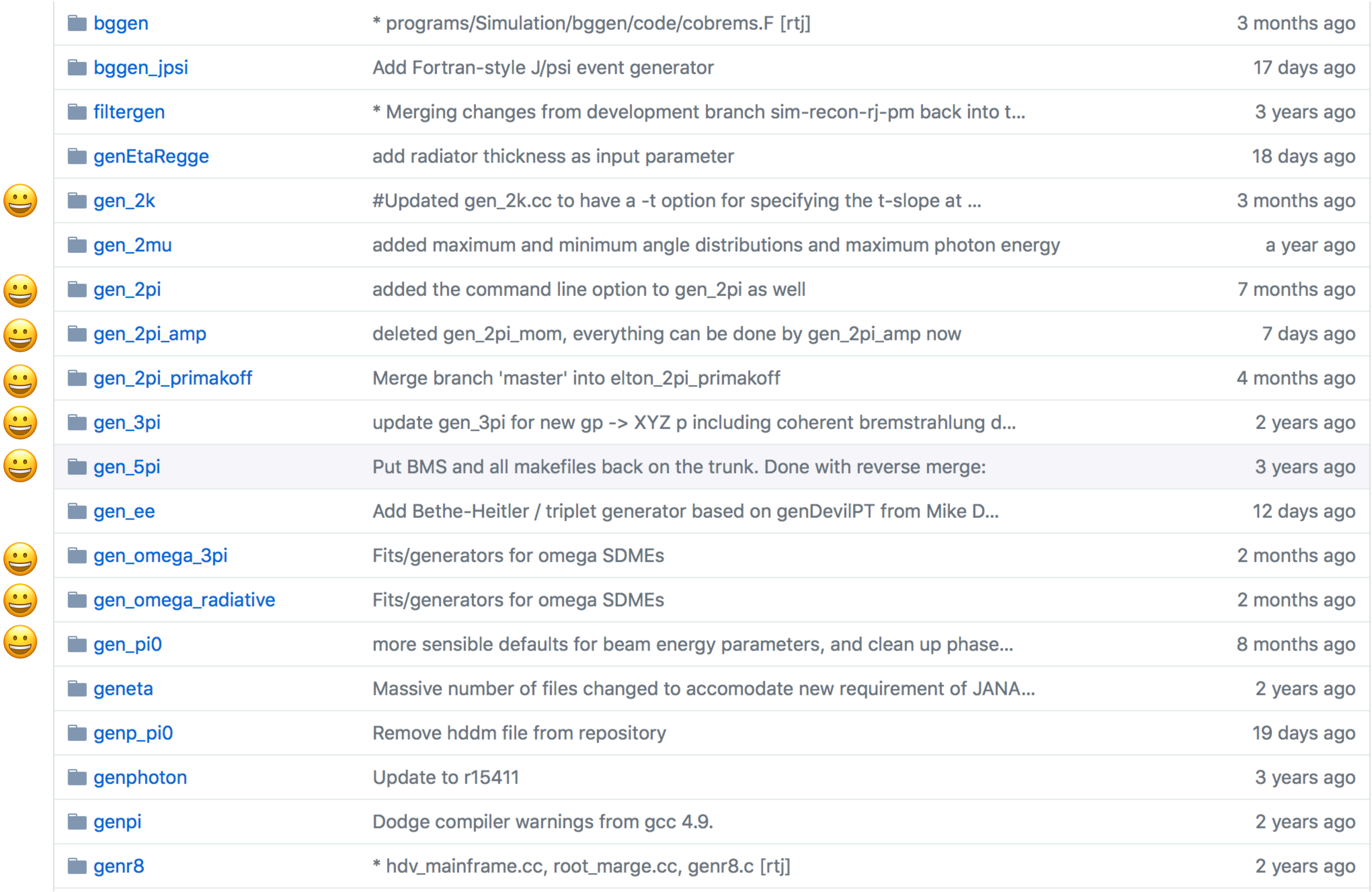
Thoughts on a common AmpTools generator

Justin Stevens
11/15/17

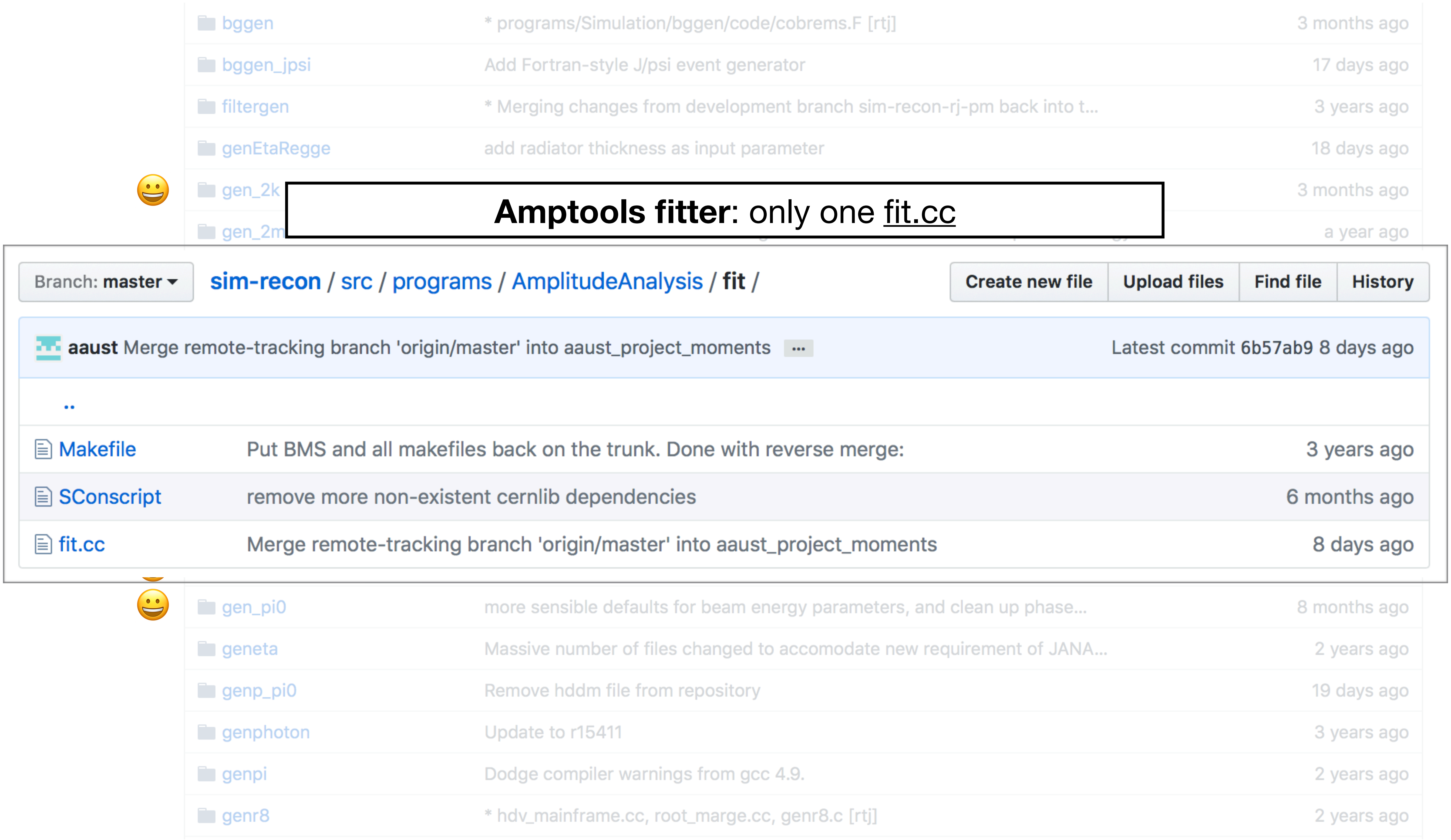
The problem…



	bggen	* programs/Simulation/bggen/code/cobrems.F [rtj]	3 months ago
	bggen_jpsi	Add Fortran-style J/psi event generator	17 days ago
	filtergen	* Merging changes from development branch sim-recon-rj-pm back into t...	3 years ago
	genEtaRegge	add radiator thickness as input parameter	18 days ago
😀	gen_2k	#Updated gen_2k.cc to have a -t option for specifying the t-slope at ...	3 months ago
	gen_2mu	added maximum and minimum angle distributions and maximum photon energy	a year ago
😀	gen_2pi	added the command line option to gen_2pi as well	7 months ago
😀	gen_2pi_amp	deleted gen_2pi_mom, everything can be done by gen_2pi_amp now	7 days ago
😀	gen_2pi_primakoff	Merge branch 'master' into elton_2pi_primakoff	4 months ago
😀	gen_3pi	update gen_3pi for new gp -> XYZ p including coherent bremstrahlung d...	2 years ago
😀	gen_5pi	Put BMS and all makefiles back on the trunk. Done with reverse merge:	3 years ago
	gen_ee	Add Bethe-Heitler / triplet generator based on genDevilPT from Mike D...	12 days ago
😀	gen_omega_3pi	Fits/generators for omega SDMEs	2 months ago
😀	gen_omega_radiative	Fits/generators for omega SDMEs	2 months ago
😀	gen_pi0	more sensible defaults for beam energy parameters, and clean up phase...	8 months ago
	geneta	Massive number of files changed to accomodate new requirement of JANA...	2 years ago
	genp_pi0	Remove hddm file from repository	19 days ago
	genphoton	Update to r15411	3 years ago
	genpi	Dodge compiler warnings from gcc 4.9.	2 years ago
	genr8	* hdv_mainframe.cc, root_marge.cc, genr8.c [rtj]	2 years ago

Amptools generators 😀: all use the same basic skeleton, but have 9 different generators (and growing…)

The problem…



Amptools fitter: only one fit.cc

bggen	* programs/Simulation/bggen/code/cobrems.F [rtj]	3 months ago
bggen_jpsi	Add Fortran-style J/psi event generator	17 days ago
filtergen	* Merging changes from development branch sim-recon-rj-pm back into t…	3 years ago
genEtaRegge	add radiator thickness as input parameter	18 days ago
😀 gen_2k		3 months ago
gen_2m		a year ago

Branch: master ▾ sim-recon / src / programs / AmplitudeAnalysis / fit /

aaust Merge remote-tracking branch 'origin/master' into aaust_project_moments … Latest commit 6b57ab9 8 days ago

..		
Makefile	Put BMS and all makefiles back on the trunk. Done with reverse merge:	3 years ago
SConscript	remove more non-existent cernlib dependencies	6 months ago
fit.cc	Merge remote-tracking branch 'origin/master' into aaust_project_moments	8 days ago

😀 gen_pi0	more sensible defaults for beam energy parameters, and clean up phase…	8 months ago
geneta	Massive number of files changed to accomodate new requirement of JANA…	2 years ago
genp_pi0	Remove hddm file from repository	19 days ago
genphoton	Update to r15411	3 years ago
genpi	Dodge compiler warnings from gcc 4.9.	2 years ago
genr8	* hdv_mainframe.cc, root_marge.cc, genr8.c [rtj]	2 years ago

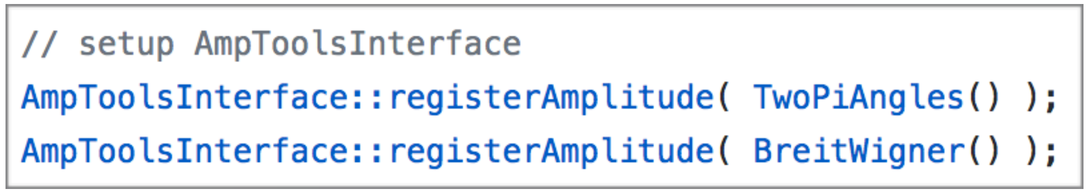
Amptools generators 😀: all use the same basic skeleton, but have 9 different generators (and growing…)

AmpTools generator layout

Configuration file and command line parameters

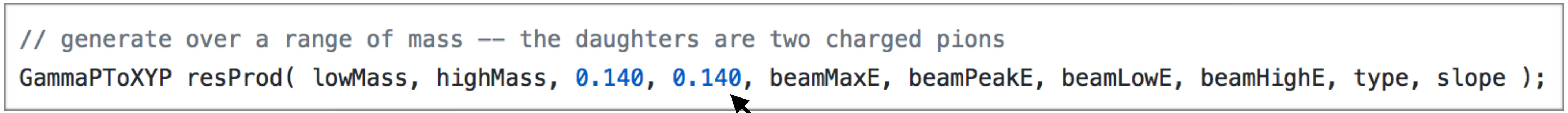
Amplitude inputs, beam energy, etc.

Define Amplitude

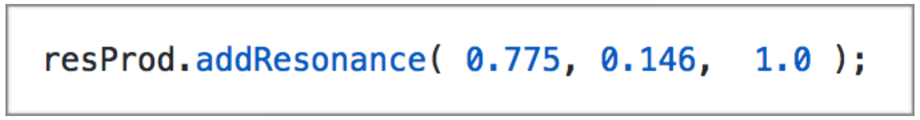


```
// setup AmpToolsInterface
AmpToolsInterface::registerAmplitude( TwoPiAngles() );
AmpToolsInterface::registerAmplitude( BreitWigner() );
```

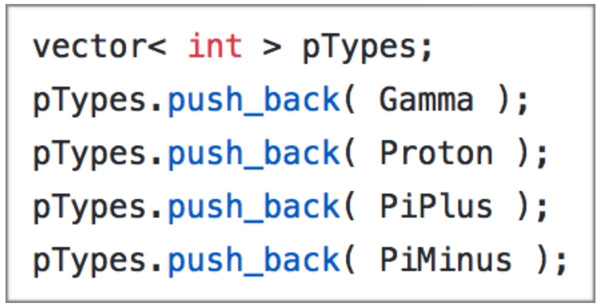
Phasespace generator



```
// generate over a range of mass -- the daughters are two charged pions
GammaPToXYP resProd( lowMass, highMass, 0.140, 0.140, beamMaxE, beamPeakE, beamLowE, beamHighE, type, slope );
```



```
resProd.addResonance( 0.775, 0.146,  1.0 );
```

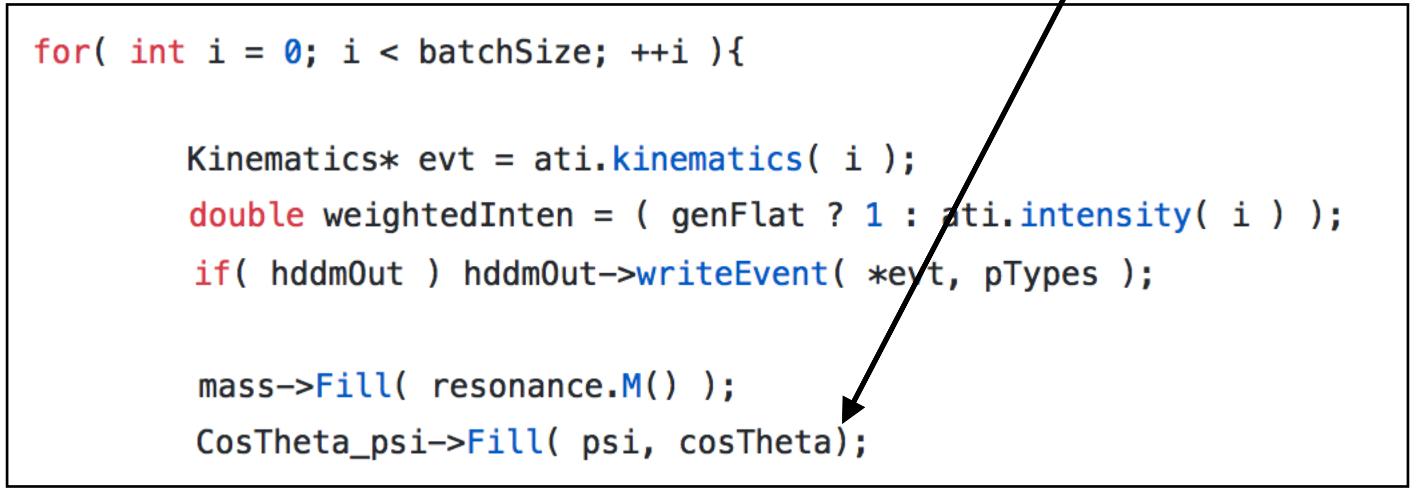


```
vector< int > pTypes;
pTypes.push_back( Gamma );
pTypes.push_back( Proton );
pTypes.push_back( PiPlus );
pTypes.push_back( PiMinus );
```

Can we write these in a general way for all AmpTools generators?

Generate events + write HDDM

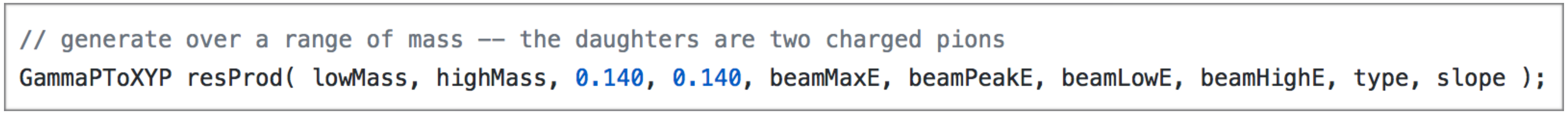
Monitoring histograms



```
for( int i = 0; i < batchSize; ++i ){

        Kinematics* evt = ati.kinematics( i );
        double weightedInten = ( genFlat ? 1 : ati.intensity( i ) );
        if( hddmOut ) hddmOut->writeEvent( *evt, pTypes );

        mass->Fill( resonance.M() );
        CosTheta_psi->Fill( psi, cosTheta);
```

Generator duplication

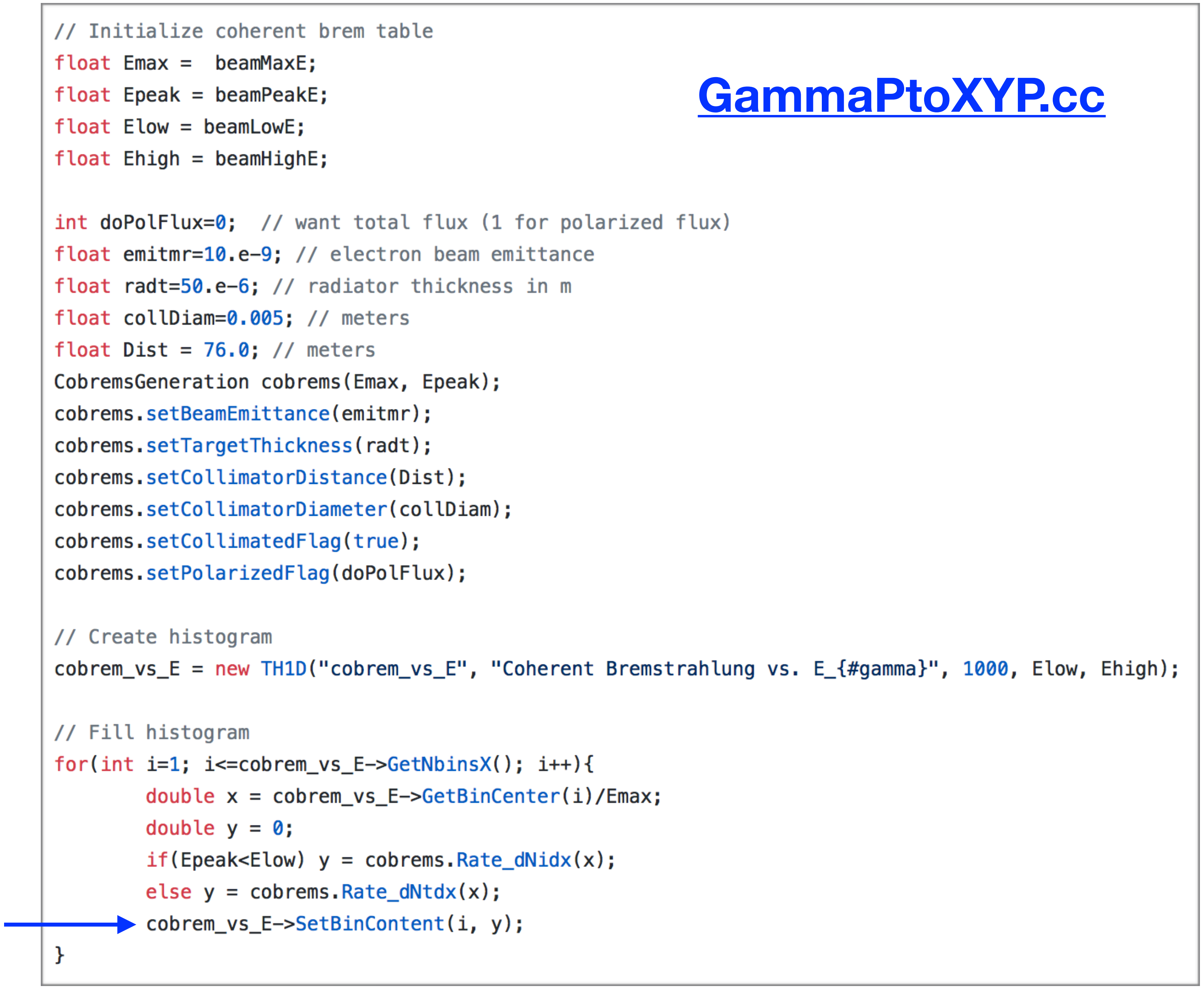
Phasespace generator



```
// generate over a range of mass -- the daughters are two charged pions
GammaPToXYP resProd( lowMass, highMass, 0.140, 0.140, beamMaxE, beamPeakE, beamLowE, beamHighE, type, slope );
```

GammaPtoXP, GammaPtoXYP, GammaPtoXYZP, etc.

All duplicate this code and set beam properties from command line



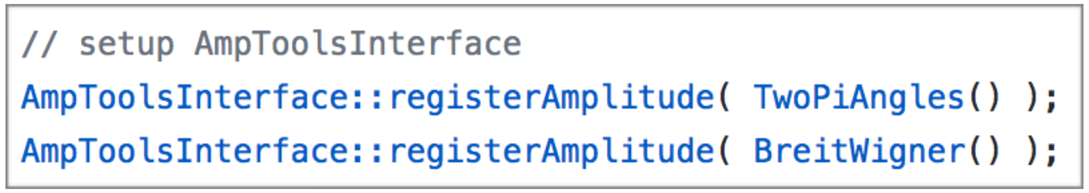
GammaPtoXYP.cc

```
// Initialize coherent brem table
float Emax =  beamMaxE;
float Epeak = beamPeakE;
float Elow = beamLowE;
float Ehigh = beamHighE;

int doPolFlux=0;  // want total flux (1 for polarized flux)
float emitmr=10.e-9; // electron beam emittance
float radt=50.e-6; // radiator thickness in m
float collDiam=0.005; // meters
float Dist = 76.0; // meters
CobremsGeneration cobrems(Emax, Epeak);
cobrems.setBeamEmittance(emitmr);
cobrems.setTargetThickness(radt);
cobrems.setCollimatorDistance(Dist);
cobrems.setCollimatorDiameter(collDiam);
cobrems.setCollimatedFlag(true);
cobrems.setPolarizedFlag(doPolFlux);

// Create histogram
cobrem_vs_E = new TH1D("cobrem_vs_E", "Coherent Bremstrahlung vs. E_{#gamma}", 1000, Elow, Ehigh);

// Fill histogram
for(int i=1; i<=cobrem_vs_E->GetNbinsX(); i++){
        double x = cobrem_vs_E->GetBinCenter(i)/Emax;
        double y = 0;
        if(Epeak<Elow) y = cobrems.Rate_dNidx(x);
        else y = cobrems.Rate_dNtdx(x);
        cobrem_vs_E->SetBinContent(i, y);
}
```

Flux vs E_{γ} to generate events →

Generator duplication

Define Amplitude

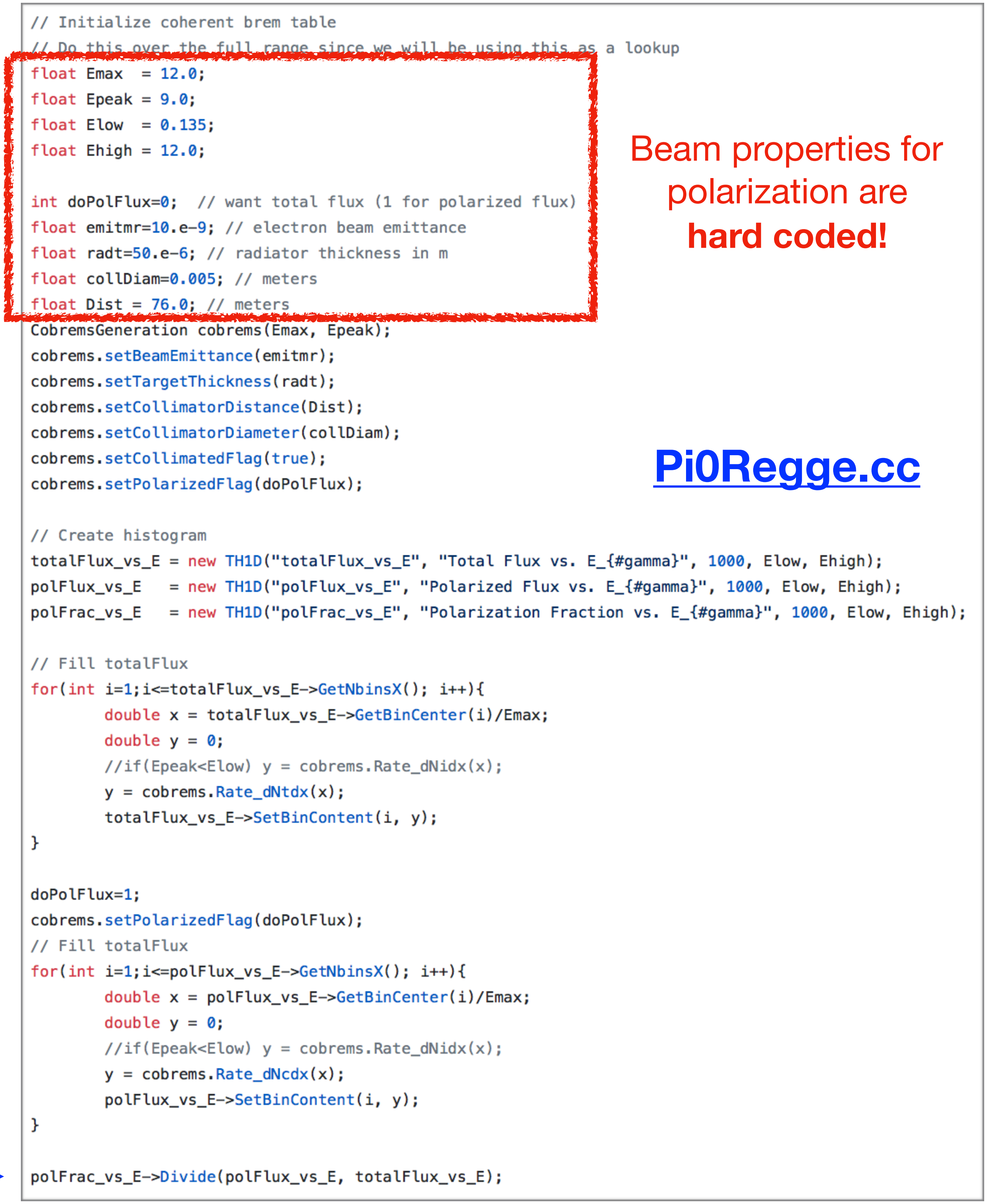


```
// setup AmpToolsInterface
AmpToolsInterface::registerAmplitude( TwoPiAngles() );
AmpToolsInterface::registerAmplitude( BreitWigner() );
```

Polarization in amplitude definition

Pi0Regge,
TwoPSAngles,
TwoPSHelicity,
TwoPiAngles,
TwoPiAnglesRadiative
TwoPiAngles_amp
TwoPiAngles_primakoff,
ThreePiAnglesSchilling
etc.

Polarization vs E_γ for amplitude →



```
// Initialize coherent brem table
// Do this over the full range since we will be using this as a lookup
float Emax  = 12.0;
float Epeak = 9.0;
float Elow  = 0.135;
float Ehigh = 12.0;

int doPolFlux=0;  // want total flux (1 for polarized flux)
float emitmr=10.e-9; // electron beam emittance
float radt=50.e-6; // radiator thickness in m
float collDiam=0.005; // meters
float Dist = 76.0; // meters
CobremsGeneration cobrems(Emax, Epeak);
cobrems.setBeamEmittance(emitmr);
cobrems.setTargetThickness(radt);
cobrems.setCollimatorDistance(Dist);
cobrems.setCollimatorDiameter(collDiam);
cobrems.setCollimatedFlag(true);
cobrems.setPolarizedFlag(doPolFlux);

// Create histogram
totalFlux_vs_E = new TH1D("totalFlux_vs_E", "Total Flux vs. E_{#gamma}", 1000, Elow, Ehigh);
polFlux_vs_E   = new TH1D("polFlux_vs_E", "Polarized Flux vs. E_{#gamma}", 1000, Elow, Ehigh);
polFrac_vs_E   = new TH1D("polFrac_vs_E", "Polarization Fraction vs. E_{#gamma}", 1000, Elow, Ehigh);

// Fill totalFlux
for(int i=1;i<=totalFlux_vs_E->GetNbinsX(); i++){
        double x = totalFlux_vs_E->GetBinCenter(i)/Emax;
        double y = 0;
        //if(Epeak<Elow) y = cobrems.Rate_dNidx(x);
        y = cobrems.Rate_dNtdx(x);
        totalFlux_vs_E->SetBinContent(i, y);
}

doPolFlux=1;
cobrems.setPolarizedFlag(doPolFlux);
// Fill totalFlux
for(int i=1;i<=polFlux_vs_E->GetNbinsX(); i++){
        double x = polFlux_vs_E->GetBinCenter(i)/Emax;
        double y = 0;
        //if(Epeak<Elow) y = cobrems.Rate_dNidx(x);
        y = cobrems.Rate_dNcdx(x);
        polFlux_vs_E->SetBinContent(i, y);
}

polFrac_vs_E->Divide(polFlux_vs_E, totalFlux_vs_E);
```

Beam properties for polarization are **hard coded!**

Pi0Regge.cc

Proposed generator layout

Configuration file and command line parameters

Amplitude inputs, beam energy, etc.

Define beam properties

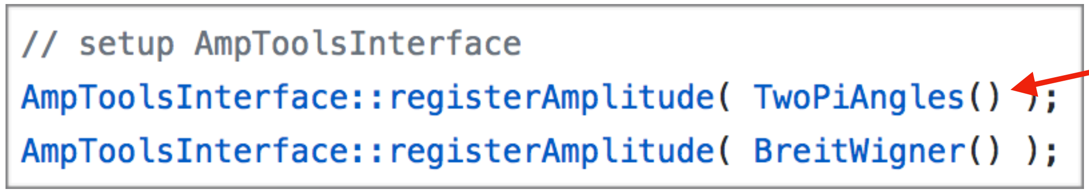
New class which creates histogram of Flux and Polarization vs E_γ

-CobremsGenerator using beam parameters from config file

-Local ROOT file: PS flux or TPOL polarization histograms (ie. from data)

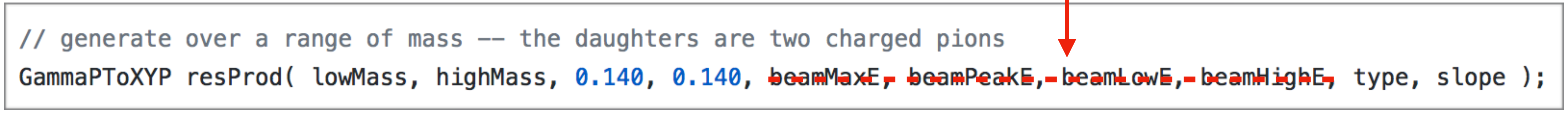
Note: this class could be used for all generators (eg. bggen, etc.)

Define Amplitude

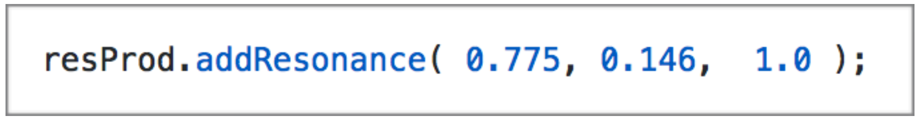


```
// setup AmpToolsInterface
AmpToolsInterface::registerAmplitude( TwoPiAngles() );
AmpToolsInterface::registerAmplitude( BreitWigner() );
```

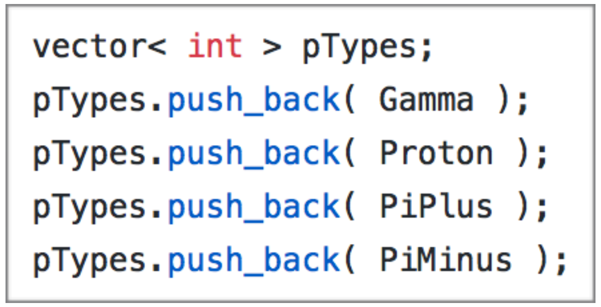
Phasespace generator



```
// generate over a range of mass -- the daughters are two charged pions
GammaPToXYP resProd( lowMass, highMass, 0.140, 0.140, ~~beamMaxE, beamPeakE, beamLowE, beamHighE,~~ type, slope );
```



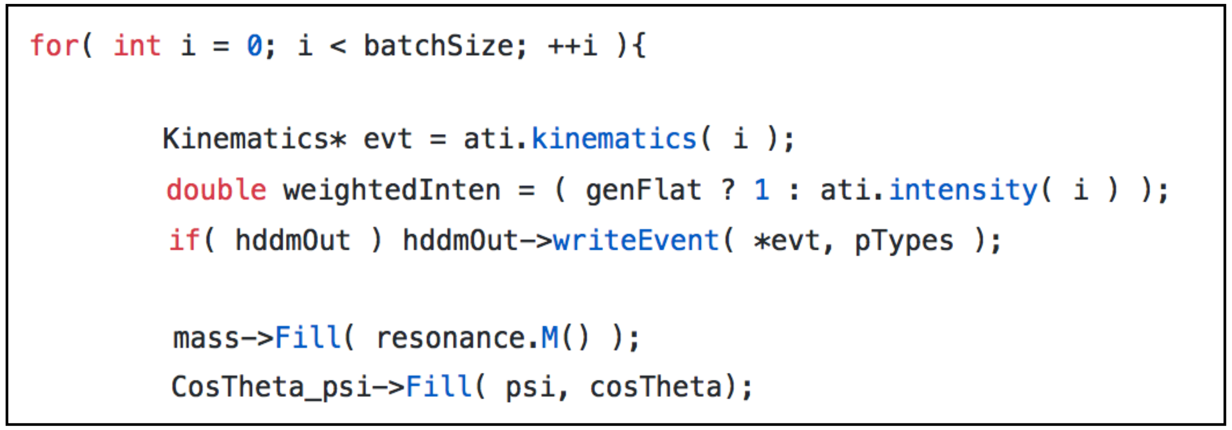
```
resProd.addResonance( 0.775, 0.146,  1.0 );
```



```
vector< int > pTypes;
pTypes.push_back( Gamma );
pTypes.push_back( Proton );
pTypes.push_back( PiPlus );
pTypes.push_back( PiMinus );
```

Generate events + write HDDM

Monitoring histograms



```
for( int i = 0; i < batchSize; ++i ){

        Kinematics* evt = ati.kinematics( i );
        double weightedInten = ( genFlat ? 1 : ati.intensity( i ) );
        if( hddmOut ) hddmOut->writeEvent( *evt, pTypes );

        mass->Fill( resonance.M() );
        CosTheta_psi->Fill( psi, cosTheta);
```

Proposed generator layout

Configuration file and command line parameters

Amplitude inputs, beam energy, etc.

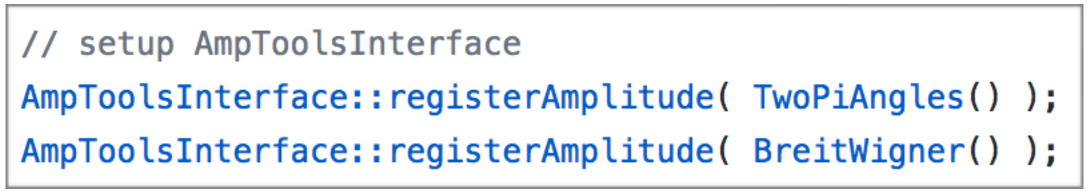
Define beam properties

New class which creates histogram of Flux and Polarization vs E$_\gamma$

-CobremsGenerator using beam parameters from config file

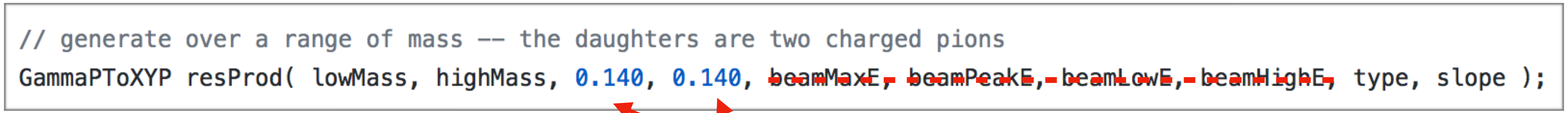
-Local ROOT file: PS flux or TPOL polarization histograms (ie. from data)

Define Amplitude

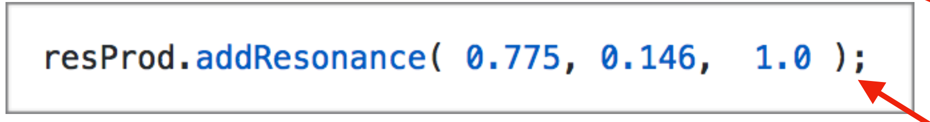


```
// setup AmpToolsInterface
AmpToolsInterface::registerAmplitude( TwoPiAngles() );
AmpToolsInterface::registerAmplitude( BreitWigner() );
```

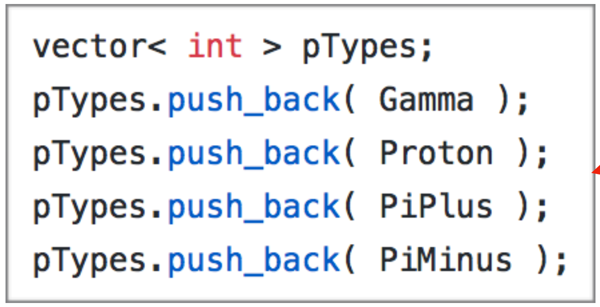
Phasespace generator



```
// generate over a range of mass -- the daughters are two charged pions
GammaPToXYP resProd( lowMass, highMass, 0.140, 0.140, ~~beamMaxE, beamPeakE, beamLowE, beamHighE,~~ type, slope );
```



```
resProd.addResonance( 0.775, 0.146,  1.0 );
```



```
vector< int > pTypes;
pTypes.push_back( Gamma );
pTypes.push_back( Proton );
pTypes.push_back( PiPlus );
pTypes.push_back( PiMinus );
```

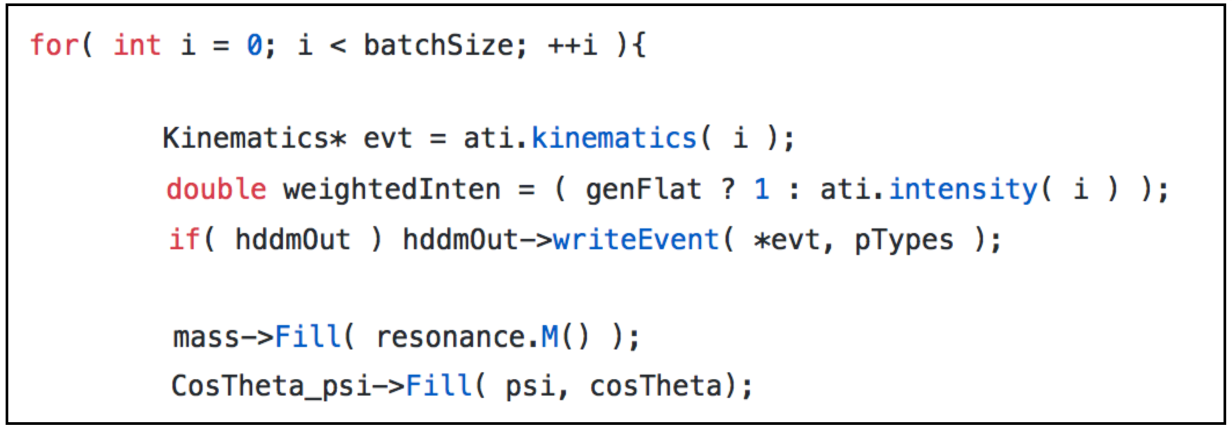
Parse config file for Phasespace generator choice and parameters (particleType.h):

gen_2pi.cfg

```
reaction Pi+Pi- gamma Pi+ Pi- p
amplitude Pi+Pi-::xpol::rhoS BreitWigner 0.775 0.146 1 2 3
```

Generate events + write HDDM

Monitoring histograms



```
for( int i = 0; i < batchSize; ++i ){

        Kinematics* evt = ati.kinematics( i );
        double weightedInten = ( genFlat ? 1 : ati.intensity( i ) );
        if( hddmOut ) hddmOut->writeEvent( *evt, pTypes );

        mass->Fill( resonance.M() );
        CosTheta_psi->Fill( psi, cosTheta);
```

Proposed generator layout

Configuration file and command line parameters

Amplitude inputs, beam energy, etc.

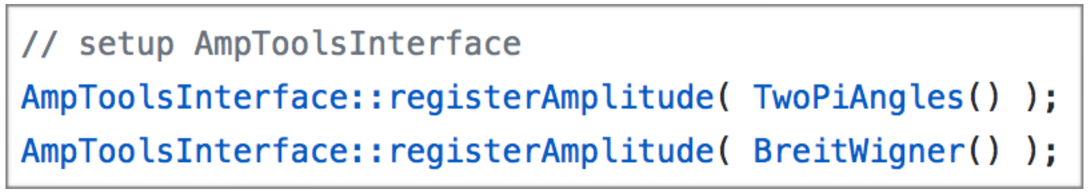
Define beam properties

New class which creates histogram of Flux and Polarization vs E_γ

-CobremsGenerator using beam parameters from config file

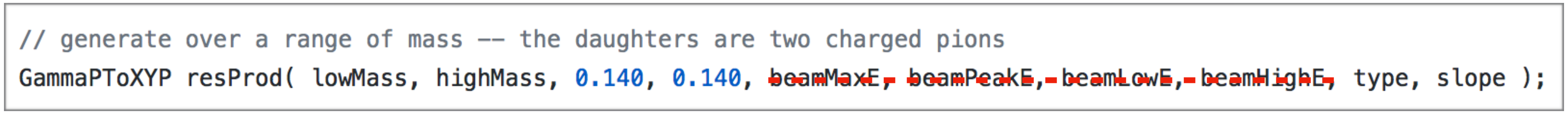
-Local ROOT file: PS flux or TPOL polarization histograms (ie. from data)

Define Amplitude

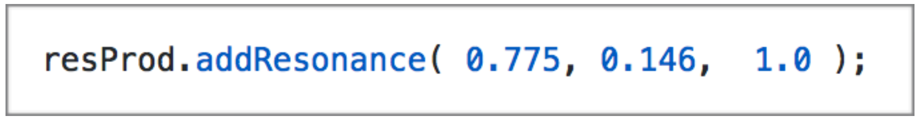


```
// setup AmpToolsInterface
AmpToolsInterface::registerAmplitude( TwoPiAngles() );
AmpToolsInterface::registerAmplitude( BreitWigner() );
```

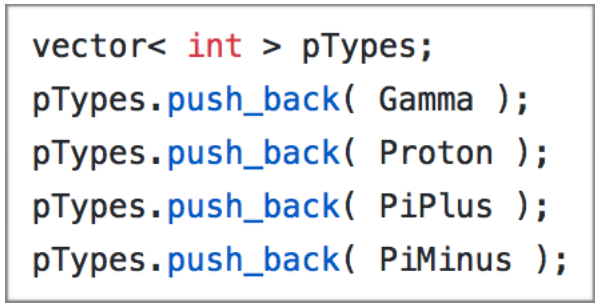
Phasespace generator



```
// generate over a range of mass -- the daughters are two charged pions
GammaPToXYP resProd( lowMass, highMass, 0.140, 0.140, ~~beamMaxE, beamPeakE, beamLowE, beamHighE,~~ type, slope );
```



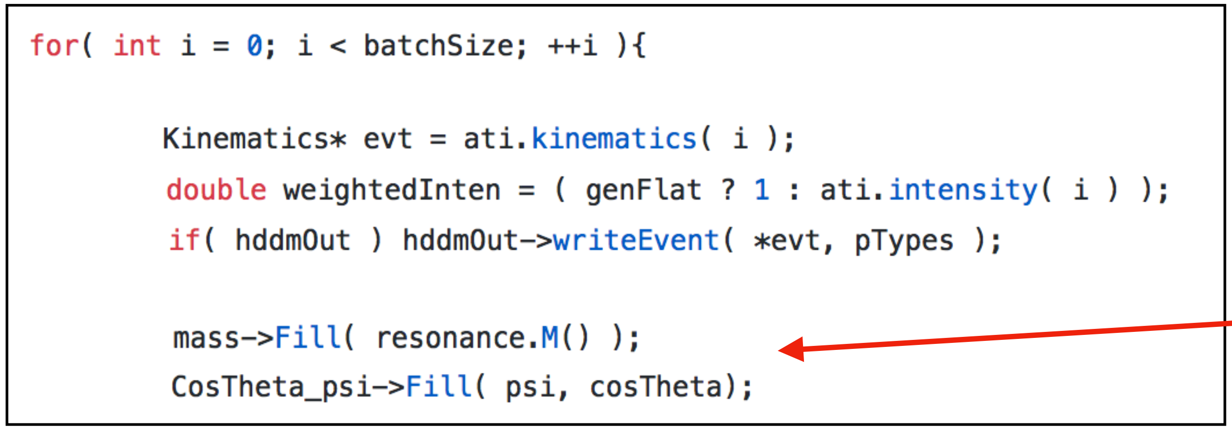
```
resProd.addResonance( 0.775, 0.146,  1.0 );
```



```
vector< int > pTypes;
pTypes.push_back( Gamma );
pTypes.push_back( Proton );
pTypes.push_back( PiPlus );
pTypes.push_back( PiMinus );
```

Generate events + write HDDM

Monitoring histograms



```
for( int i = 0; i < batchSize; ++i ){

        Kinematics* evt = ati.kinematics( i );
        double weightedInten = ( genFlat ? 1 : ati.intensity( i ) );
        if( hddmOut ) hddmOut->writeEvent( *evt, pTypes );

        mass->Fill( resonance.M() );
        CosTheta_psi->Fill( psi, cosTheta);
```

User writes custom class to create and fill unique histograms for their generator

Common amplitude definitions?

Many different amplitudes for two-particle final state

TwoPSAngles, TwoPSHelicity, TwoPiAngles, TwoPiAngles_primakoff, TwoPiAngles_amp, etc.

Similarities:

-Same final state with two Pseudoscalars

-Begs for a common class for at least the kinematic quantities which are the same

Possible differences in gen/fit due to:

-Different frame choice (config file parameter?)

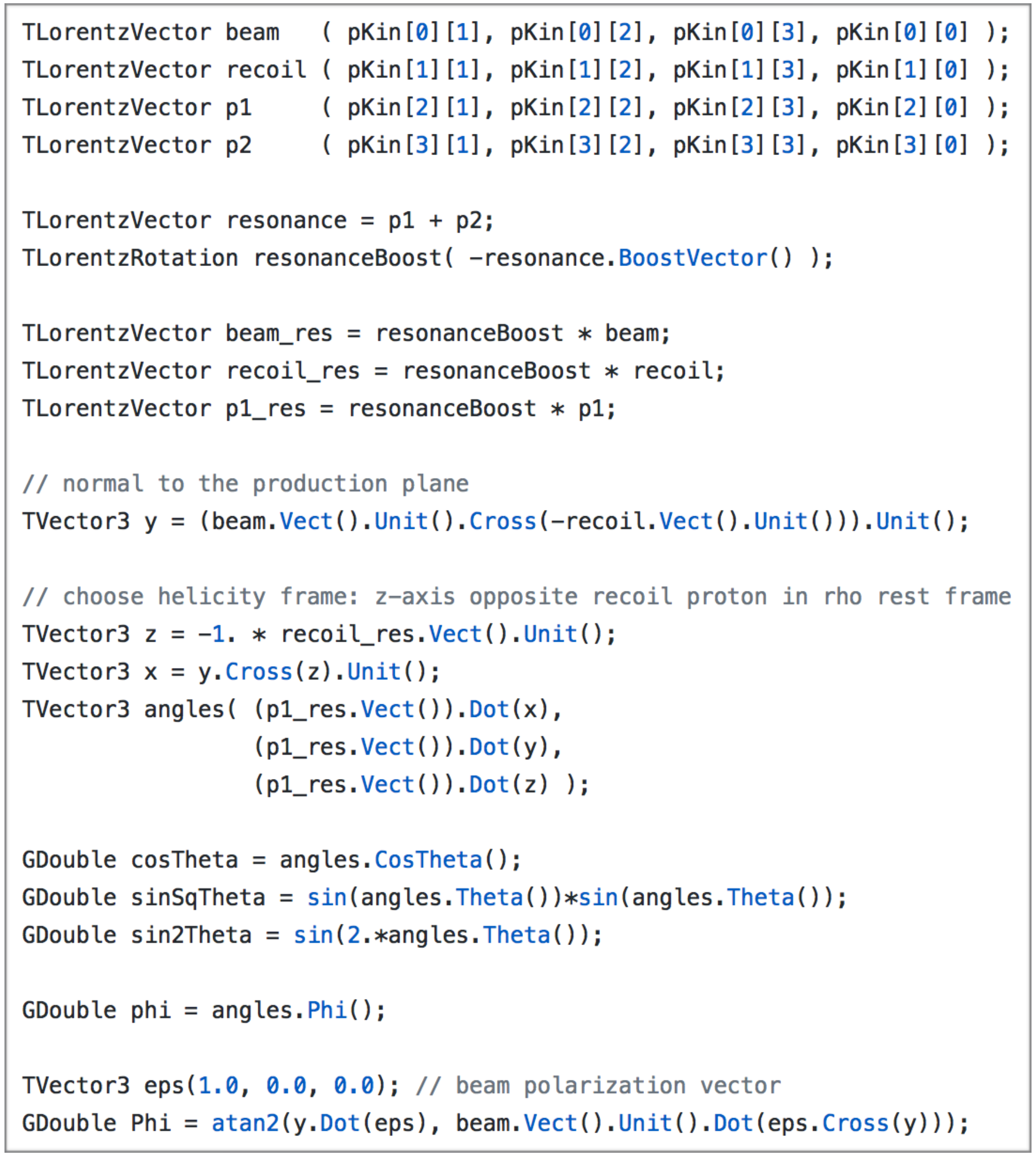
-just a choice of the z-axis

-Different parameters

-Can we have a single intensity function for the SDMEs (in 2 pi or 3 pi)?

Similar functionality needed in “PlotGenerator” codes for plotting fit results

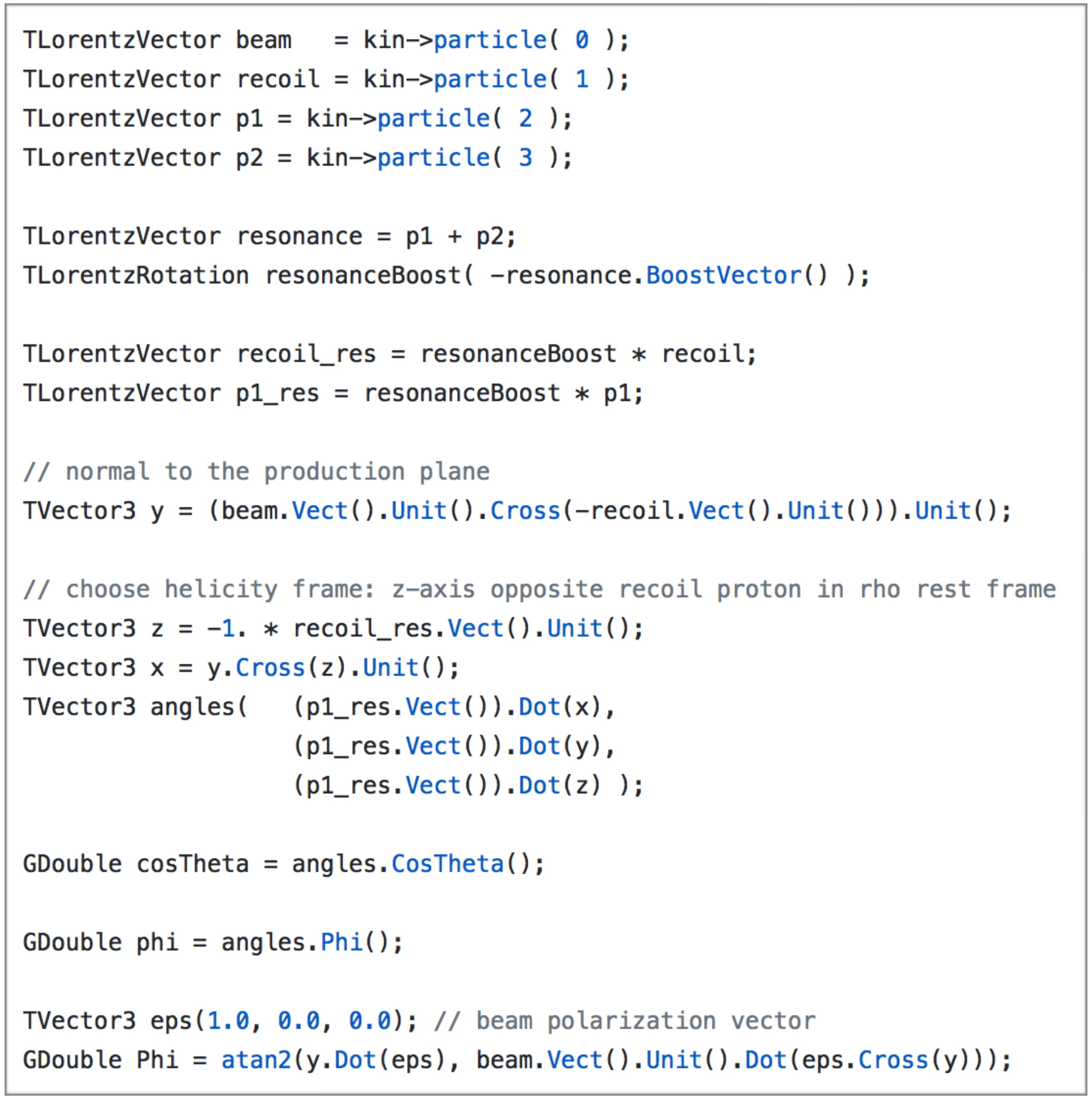
TwoPiAngles.cc



```
TLorentzVector beam   ( pKin[0][1], pKin[0][2], pKin[0][3], pKin[0][0] );
TLorentzVector recoil ( pKin[1][1], pKin[1][2], pKin[1][3], pKin[1][0] );
TLorentzVector p1     ( pKin[2][1], pKin[2][2], pKin[2][3], pKin[2][0] );
TLorentzVector p2     ( pKin[3][1], pKin[3][2], pKin[3][3], pKin[3][0] );

TLorentzVector resonance = p1 + p2;
TLorentzRotation resonanceBoost( -resonance.BoostVector() );

TLorentzVector beam_res = resonanceBoost * beam;
TLorentzVector recoil_res = resonanceBoost * recoil;
TLorentzVector p1_res = resonanceBoost * p1;

// normal to the production plane
TVector3 y = (beam.Vect().Unit().Cross(-recoil.Vect().Unit())).Unit();

// choose helicity frame: z-axis opposite recoil proton in rho rest frame
TVector3 z = -1. * recoil_res.Vect().Unit();
TVector3 x = y.Cross(z).Unit();
TVector3 angles( (p1_res.Vect()).Dot(x),
                 (p1_res.Vect()).Dot(y),
                 (p1_res.Vect()).Dot(z) );

GDouble cosTheta = angles.CosTheta();
GDouble sinSqTheta = sin(angles.Theta())*sin(angles.Theta());
GDouble sin2Theta = sin(2.*angles.Theta());

GDouble phi = angles.Phi();

TVector3 eps(1.0, 0.0, 0.0); // beam polarization vector
GDouble Phi = atan2(y.Dot(eps), beam.Vect().Unit().Dot(eps.Cross(y)));
```

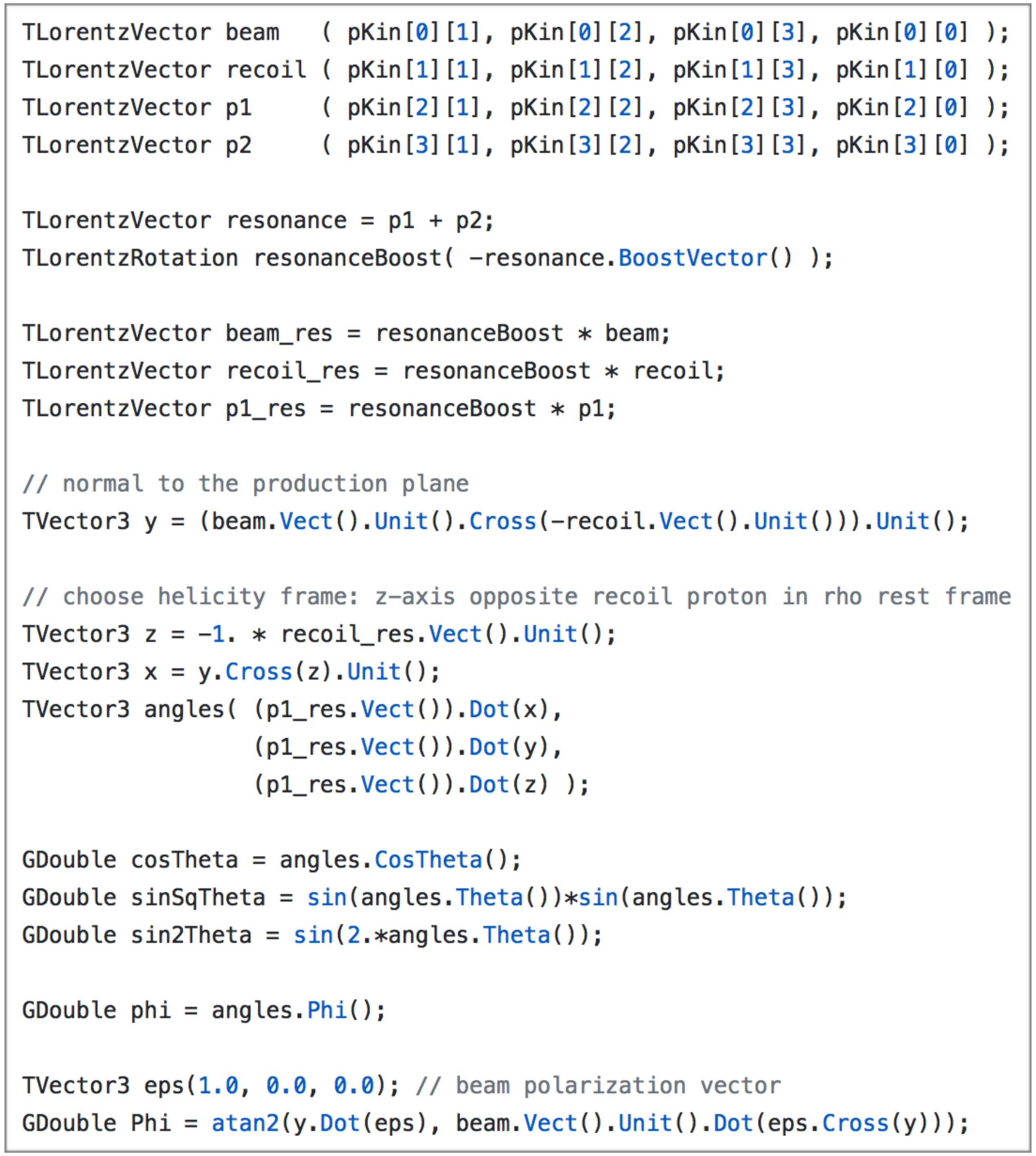
Common amplitude definitions?

TwoPiPlotGenerator.cc



```
TLorentzVector beam   = kin->particle( 0 );
TLorentzVector recoil = kin->particle( 1 );
TLorentzVector p1 = kin->particle( 2 );
TLorentzVector p2 = kin->particle( 3 );

TLorentzVector resonance = p1 + p2;
TLorentzRotation resonanceBoost( -resonance.BoostVector() );

TLorentzVector recoil_res = resonanceBoost * recoil;
TLorentzVector p1_res = resonanceBoost * p1;

// normal to the production plane
TVector3 y = (beam.Vect().Unit().Cross(-recoil.Vect().Unit())).Unit();

// choose helicity frame: z-axis opposite recoil proton in rho rest frame
TVector3 z = -1. * recoil_res.Vect().Unit();
TVector3 x = y.Cross(z).Unit();
TVector3 angles(   (p1_res.Vect()).Dot(x),
                   (p1_res.Vect()).Dot(y),
                   (p1_res.Vect()).Dot(z) );

GDouble cosTheta = angles.CosTheta();

GDouble phi = angles.Phi();

TVector3 eps(1.0, 0.0, 0.0); // beam polarization vector
GDouble Phi = atan2(y.Dot(eps), beam.Vect().Unit().Dot(eps.Cross(y)));
```

TwoPiAngles.cc



```
TLorentzVector beam   ( pKin[0][1], pKin[0][2], pKin[0][3], pKin[0][0] );
TLorentzVector recoil ( pKin[1][1], pKin[1][2], pKin[1][3], pKin[1][0] );
TLorentzVector p1     ( pKin[2][1], pKin[2][2], pKin[2][3], pKin[2][0] );
TLorentzVector p2     ( pKin[3][1], pKin[3][2], pKin[3][3], pKin[3][0] );

TLorentzVector resonance = p1 + p2;
TLorentzRotation resonanceBoost( -resonance.BoostVector() );

TLorentzVector beam_res = resonanceBoost * beam;
TLorentzVector recoil_res = resonanceBoost * recoil;
TLorentzVector p1_res = resonanceBoost * p1;

// normal to the production plane
TVector3 y = (beam.Vect().Unit().Cross(-recoil.Vect().Unit())).Unit();

// choose helicity frame: z-axis opposite recoil proton in rho rest frame
TVector3 z = -1. * recoil_res.Vect().Unit();
TVector3 x = y.Cross(z).Unit();
TVector3 angles( (p1_res.Vect()).Dot(x),
                 (p1_res.Vect()).Dot(y),
                 (p1_res.Vect()).Dot(z) );

GDouble cosTheta = angles.CosTheta();
GDouble sinSqTheta = sin(angles.Theta())*sin(angles.Theta());
GDouble sin2Theta = sin(2.*angles.Theta());

GDouble phi = angles.Phi();

TVector3 eps(1.0, 0.0, 0.0); // beam polarization vector
GDouble Phi = atan2(y.Dot(eps), beam.Vect().Unit().Dot(eps.Cross(y)));
```

Similar functionality needed in “PlotGenerator” codes for plotting fit results

Summary

- AmpTools generators are proliferating 😀, but some common infrastructure would help streamline these
  - **Goal:** converge on a single “gen_amp.cc” which parses config file to setup generator (reaction, particle types, masses, etc.)
  - Common creation of beam histograms for consistent phasespace generation (flux) and amplitude calculation (polarization)
- A library of common kinematic calculations in amplitude definitions (eg. angles in two pseudoscalar production) would reduce code duplication
  - There is probably a clever, more general set of decay angle calculations that could extend this to higher multiplicity reactions
- **Reminder:** this all has to remain consistent with a single fit.cc which uses the same amplitude definitions