

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 Using Joy's S^3 Quaternion Model. With 3D Vectors!

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

```
In[780]:= << 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};  
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; (*Adjustable parameters for fine tuning*)
```

Generating Particle Data with Three Independent Do-Loops

```
In[819]:= Do[s = RandomPoint[Sphere[]]; (*Singlet 3D vector*) (*Hidden Variable*)
   $\theta_1 = \text{ToSphericalCoordinates}[s][[3]] * 180 / \pi;$ 
   $\theta_2 = \text{ToSphericalCoordinates}[s][[2]];$ 
   $\lambda_1[[i]] = \beta \left( \cos\left[\frac{\theta_1}{\phi}\right]^2 \right);$ 
   $\lambda_2[[i]] = \left( \cos\left[\frac{\theta_2 * \xi}{2}\right]^2 \right);$ 
  Ls1[[i]] = s.Qcoordinates; (*Convert to quaternion coordinates*)
  Ls2[[i]] = -s.Qcoordinates, {i, m}]
```

```
In[820]:= 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]] >  $\lambda_1[[i]]$ ,
    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]] >  $\lambda_2[[i]]$ ,
    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}];
```

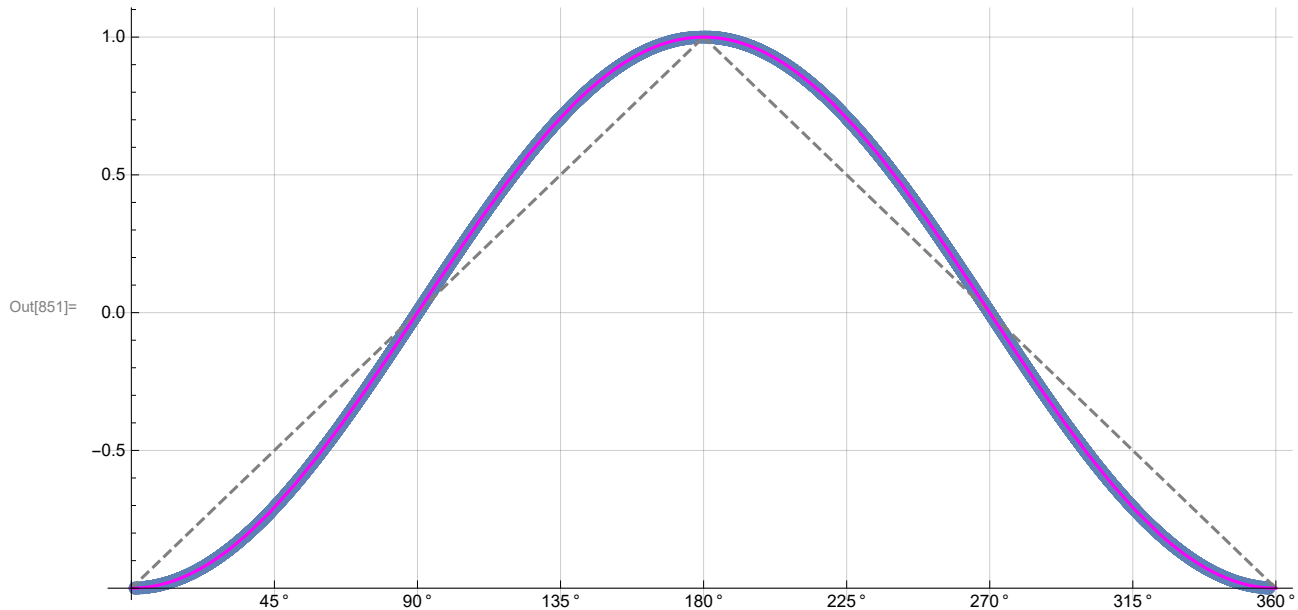
```
In[822]:= Do[b = RandomPoint[Sphere[]]; (*Detector 3D vector angle*)
  bb1[[i]] = b;
  Db = b.Qcoordinates; (*Convert to quaternion coordinates*)
  Db1[[i]] = Db;
  qb = Ls2[[i]] ** Db;
  qb1[[i]] = qb;
  bq = -Ls2[[i]] ** Db;
  bq1[[i]] = bq;
  If[Abs[Re[qb]] >  $\lambda_1[[i]]$ ,
    qB1 = Re[Db ** 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]] >  $\lambda_2[[i]]$ ,
    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}];
```

Product Calculation Verification

```

In[824]:= m3 = 20000;
DA = Take[Da1, m3];
DB = Take[Db1, m3];
Ls11 = Take[Ls1, m3];
Ls22 = Take[Ls2, m3];
outA11 = Table[{0, 0}, m3];
outA22 = Table[{0, 0}, m3];
outB11 = Table[{0, 0}, m3];
outB22 = Table[{0, 0}, m3];
rab1 = ConstantArray[0, m3];
rab2 = ConstantArray[0, m3];
angleθ = ConstantArray[0, m3 * 2];
AB6 = ConstantArray[0, m3 * 2];
AB7 = Table[{0, 0}, m3 * 2];
Do[If[Abs[Re[qa1[[i]]]] > λ1[[i]], qa11 = Re[DA[[i]] ** Ls11[[i]]],
  qa11 = Sign[{aq1[[i]][[2]], aq1[[i]][[3]], aq1[[i]][[4]]}.Qcoordinates]];
outA11[[i]] = {aa1[[i]], qa11};
If[Abs[Re[qa1[[i]]]] > λ2[[i]], qa22 = Re[-DA[[i]] ** Ls11[[i]]],
  qa22 = Sign[{qa1[[i]][[2]], qa1[[i]][[3]], qa1[[i]][[4]]}.Qcoordinates]];
outA22[[i]] = {aa1[[i]], qa22};
If[Abs[Re[qb1[[i]]]] > λ1[[i]], qb11 = Re[DB[[i]] ** Ls22[[i]]],
  qb11 = Sign[{bq1[[i]][[2]], bq1[[i]][[3]], bq1[[i]][[4]]}.Qcoordinates]];
outB11[[i]] = {bb1[[i]], qb11};
If[Abs[Re[qb1[[i]]]] > λ2[[i]], qb22 = Re[-DB[[i]] ** Ls22[[i]]],
  qb22 = Sign[{qb1[[i]][[2]], qb1[[i]][[3]], qb1[[i]][[4]]}.Qcoordinates]];
outB22[[i]] = {bb1[[i]], qb22};
rab2[[i]] = {aq1[[i]][[2]], aq1[[i]][[3]], aq1[[i]][[4]]}.
  {bq1[[i]][[2]], bq1[[i]][[3]], bq1[[i]][[4]]};
rab1[[i]] = {qa1[[i]][[2]], qa1[[i]][[3]], qa1[[i]][[4]]}.
  {qb1[[i]][[2]], qb1[[i]][[3]], qb1[[i]][[4]]}, {i, m3}
outAq = Catenate[{outA11, outA22}];
outBq = Catenate[{outB11, outB22}];
rab = Catenate[{rab1, rab2}];
a6 = outAq[[All, 1]];
b6 = outBq[[All, 1]];
A6 = outAq[[All, 2]];
B6 = outBq[[All, 2]];
Do[φA1 = ArcTan[a6[[i]][[1]], a6[[i]][[2]]] / 50;
  φB1 = ArcTan[b6[[i]][[2]], b6[[i]][[1]]] / 50;
  If[φA1 * φB1 > 0, angleθ[[i]] = ArcCos[a6[[i]].b6[[i]] * 180 / π,
    angleθ[[i]] = (2 π - ArcCos[a6[[i]].b6[[i]]) * 180 / π];
  AB6[[i]] = A6[[i]] * B6[[i]] - rab[[i]];
  AB7[[i]] = {angleθ[[i]], AB6[[i]]}, {i, m3 * 2}
qsim2 = ListPlot[AB7, 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}];
negcos2 = Plot[-Cos[x Degree], {x, 0, 360}, PlotStyle → {Magenta}];
Show[qsim2, p1, p2, negcos2]

```



Blue is data and magenta is the -cosine curve for an exact match!

Statistical Analysis of the Particle Data Received from Alice and Bob

```
In[852]:= 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[phiA1 = ArcTan[a1[[i]][[1]], a1[[i]][[2]]] / 50;
phiB1 = ArcTan[b1[[i]][[2]], b1[[i]][[1]]] / 50;
If[phiA1 * phiB1 > 0, th[[i]] = ArcCos[a1[[i]].b1[[i]]],
th[[i]] = 2 pi - ArcCos[a1[[i]].b1[[i]]];
theta[[i]] = Round[th[[i]] * 180 / pi] + 1;
theta = theta[[i]];
aliceD = qA[[i]]; bobD = qB[[i]];
If[aliceD == 1, nAP[[theta]] ++];
If[bobD == 1, nBP[[theta]] ++];
If[aliceD == -1, nAN[[theta]] ++];
If[bobD == -1, nBN[[theta]] ++];
If[aliceD == 1 && bobD == 1, nPP[[theta]] ++];
If[aliceD == 1 && bobD == -1, nPN[[theta]] ++];
If[aliceD == -1 && bobD == 1, nNP[[theta]] ++];
If[aliceD == -1 && bobD == -1, nNN[[theta]] ++], {i, m2}]
```

Calculating Mean Values of AB

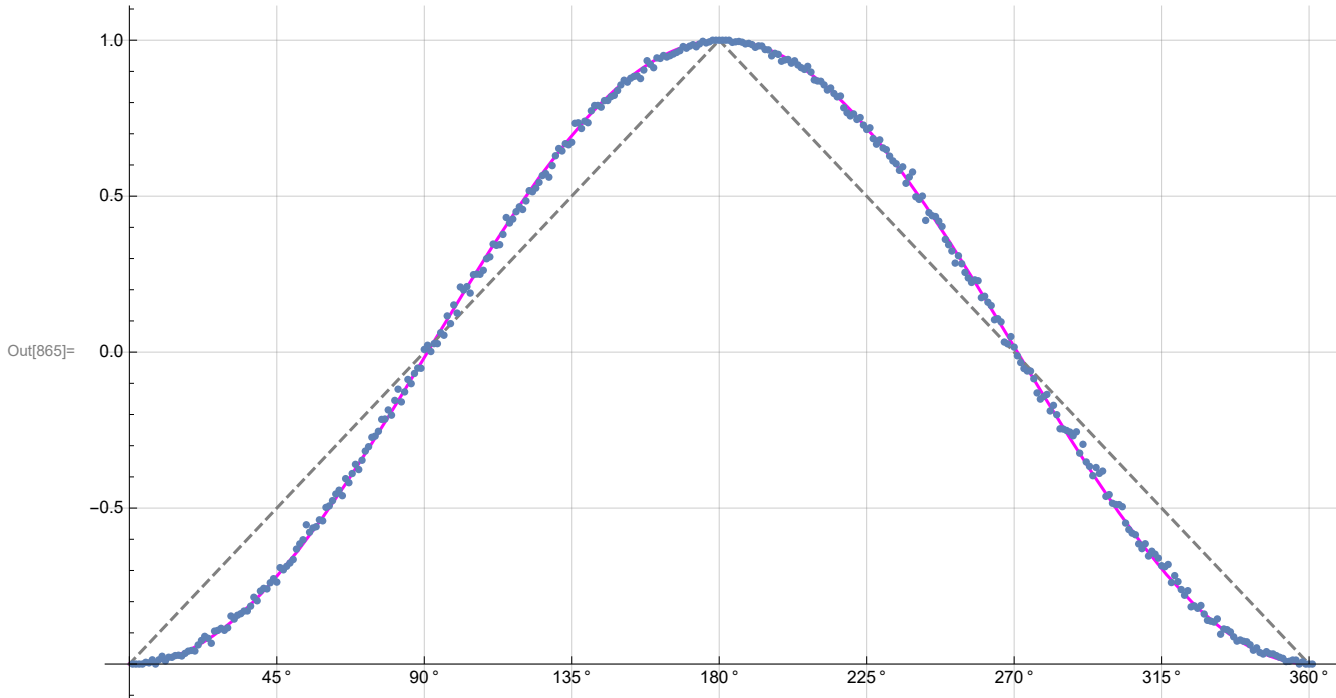
```
In[857]:= mean = ConstantArray[0, trialDeg];
sum1 = ConstantArray[0, trialDeg];
sum2 = ConstantArray[0, trialDeg];
Do[sum1[[i]] = (nPP[[i]] + nNN[[i]] - nPN[[i]] - nNP[[i]]);
sum2[[i]] = nPP[[i]] + nPN[[i]] + nNP[[i]] + nNN[[i]] + 0.0000001;
mean[[i]] = sum1[[i]] / sum2[[i]], {i, trialDeg}]
```

Plotting the Results and Comparing Mean Values with -Cosine Function

```

In[861]:= 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]

```



Computing Averages

```

In[866]:= 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 = 0.000088
AveB = 0.000116
P(A+) = 0.500068
P(B+) = 0.50009
Total Events = 999260
Ave ++ = 0.249941
Ave -- = 0.249819
Ave +- = 0.250161
Ave -+ = 0.250079
Approx. CHSH = 2.71032

In[894]:= Eab = TrigReduce[
$$\frac{\sin[(\eta_{ab})/2]^2}{2} + \frac{\sin[(\eta_{ab})/2]^2}{2} - \frac{\cos[(\eta_{ab})/2]^2}{2} - \frac{\cos[(\eta_{ab})/2]^2}{2}] / \text{totP};
Print["E(a, b) = ", Eab];
E(a, b) = -1. Cos[\eta_{ab}]$$

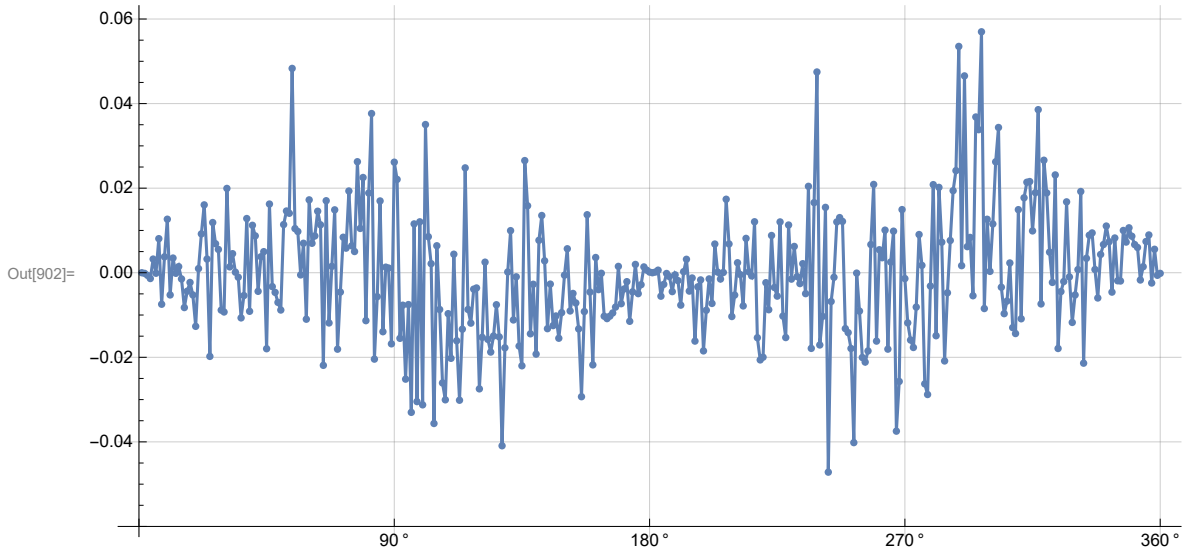
```

Calculating Deviation from -Cosine Curve

```

In[896]:= 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}]

```



```

In[903]:= N[Mean[Abs[dev3]]]
N[Mean[dev3]]

```

```
Out[903]= 0.0111951
```

```
Out[904]= -0.0003332
```