Simulation Based on Michel Fodje's epr-simple simulation translated from Python to Mathematica by John Reed 13 Nov 2013 Plus Quaternions
Modified by Fred Diether for Completely Local-Realistic Dec. 2021
Includes Joy's S^3 Quaternion Model. With 3D Vectors!

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

In[ ]:=
```
<< 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 = 1000000;
trialDeg = 361;
Ls1 = ConstantArray[0, m];
Ls2 = ConstantArray[0, m];
λ1 = ConstantArray[0, m];
λ2 = ConstantArray[0, m];
Da1 = ConstantArray[0, m];
Db1 = ConstantArray[0, m];
qA = ConstantArray[0, m];
qB = ConstantArray[0, m];
qa = ConstantArray[0, m];
qb = ConstantArray[0, m];
aq1 = ConstantArray[0, m];
qa1 = ConstantArray[0, m];
bq1 = ConstantArray[0, m];
qb1 = ConstantArray[0, m];
aa1 = ConstantArray[0, m];
bb1 = ConstantArray[0, m];
outA1 = Table[{0, 0}, m];
outA2 = Table[{0, 0}, m];
outB1 = Table[{0, 0}, m];
outB2 = Table[{0, 0}, m];
a1 = ConstantArray[0, m];
b1 = 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];
nAN = ConstantArray[0, trialDeg];
nBN = ConstantArray[0, trialDeg];
ϕ = 3; β = 0.284; ξ = 0.892; (*Adustable parameters for fine tuning*)
```

Generating Particle Data with Three Independent Do-Loops

```
In[•]:= Do[s = RandomPoint[Sphere[]];  (*Singlet 3D vector*)  (*Hidden Variable*)
 Θ1 = ToSphericalCoordinates[s][[3]] * 180 / π;
 Θ2 = ToSphericalCoordinates[s][[2]];
 λ1[[i]] = β (Cos[Θ1 / ϕ]^2);
 λ2[[i]] = (Cos[Θ2 * ξ / 2]^2);
 Ls1[[i]] = s.Qcoordinates;  (*Convert to quaternion coordinates*)
 Ls2[[i]] = -s.Qcoordinates, {i, m}]
```

```
In[•]:= Do[a = RandomPoint[Sphere[]];  (*Detector 3D vector angle*)
 aa1[[i]] = a;
 Da = a.Qcoordinates;  (*Convert to quaternion coordinates*)
 Da1[[i]] = Da;
 qa = Da ** Ls1[[i]];
 qa1[[i]] = qa;
 aq = -Da ** Ls1[[i]];
 aq1[[i]] = aq;
 If[Abs[Re[qa]] > λ1[[i]],
  qA1 = Re[Da ** Limit[Ls1[[i]], Ls1[[i]] → Sign[Re[Da ** Ls1[[i]]]] Da]],
  qA1 = Sign[{aq[[2]], aq[[3]], aq[[4]]}.Qcoordinates]];
 outA1[[i]] = {a, qA1};
 If[Abs[Re[qa]] > λ2[[i]],
  qA2 = Re[-Da ** Limit[Ls1[[i]], Ls1[[i]] → Sign[Re[Da ** Ls1[[i]]]] Da]],
  qA2 = Sign[{qa[[2]], qa[[3]], qa[[4]]}.Qcoordinates]];
 outA2[[i]] = {a, qA2}, {i, m}]
outA = Catenate[{outA1, outA2}];
```

```
In[•]:= Do[b = RandomPoint[Sphere[]];  (*Detector 3D vector angle*)
 bb1[[i]] = b;
 Db = b.Qcoordinates;  (*Convert to quaternion coordinates*)
 Db1[[i]] = Db;
 qb = Ls2[[i]] ** Db;
 qb1[[i]] = qb;
 bq = -Ls2[[i]] ** Db;
 bq1[[i]] = bq;
 If[Abs[Re[qb]] > λ1[[i]],
  qB1 = Re[Db ** Limit[Ls2[[i]], Ls2[[i]] → Sign[Re[Db ** Ls2[[i]]]] Db]],
  qB1 = Sign[{bq[[2]], bq[[3]], bq[[4]]}.Qcoordinates]];
 outB1[[i]] = {b, qB1};
 If[Abs[Re[qb]] > λ2[[i]],
  qB2 = Re[-Db ** Limit[Ls2[[i]], Ls2[[i]] → Sign[Re[Db ** Ls2[[i]]]] Db]],
  qB2 = Sign[{qb[[2]], qb[[3]], qb[[4]]}.Qcoordinates]];
 outB2[[i]] = {b, qB2}, {i, m}]
outB = Catenate[{outB1, outB2}];
```

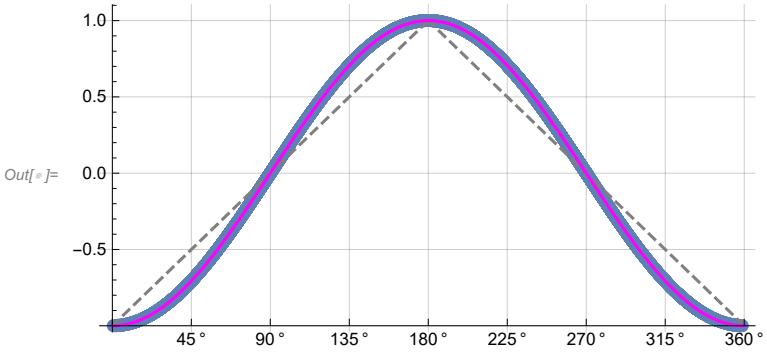
Verification of the Analytical 3-Sphere Model Based on Geometric Algebra using Quaternions

In[ ]:=
```
q = 0; t = 0;
m3 = 20000;
DA = Take[Da1, m3];
DB = Take[Db1, m3];
Ls11 = Take[Ls1, m3];
Ls22 = Take[Ls2, m3];
QA = DA ** Ls11;
QB = Ls22 ** DB;
r0 = ConstantArray[0, m3];
r1 = ConstantArray[0, m3];
r2 = ConstantArray[0, m3];
QAB = ConstantArray[0, m3];
plotq = Table[{0, 0}, m3];
(*qA=Re[DA**Limit[Ls11[[i]],Ls11[[i]]→Sign[Re[DA**Ls11[[i]]]]DA]];
qB=Re[DB**Limit[Ls22[[i]],Ls22[[i]]→Sign[Re[DB[[i]]**Ls22[[i]]]]DB]];*)
(*These two lines moved to the A and B Do-loops
  for further proper local processing*)
```

In[ ]:=
```
Do[r1[[i]] = {QA[[i]][[2]], QA[[i]][[3]], QA[[i]][[4]]};
 r2[[i]] = {QB[[i]][[2]], QB[[i]][[3]], QB[[i]][[4]]};
 QAB[[i]] = Re[QA[[i]]] * Re[QB[[i]]] - r1[[i]].r2[[i]];
 r0[[i]] = (Re[qa1[[i]]] Limit[Cross[s2, bb1[[i]]], s2 → Sign[Re[qb1[[i]]]] bb1[[i]]] +
     Re[qb1[[i]]] Limit[Cross[aa1[[i]], s1], s1 → Sign[Re[qa1[[i]]]] aa1[[i]]] -
     Limit[Cross[aa1[[i]], s1], s1 → Sign[Re[qa1[[i]]]] aa1[[i]]] *
      Limit[Cross[s2, bb1[[i]]], s2 → Sign[Re[qb1[[i]]]] bb1[[i]]]) /
   (Sin[ArcCos[aa1[[i]].bb1[[i]]]]);
 q = {QAB[[i]], r0[[i]][[1]], r0[[i]][[2]], r0[[i]][[3]]}.Qcoordinates2;
 ϕA = ArcTan[aa1[[i]][[1]], aa1[[i]][[2]]] / 50;
 ϕB = ArcTan[bb1[[i]][[2]], bb1[[i]][[1]]] / 50;
 If[ϕA * ϕB > 0, angleΘ = ArcCos[aa1[[i]].bb1[[i]]] * 180 / π,
  angleΘ = (2 π - ArcCos[aa1[[i]].bb1[[i]]]) * 180 / π];
 t = t + q[[1]];
 plotq[[i]] = {angleΘ, q[[1]]}, {i, m3}]
Meanq = t / m; (*shows vanishing of the non-real part iJK*)
Print["Meanq = ", Meanq]
qsim = ListPlot[plotq, PlotMarkers → {Automatic, Small},
   AspectRatio → 8 / 16, Ticks → {{{0, 0 °}, {45, 45 °}, {90, 90 °}, {135, 135 °},
       {180, 180 °}, {225, 225 °}, {270, 270 °}, {315, 315 °}, {360, 360 °}}, Automatic},
   GridLines → Automatic, AxesOrigin → {0, -1.0}];
p1 = Plot[-1 + 2 x Degree / π, {x, 0, 180}, PlotStyle → {Gray, Dashed}];
p2 = Plot[3 - 2 x Degree / π, {x, 180, 360}, PlotStyle → {Gray, Dashed}];
negcos1 = Plot[-Cos[x Degree], {x, 0, 360}, PlotStyle → {Magenta}];
Show[qsim, p1, p2, negcos1]
```

```
Meanq = -0.0000566344
```

Out[ ]=



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

Statistical Analysis of the Particle Data Received from Alice and Bob

```
In[•]:= m2 = 2 m;
theta = ConstantArray[0, m2];
th = ConstantArray[0, m2];
a1 = outA[[All, 1]];
qA = outA[[All, 2]];
b1 = outB[[All, 1]];
qB = outB[[All, 2]];
Do[ϕA1 = ArcTan[a1[[i]][[1]], a1[[i]][[2]]] / 50;
 ϕB1 = ArcTan[b1[[i]][[2]], b1[[i]][[1]]] / 50;
 If[ϕA1 * ϕB1 > 0, th[[i]] = ArcCos[a1[[i]].b1[[i]]],
  th[[i]] = 2 π - ArcCos[a1[[i]].b1[[i]]]];
 theta[[i]] = Round[th[[i]] * 180 / π] + 1;
 Θ = theta[[i]];
 aliceD = qA[[i]]; bobD = qB[[i]];
 If[aliceD == 1, nAP[[Θ]] ++];
 If[bobD == 1, nBP[[Θ]] ++];
 If[aliceD == -1, nAN[[Θ]] ++];
 If[bobD == -1, nBN[[Θ]] ++];
 If[aliceD == 1 && bobD == 1, nPP[[Θ]] ++];
 If[aliceD == 1 && bobD == -1, nPN[[Θ]] ++];
 If[aliceD == -1 && bobD == 1, nNP[[Θ]] ++];
 If[aliceD == -1 && bobD == -1, nNN[[Θ]] ++], {i, m2}]
```

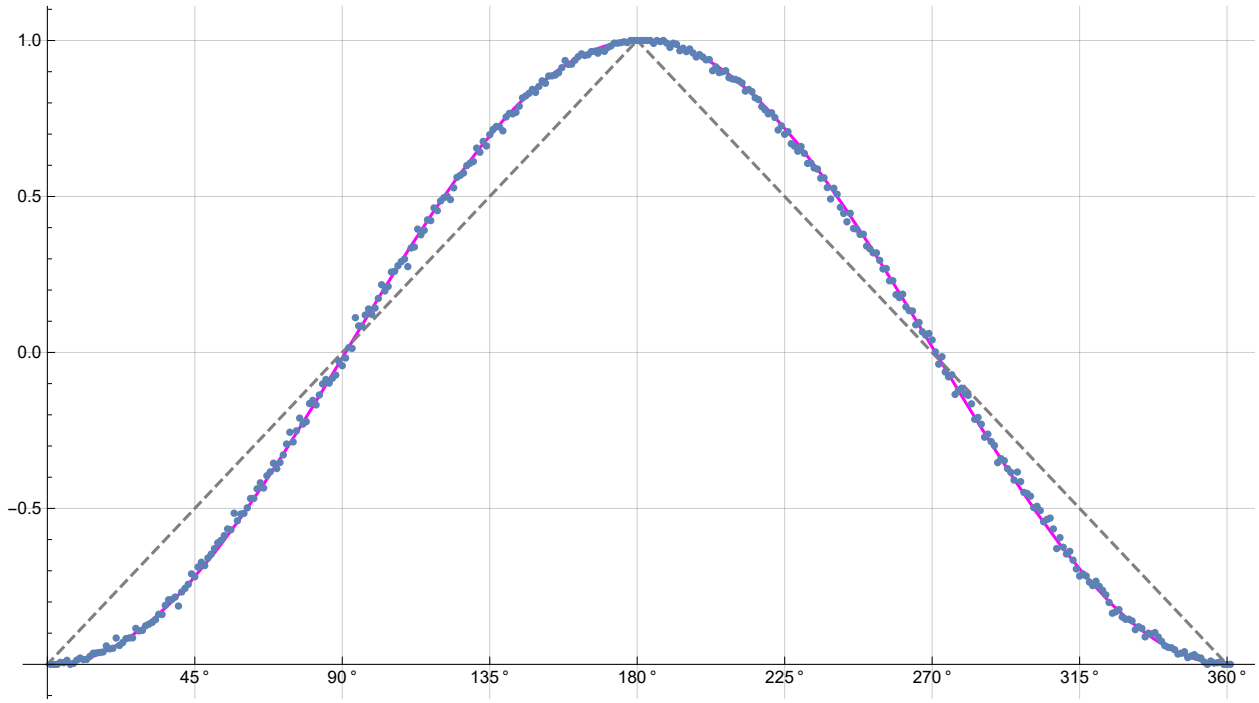
Calculating Mean Values of AB

```
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 and Comparing Mean Values with -Cosine Function

In[ ]:=
```
simulation = ListPlot[mean, PlotMarkers → {Automatic, Tiny}];
negcos =
  Plot[-Cos[x Degree - 1 Degree], {x, 0, 361}, PlotStyle → {Magenta}, AspectRatio → 9/16, Ticks →
    {{{0, 0 °}, {45, 45 °}, {90, 90 °}, {135, 135 °}, {180, 180 °}, {225, 225 °}, {270, 270 °},
      {315, 315 °}, {360, 360 °}}, Automatic}, GridLines → Automatic, AxesOrigin → {0, -1.0}];
p1 = Plot[-1 + 2 x Degree / π, {x, 0, 180}, PlotStyle → {Gray, Dashed}];
p2 = Plot[3 - 2 x Degree / π, {x, 180, 360}, PlotStyle → {Gray, Dashed}];
Show[negcos, p1, p2, simulation]
```

Out[ ]=



Computing Averages

```
In[•]:= A1 = ConstantArray[0, m2];
        B1 = ConstantArray[0, m2];
        Do[If[qA[[i]] == 1 || qA[[i]] == -1, A1[[i]] = qA[[i]]];
         If[qB[[i]] == 1 || qB[[i]] == -1, B1[[i]] = qB[[i]]], {i, m2}]
        AveA = N[Sum[A1[[i]], {i, m2}] / m2];
        AveB = N[Sum[B1[[i]], {i, m2}] / m2];
        Print["AveA = ", AveA];
        Print["AveB = ", AveB];
        PAP = N[Sum[nAP[[i]], {i, trialDeg}]];
        PBP = N[Sum[nBP[[i]], {i, trialDeg}]];
        PAN = N[Sum[nAN[[i]], {i, trialDeg}]];
        PBN = N[Sum[nBN[[i]], {i, trialDeg}]];
        PA1 = PAP / (PAP + PAN);
        PB1 = PBP / (PBP + PBN);
        Print["P(A+)= ", PA1]
        Print["P(B+)= ", PB1]
        totAB = Sum[nPP[[i]] + nNN[[i]] + nPN[[i]] + nNP[[i]], {i, trialDeg}];
        Print["Total Events = ", totAB]
        PP = N[Sum[nPP[[i]], {i, trialDeg}] / totAB];
        NN = N[Sum[nNN[[i]], {i, trialDeg}] / totAB];
        PN = N[Sum[nPN[[i]], {i, trialDeg}] / totAB];
        NP = N[Sum[nNP[[i]], {i, trialDeg}] / totAB];
        totP = PP + NN + PN + NP;
        Print["Ave ++ = ", PP]
        Print["Ave -- = ", NN]
        Print["Ave +- = ", PN]
        Print["Ave -+ = ", NP]
        CHSH = Abs[N[mean[[23]]] - N[mean[[135]]] + N[mean[[68]]] + N[mean[[45]]]];
        Print["Approx. CHSH = ", CHSH]

        AveA = -0.0001275

        AveB = 0.0001895

        P(A+)= 0.499901

        P(B+)= 0.500147

        Total Events = 998425

        Ave ++ = 0.250175

        Ave -- = 0.250195

        Ave +- = 0.249914

        Ave -+ = 0.249716

        Approx. CHSH = 2.73016
```

$$\text{In[•]:=}\quad E_{ab} = \text{TrigReduce}\left[\frac{\text{Sin}\left[(\eta_{ab})/2\right]^2}{2} + \frac{\text{Sin}\left[(\eta_{ab})/2\right]^2}{2} - \frac{\text{Cos}\left[(\eta_{ab})/2\right]^2}{2} - \frac{\text{Cos}\left[(\eta_{ab})/2\right]^2}{2}\right] \Big/ \text{totP};$$

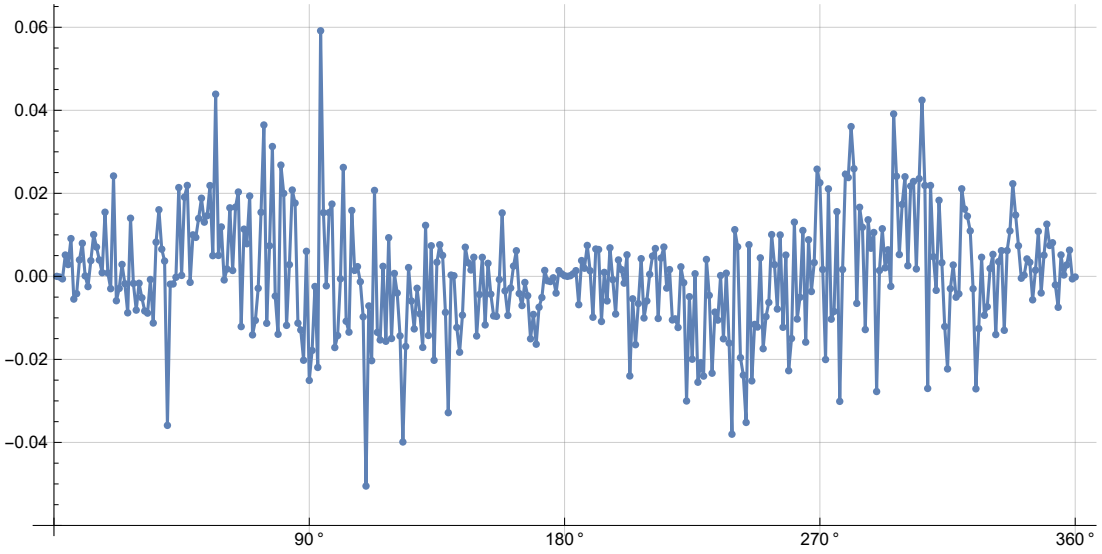
```
        Print["E(a, b) = ", Eab]

        E(a, b) = -1. Cos[ηab]
```

Calculating Deviation from -Cosine Curve

```
dev1 = ConstantArray[2, 360];
dev2 = ConstantArray[2, 360];
dev3 = ConstantArray[2, 360];
Do[dev1 = mean[[i]];
 dev2[[i]] = {dev1, i}, {i, 360}]
devang = dev2[[All, 2]];
Do[dev3[[i]] = mean[[i]] + Cos[devang[[i]] Degree - 1 Degree], {i, 360}]
ListPlot[N[dev3], PlotMarkers → {Automatic, Tiny}, Joined → True, AspectRatio → 1 / 2,
 Ticks → {{{0, 0 °}, {90, 90 °}, {180, 180 °}, {270, 270 °}, {360, 360 °}}, Automatic},
 GridLines → Automatic, AxesOrigin → {0, -0.06}]
```

Out[ ]=



In[ ]:=
```
N[Mean[Abs[dev3]]]
N[Mean[dev3]]
```

Out[ ]= 0.0104776

Out[ ]= 0.000101834