

**Simulation Based on Joy Christian's original 3-Sphere Model.  
Parts of Quaternion Code by John Reed.  
Modified by Fred Diether Dec. 2021. With 3D Vectors!**

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

```
In[963]:= << 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 = 30000;  
trialDeg = 361;  
Ls1 = ConstantArray[0, m];  
Ls2 = ConstantArray[0, m];  
Da1 = ConstantArray[0, m];  
Db1 = ConstantArray[0, m];  
qa1 = ConstantArray[0, m];  
qb1 = ConstantArray[0, m];  
outA = Table[{0, 0}, m];  
outB = Table[{0, 0}, m];  
a1 = ConstantArray[0, m];  
b1 = ConstantArray[0, m];
```

**Generating Particle Data with Three Independent Do-Loops**

```
In[982]:= Do[s = RandomPoint[Sphere[]]; (*Singlet 3D vector*) (*Hidden Variable*)  
Ls1[[i]] = s.Qcoordinates; (*Convert to quaternion coordinates*)  
Ls2[[i]] = -s.Qcoordinates, {i, m}]
```

```
In[983]:= Do[a = RandomPoint[Sphere[]]; (*Detector 3D vector angle*)  
a1[[i]] = a;  
Da = a.Qcoordinates; (*Convert to quaternion coordinates*)  
Da1[[i]] = Da;  
qa = Da ** Ls1[[i]];  
qa1[[i]] = qa;  
qA = Re[Da ** Limit[Ls1[[i]], Ls1[[i]] → Sign[Re[Da ** Ls1[[i]]]] Da]];  
outA[[i]] = {a, qA}, {i, m}]
```

```
In[984]:= Do[b = RandomPoint[Sphere[]]; (*Detector 3D vector angle*)  
b1[[i]] = b;  
Db = b.Qcoordinates; (*Convert to quaternion coordinates*)  
Db1[[i]] = Db;  
qb = Ls2[[i]] ** Db;  
qb1[[i]] = qb;  
qB = Re[Db ** Limit[Ls2[[i]], Ls2[[i]] → Sign[Re[Db ** Ls2[[i]]]] Db]];  
outB[[i]] = {b, qB}, {i, m}]
```

**VERIFICATION OF THE ANALYTICAL 3-SPHERE MODEL BASED ON GEOMETRIC ALGEBRA  
USING QUATERNIONS**

```

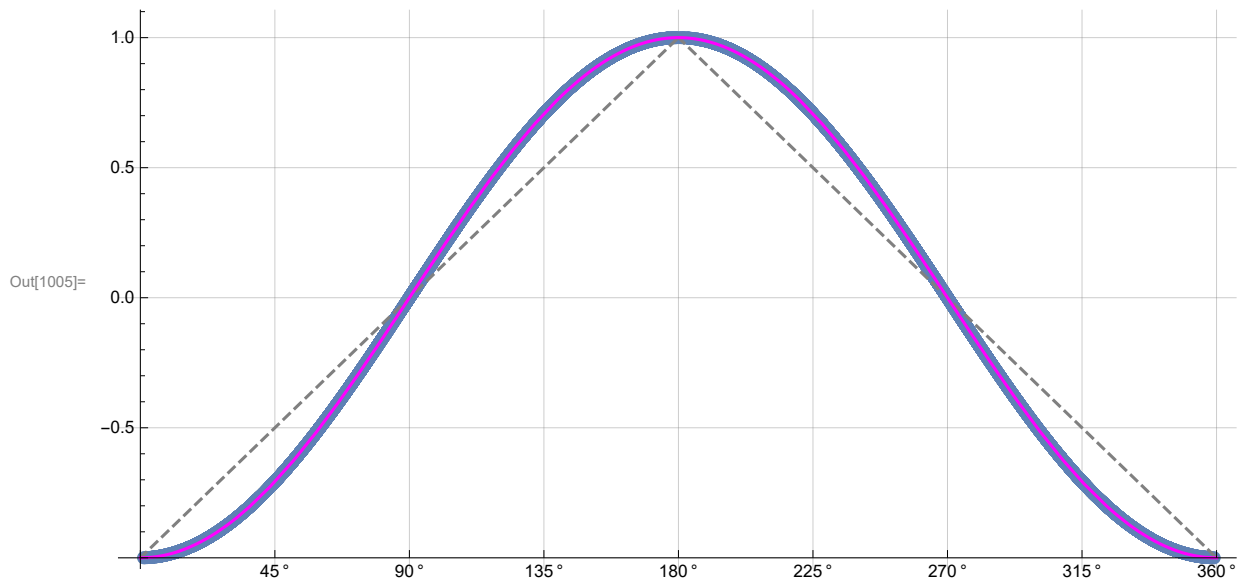
In[985]:= q = 0; t = 0; u = 0;
r0 = ConstantArray[0, m];
r1 = ConstantArray[0, m];
r2 = ConstantArray[0, m];
QA1 = ConstantArray[0, m];
QB1 = ConstantArray[0, m];
QAB = ConstantArray[0, m];
θ = ConstantArray[0, m];
plotq = Table[{0, 0}, m];
A = outA[[All, 2]];
B = outB[[All, 2]];

In[996]:= Do[r1[[i]] = ({qa1[[i]][[2]], qa1[[i]][[3]], qa1[[i]][[4]]});
r2[[i]] = ({qb1[[i]][[2]], qb1[[i]][[3]], qb1[[i]][[4]]});
QA1[[i]] = Flatten[{Re[qa1[[i]]], r1[[i]]}.Qcoordinates2;
QB1[[i]] = Flatten[{Re[qb1[[i]]], r2[[i]]}.Qcoordinates2;
QAB[[i]] = Re[QA1[[i]] ** QB1[[i]]];
r0[[i]] = (Re[qa1[[i]]] Limit[Cross[s2, b1[[i]]], s2 → Sign[Re[qb1[[i]]]] b1[[i]] +
Re[qb1[[i]]] Limit[Cross[a1[[i]], s1], s1 → Sign[Re[qa1[[i]]]] a1[[i]] -
Limit[Cross[a1[[i]], s1], s1 → Sign[Re[qa1[[i]]]] a1[[i]] *
Limit[Cross[s2, b1[[i]]], s2 → Sign[Re[qb1[[i]]]] b1[[i]]) /
(Sin[ArcCos[a1[[i]].b1[[i]]]));
q = {Re[QAB[[i]]], r0[[i]][[1]], r0[[i]][[2]], r0[[i]][[3]].Qcoordinates2;
t = t + A[[i]];
u = u + B[[i]];
AveA = t / m;
AveB = u / m;
φA = ArcTan[a1[[i]][[1]], a1[[i]][[2]]] / 50;
φB = ArcTan[b1[[i]][[2]], b1[[i]][[1]]] / 50;
If[φA * φB > 0, θ = ArcCos[a1[[i]].b1[[i]]] * 180 / π,
θ = (2 π - ArcCos[a1[[i]].b1[[i]]) * 180 / π];
plotq[[i]] = {θ, Re[q]}, {i, m}
Print["<A> = ", AveA]
Print["<B> = ", AveB]
meanq = Mean[plotq[[All, 2]]];
Print["Imaginary part vanishes. meanq = ", meanq]
sim = 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}];
negcos = Plot[-Cos[x Degree], {x, 0, 360}, PlotStyle → {Magenta}];
Show[sim, p1, p2, negcos]

<A> = -0.00593333
<B> = -0.00113333

Imaginary part vanishes. meanq = -0.00518224

```



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