KFParticle: sPHENIX Workfest 2020 Summary

Yasser Corrales Morales, **Cameron Dean**, Jin Huang, Yuanjing Ji

With thanks to Ivan Kisel and Yuri Fisyak

KFParticle: preworkshop status and plans

- KFParticle allows users to build particles directly from position, momentum and covariance matrices
  - Particle = (x, y, z, px, py, pz, pE) Last part is optional
- KFParticle is well developed and in use by other collaborations
- KFParticle can then combine daughters into mothers (if you know what you're looking for)
  OR
- Use KFParticleFinder to construct a set of decays
- The challenge: Build KFParticle[Finder] into our framework and use on DSTs

Beta version of sPHENIX analysis package

```
/// KFParticle constructor
KFParticle_sPHENIX::KFParticle_sPHENIX():
    SubsysReco("KFPARTICLE"),
    m_dst_track()
{}

KFParticle_sPHENIX::~KFParticle_sPHENIX(){} /// KFParticle destructor

int KFParticle_sPHENIX::Init(PHCompositeNode *topNode)
{
  //if (Verbosity() > 5) { printf("Beginning Init in KFParticle_sPHENIX\n");}
  return 0;
}

int KFParticle_sPHENIX::process_event(PHCompositeNode *topNode)
{
    //if (Verbosity() > 5) { printf("Beginning process_event in KFParticle_sPHENIX\n");}
    makeParticle(topNode, -1);
    return Fun4AllReturnCodes::EVENT_OK;
}
```

Built in constructors, inits and process_event to pass package to server

Can extract required information from each track

```
std::vector<float> KFParticle_sPHENIX::getTrack(PHCompositeNode *topNode) /// Build 6x
{
    std::vector<float> particleVect;
    particleVect.push_back(m_dst_track->get_x());
    particleVect.push_back(m_dst_track->get_y());
    particleVect.push_back(m_dst_track->get_z());
    particleVect.push_back(m_dst_track->get_px());
    particleVect.push_back(m_dst_track->get_py());
    particleVect.push_back(m_dst_track->get_pz());

    return particleVect;
}

std::vector<std::vector<float>> KFParticle_sPHENIX::getCov(PHCompositeNode *topNode) /
{
   std::vector<std::vector<float>> covarianceMat;
   covarianceMat.resize(6, std::vector<float>(6, 0));
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; i < 6; j++)
        {
            covarianceMat[i][j] = m_dst_track->get_error(i, j);
        }
    }

    return covarianceMat;
}
```

```
KFParticle KFParticle_sPHENIX::makeParticle(PHCompositeNode *topNode, const int massHypothesis)
constraints
{
/*
#ifdef HomogeneousField
  KFParticle::SetField(-1.5);
#endif
  float point[3] = {0.f};
  float b[3] = {0.f};
*/
  const float *f_trackParameters = &KFParticle_sPHENIX::getTrack(topNode)[0];
  const float *f_trackCovariance = &KFParticle_sPHENIX::getCov(topNode)[0][0];

  KFPTrack kfp_track;

  kfp_track.SetParameters(f_trackParameters);
  kfp_track.SetCovarianceMatrix(f_trackCovariance);
  kfp_track.SetNDF(m_dst_track->get_ndf());
  kfp_track.SetChi2(m_dst_track->get_chisq());
  kfp_track.SetCharge(m_dst_track->get_charge());

  KFParticle particle(kfp_track, massHypothesis);
  //particle.GetFieldValue(point,b);

  return particle;
}
```

Use this to return a particle with charge and mass (if set)

```
float KFParticle_sPHENIX::eventDIRA(KFParticle particle, KFPVertex vertex)
{

  TMatrixD flightVector(3, 1), momVector(3, 1);
  flightVector(0,0) = particle.GetX() - vertex.GetX();
  flightVector(1,0) = particle.GetY() - vertex.GetY();
  flightVector(2,0) = particle.GetZ() - vertex.GetZ();

  momVector(0, 0) = particle.GetPx();
  momVector(1, 0) = particle.GetPy();
  momVector(2, 0) = particle.GetPz();

  TMatrixD momDotFD(1,1); //Calculate momentum dot flight distance
  momDotFD = TMatrixD(momVector, TMatrixD::kTransposeMult, flightVector);
  float f_momDotFD = momDotFD(0,0);

  TMatrixD sizeOfMom(1,1); //Calculates the length of the momentum vector
  sizeOfMom = TMatrixD(momVector, TMatrixD::kTransposeMult, momVector);
  float f_sizeOfMom = std::sqrt(sizeOfMom(0,0));

  TMatrixD sizeOfFD(1,1); //Calculates the length of the flight distance vector
  sizeOfFD = TMatrixD(flightVector, TMatrixD::kTransposeMult, flightVector);
  float f_sizeOfFD = std::sqrt(sizeOfFD(0,0));

  return f_momDotFD/(f_sizeOfMom*f_sizeOfFD); //returns the DIRA
}
```

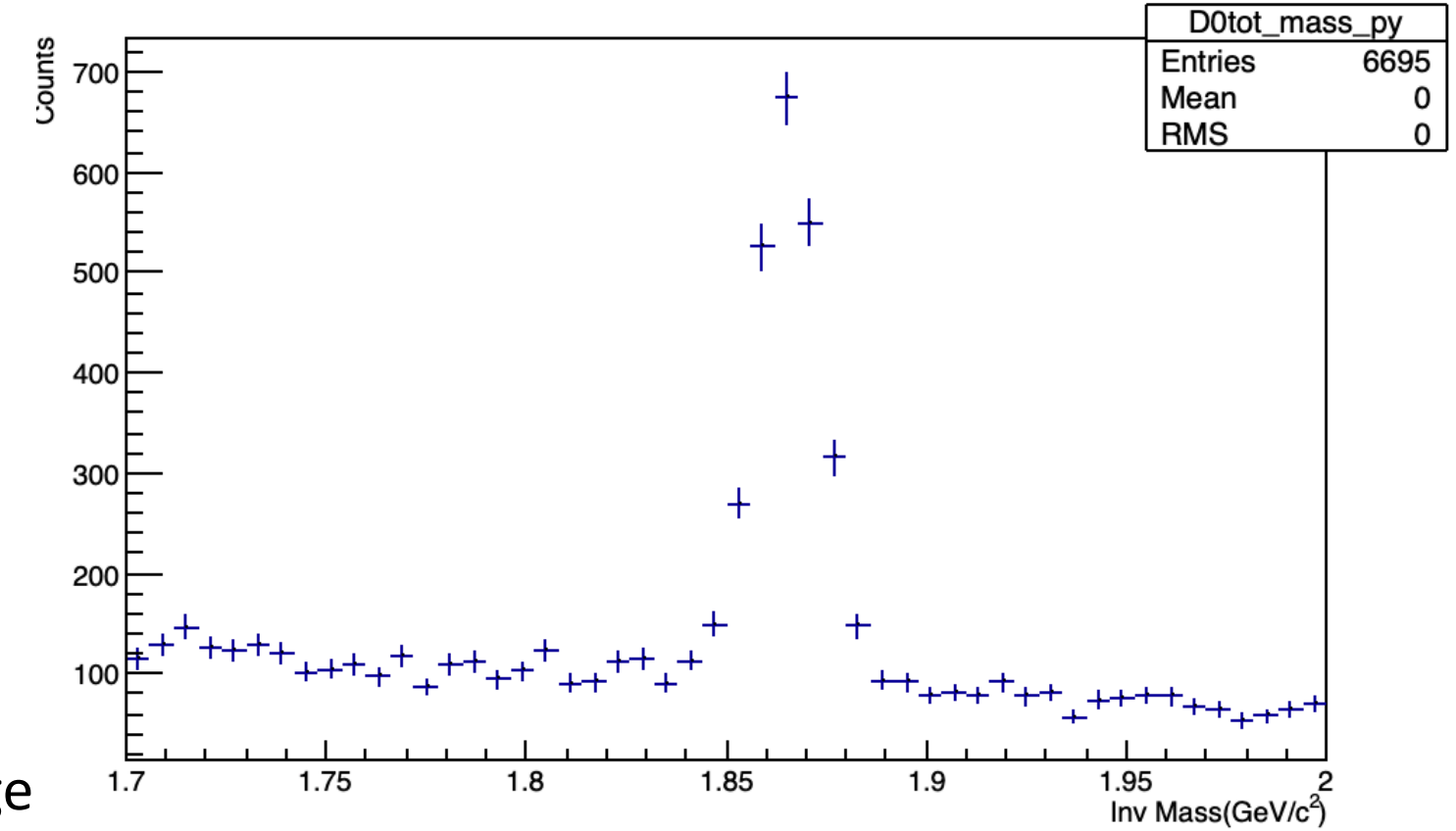
- Added in some extra functionality to improve analyses
- eventDIRA calculates the cosine of the angle between the flight direction and momentum, rejects bad mother candidates
- flightDistanceChi2 calculates the chi2 of the particles flight direction, can be used to reject events away from vertices or find daughter tracks
- getCombination is a very quick function to select particle combinations with chi-squared cuts, minimum daughter DCA requirements, mass windows and decay length windows (Requires thought as some selections can bias your sample)

```
float KFParticle_sPHENIX::flightDistanceChi2(KFParticle particle, KFPVertex vertex)
{
  TMatrixD flightVector(3, 1), flightDistanceCovariance(3, 3);

  flightVector(0,0) = particle.GetX() - vertex.GetX();
  flightVector(1,0) = particle.GetY() - vertex.GetY();
  flightVector(2,0) = particle.GetZ() - vertex.GetZ();

  // The vertex covariance matrix in KFParticle is missing a function to allow 2D assignement of covariance matrix. Correct
  //for (int i = 0; i < 3; i++) { for (int j = 0; j < 3; j++) { flightDistanceCovariance(i, j) = particle.GetCovariance(i,
  flightDistanceCovariance(0, 0) = particle.GetCovariance(0, 0) + vertex.GetCovariance(0);
  flightDistanceCovariance(1, 0) = flightDistanceCovariance(0, 1) = particle.GetCovariance(1, 0) + vertex.GetCovariance(1);
  flightDistanceCovariance(1, 1) = particle.GetCovariance(1, 1) + vertex.GetCovariance(2);
  flightDistanceCovariance(2, 0) = flightDistanceCovariance(0, 2) = particle.GetCovariance(2, 0) + vertex.GetCovariance(3);
  flightDistanceCovariance(2, 1) = flightDistanceCovariance(1, 2) = particle.GetCovariance(2, 1) + vertex.GetCovariance(4);
  flightDistanceCovariance(2, 2) = particle.GetCovariance(2, 2) + vertex.GetCovariance(5);


  TMatrixD anInverseMatrix(3,3);
  anInverseMatrix = flightDistanceCovariance.Invert();
  TMatrixD m_chi2Value(1,1);
  m_chi2Value = TMatrixD(flightVector, TMatrixD::kTransposeMult, anInverseMatrix*flightVector);
  return m_chi2Value(0,0);
}
```

```
KFParticle KFParticle_sPHENIX::getCombination(const KFParticle *vDaughters[], int nDaughters, const KFPVertex vertex,
                                              float fdChi2Req, float dcaReq, float diraReq,
                                              float minMass, float maxMass, float minLength, float maxLength)
{
   KFParticle mother;
   const KFParticle p_vertex(vertex);
   mother.Construct(vDaughters, nDaughters, &p_vertex, -1);

   float calculated_fdChi2 = KFParticle_sPHENIX::flightDistanceChi2(mother, vertex);
   float calculated_dira = KFParticle_sPHENIX::eventDIRA(mother, vertex);
   float calculated_mass = mother.GetMass();
   float calculated_length = mother.GetDecayLength();

   std::vector <float> calculated_dca;
   for (int i = 0; i < nDaughters; ++i)
   {
     for (int j = 0; j < nDaughters; ++j)
     {
       if (i != j) { calculated_dca.push_back(vDaughters[i]->GetDistanceFromParticleXY(*vDaughters[j])); }
     }
   }

  auto d_minmaxDCA = minmax_element(calculated_dca.begin(), calculated_dca.end());
  float f_maxDCA = *d_minmaxDCA.second;

   if (calculated_fdChi2 > fdChi2Req || calculated_dira < diraReq || f_maxDCA > dcaReq ||
       calculated_mass < minMass || calculated_mass > maxMass ||
       calculated_length < minLength || calculated_length > maxLength)
   { printf("Combination was not good\n"); }

   else printf("Found a good combination\n");

   return mother;
}
```

Output

- Jin produced a large sample to D->Kpi events
- Yuanjing was able to reconstruct the events with KFParticle only (no KFParticleFinder)



- Next steps
  1. Improve package
  2. Implement KFParticleFinder
  3. Make user friendly