDirections[k][0];

   AliceDirections[k][1] = abDirections[k][1];

   AliceDirections[k][2] = abDirections[k][2];

   BobDirections[k][0]   = abDirections[k][3];

   BobDirections[k][1]   = abDirections[k][4];

   BobDirections[k][2]   = abDirections[k][5];

 }

}




var tempLine = new Array();
```

```
var Data_Val = document.getElementById('out_measurements').value;

var data_rows = Data_Val.split('\n');


var directionIndex = 1;

var beginNewDirection = false;


     var a = new Array(3);

a[0] = AliceDirections[0][0];

a[1] = AliceDirections[0][1];

a[2] = AliceDirections[0][2];

      var b = new Array(3);

b[0] = BobDirections[0][0];

b[1] = BobDirections[0][1];

b[2] = BobDirections[0][2];

var sum = 0;


for (var ii=0;ii<data_rows.length;ii++)

{

 //parse the input line

   var vect = data_rows[ii].split(',');

   if(vect.length == 4)

     tempLine = data_rows[ii].split(',');


 //see if a new direction index is starting

 if (directionIndex != tempLine[0])

 {

  beginNewDirection = true;

 }


 if(!beginNewDirection)
```

```
{
 var sharedRandomnessIndex = tempLine[1];
 var sharedRandomness = angMom[sharedRandomnessIndex];
 var aliceOutcome = tempLine[2];
 var bobOutcome = tempLine[3];
 sum = sum + aliceOutcome*bobOutcome;
}


if (beginNewDirection)
{
 //finish computation
 var epsilon = sum/newTotAngMom;
 var angle = Math.acos(Dot(a, b));

 boardCorrelations.createElement('point', [angle,epsilon],{size:0.1,withLabel:false});

 var diffLinear = epsilon - (-1+2/Math.PI*angle);
 varianceLinear = varianceLinear + diffLinear*diffLinear;
 var diffCosine = epsilon + Math.cos(angle);
 varianceCosine = varianceCosine + diffCosine*diffCosine;

 //reset and start a new cycle
 directionIndex = tempLine[0];
 a[0] = AliceDirections[directionIndex-1][0];
 a[1] = AliceDirections[directionIndex-1][1];
 a[2] = AliceDirections[directionIndex-1][2];
 b[0] = BobDirections[directionIndex-1][0];
 b[1] = BobDirections[directionIndex-1][1];
 b[2] = BobDirections[directionIndex-1][2];
```

```
  sum = 0;
  var sharedRandomnessIndex = tempLine[1];
  var sharedRandomness = angMom[sharedRandomnessIndex];
  var aliceOutcome = tempLine[2];
  var bobOutcome = tempLine[3];
  sum = sum + aliceOutcome*bobOutcome;
  beginNewDirection = false;
 }


  }
 //finish computation for last element of the loop above
 var epsilon = sum/newTotAngMom;
 var angle = Math.acos(Dot(a, b));


 boardCorrelations.createElement('point', [angle,epsilon],{size:0.1,withLabel:false});


 var diffLinear = epsilon - (-1+2/Math.PI*angle);
 varianceLinear = varianceLinear + diffLinear*diffLinear;
 var diffCosine = epsilon + Math.cos(angle);
 varianceCosine = varianceCosine + diffCosine*diffCosine;


 //display total fit
 boardCorrelations.createElement('text',[2.0, -0.7, 'Linear Fitting: ' +
 varianceLinear],{});
 boardCorrelations.createElement('text',[2.0, -0.8, 'Cosine Fitting: ' +
 varianceCosine],{});
 boardCorrelations.createElement('text',[2.0, -0.9, 'Cosine/Linear: ' +
varianceCosine/varianceLinear],{});


 boardCorrelations.unsuspendUpdate();
```

```
};

var clearBoard = function()
{
 JXG.JSXGraph.freeBoard(boardCorrelations);
 boardCorrelations =
JXG.JSXGraph.initBoard('jxgboxCorrelations',{boundingbox:[-0.20, 1.25,
3.4, -1.25],axis:true,

showCopyright:false});
 boardCorrelations.create('functiongraph', [function(t){ return
-Math.cos(t); }, -Math.PI*10, Math.PI*10],{strokeColor:

"#66ff66", strokeWidth:2,highlightStrokeColor: "#66ff66",
highlightStrokeWidth:2});
 boardCorrelations.create('functiongraph', [function(t){ return
-1+2/Math.PI*t; }, 0, Math.PI],{strokeColor: "#6666ff",

strokeWidth:2,highlightStrokeColor: "#6666ff", highlightStrokeWidth:2});
};

var clearInput = function()
{
 document.getElementById('in_data').value = '';
};

var clearTestDir = function()
{
 document.getElementById('in_test').value = '';
```

```
};

var clearOutput = function()
{
 document.getElementById('out_measurements').value = '';
};

var generateTestDir = function()
{
   clearBoard();
 var totTestDir = document.getElementById('totTestDir').value;
 var testDir = new Array(totTestDir);
 var strData = "";
 for(var i=0; i<totTestDir; i++)
 {
  //first is Alice, second is Bob
  testDir[i] = RandomDirection();
  strData = strData + testDir[i][0] + ", " + testDir[i][1] + ", " +
testDir[i][2]+ ", " ;
  testDir[i] = RandomDirection();
  strData = strData + testDir[i][0] + ", " + testDir[i][1] + ", " +
testDir[i][2] + '\n';
 }

 document.getElementById('in_test').value = strData;
};

var generateRandomData = function()
{
   clearBoard();
```

```
 var totAngMoms = document.getElementById('totAngMom').value;

 var angMom = new Array(totAngMoms);

 var strData = "";

 for(var i=0; i<totAngMoms; i++)

 {

  angMom[i] = RandomDirection();

  strData = strData + angMom[i][0] + ", " + angMom[i][1] + ", " +
angMom[i][2] + '\n';

 }


 document.getElementById('in_data').value = strData;

};


var apendResults= function(newData)

{

 var existingData = document.getElementById('out_measurements').value;

 existingData = existingData + newData;

 document.getElementById('out_measurements').value = existingData;

};



function GenerateAliceOutputFromSharedRandomness(direction, sharedRandomness3DVector) {

   var dot = Dot(direction, sharedRandomness3DVector);

            var rand = Math.random();


            if ((dot > 0) && (rand <= 0.5) && (Math.acos(dot) > Math.PI*(1/2-1/8)) ) return 0;

            if ((dot <= 0) && (rand <= 0.5) && (Math.acos(dot) < Math.PI*(1/2+1/8)) ) return 0;


   //replace this with your own function returning +1 or -1

   if (dot > 0)
```

```
        return +1;
    else
        return -1;
//}
};

function GenerateBobOutputFromSharedRandomness(direction, sharedRandomness3DVector) {
    var dot = Dot(direction, sharedRandomness3DVector);
            var rand = Math.random();

            if ((dot > 0) && (rand <= 0.5) && (Math.acos(dot) > Math.PI*(1/2-1/8)) ) return 0;
            if ((dot <= 0) && (rand <= 0.5) && (Math.acos(dot) < Math.PI*(1/2+1/8)) ) return 0;

    //replace this with your own function returning +1 or -1
    if (dot > 0)
        return -1;
    else
        return +1;
//}
};

var boardCorrelations = JXG.JSXGraph.initBoard('jxgboxCorrelations',
{axis:true, boundingbox: [-0.25, 1.25, 3.4, -1.25],
showCopyright:false});

clearBoard();
generateRandomData();
generateTestDir();
generateData();
plotData();
```