ACTS seed reconstruction

Barak Schmookler

with help from Emma and Rey

Motivation

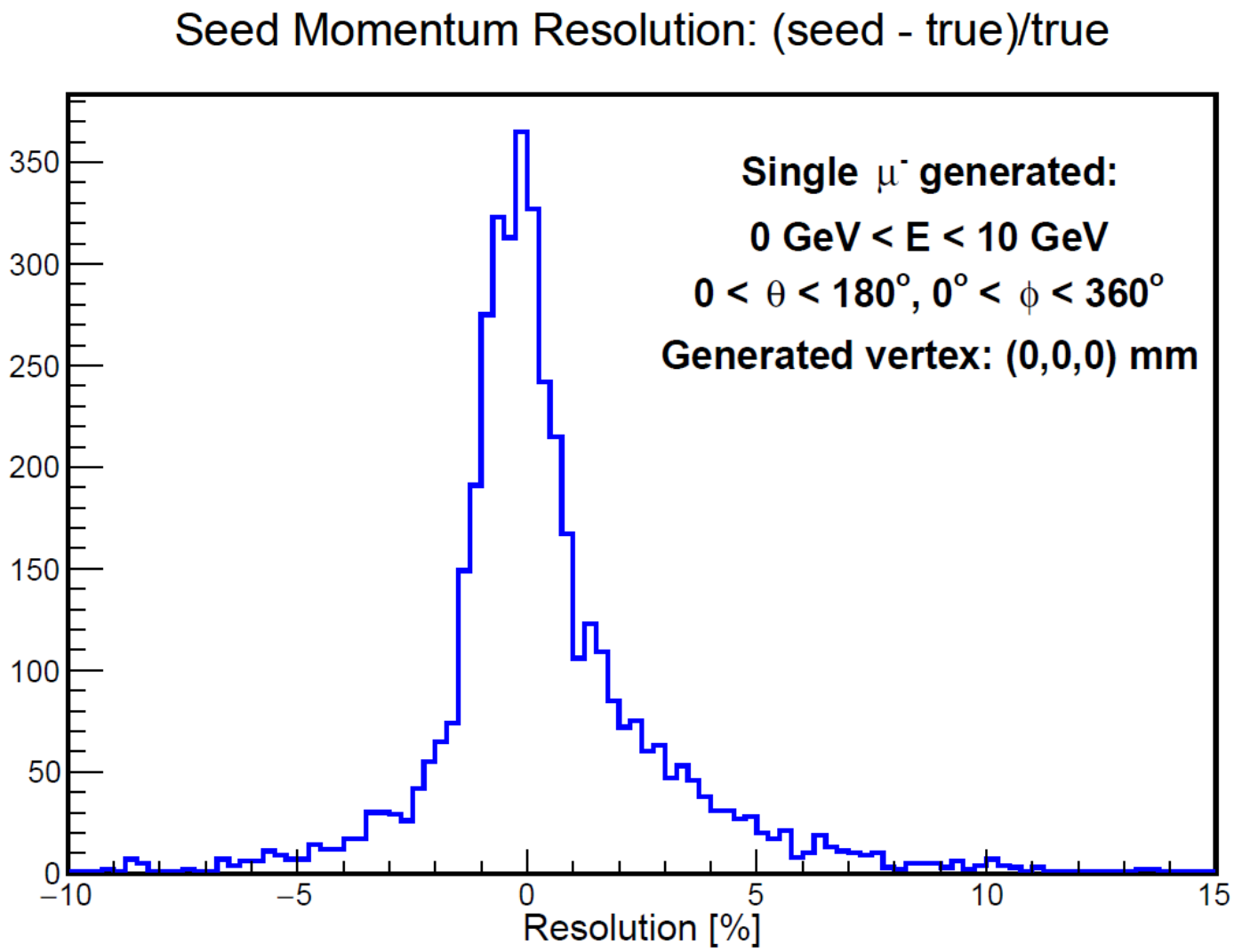
- Reconstruction of the seed parameters is done in this file:

  https://github.com/eic/EICrecon/blob/main/src/algorithms/tracking/TrackSeeding.cc

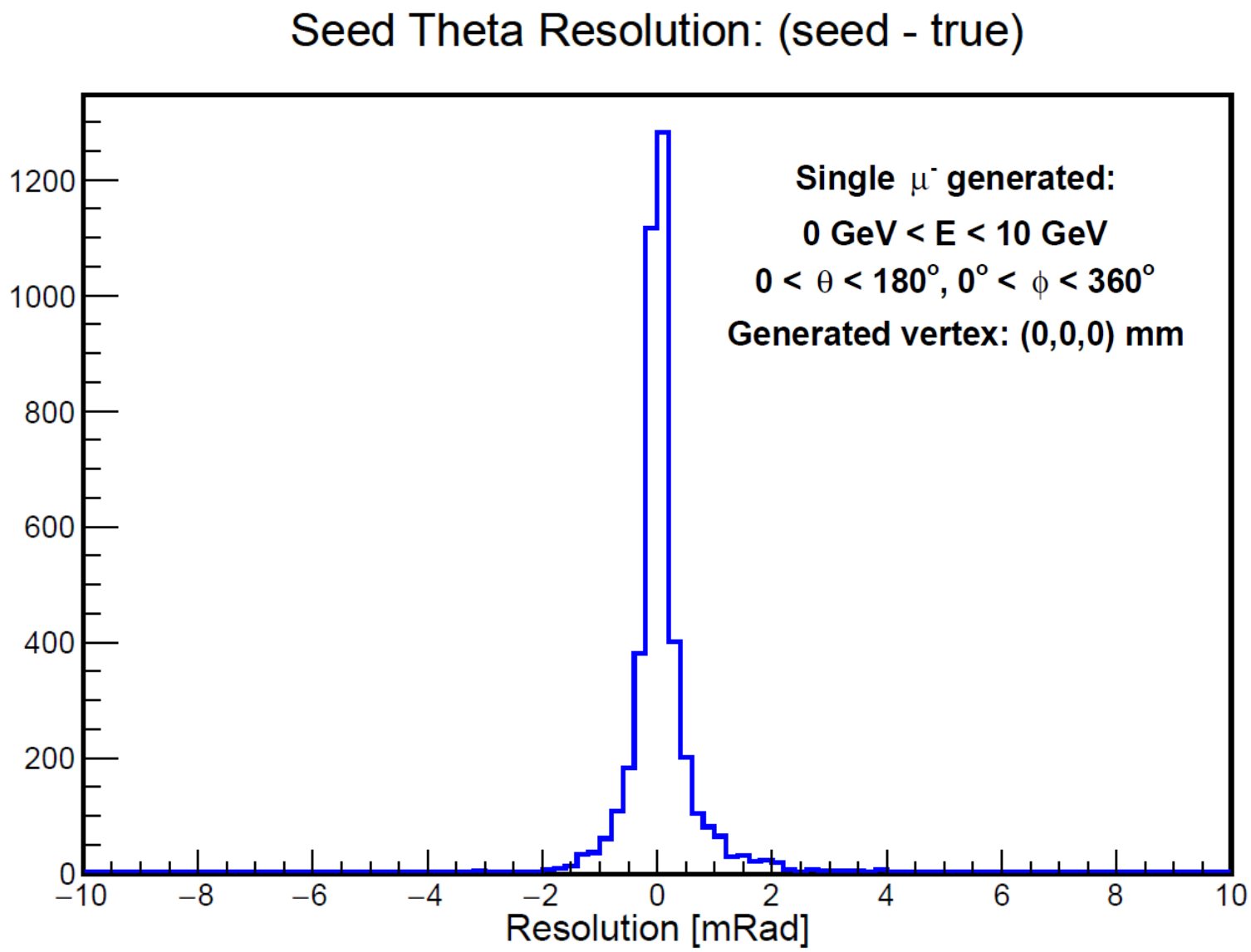
- We want to compare the reconstructed seed parameters – momentum (q/p), theta, phi, ACTS positions (a,b) – to the generated particle.
- This will allow us to check if our seed reconstruction is reasonable, and it will provide guidance for initial values for the CKF covariance matrix.

Seed momentum reconstruction

See unit fix in
https://github.com/eic/EICrecon/pull/544



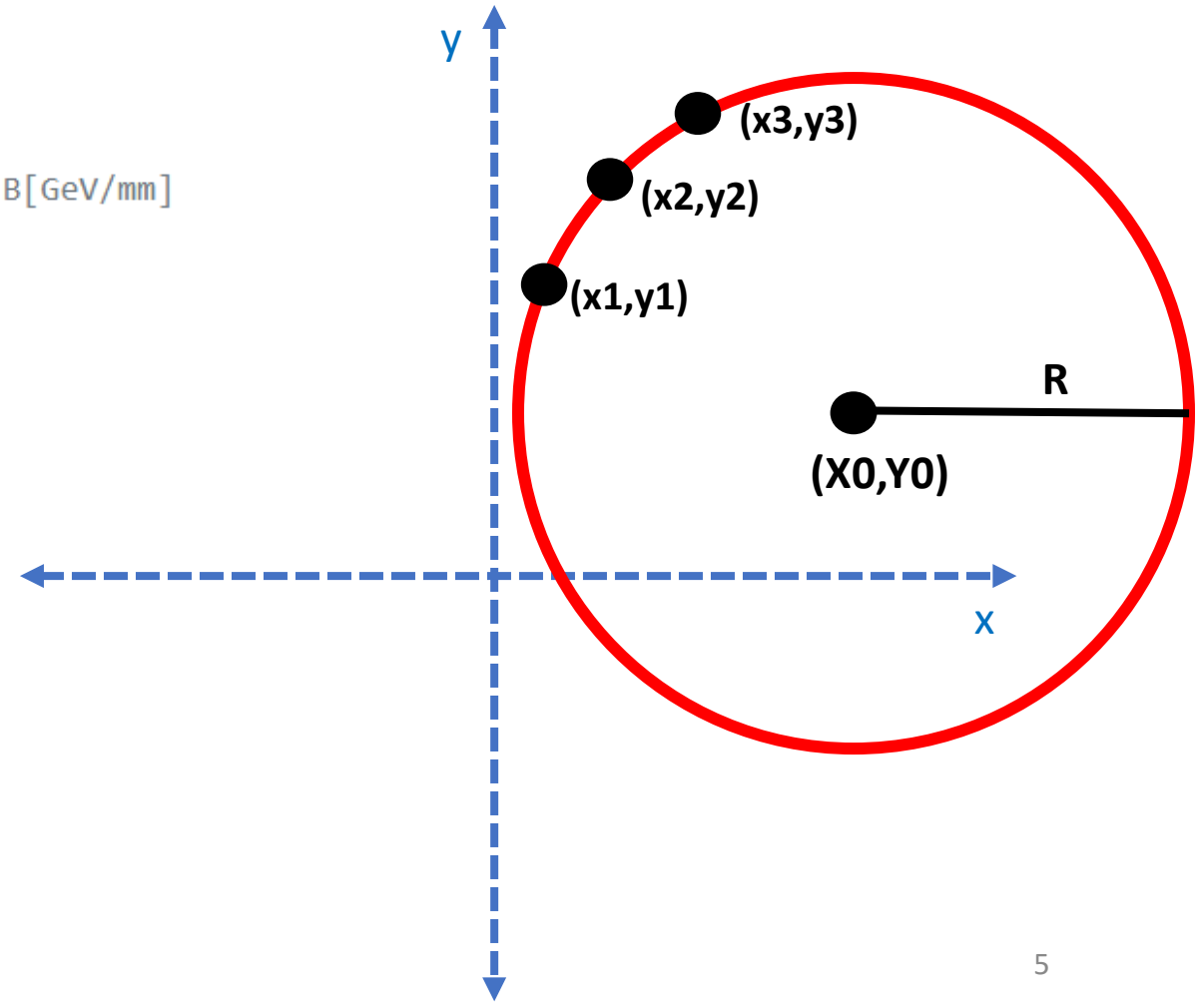
Seed theta reconstruction



How the seed (transverse) momentum is reconstructed

```
85        auto RX0Y0 = circleFit(xyHitPositions);
86        float R = std::get<0>(RX0Y0);
87        float X0 = std::get<1>(RX0Y0);
88        float Y0 = std::get<2>(RX0Y0);
97        float pt = R * m_cfg.m_bFieldInZ; // pt[GeV] = R[mm] * B[GeV/mm]
```

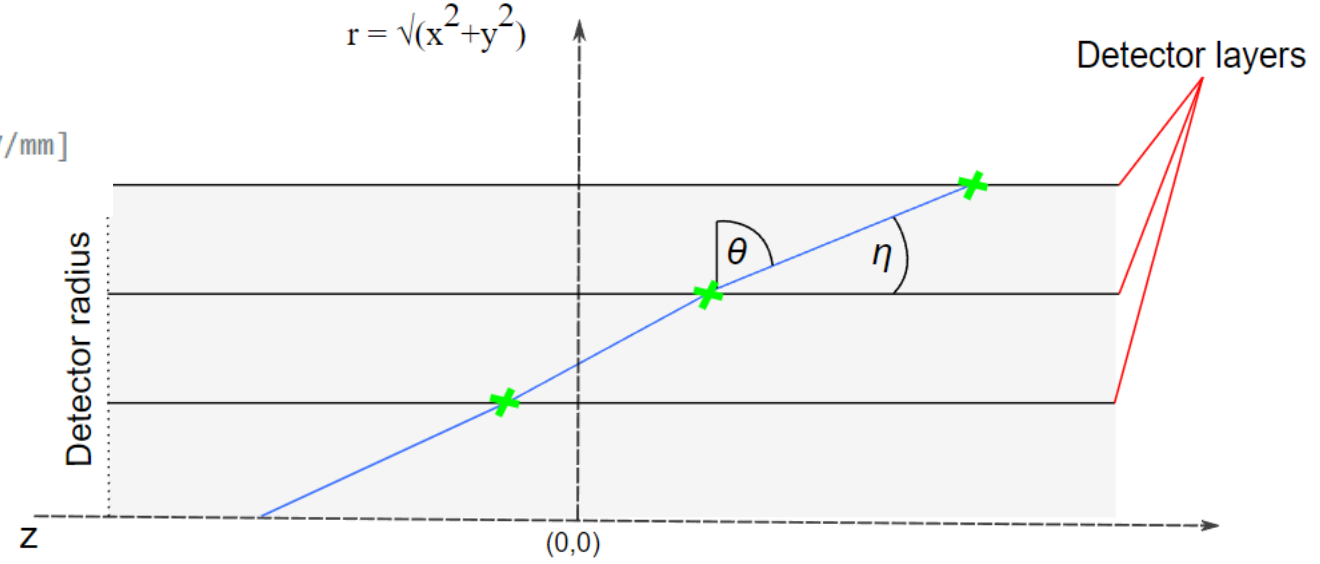
circleFit determines the circle radius and center in the (x,y) plane. The function can fit more than 3 points – so it may be more complex than necessary – but it seems to work well.



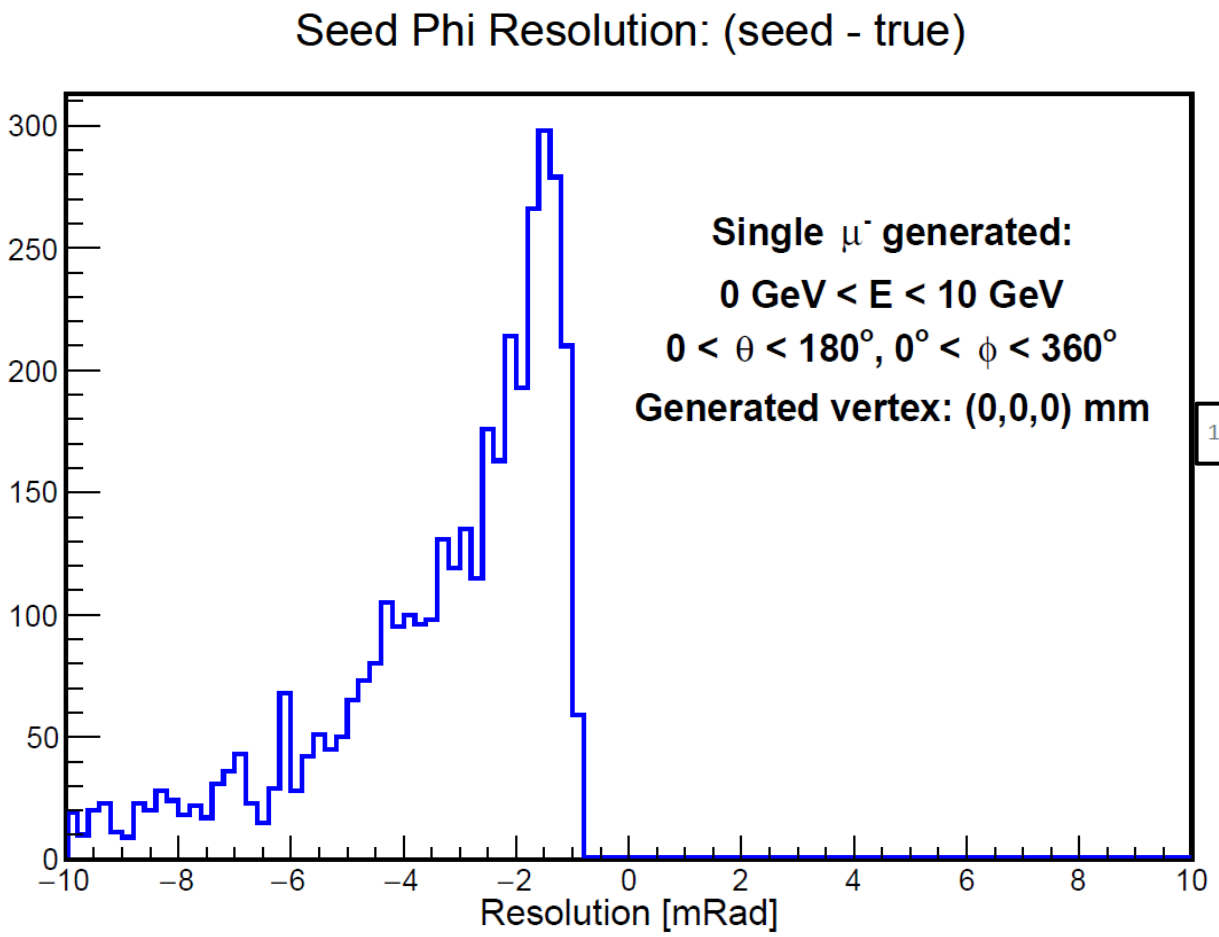
How the seed theta is reconstructed

```
89      auto slopeZ0 = lineFit(rzHitPositions);
92     float theta = atan(1./std::get<0>(slopeZ0));
93     // normalize to 0<theta<pi
94     if(theta < 0)
95       { theta += M_PI; }
96     float eta = -log(tan(theta/2.));
97     float pt = R * m_cfg.m_bFieldInZ; // pt[GeV] = R[mm] * B[GeV/mm]
98     float p = pt * cosh(eta);
```

lineFit makes a straight line fit of the three seed points in the r-z plane to determine theta.



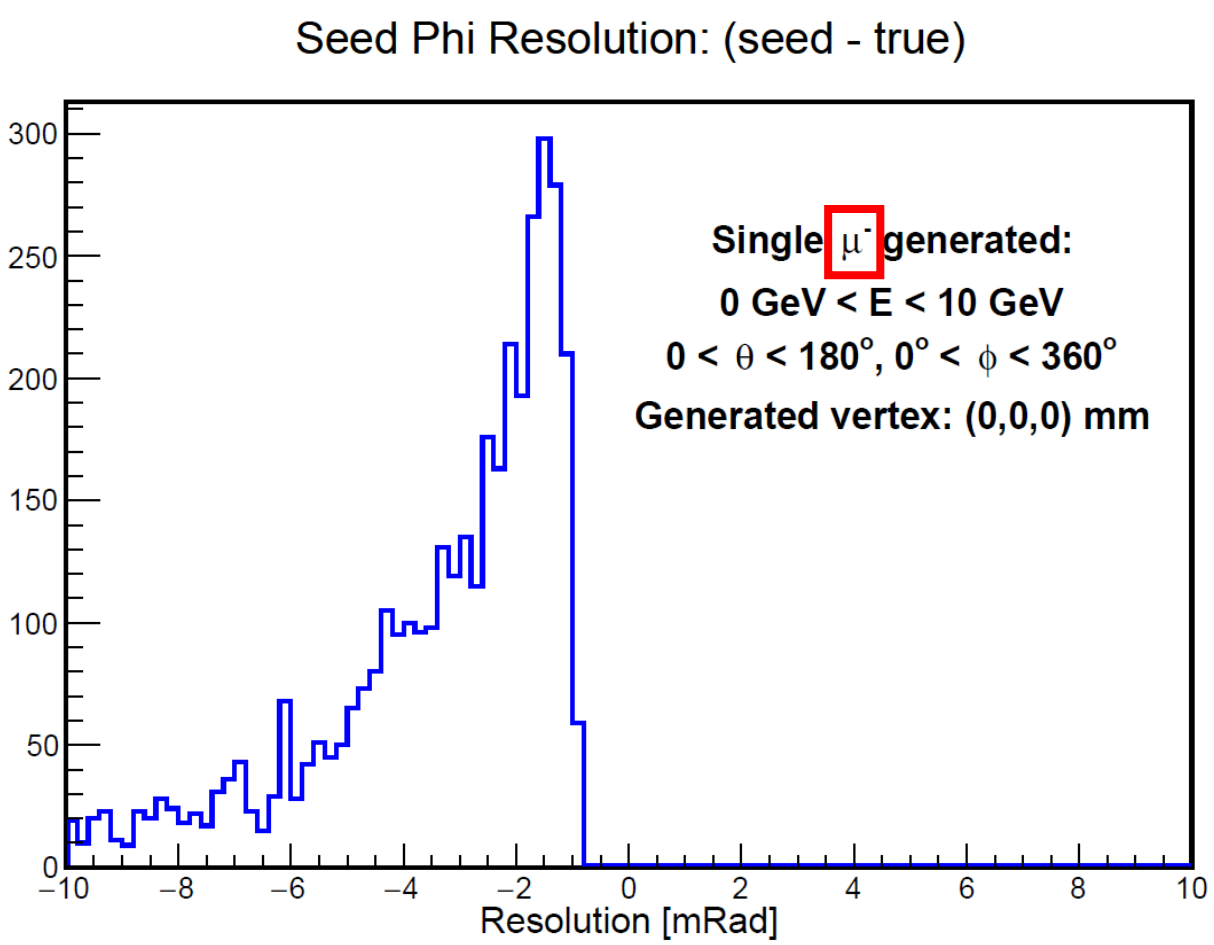
Seed phi reconstruction

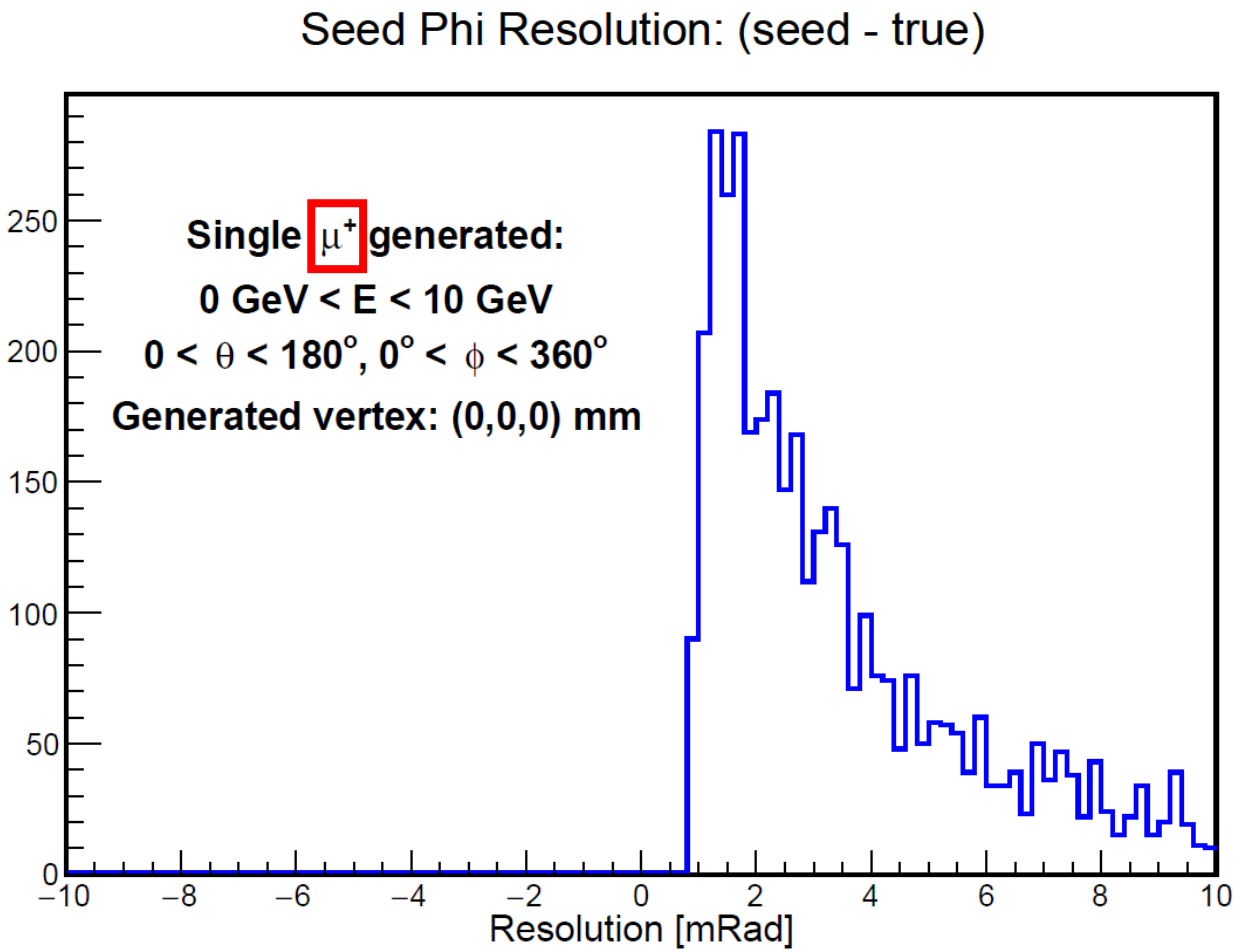


The seed phi is calculated based on position of first seed point.

```
120         atan2(xyHitPositions.at(0).second, xyHitPositions.at(0).first), // phi of first hit (rad)
```

Seed phi reconstruction





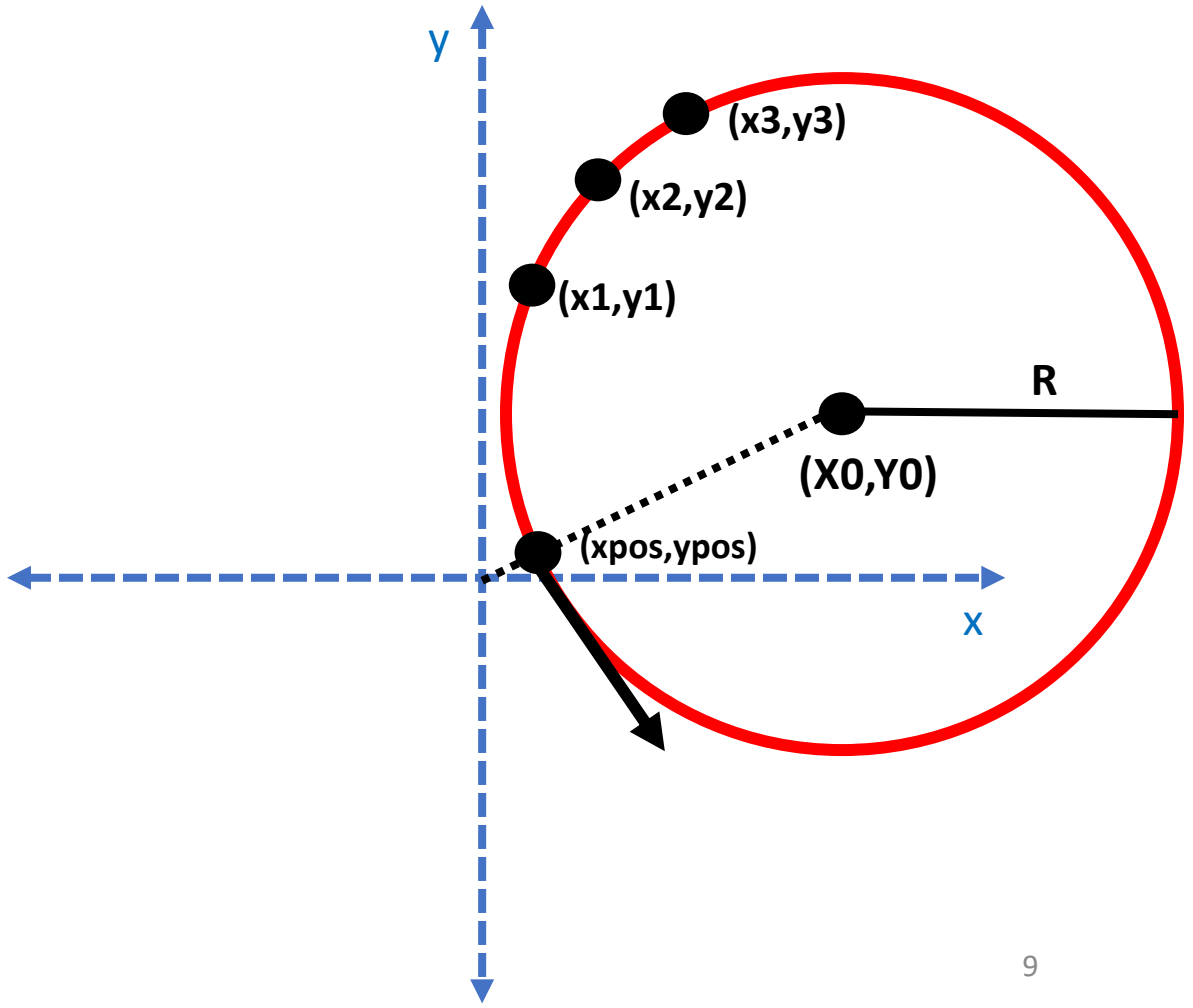
See 'reversed' effect for positive muons, as expected.

How to reconstruct the correct seed phi

```
85          auto RX0Y0 = circleFit(xyHitPositions);
86          float R = std::get<0>(RX0Y0);
87          float X0 = std::get<1>(RX0Y0);
88          float Y0 = std::get<2>(RX0Y0);
101         const auto xypos = findRoot(RX0Y0);
```

findRoot finds the point of closest approach on the circle to the origin – (xpos,ypos).

So, we need to find the angle of the vector going through (xpos,ypos) that is tangential to the circle.



How to reconstruct the correct seed phi

```
//Calculate phi at xypos
auto xpos = xypos.first;
auto ypos = xypos.second;

auto vxpos = -1.*charge*(ypos-Y0);
auto vypos = charge*(xpos-X0);

auto phi = atan2(vypos,vxpos);
```

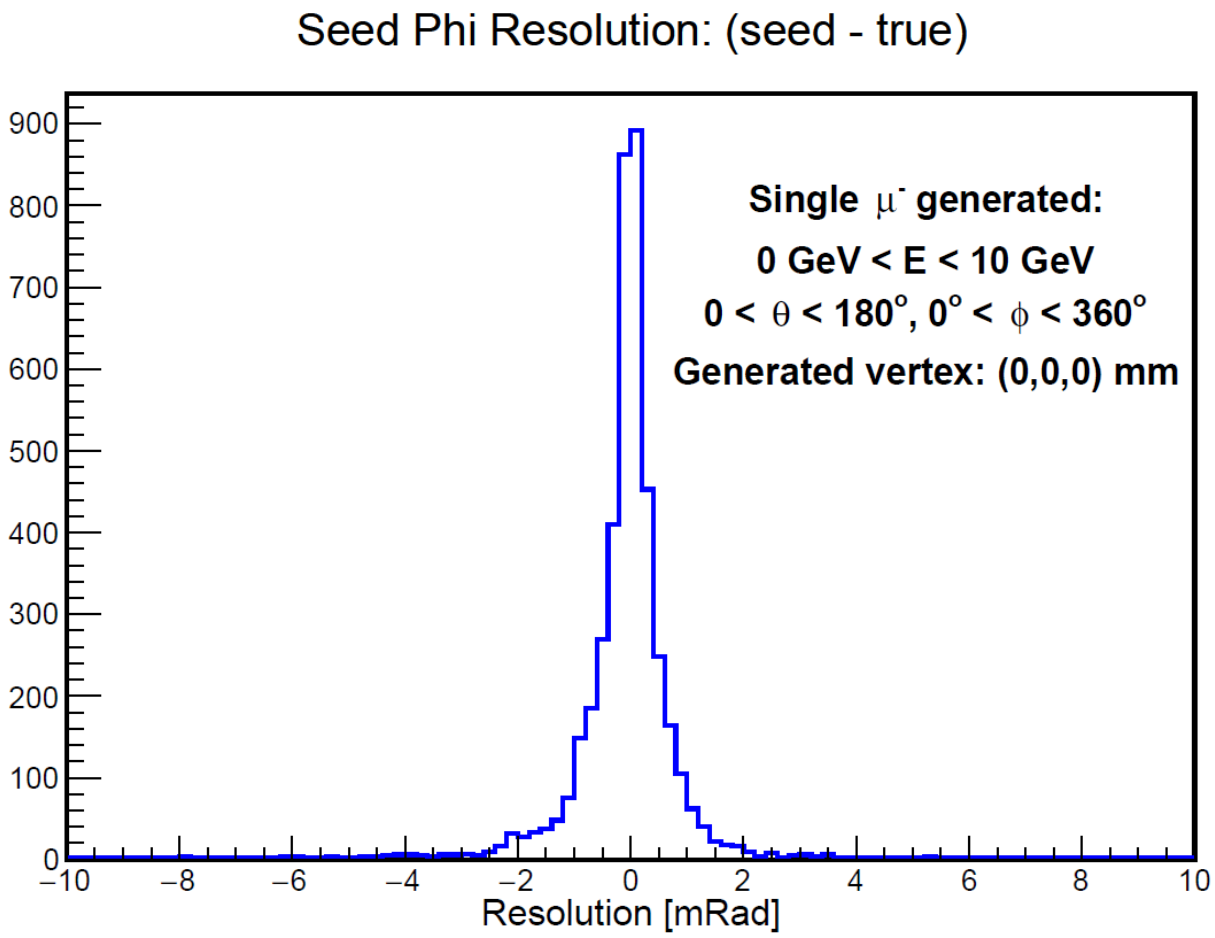
Vector from center of circle to point closest to origin:

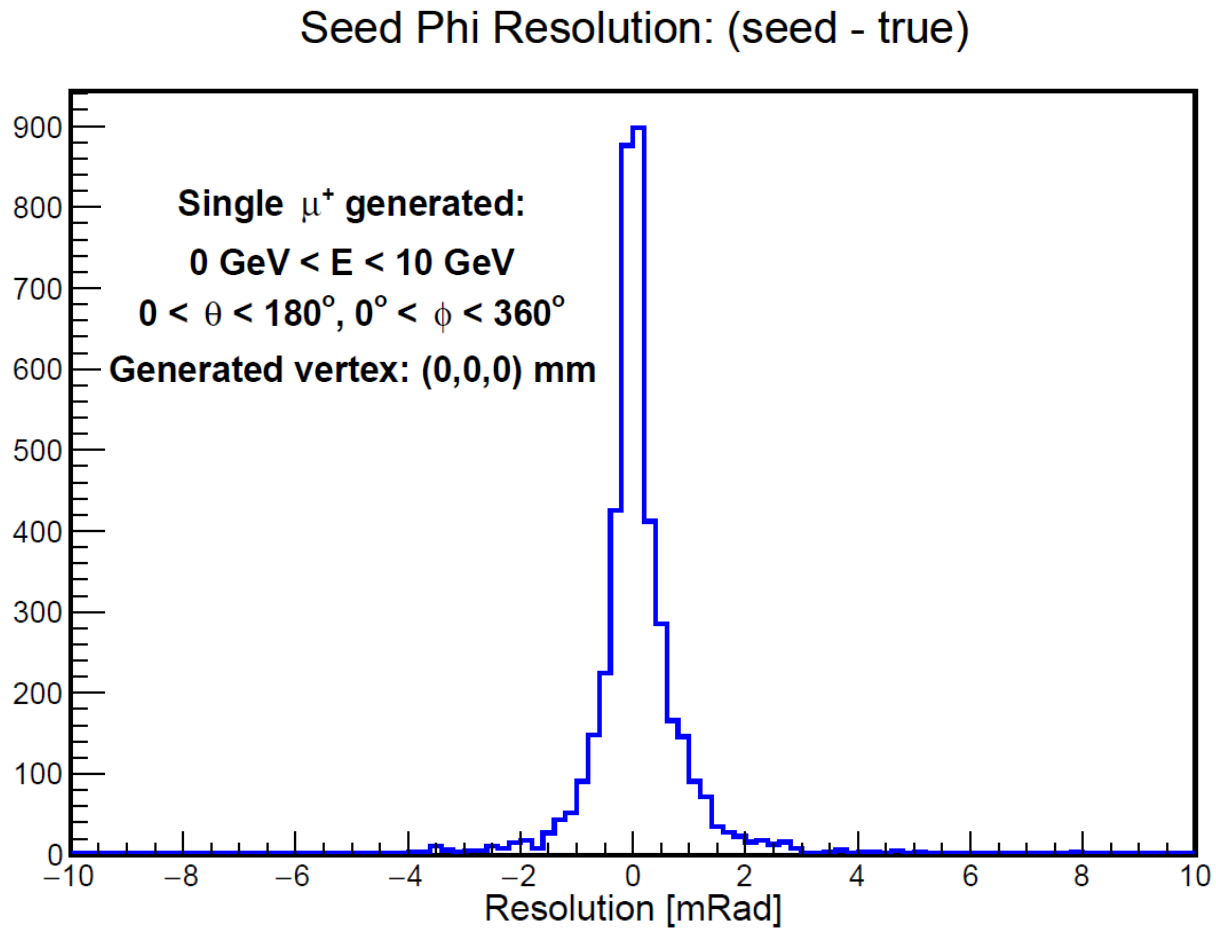
$$< (xpos\ - X0), (ypos\ - YO >$$

Vector tangential to circle at point closest to origin:

$$\pm < (ypos - Y0), -(xpos - X0) >$$

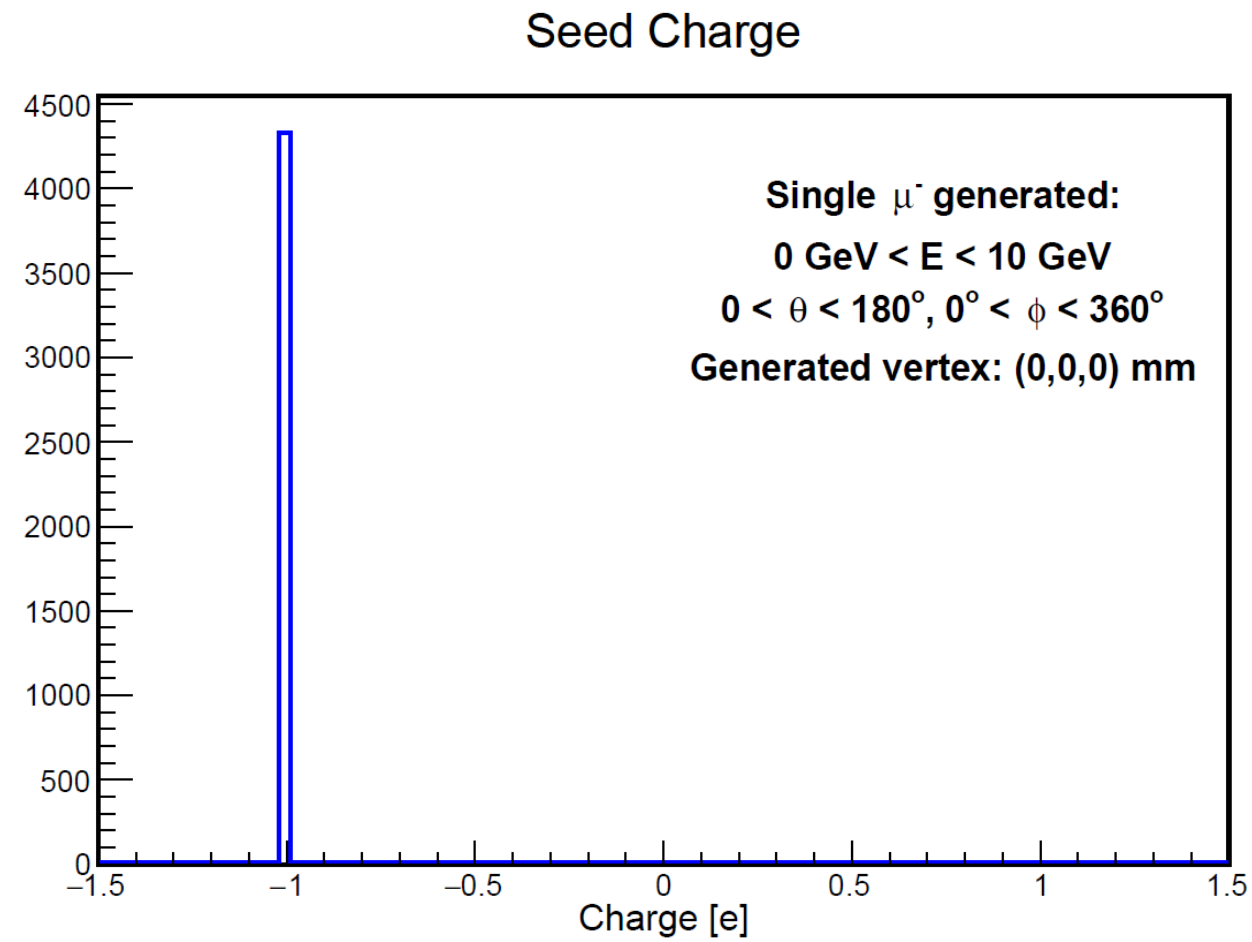
Seed phi reconstruction after fix

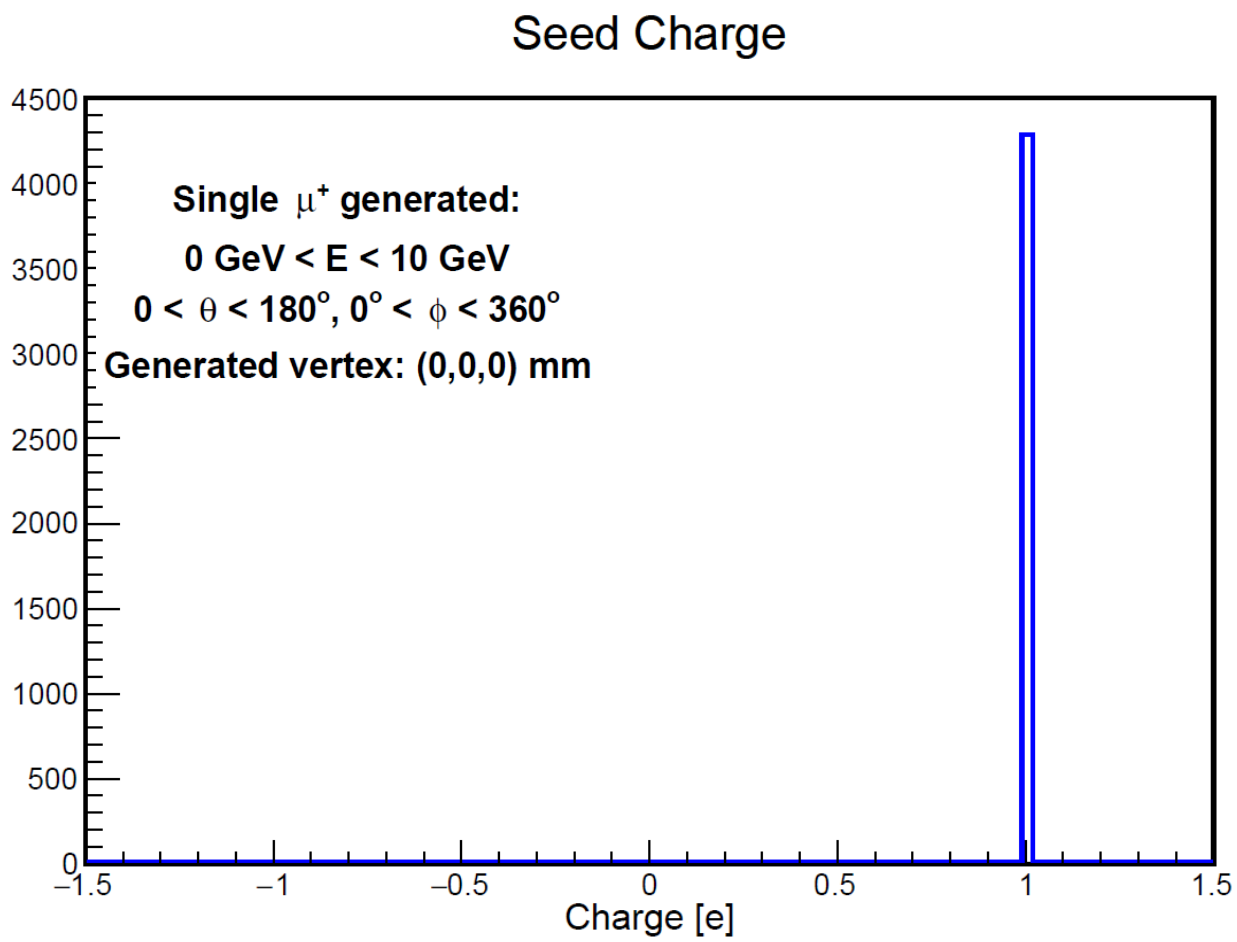




Looks good for both positive and negative particles!

Seed charge reconstruction

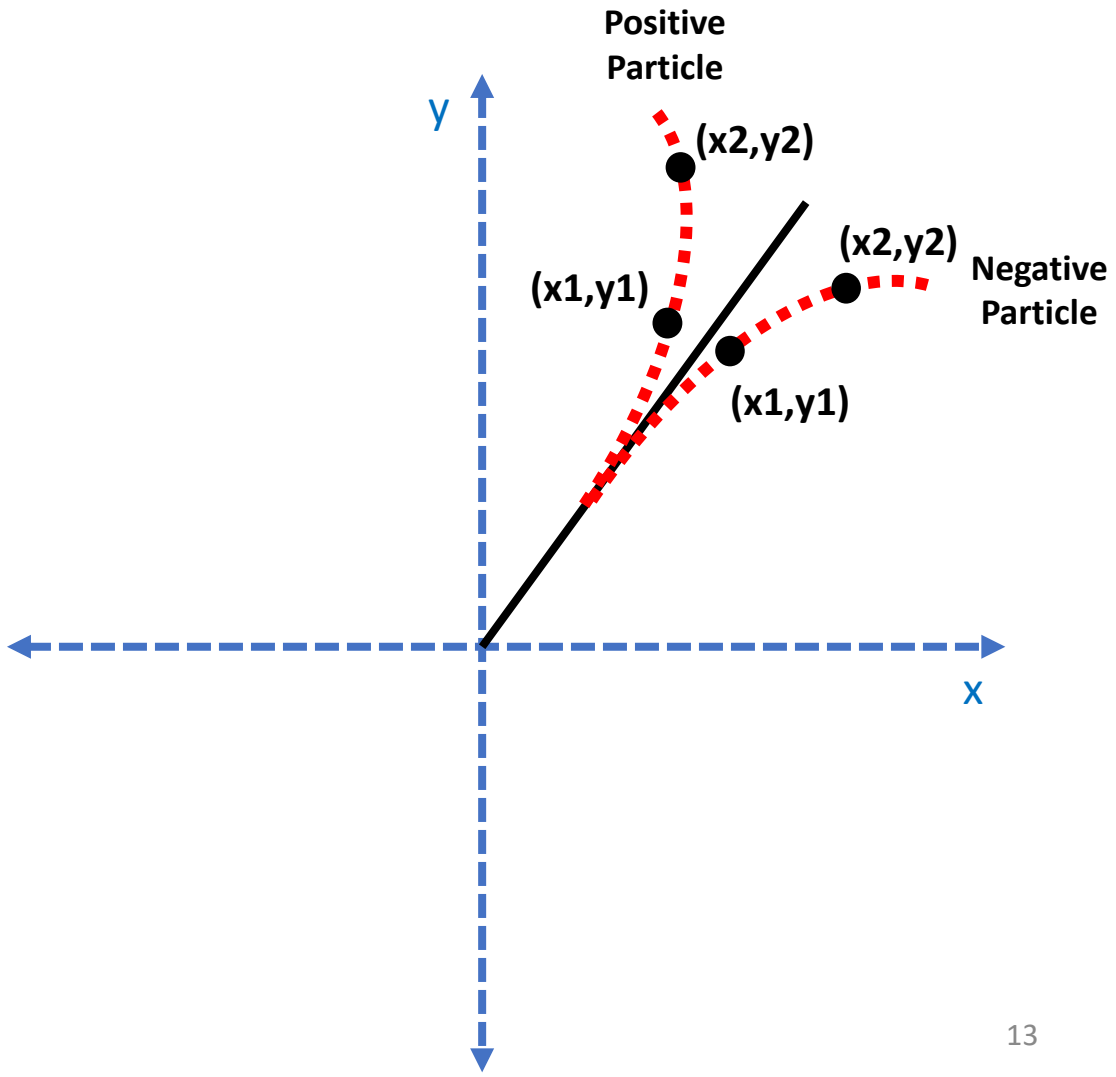




How the seed charge is reconstructed

```
91          int charge = determineCharge(xyHitPositions);
```

determineCharge compares the first 2 seed hits and considers which way they ‘fall off’ a line.



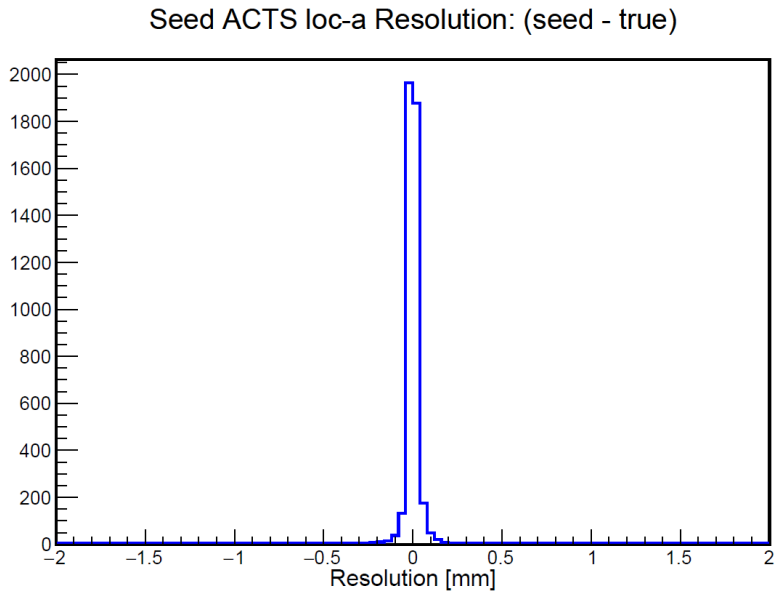
Seed position reconstruction

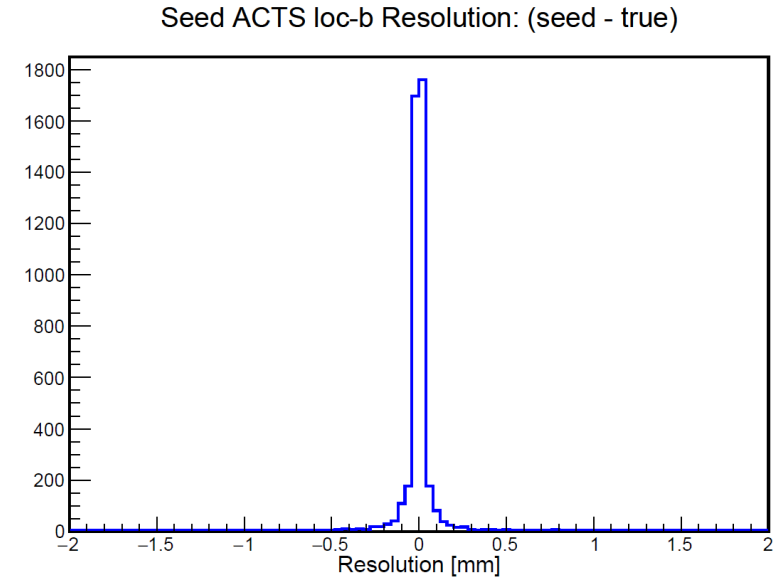
Single μ^- generated:

0 GeV < E < 10 GeV

0 < θ < 180°, 0° < ϕ < 360°

Generated vertex: (0,0,0) mm





Top plot: **Seed ACTS loc.a –** $\sqrt{gen.vert.x^2 + gen.\ vert.y^2}$

Bottom plot: **Seed ACTS loc.b – gen.vert.z**

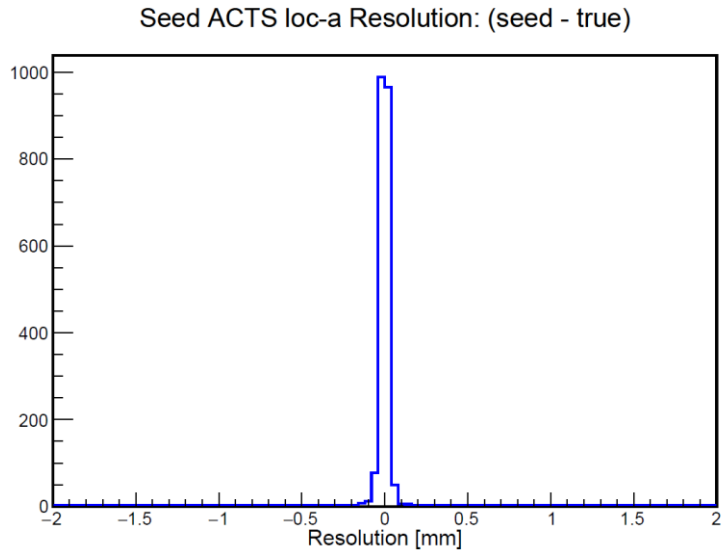
Seed position reconstruction

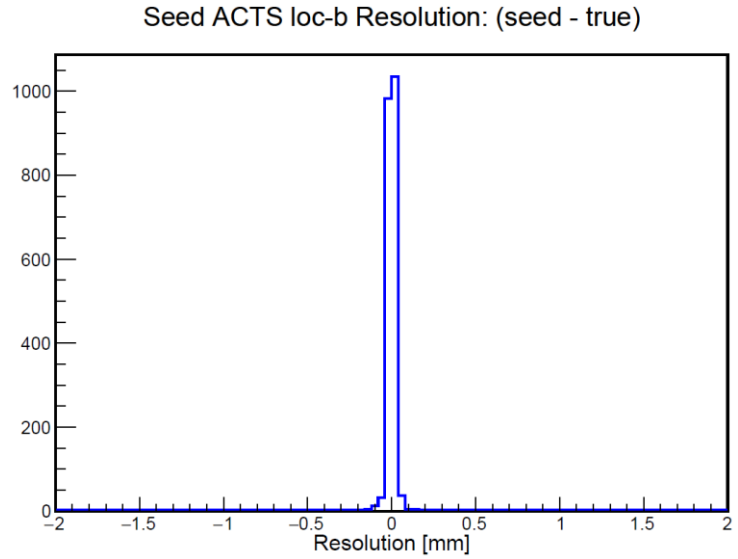
Single μ^- generated:

0 GeV < E < 10 GeV

0 < θ < 180°, 0° < ϕ < 360°

Generated vertex: (0,0,+10) mm





Top plot: **Seed ACTS loc.a –** $\sqrt{gen.vert.x^2 + gen.\ vert.y^2}$

Bottom plot: **Seed ACTS loc.b – gen.vert.z**

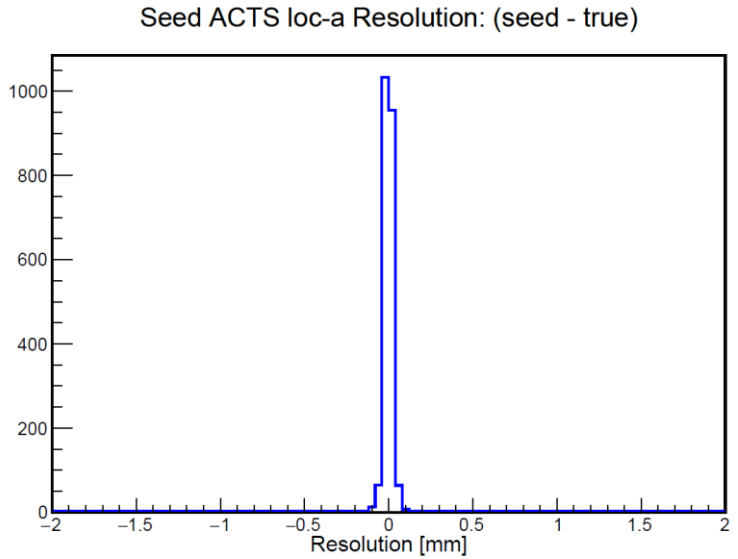
Seed position reconstruction

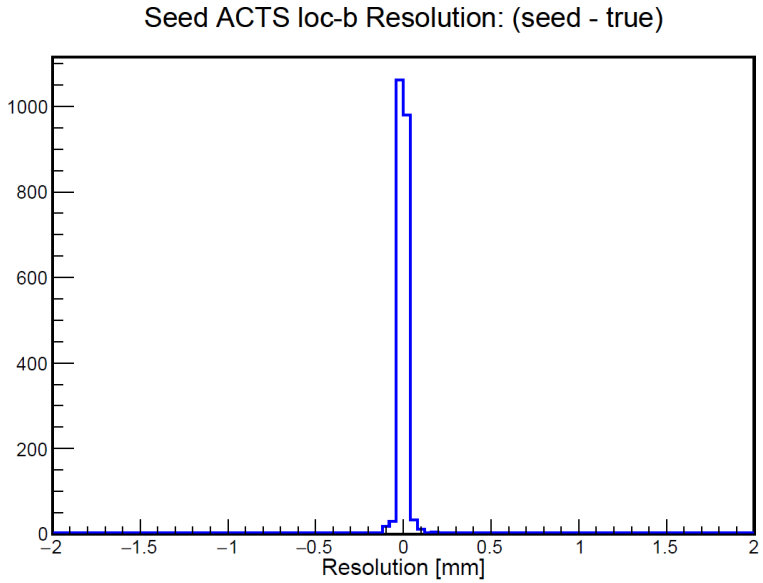
Single μ^- generated:

0 GeV < E < 10 GeV

0 < θ < 180°, 0° < ϕ < 360°

Generated vertex: (0,0,-10) mm





Top plot: **Seed ACTS loc.a – $\sqrt{gen.vert.x^2 + gen.\ vert.y^2}$**

Bottom plot: **Seed ACTS loc.b – gen.vert.z**

How are the seed positions calculated

```
101   const auto xypos = findRoot(RX0Y0);
102   const float z0 = seed.z();
103   auto perigee = Acts::Surface::makeShared<Acts::PerigeeSurface>(Acts::Vector3(0,0,0));
104   Acts::Vector3 global(xypos.first, xypos.second, z0);
105
106   auto local = perigee->globalToLocal(m_geoSvc->getActsGeometryContext(),
107                                       global, Acts::Vector3(1,1,1));
108
109   Acts::Vector2 localpos(sqrt(square(xypos.first) + square(xypos.second)), z0);
110   if(local.ok())
111     {
112       localpos = local.value();
113     }
```

Seed distance of closest approach in global coordinates.

How are the seed positions calculated

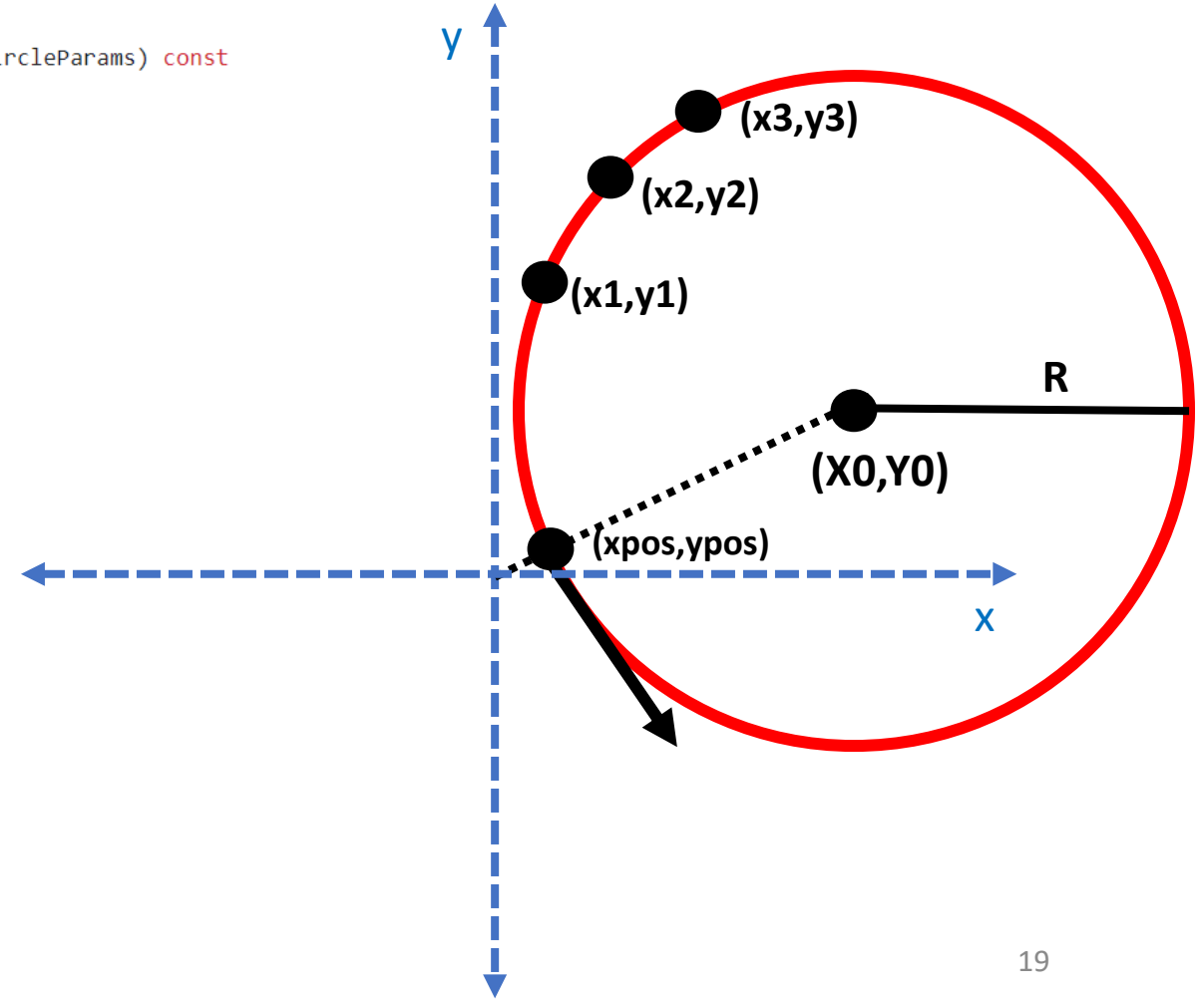
```
101  const auto xypos = findRoot(RX0Y0);
102  const float z0 = seed.z();
103  auto perigee = Acts::Surface::makeShared<Acts::PerigeeSurface>(Acts::Vector3(0,0,0));
104  Acts::Vector3 global(xypos.first, xypos.second, z0);
105
106  auto local = perigee->globalToLocal(m_geoSvc->getActsGeometryContext(),
107                                      global, Acts::Vector3(1,1,1));
108
109  Acts::Vector2 localpos(sqrt(square(xypos.first) + square(xypos.second)), z0);
110  if(local.ok())
111    {
112      localpos = local.value();
113    }
```

Conversion to ACTS local coordinates (perigee surface).

Not sure we need this line, since it is overwritten. Or we can keep that line and delete the rest.

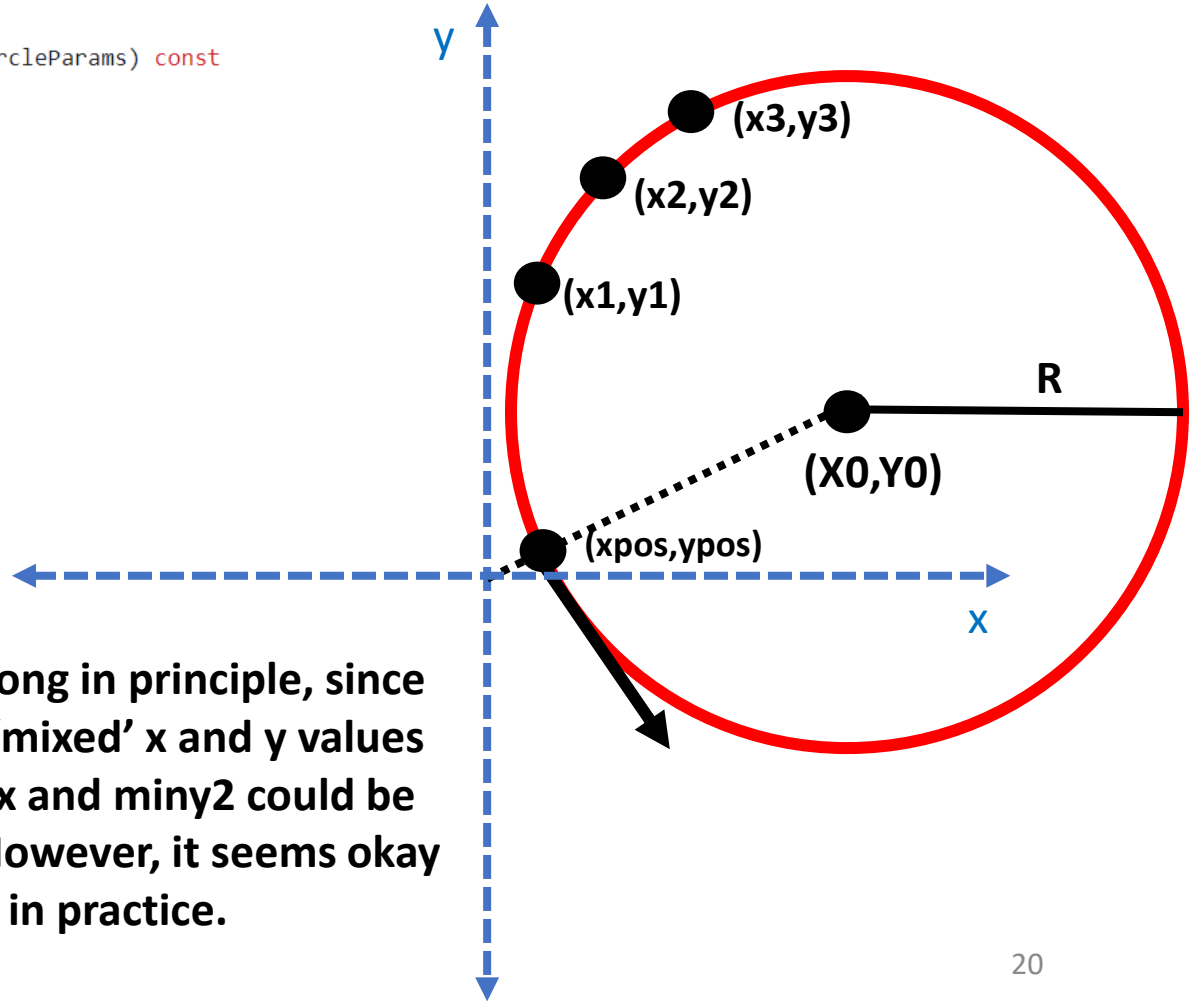
How are the seed positions calculated

```
134  std::pair<float, float> eicrecon::TrackSeeding::findRoot(std::tuple<float,float,float>& circleParams) const
135  {
136    const float R = std::get<0>(circleParams);
137    const float X0 = std::get<1>(circleParams);
138    const float Y0 = std::get<2>(circleParams);
139    const double miny = (std::sqrt(square(X0) * square(R) * square(Y0) + square(R)
140                        * pow(Y0,4)) + square(X0) * Y0 + pow(Y0, 3))
141      / (square(X0) + square(Y0));
142
143    const double miny2 = (-std::sqrt(square(X0) * square(R) * square(Y0) + square(R)
144                        * pow(Y0,4)) + square(X0) * Y0 + pow(Y0, 3))
145      / (square(X0) + square(Y0));
146
147    const double minx = std::sqrt(square(R) - square(miny - Y0)) + X0;
148    const double minx2 = -std::sqrt(square(R) - square(miny2 - Y0)) + X0;
149
150    /// Figure out which of the two roots is actually closer to the origin
151    const float x = ( std::abs(minx) < std::abs(minx2)) ? minx:minx2;
152    const float y = ( std::abs(miny) < std::abs(miny2)) ? miny:miny2;
153    return std::make_pair(x,y);
154  }
```



How are the seed positions calculated

```
134 std::pair<float, float> eicrecon::TrackSeeding::findRoot(std::tuple<float,float,float>& circleParams) const
135 {
136   const float R = std::get<0>(circleParams);
137   const float X0 = std::get<1>(circleParams);
138   const float Y0 = std::get<2>(circleParams);
139   const double miny = (std::sqrt(square(X0) * square(R) * square(Y0) + square(R)
140                       * pow(Y0,4)) + square(X0) * Y0 + pow(Y0, 3))
141     / (square(X0) + square(Y0));
142
143   const double miny2 = (-std::sqrt(square(X0) * square(R) * square(Y0) + square(R)
144                       * pow(Y0,4)) + square(X0) * Y0 + pow(Y0, 3))
145     / (square(X0) + square(Y0));
146
147   const double minx = std::sqrt(square(R) - square(miny - Y0)) + X0;
148   const double minx2 = -std::sqrt(square(R) - square(miny2 - Y0)) + X0;
149
150   /// Figure out which of the two roots is actually closer to the origin
151   const float x = ( std::abs(minx) < std::abs(minx2)) ? minx:minx2;
152   const float y = ( std::abs(miny) < std::abs(miny2)) ? miny:miny2;
153   return std::make_pair(x,y);
154 }
```



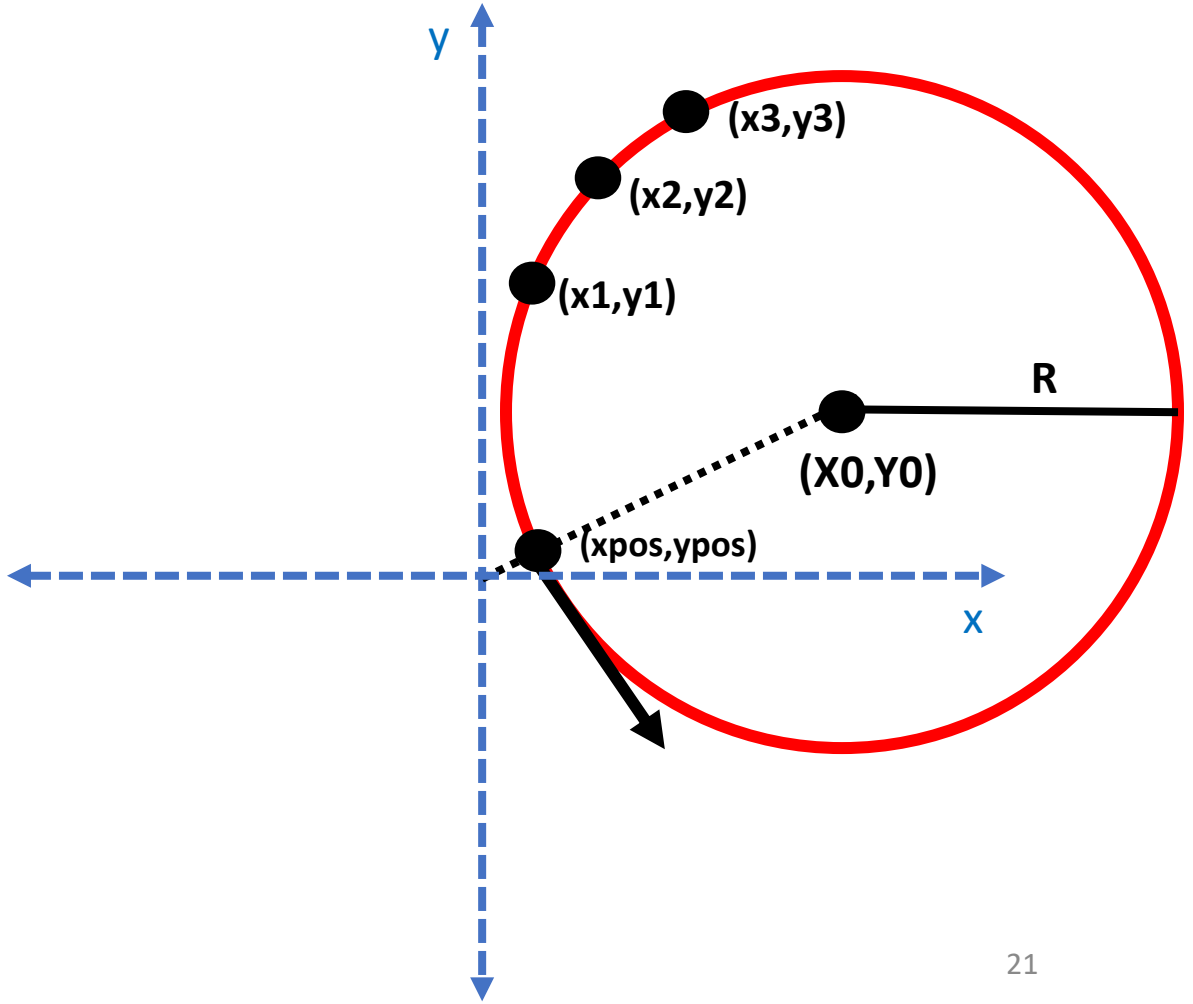
This is wrong in principle, since it allows ‘mixed’ x and y values (e.g. minx and miny2 could be chosen). However, it seems okay in practice.

How are the seed positions calculated

Maybe a simpler approach:

Use the geometry ‘fun fact’ that the line from the origin to (xpos,ypos) also passes through the center of the circle. Then we can easily calculate (xpos,ypos) by constructing the vector to (X0,Y0).

If X02 + Y0**2 > R**2:**
D = sqrt(X02 + Y0**2) – R**
U = (X0 , Y0) / sqrt(X02 + Y0**2)**
(xpos,ypos) = D*U
If X02 + Y0**2 < R**2:**
D = R - sqrt(X02 + Y0**2)**
U = -1.* (X0 , Y0) / sqrt(X02 + Y0**2)**
(xpos,ypos) = D*U



Next steps

- This study was done with the default set of seed parameters. Emma is working on a similar study using the updated seed finder parameters.