Fit parameters

Setup equations

```
Off[General::spell1];
Off[General::spell];

<< Graphics`

<< Statistics`NonlinearFit`

<< Statistics`DataManipulation`

SetDirectory["c:/cygwin/home/meister/Development/cando/src/tools"];

RT = 0.002 * 300.0;

BINS = 41;

rotationMatrixX[a_] := {
   { 1.0, 0.0, 0.0, 0.0 },
   { 0.0, Cos[a], Sin[a], 0.0 },
   {0.0, -Sin[a], Cos[a], 0.0},
   {0.0, 0.0, 0.0, 1.0 } };

rotationMatrixY[a_] := {
   {Cos[a], 0.0, -Sin[a], 0.0 },
   {0.0, 1.0, 0.0, 0.0 },
   {Sin[a], 0.0, Cos[a], 0.0 },
   {0.0, 0.0, 0.0, 1.0 }};

rotationMatrixZ[a_] := {
   {Cos[a], Sin[a], 0.0, 0.0 },
   {-Sin[a], Cos[a], 0.0, 0.0 },
   {0.0, 0.0, 1.0, 0.0 },
   {0.0, 0.0, 0.0, 1.0 }};

translationMatrix[v_] := {
   { 1.0, 0.0, 0.0, v[[1]]},
   {0.0, 1.0, 0.0, v[[2]]},
   {0.0, 0.0, 1.0, v[[3]]},
   {0.0, 0.0, 0.0, 1.0 }};

segments[d_] :=
 Cases[d, XMLElement["segment", {___, "name" → nm_, ___}, {___}] :→ nm, Infinity]
```

```
getSegment[d_, seg_] :=
 Cases[d, XMLElement["segment", {___, "name" → nm_, ___}, data_] ⧴ data, Infinity]

directions[d_] :=
 Cases[d, XMLElement["frame", {___, "x" → x_, "y" → y_, "z" → z_, ___}, ___] ⧴
   ToExpression[{x, y, z}], Infinity]

distance[d_] :=
 Cases[d, XMLElement["frame", {___, "r" → r_, ___}, ___] ⧴ ToExpression[r], Infinity]

rotx[d_] :=
 Cases[d, XMLElement["frame", {___, "rotx" → v_, ___}, ___] ⧴ ToExpression[v], Infinity]

roty[d_] :=
 Cases[d, XMLElement["frame", {___, "roty" → v_, ___}, ___] ⧴ ToExpression[v], Infinity]

rotz[d_] :=
 Cases[d, XMLElement["frame", {___, "rotz" → v_, ___}, ___] ⧴ ToExpression[v], Infinity]

inertiaTensor[pnts_] := Module[{avg, relPnts, ixx, iyy, izz, ixy, ixz, iyz, m},
   avg = Mean[pnts];
   relPnts = Map[# - avg &, pnts];
   ixx = Total[Map[#[[2]]^2 + #[[3]]^2 &, relPnts]];
   iyy = Total[Map[#[[1]]^2 + #[[3]]^2 &, relPnts]];
   izz = Total[Map[#[[1]]^2 + #[[2]]^2 &, relPnts]];
   ixy = Total[Map[#[[1]] * #[[2]] &, relPnts]];
   ixz = Total[Map[#[[1]] * #[[3]] &, relPnts]];
   iyz = Total[Map[#[[2]] * #[[3]] &, relPnts]];
   m = {{ixx, ixy, ixz},
     {ixy, iyy, iyz},
     {ixz, iyz, izz}}
  ];

diagonalizedInertiaTensor[pnts_] := inertiaTensor[pnts] // Eigensystem
```

```
transformToPrincipleAxes[pnts_] := Module[{es, vx, vy, vz, g, l, tpnts},
   vx = Mean[pnts];
   vx *= 1.0 / Sqrt[vx.vx];
   es = diagonalizedInertiaTensor[pnts];
   vy = es[[2]][[3]];
   vz = es[[2]][[2]];
   Print["vy.vz=", vy.vz];
   vy = vy - (vy.vx) * vx;
   vy *= 1.0 / Sqrt[vy.vy];
   vz = vz - (vz.vx) * vx;
   vz *= 1.0 / Sqrt[vz.vz];
   Print["vY=", vy, " lvY=", Sqrt[vy.vy]];
   Print["vZ=", vz, " lvZ=", Sqrt[vz.vz]];
   Print["vX.vY=", vx.vy];
   Print["vX.vZ=", vx.vz];
   Print["vY.vZ=", vy.vz];
   m = {vx, vy, vz};
   Print[m // MatrixForm];
   tpnts = Table[m.pnts[[i]], {i, 1, Length[pnts]}]
  ];

directionGraphics[dir_] :=
  {Graphics3D[{PointSize[0.01], Sequence[Point /@ dir]}], principleAxes[dir]};

binValues[pnts_] := Module[{},
   amin = Min[pnts];
   amax = Max[pnts];
   bins = BINS;
   width = (amax - amin) / bins;
   xstart = amin + width / 2;
   bc = BinCounts[pnts, {amin, amax, width}];
   Table[{xstart + width * (i - 1), bc[[i]] / Length[bc]}, {i, 1, Length[bc]}]
  ];

parameterizeSimple[pnts_] := Module[{avg, stdev},
  avg = Mean[pnts];
  stdev = StandardDeviation[pnts];
  {avg, stdev}]

pop[x_, x0_, k_, coef_] := coef E^-(k (x - x0) ^2) / (RT)
```

```
parameterize[nm_, pnts_, pr_: All] :=
 Block[{df = $DisplayFunction, $DisplayFunction = Identity,
   binvals, coefStart, xstart, res, fitPlot, dataPlot},
  binvals = binValues[pnts];
  dataPlot = ListPlot[binvals,
    PlotStyle → {PointSize[0.02]}, PlotLabel → ToString[ nm ], PlotRange → pr];
  coefStart = Max[Transpose[binvals][[2]] / 2.0];
  xstart = Mean[pnts];
  res = NonlinearRegress[binvals, pop[x, x0, k, coef],
      {x}, {{x0, xstart}, {k, 10.0}, {coef, coefStart}}][[1]][[2]];
  fitPlot = Plot[pop[x, x0, k, coef] /. res, {x, Min[pnts], Max[pnts]}];
  $DisplayFunction = df;
  Show[dataPlot, fitPlot, PlotRange → All];
  res]

parameterizeEverything[data_, nm_] := {segment → nm,
  {rotx → parameterize["rotx", rotx[data]]},
  {roty → parameterize["roty", roty[data]]},
  {rotz → parameterize["rotz", rotz[data]]},
  {dist → parameterize["distance", distance[data]]}}

doAll[fn_] := Module[{data, pnts, segs, results},
   data = Import[fn];
   segs = segments[data];
   Do[
    pnts = getSegment[data, segs[[i]]];
    results = parameterizeEverything[pnts, segs[[i]]];
    Print["Segment = ", segs[[i]]];
    Print[results];
    , {i, Length[segs]}];
  ];

writeOneParameter[strm_, scaleForces_, nm_, parm_] := Block[{},
   WriteString[strm, "<", nm, " x0=\"", ToString[x0 /. parm],
     "\" k=\"", ToString[(k /. parm) * scaleForces], "\"/>\n"];
];
```

```
writeXmlParameters[fn_, nm_, scaleForces_, rxp_,
   ryp_, rzp_, ryyp_, rzzp_, distp_, mt_] := Block[{strm, mx},
   strm = OpenWrite[fn];
   WriteString[strm, "<Parameters name=\"" <> nm <> "\">\n"];
   writeOneParameter[strm, scaleForces, "rotx", rxp];
   writeOneParameter[strm, scaleForces, "roty", ryp];
   writeOneParameter[strm, scaleForces, "rotz", rzp];
   writeOneParameter[strm, scaleForces, "rotyy", ryyp];
   writeOneParameter[strm, scaleForces, "rotzz", rzzp];
   writeOneParameter[strm, scaleForces, "dist", distp];
   WriteString[strm, "<directionTransform>\n"];
   mx = mt // Flatten;
   Do[WriteString[strm, " ", ToString[mx[[i]]]], {i, 1, Length[mx]}];
   WriteString[strm, "\n</directionTransform>\n"];
   WriteString[strm, "</Parameters>\n"];
   Close[strm];
  ];
```

Process one of the segments in the middle of a sequence.

```
data = Import["out/_middle_parameters.xml"];
```

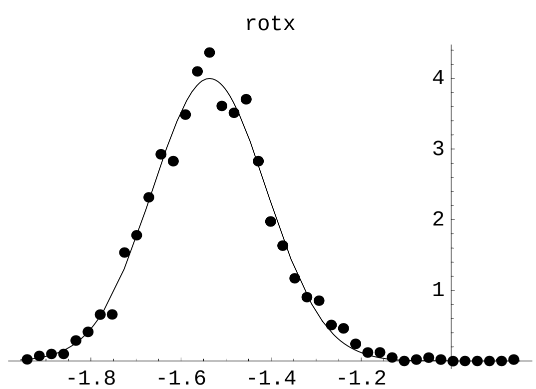
```
segs = segments[data];
```

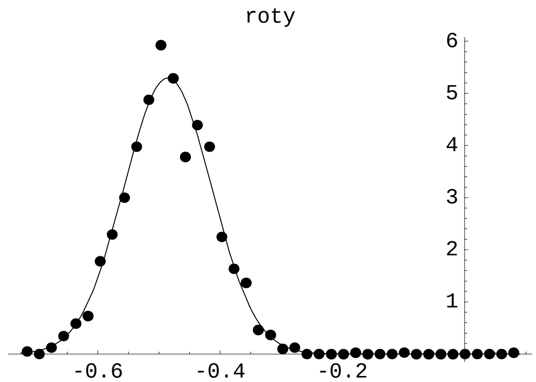
```
segmentToProcess = segs[[1]]
```

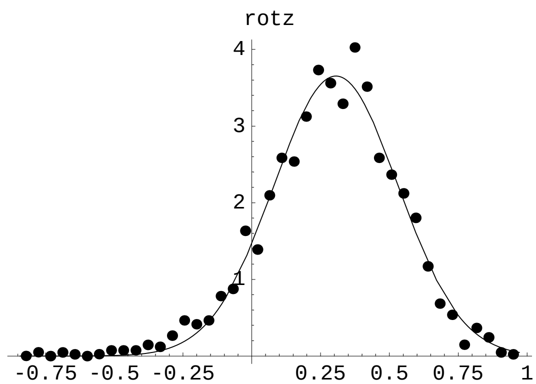
```
dkp-OSS2+OSS3
```

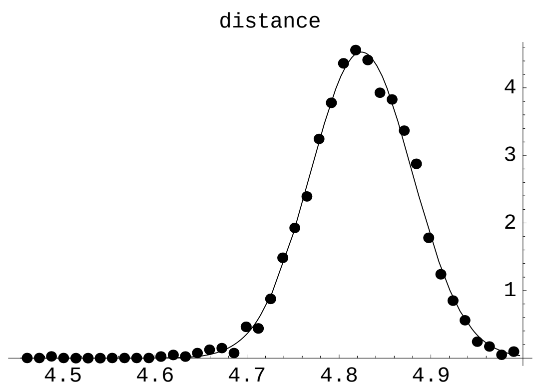
```
framesToProcess = getSegment[data, segmentToProcess];
```

```
rotDistParams = parameterizeEverything[framesToProcess, segmentToProcess]
```










```
{segment → dkp−OSS2+OSS3, {rotx → {x0 → −1.5366, k → 31.2036, coef → 2.39837}},
 {roty → {x0 → −0.485379, k → 98.3704, coef → 3.17604}},
 {rotz → {x0 → 0.305575, k → 9.75574, coef → 2.19112}},
 {dist → {x0 → 4.82446, k → 163.312, coef → 2.71791}}}
```

doAll["out/_lead_parameters.xml"];

doAll["out/_tail_parameters.xml"];

Parameterize the direction vector

Setup calculation

```
<< Geometry`Rotations`
```

```
dir = directions[framesToProcess];
```

For testing, use mangleMatrix to first rotate the points g*d knows where and then optimize it back onto the X-axis

mangleMatrix = RotationMatrix3D[90*0.0174533,10*0.0174533,0]

sd = mangleMatrix.#&/@dir;

```
sd = dir;
```

```
Mean[sd]
```

```
{0.921451, 0.151647, 0.325723}
```

angleY[rotm_,pnt_]:=ArcCos[((rotm.pnt).{1,0,0})[[2]]]

angleZ[rotm_,pnt_]:=ArcCos[((rotm.pnt).{1,0,0})[[3]]]

en[phi_,theta_,psi_,pnt_]:=angleY[RotationMatrix3D[phi,theta,psi],pnt]^2+angleZ[RotationMatrix3D[phi,theta,psi],pnt]^2

```
tf[rotm_, pnt_] := rotm.pnt
```

```
anglez[rotm_, pnt_] :=
 ArcSin[tf[rotm, pnt][[2]] / Sqrt[tf[rotm, pnt][[1]]^2 + tf[rotm, pnt][[2]]^2]]
```

```
enz[rotm_, pnt_] := anglez[rotm, pnt]^2
```

```
angley[rotm_, pnt_] :=
 ArcSin[tf[rotm, pnt][[3]] / Sqrt[tf[rotm, pnt][[1]]^2 + tf[rotm, pnt][[3]]^2]]
```

```
eny[rotm_, pnt_] := angley[rotm, pnt]^2
```

```
en[phi_, theta_, psi_, pnt_] := enz[RotationMatrix3D[phi, theta, psi], pnt] +
  10 * eny[RotationMatrix3D[phi, theta, psi], pnt]
```

```
e[phi_, theta_, psi_, pnts_] := Total[en[phi, theta, psi, #] & /@ pnts]
```

Carry out the minimization

First compile a function that just uses every 20th data point

```
tsd = Take[sd, {1, Length[sd], 20}];
```

```
Length[tsd]
```

```
98
```

```
e[0, 0, 0, tsd]
```

```
126.091
```

```
ceshort = Compile[{phif, thetaf, psif}, e[phif, thetaf, psif, tsd]]
```

```
CompiledFunction[{phif, thetaf, psif}, e[phif, thetaf, psif, tsd], -CompiledCode-]
```

```
ceshort[0, 0, 0]
```

```
126.091
```

```
Timing[sres = Minimize[ceshort[phif, thetaf, psif], {phif, thetaf, psif}]]
```

```
CompiledFunction::cfsa : Argument phif at position 1 should be a machine-size real number. More…
```

```
{7.516 Second, {5.73349, {phif → -1.19957, psif → 1.39821, thetaf → 0.343771}}}
```

Now compile the entire function and minimize from this starting point

```
ce = Compile[{{phi, _Real}, {theta, _Real}, {psi, _Real}}, e[phi, theta, psi, sd]]
```

```
CompiledFunction[{phi, theta, psi}, e[phi, theta, psi, sd], -CompiledCode-]
```

```
ce[phif, thetaf, psif] /. sres[[2]]
```

```
CompiledFunction::cfsa : Argument phif at position 1 should be a machine-size real number. More…
```

```
102.692
```

```
start = {{phi, phif}, {theta, thetaf}, {psi, psif}} /. sres[[2]]
```

```
{{phi, -1.19957}, {theta, 0.343771}, {psi, 1.39821}}
```

```
Timing[res = FindMinimum[ce[phi, theta, psi], start ]]
```

```
CompiledFunction::cfsa : Argument phi at position 1 should be a machine-size real number. More…
```

```
{35.39 Second, {100.49, {phi → -1.37022, theta → 0.33596, psi → 1.53475}}}
```

Timing[resall = Minimize[ce[phi,theta,psi],{phi,theta,psi}]]

Show results

```
res
```

```
{100.49, {phi → -1.37022, theta → 0.33596, psi → 1.53475}}
```

```
rm = RotationMatrix3D[phi, theta, psi] /. res[[2]]
```

```
{{0.931746, 0.152656, 0.329461},
 {-0.165762, 0.986094, 0.0118816}, {-0.323066, -0.0656827, 0.944094}}
```

```
rotatedSd = rm.# & /@ sd;
```

```
dg[dir_] := {Graphics3D[{PointSize[0.005], Sequence[Point /@ dir]}]};
```

```
gsd = dg[sd]
```

```
{- Graphics3D -}
```

```
gRotatedSd = dg[rotatedSd]
```

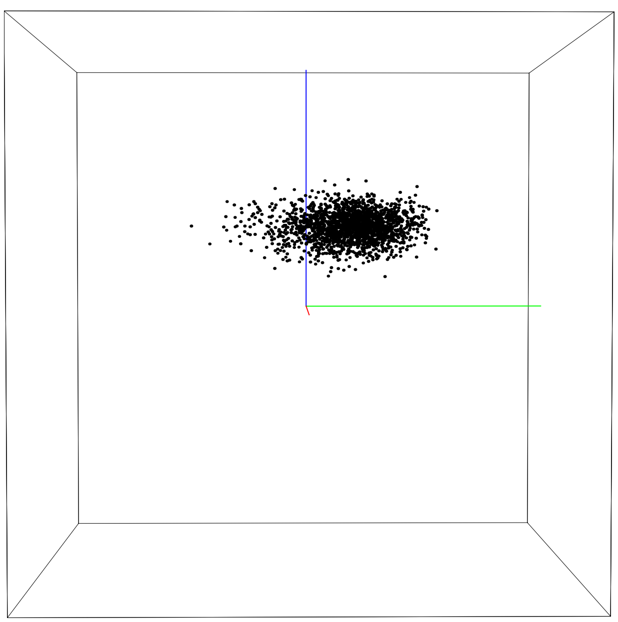
```
{- Graphics3D -}
```

```
axes = Graphics3D[{
    RGBColor[1, 0, 0], Line[{{0, 0, 0}, {1, 0, 0}}],
    RGBColor[0, 1, 0], Line[{{0, 0, 0}, {0, 1, 0}}],
    RGBColor[0, 0, 1], Line[{{0, 0, 0}, {0, 0, 1}}]}];
```

```
vp =
 ViewPoint -> {3.382, -0.040, 0.109}
```

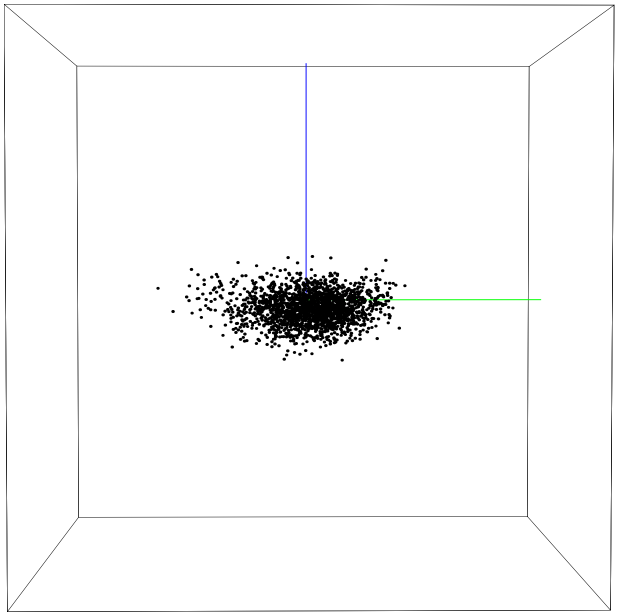
```
ViewPoint → {3.382, -0.04, 0.109}
```

```
Show[gsd, axes,
 PlotRange → {{-1.1, 1.1}, {-1.1, 1.1}, {-1.1, 1.1}},
 vp
]
```

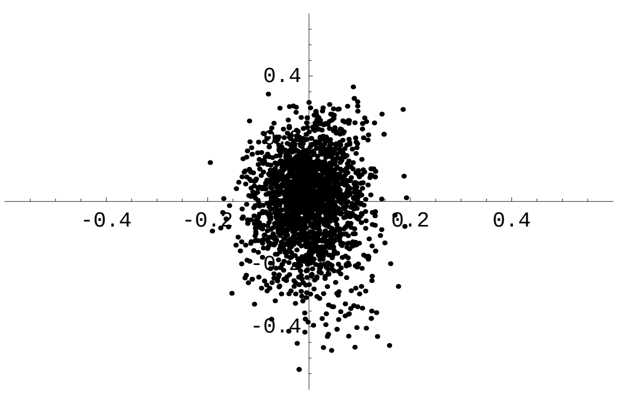


```
- Graphics3D -
```

```
Show[gRotatedSd, axes,
 PlotRange → {{-1.1, 1.1}, {-1.1, 1.1}, {-1.1, 1.1}},
 vp
]
```



```
- Graphics3D -

anglePoint[rotm_, pnt_] := {angley[rotm, pnt], anglez[rotm, pnt]}

anglePoints[rotm_, pnts_] := Table[anglePoint[rotm, pnts[[i]]], {i, 1, Length[pnts]}]

aps = anglePoints[rm, sd];

rng = 0.6;

ListPlot[aps, PlotRange → {{-rng, rng}, {-rng, rng}}];
```

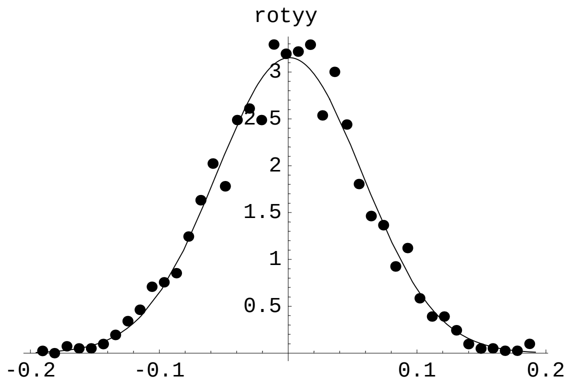



```
invrm = Inverse[rm]

{{0.931746, -0.165762, -0.323066},
 {0.152656, 0.986094, -0.0656827}, {0.329461, 0.0118816, 0.944094}}
```

```
invrm // MatrixForm
```

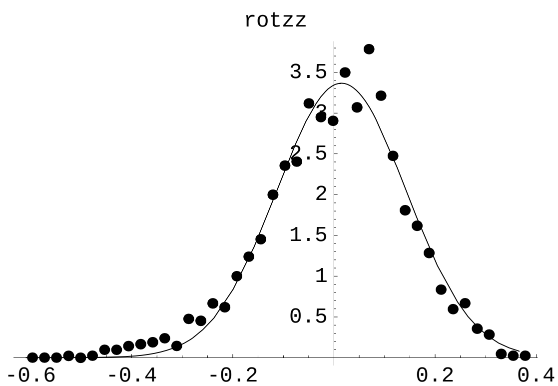
$$\begin{pmatrix} 0.931746 & -0.165762 & -0.323066 \\ 0.152656 & 0.986094 & -0.0656827 \\ 0.329461 & 0.0118816 & 0.944094 \end{pmatrix}$$

```
rotyyParameters = parameterize["rotyy", Transpose[aps][[1]], {-0.6, 0.6}]
```



{x0 → 0.00107446, k → 156.042, coef → 1.8916}

```
rotzzParameters = parameterize["rotzz", Transpose[aps][[2]], {-0.6, 0.6}]
```



{x0 → 0.0148487, k → 30.3664, coef → 2.02033}

Test the direction transform

```
Mean[sd]
```

{0.921451, 0.151647, 0.325723}

```
invrm.{1, 0, 0}
```

{0.931746, 0.152656, 0.329461}

```
yy[rotzz_] := {Cos[rotzz], Sin[rotzz], 0};
zz[rotyy_] := {Cos[rotyy], 0, -Sin[rotyy]}
```

```
dd[rotyy_, rotzz_] := yy[rotzz] + zz[rotyy]
```

```
ddn[rotyy_, rotzz_] := dd[rotyy, rotzz] / (Sqrt[dd[rotyy, rotzz].dd[rotyy, rotzz]])
```

```
ddn[10 * 0.0174533, 0]
```

```
{0.996195, 0, -0.0871558}
```

```
rdd[ryy_, rzz_] := invrm.ddn[ryy, rzz]
```

```
rdd[0, 0]
```

```
{0.931746, 0.152656, 0.329461}
```

```
Mean[sd]
```

```
{0.921451, 0.151647, 0.325723}
```

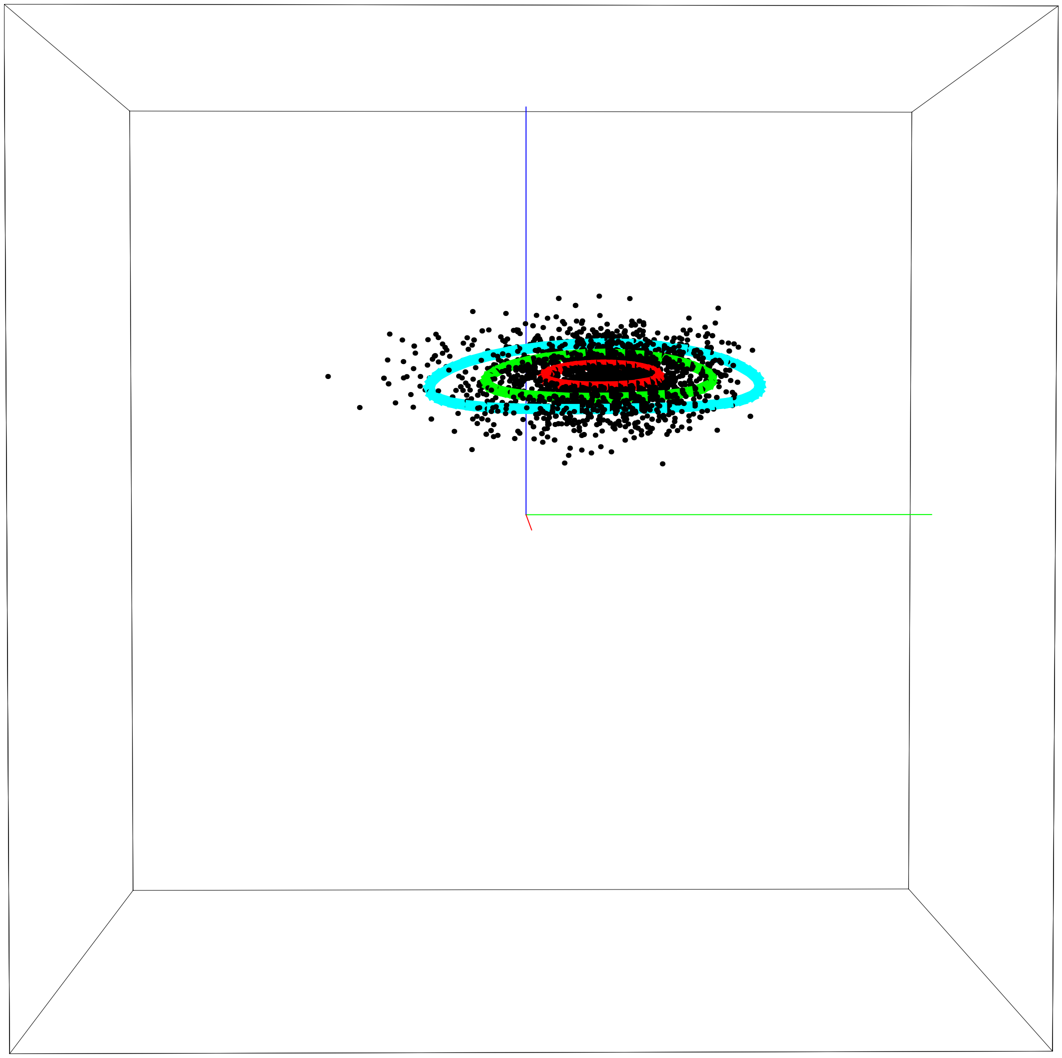
```
Clear[ld, s]
```

```
ld[s_] = Table[rdd[Cos[i * 0.0174533] * s * 0.1, Sin[i * 0.0174533] * s * 0.5], {i, 0, 360, 10}];
```

```
gl = Graphics3D[{Thickness[0.01],
   RGBColor[1, 0, 0], Line[ld[0.5]],
   RGBColor[0, 1, 0], Line[ld[1]],
   RGBColor[0, 1, 1], Line[ld[1.5]]
  }]
```

```
- Graphics3D -
```

```
Show[gl, gsd, axes,
  PlotRange → {{-1.1, 1.1}, {-1.1, 1.1}, {-1.1, 1.1}},
  ViewPoint -> {3.382, -0.040, 0.109}
 ];
```



Now pull all of the parameters together

```
rotDistParams
```

```
{segment → dkp-OSS2+OSS3, {rotx → {x0 → -1.5366, k → 31.2036, coef → 2.39837}},
 {roty → {x0 → -0.485379, k → 98.3704, coef → 3.17604}},
 {rotz → {x0 → 0.305575, k → 9.75574, coef → 2.19112}},
 {dist → {x0 → 4.82446, k → 163.312, coef → 2.71791}}}
```

```
flatRotDistParams = Flatten[rotDistParams, 1]
```

```
{segment → dkp-OSS2+OSS3, rotx → {x0 → -1.5366, k → 31.2036, coef → 2.39837},
 roty → {x0 → -0.485379, k → 98.3704, coef → 3.17604},
 rotz → {x0 → 0.305575, k → 9.75574, coef → 2.19112},
 dist → {x0 → 4.82446, k → 163.312, coef → 2.71791}}
```

Here are the important parameters, their equilibrium values (x0) and their force constants (k)

```
rotxParameters = rotx /. flatRotDistParams
```

```
{x0 → -1.5366, k → 31.2036, coef → 2.39837}
```

```
rotyParameters = roty /. flatRotDistParams
```

```
{x0 → -0.485379, k → 98.3704, coef → 3.17604}
```

```
rotzParameters = rotz /. flatRotDistParams
```

```
{x0 → 0.305575, k → 9.75574, coef → 2.19112}
```

```
distParameters = dist /. flatRotDistParams
```

```
{x0 → 4.82446, k → 163.312, coef → 2.71791}
```

■ "invrm" is the transformation matrix to convert from "direction cloud space" to "monomer space"

```
invrm // MatrixForm
```

$$\begin{pmatrix} 0.931746 & -0.165762 & -0.323066 \\ 0.152656 & 0.986094 & -0.0656827 \\ 0.329461 & 0.0118816 & 0.944094 \end{pmatrix}$$

```
rotyyParameters
```

```
{x0 → 0.00107446, k → 156.042, coef → 1.8916}
```

```
rotzzParameters
```

```
{x0 → 0.0148487, k → 30.3664, coef → 2.02033}
```

■ Now generate a transformation matrix for a set of parameters

```
monomerDirection[trotyy_, trotzz_] := invrm.ddn[trotyy, trotzz];
```

```
monomerTransformRaw[ trotx_, troty_, trotz_, tx_, ty_, tz_, tdist_] :=
  Block[{mx, my, mz, mt, v},
   mx = rotationMatrixX[-trotx];
   my = rotationMatrixY[-troty];
   mz = rotationMatrixZ[-trotz];
   v = {tx, ty, tz};
   v = v / Sqrt[v.v];
   mt = translationMatrix[v * tdist];
   mt.mz.my.mx];

monomerTransform[ trotx_, troty_, trotz_, trotyy_, trotzz_, tdist_] := Block[{dir},
   dir = monomerDirection[trotyy, trotzz];
   m = monomerTransformRaw[trotx, troty, trotz, dir[[1]], dir[[2]], dir[[3]], tdist]
  ];

writeXmlParameters["dkp-pro4SS+pro4SS.xml", "atest", 1, rotxParameters, rotyParameters,
 rotzParameters, rotyyParameters, rotzzParameters, distParameters, invrm // Flatten]

(k /. rotyyParameters) * 100

15604.2
```

Axis definitions

```
vO = position of "CG";
vX1 = [position of "CB"] - vO;
vX2 = [position of "CG"] - vO;
vX = [(vX1 + vX2) * -1.0].normalized ();
vY = [(vX1 - vX2)].normalized ();
vZ = vX.cross (vY);
```