Simulation Based on Joy Christian's updated 3-Sphere model and Michel Fodje's epr-simple simulation translated from Python to Mathematica by John Reed 13 Nov 2013 plus some Quaternion Parts and using John Reed's trial number matching code. Modified, Created by Fred Diether for Completely Local-Realistic Feb. 2022

Load Quaternion Package, Set Run Time Parameters, Initialize Arrays and Tables

In[229]:=
```
<< Quaternions`
β0 = Quaternion[1, 0, 0, 0];
β1 = Quaternion[0, 1, 0, 0];
β2 = Quaternion[0, 0, 1, 0];
β3 = Quaternion[0, 0, 0, 1];
Qcoordinates = {β1, β2, β3};
Qcoordinates2 = {β0, β1, β2, β3};
m = 5 000 000; (*Number of events to perform.*)
trialDeg = 720;
ss = ConstantArray[0, m];
λ = ConstantArray[0, m];
Ls1 = ConstantArray[0, m];
Ls2 = ConstantArray[0, m];
a1 = ConstantArray[0, m];
b1 = ConstantArray[0, m];
aa1 = ConstantArray[0, m];
bb1 = ConstantArray[0, m];
qa1 = ConstantArray[0, m];
qb1 = ConstantArray[0, m];
outqA = Table[{0, 0, 0}, m];
outqB = Table[{0, 0, 0}, m];
λ1 = ConstantArray[0, m];
λ2 = ConstantArray[0, m];
m2 = 20 000;
r0 = ConstantArray[0, m2];
r1 = ConstantArray[0, m2];
r2 = ConstantArray[0, m2];
QAB = ConstantArray[0, m2];
plotq = Table[{0, 0}, m2];
ssca = ConstantArray[0, m];
sscb = ConstantArray[0, m];
nPP = ConstantArray[0, trialDeg];
nNN = ConstantArray[0, trialDeg];
nPN = ConstantArray[0, trialDeg];
nNP = ConstantArray[0, trialDeg];
nAP = ConstantArray[0, trialDeg];
nBP = ConstantArray[0, trialDeg];
ϕ = 3; β = 0.23; ξ = -15;(*Adjustable parameters*)
```

Generating Particle Data with Three Independent Do-Loops and Implement Hidden Variable Mechanisms

We note here that the Limit[x, x → Sign[x]] = Sign[x] so that we just use the sign function in the A and B Do-loops instead of the limits.

In[266]:=
```
Do[s = RandomPoint[Sphere[]];  (*Singlet 3D vector; Hidden Variable*)
 Θ1 = ToSphericalCoordinates[s][[3]] * 180 / π;
 ss[[k]] = Θ1;
 λ[[k]] = β (Cos[Θ1 / ϕ]^2);  (*Hidden variable mechanism*)
 Ls1[[k]] = s.Qcoordinates;  (*Convert to quaternion coordinates; A particle spin*)
 Ls2[[k]] = -s.Qcoordinates, {k, m}]  (*B particle spin plus conservation of angular momentum*)
```

In[267]:=
```
Do[a = RandomInteger[{-179, 180}];  (*Detector 2D vector angle 1 degree increments*)
 a1[[k]] = a;
 aa = N[Flatten[{FromPolarCoordinates[{1, a * π / 180}], 0.000001}]];
 aa1[[k]] = aa;
 Da = aa.Qcoordinates;  (*Convert to quaternion coordinates; A detector*)
 qa = Da ** Ls1[[k]];  (*Detector - particle interaction*)
 qa1[[k]] = qa;
 If[Abs[Re[qa]] > λ[[k]], qA = Sign[Re[qa]], qA = Sign[Sin[(a - ss[[k]] + ξ) Degree]]];
 A0 = Sign[Sin[(a - ss[[k]] + ξ) Degree]];
 outqA[[k]] = {a, qA, A0};
 If[Abs[Re[qa]] > λ[[k]], λ1[[k]] = 0, λ1[[k]] = k], {k, m}]  (*Hidden variable mechanism*)
outqA3 = outqA;
```

In[269]:=
```
Do[b = RandomInteger[{-179, 180}];  (*Detector 2D vector angle 1 degree increments*)
 b1[[k]] = b;
 bb = N[Flatten[{FromPolarCoordinates[{1, b * π / 180}], 0.000001}]];
 bb1[[k]] = bb;
 Db = bb.Qcoordinates;  (*Convert to quaternion coordinates; B detector*)
 qb = Ls2[[k]] ** Db;  (*Detector - particle interaction*)
 qb1[[k]] = qb;
 If[Abs[Re[qb]] > λ[[k]], qB = Sign[Re[qb]], qB = -Sign[Sin[(b - ss[[k]] + ξ) Degree]]];
 B0 = -Sign[Sin[(b - ss[[k]] + ξ) Degree]];
 outqB[[k]] = {b, qB, B0};
 If[Abs[Re[qb]] > λ[[k]], λ2[[k]] = 0, λ2[[k]] = k], {k, m}]  (*Hidden variable mechanism*)
outqB3 = outqB;
```

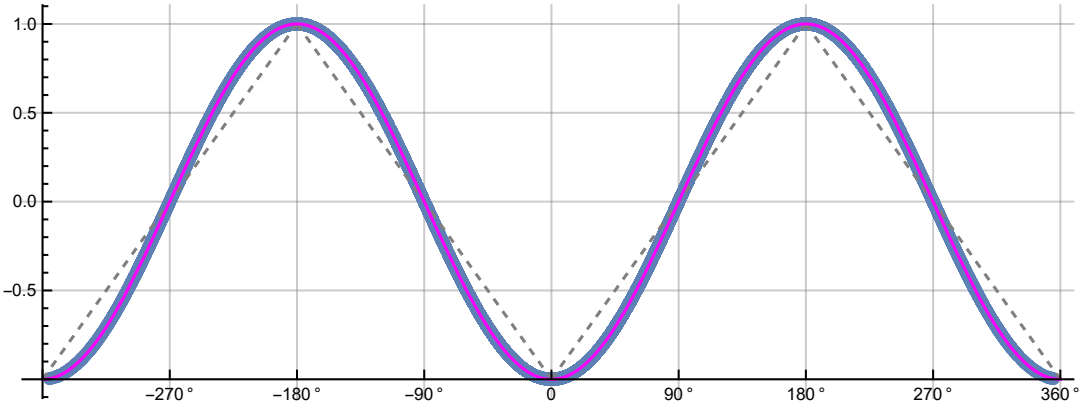
Verification of the Analytical 3-Sphere Model Product Calculation Prediction

In[ ]:=
```
(*qA=Re[Da**Limit[Ls1[[k]],Ls1[[k]]→Sign[Re[Da**Ls1[[k]]]]Da]];
qB=Re[Db**Limit[Ls2[[k]],Ls2[[k]]→Sign[Re[Db**Ls2[[k]]]]Db]];*)
(*The above two lines are moved to the independent A and B Do-loops
  and modified for proper calculation of the A and B outcomes.*)
Do[r1[[k]] = {qa1[[k]][[2]], qa1[[k]][[3]], qa1[[k]][[4]]};
 r2[[k]] = {qb1[[k]][[2]], qb1[[k]][[3]], qb1[[k]][[4]]};
 QAB[[k]] = Re[qa1[[k]]] * Re[qb1[[k]]] - r1[[k]].r2[[k]];
r0[[k]] = (Re[qa1[[k]]] Limit[Cross[s2, bb1[[k]]], s2 → Sign[Re[qb1[[k]]]] bb1[[k]]]
      + Re[qb1[[k]]] Limit[Cross[aa1[[k]], s1], s1 → Sign[Re[qa1[[k]]]] aa1[[k]]]
      - Cross[Limit[Cross[aa1[[k]], s1], s1 → Sign[Re[qa1[[k]]]] aa1[[k]]],
       Limit[Cross[s2, bb1[[k]]], s2 → Sign[Re[qb1[[k]]]] bb1[[k]]]) /
   (Sin[ArcCos[aa1[[k]].bb1[[k]]]]);
 q = {Re[QAB[[k]]], r0[[k]][[1]], r0[[k]][[2]], r0[[k]][[3]]}.Qcoordinates2;
 θ = a1[[k]] - b1[[k]];
 plotq[[k]] = {θ, Re[q]}, {k, m2}]
```

In[ ]:=
```
Print["Typical q = ", q];
meanq = Mean[plotq[[All, 2]]];
Print["Imaginary part vanishes. meanq = ", meanq];
sim1 = ListPlot[plotq, PlotMarkers → {Automatic, Small}, AspectRatio → 3 / 8,
   Ticks → {{{90, 90 °}, {-90, -90 °}, {-180, -180 °}, {180, 180 °}, {0, 0 °},
       {-270, -270 °}, {270, 270 °}, {-360, -360 °}, {360, 360 °}}, Automatic},
   GridLines → Automatic, AxesOrigin → {-360, -1.0}];
p1 = Plot[-1 + 2 x1 Degree / π, {x1, 0, 180}, PlotStyle → {Gray, Dashed}];
p2 = Plot[3 - 2 x2 Degree / π, {x2, 180, 360}, PlotStyle → {Gray, Dashed}];
p3 = Plot[3 + 2 x3 Degree / π, {x3, -360, -180}, PlotStyle → {Gray, Dashed}];
p4 = Plot[-1 - 2 x4 Degree / π, {x4, -180, 0}, PlotStyle → {Gray, Dashed}];
negcos1 = Plot[-Cos[x Degree], {x, -360, 360}, PlotStyle → {Magenta}];
Show[sim1, p1, p2, p3, p4, negcos1]
```

```
Typical q = Quaternion[0.819152, 0., 0., 0.]

Imaginary part vanishes. meanq = 0.00166173
```

Out[ ]=



Blue is the correlation data, magenta is the negative cosine curve for an exact match.

Statistical Analysis of the Particle Data Received from Alice and Bob

Spinorial Sign Change Corrections in A and B

For the spinorial sign change corrections we will need,

$$\mathbf{q}(\eta_{\mathbf{sn}} + \delta\,\pi,\, \mathbf{r}) = (-1)^{\delta}\,\mathbf{q}(\eta_{\mathbf{sn}},\, \mathbf{r}) \quad \text{for } \delta = 0, 1, 2, 3, \ldots$$

In[ ]:=
```
Do[If[λ2[[k]] == k && outqA[[k]][[2]] ≠ outqA[[k]][[3]], outqA3[[k]][[2]] = outqA[[k]][[2]] * -1];
 (*Spinorial sign change correction*)
 If[λ1[[k]] == k && outqB[[k]][[2]] ≠ outqB[[k]][[3]], outqB3[[k]][[2]] = outqB[[k]][[2]] * -1];
 (*Spinorial sign change correction*)
 If[λ2[[k]] == k && outqA[[k]][[2]] ≠ outqA[[k]][[3]], ssca[[k]] = 1, ssca[[k]] = 0];
 If[λ1[[k]] == k && outqB[[k]][[2]] ≠ outqB[[k]][[3]], sscb[[k]] = 1, sscb[[k]] = 0], {k, m}]
A = outqA3[[All, 2]];
B = outqB3[[All, 2]];
N[Total[ssca] / m] * 100 (*Percentage of spinorial sign changes.*)
N[Total[sscb] / m] * 100
```

Out[ ]= 3.31684

Out[ ]= 3.30958

In[ ]:=
```
Do[θ2 = a1[[k]] - b1[[k]] + 360;  (*Angles shifted by 360 since θ2 is an index*)
 aliceD = A[[k]]; bobD = B[[k]];
 If[aliceD == 1, nAP[[θ2]]++];
 If[bobD == 1, nBP[[θ2]]++];
 If[aliceD == 1 && bobD == 1, nPP[[θ2]]++];
 If[aliceD == 1 && bobD == -1, nPN[[θ2]]++];
 If[aliceD == -1 && bobD == 1, nNP[[θ2]]++];
 If[aliceD == -1 && bobD == -1, nNN[[θ2]]++], {k, m}]
```

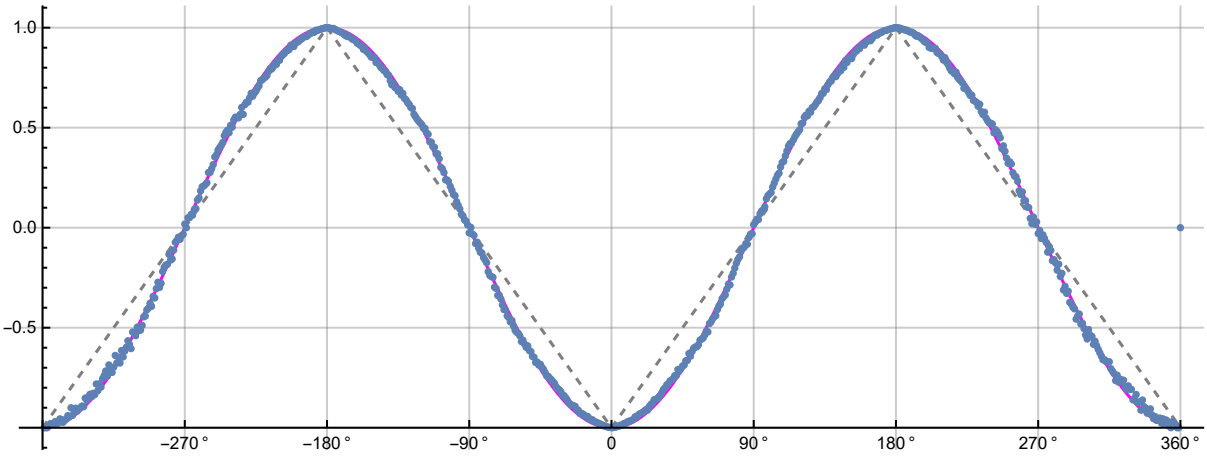
Calculating Mean Values of AB

In[ ]:=
```
mean = ConstantArray[0, trialDeg];
sum1 = ConstantArray[0, trialDeg];
sum2 = ConstantArray[0, trialDeg];
Do[sum1[[i]] = (nPP[[i]] + nNN[[i]] - nPN[[i]] - nNP[[i]]);
 sum2[[i]] = nPP[[i]] + nPN[[i]] + nNP[[i]] + nNN[[i]] + 0.0000001;
mean[[i]] = sum1[[i]] / sum2[[i]], {i, trialDeg}]
```

Plotting the Results Comparing Mean Values with -Cosine Curve

In[ ]:=
```
sim2 = ListPlot[mean, PlotMarkers → {Automatic, Tiny}];
negcos2 = Plot[-Cos[x7 Degree], {x7, 0, 720}, PlotStyle → {Magenta}, AspectRatio → 3 / 8,
   Ticks → {{{0, -360 °}, {90, -270 °}, {180, -180 °}, {270, -90 °}, {360, 0 °}, {450, 90 °},
      {540, 180 °}, {630, 270 °}, {720, 360 °}}, Automatic}, GridLines → Automatic,
   AxesOrigin → {0, -1.0}];
p5 = Plot[-5 + 2 x5 Degree / π, {x5, 360, 540}, PlotStyle → {Gray, Dashed}];
p6 = Plot[7 - 2 x6 Degree / π, {x6, 540, 720}, PlotStyle → {Gray, Dashed}];
Show[negcos2, p1, p2, p5, p6, sim2]
```

Out[ ]=



Computing Total Events Detected and Averages

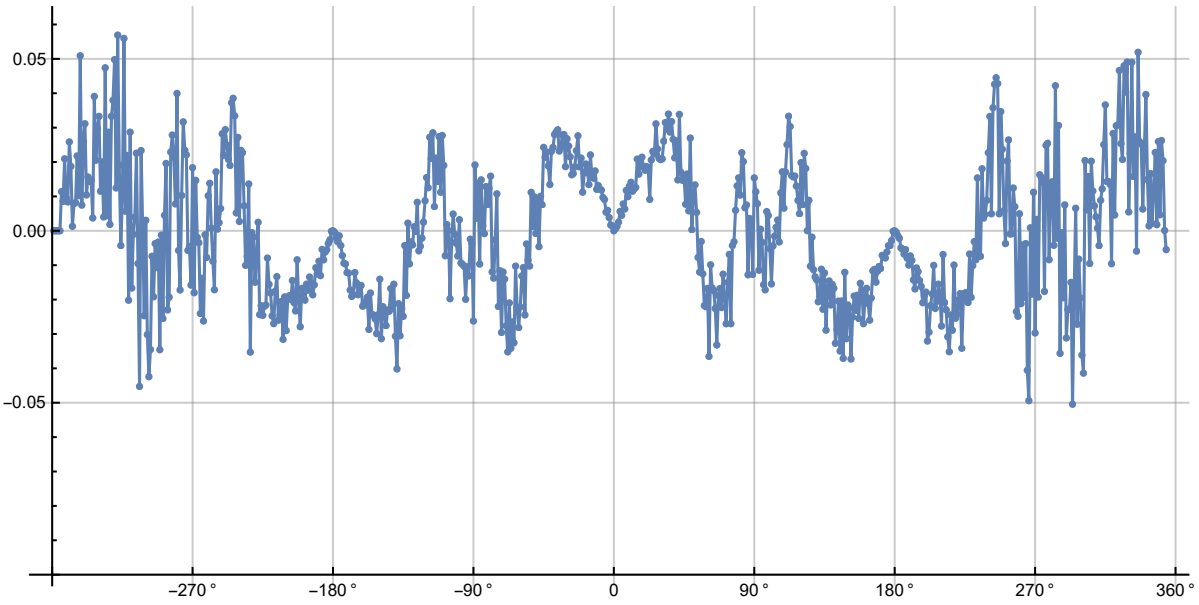
In[ ]:=
```
totAB = Total[nPP + nNN + nPN + nNP];
AveA = N[Total[A] / totAB];
AveB = N[Total[B] / totAB];
PAP = Total[nAP];
PBP = Total[nBP];
PA1 = N[PAP / totAB];
PB1 = N[PBP / totAB];
PP = N[Total[nPP] / totAB ];
NN = N[Total[nNN] / totAB ];
PN = N[Total[nPN] / totAB ];
NP = N[Total[nNP] / totAB ];
CHSH = Abs[N[mean[[315]] - N[mean[[225]] + N[mean[[405]] + N[mean[[45]]];
Print["Total Events Detected = ", totAB];
Print["AveA = ", AveA];
Print["AveB = ", AveB];
Print["P(A+) = ", PA1];
Print["P(B+) = ", PB1];
Print["P(++) = ", PP];
Print["P(--) = ", NN];
Print["P(+-) = ", PN];
Print["P(-+) = ", NP];
Print["Approx. CHSH = ", CHSH];
```

```
Total Events Detected = 5 000 000

AveA = 0.00025

AveB = -0.0000116

P(A+) = 0.500125

P(B+) = 0.499994

P(++) = 0.249842

P(--) = 0.249723

P(+-) = 0.250283

P(-+) = 0.250152

Approx. CHSH = 2.75407
```

Computing Deviation from Negative Cosine Curve

In[ ]:=
```
devang = ConstantArray[0, 714];
dev = ConstantArray[0, 714];
Do[devang[[i]] = i;
dev[[i]] = mean[[i]] + Cos[devang[[i]] Degree], {i, 6, 714}]
ListPlot[N[dev], PlotMarkers → {Automatic, Tiny}, Joined → True,
 AspectRatio → 8 / 16, Ticks → {{{0, -360 °}, {90, -270 °}, {180, -180 °}, {270, -90 °},
    {360, 0 °}, {450, 90 °}, {540, 180 °}, {630, 270 °}, {720, 360 °}}, Automatic},
 GridLines → Automatic, AxesOrigin → {0, -0.1}]
```

Out[ ]=



In[ ]:=
```
Mean[N[dev]] // PercentForm
Mean[N[Abs[dev]]]
```

Out[ ]//PercentForm=
```
-0.01131%
```

Out[ ]=
```
0.016433
```

In[ ]:=
```
-0.000168243; 0.0218532; (*deviations, 1 million events*)
```