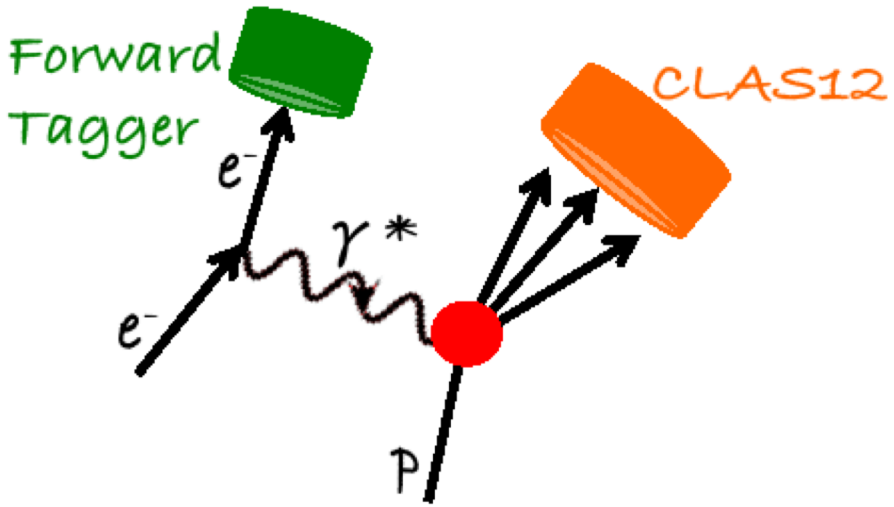
Quasi-real photoproduction in CLAS12-MesonEx

CLAS12-MesonEx

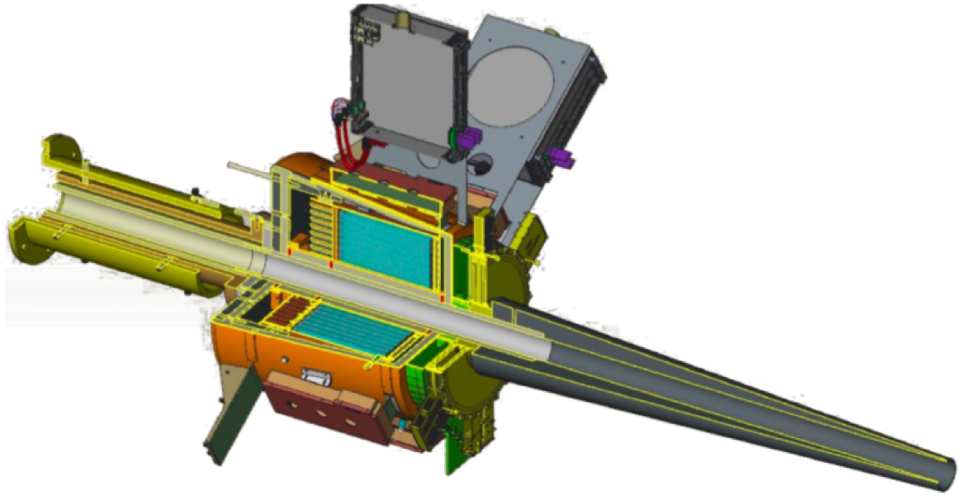
Meson Spectroscopy program with low-Q^2 electroproduction (quasi-real photoproduction)

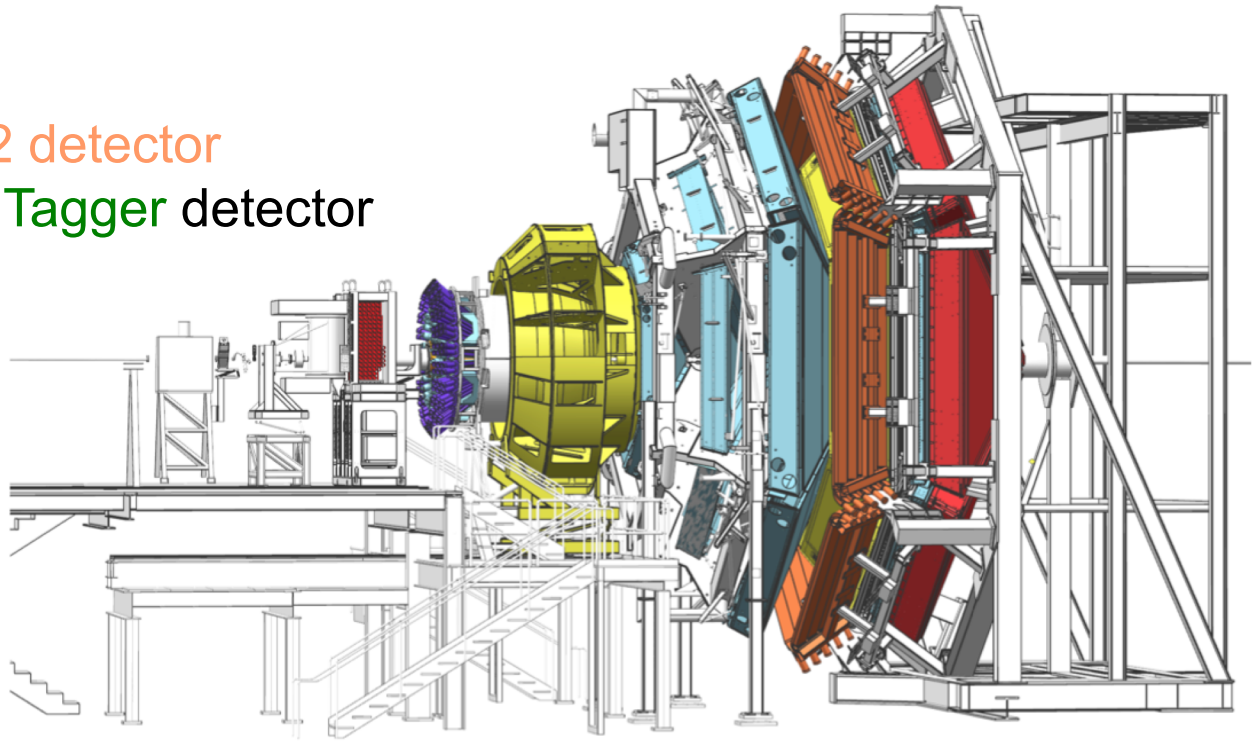


Advantages

- A
- B
- C

- Final state hadrons measured with the CLAS12 detector
- Scattered electron measured with the Forward Tagger detector





CLAS12 detector

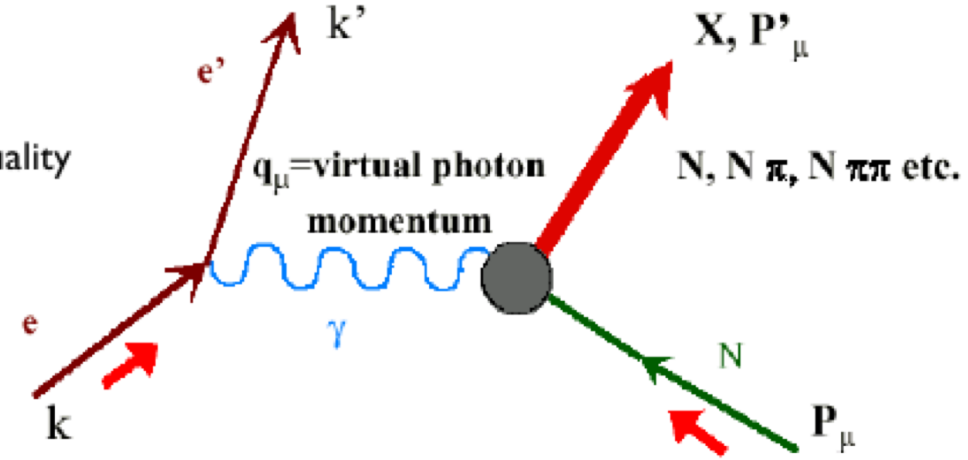
Forward Tagger detector

Low Q^2 electron scattering kinematics

Relevant kinematic variables:

$$Q^2 = 4EE' \sin^2(\theta_e/2)$$
$$E_\gamma = \nu = E - E'$$
$$W^2 = M_p^2 + 2M_p\nu - Q^2$$

$-q^m q_m = Q^2$ = photon virtuality
s = CM total energy
t = momentum transfer



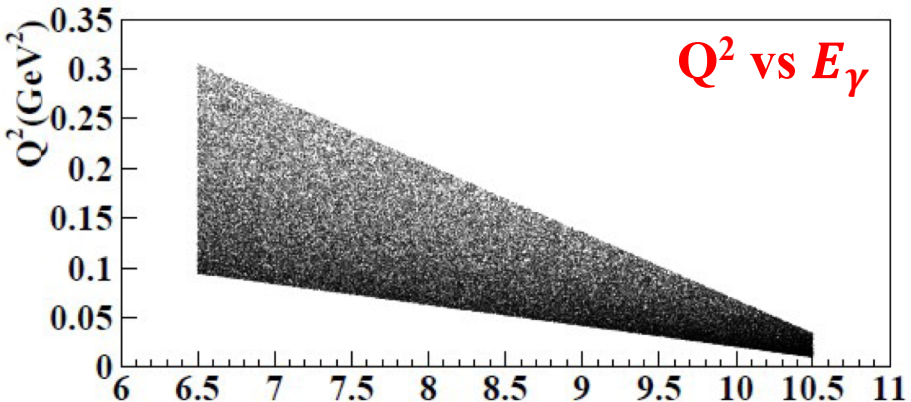
Virtual photon polarization is defined event by event:

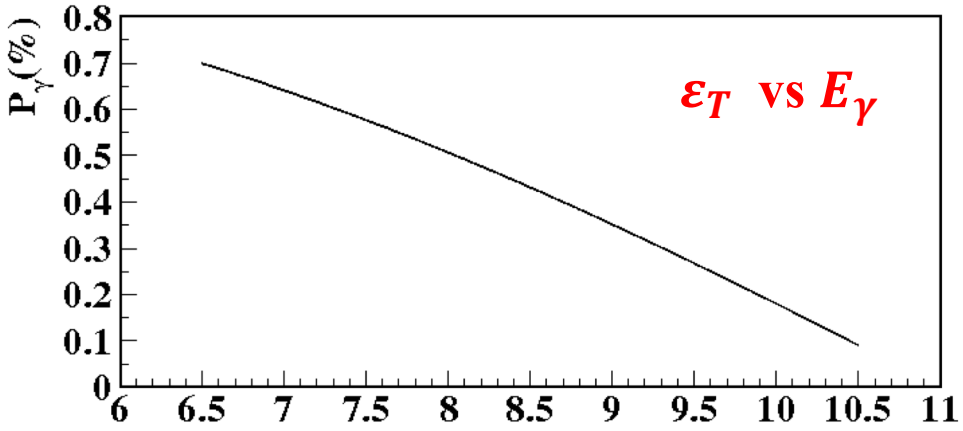
$$\varepsilon_T = [1 + 2\frac{Q^2+\nu^2}{Q^2}\tan^2(\theta_e/2)]^{-1}$$

Transverse linear polarization

$$\varepsilon_L = \frac{Q^2}{\nu^2}\varepsilon \simeq 0$$

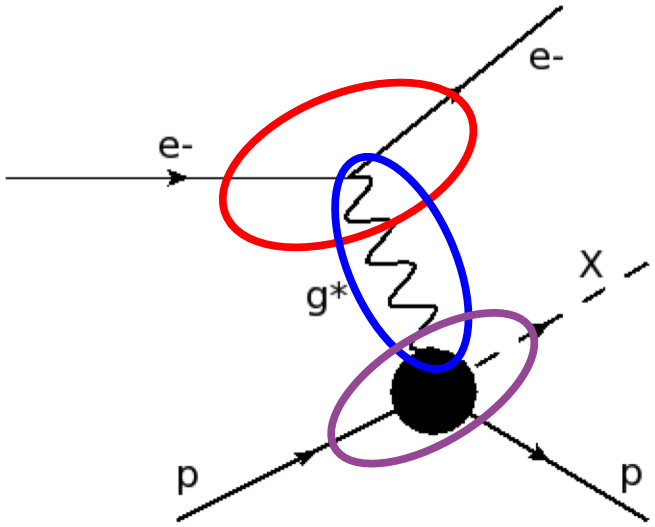
Longitudinal polarization





Photoproduction amplitudes with low Q^2 electron scattering

Goal: in the low Q^2 limit, develop an analysis framework where one can use photoproduction amplitudes, having the electron scattering part automatically included.



In the one-photon-exchange approximation:

$$\mathcal{M} = e^2 \overline{u}(p', h')\gamma^\mu u(p, h)\, \frac{g_{\mu\nu}}{q^2}\, J^\nu(k, k', w, s, s')$$

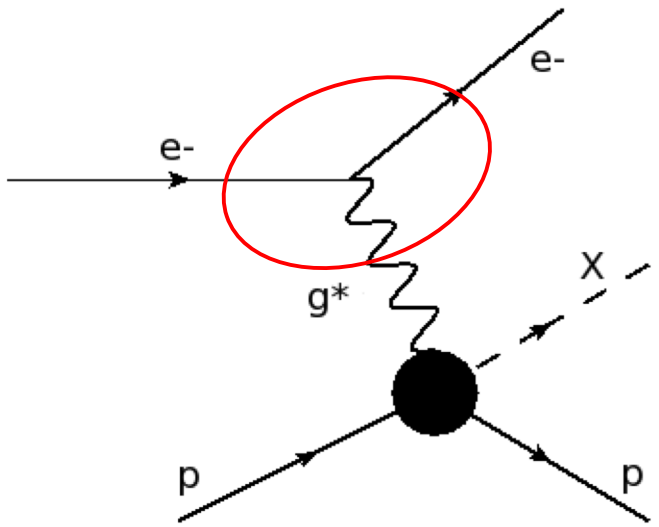
Summing over virtual photon helicity
(virtual photon pol. vectors completeness relation + current conservation):

$$\mathcal{M} = -\frac{e^2}{q^2} \sum_{\lambda=-1}^{1} (\varepsilon_\mu^*(k, \lambda)\overline{u}\gamma^\mu u)(\varepsilon_\nu(k, \lambda) J^\nu)$$

Resulting amplitude is the factorized into two terms:

- Emission of a quasi-real photon (m ~ 0) by the electron
- Photoproduction of the final state NX on the nucleon

Leptonic vertex calculation



$$M = -\frac{e^2}{q^2} \sum_{\lambda=1,0,-1} \underline{(\varepsilon^*_\mu(k,\lambda)\bar{u}\gamma^\mu u)}(\varepsilon_\nu(k,\lambda)J^\nu)$$

Leptonic vertex:
quasi-real photon emission

$$L(\lambda, s, s') = \varepsilon^*_\mu(k,\lambda)\bar{u}(p,s)\gamma^\mu u(p',s')$$

The calculation is performed projecting the electron spin on the helicity base and working in the GJ reference frame (virtual photon moving along Z axis, nucleon scattering in the XY plane)

$$L(\lambda = 1, s = s' = 1) = 2\sqrt{2EE'}cos(\theta/2)sin(\theta'/2)e^{-i\phi'}$$
$$L(\lambda = -1, s = s' = 1) = -2\sqrt{2EE'}sin(\theta/2)cos(\theta'/2)e^{i\phi}$$
$$L(\lambda = 0, s = s' = \pm 1) = 2\frac{\sqrt{EE'}}{Q^2}(E_\gamma + P_\gamma)sin(\theta/2)sin(\theta'/2)e^{\pm i(\phi-\phi')}$$
$$L(\lambda = 1, s = s' = -1) = 2\sqrt{2EE'}sin(\theta/2)cos(\theta'/2)e^{-i\phi}$$
$$L(\lambda = -1, s = s' = -1) = -2\sqrt{2EE'}cos(\theta/2)sin(\theta'/2)e^{i\phi'}$$

$L(\lambda, s \neq s') = 0$ No spin-flip at high energy!

E, θ, ϕ: impinging electron
E', θ', ϕ': scattered electron

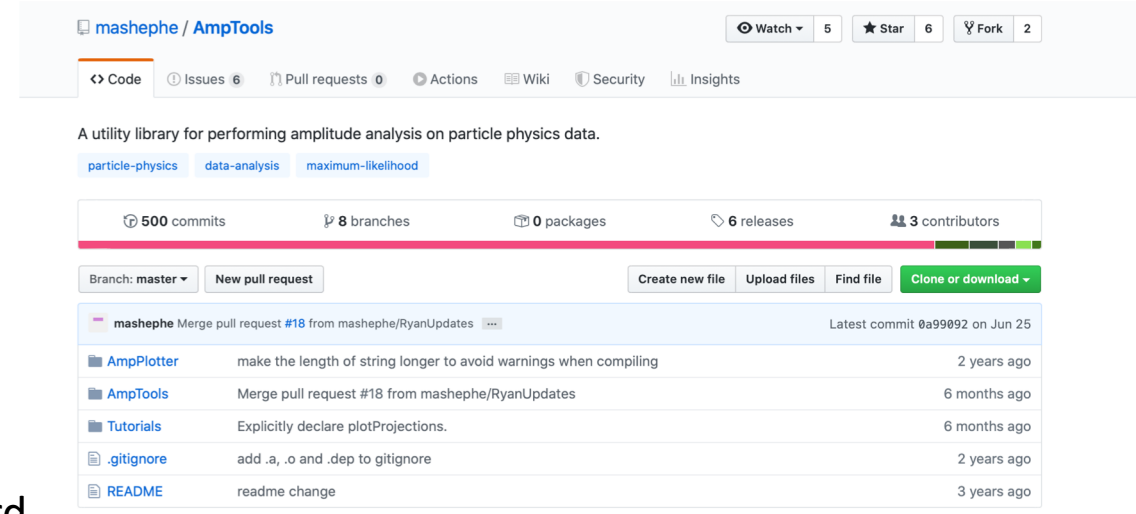
V. Mathieu

AmpTools

The AmpTools package is a collection of libraries that are useful for performing unbinned maximum likelihood fits to data using a set of interfering amplitudes.

All the technical aspects are handled within the software in a (almost) transparent way to the user: this includes data readout, event handling, fit performing.

Thus the user can just focus on the physics.



The AmpTools software:

- Developed at Indiana University
  - Main developers: H. Matevosyan, R. Mitchell, and M. Shepherd
- Fully written in C++
- Freely available at https://github.com/mashephe/AmpTools/
  - Requires ROOT to be installed (TLorentzVector class is used to handle 4-vectors)

- A very nice presentation about its features (with more details than this talk): http://www.ge.infn.it/~athos12/ATHOS/Program_files/mitchell_athos12.pdf

AmpTools

AmpTools philosophy: provide a full set of C++ classes, with virtual methods (i.e. functions) that the user has to complete (override) according to the specific application.

- Data readout
- Amplitudes
- Visualization (plots, histograms)..

No a-priori choices are built in in the framework

Example: the "CLAS12PhotonsDataReader" (class used to read data from MC developed for my PhD thesis)

```
class Clas12PhotonsDataReader : public UserDataReader<Clas12PhotonsDataReader >
```

Derive the class from the AmpTools Data Reader

```
Kinematics*
Clas12PhotonsDataReader::getEvent(){
        ...
        MCTree->GetEntry(n);
        "Kinematics=Entry"; //a little bit more
                              complicated
        n++;
        ...
}
```

The (only) method that has to be completed (overridden) by the user is the "getEvent".

Returns a "Kinematics" object (a class containing 4-vectors of particles).

(In this specific case, MC data was read from a ROOT file)

AmpTools reaction amplitudes

AmpTools amplitude: an "amplitude", practically, is simply a function that takes as input the 4-vectors of the involved particles for a given reaction and returns a complex number. The "amplitude" can spread from very simple (a constant) to very complex.

The reaction intensity (per each event) is written as a incoherent sum of coherent sums

$$I(\tau) = \sum_i |\sum_j V_{i,j} A_{i,j}(\tau)|^2$$

I: intensity of the reaction

$\boldsymbol{\tau}$: complete set of kinematic variables for the final state

i: index of the incoherent sum (typically, it runs on spin configurations)

j: index of the coherent sum

$\mathbf{A_{i,j}}$**:** the set of complex amplitudes, *user defined*

$\mathbf{V_{i,j}}$: a (complex) normalization factor for each amplitude

It is up to the user to write the amplitudes that he needs for his specific reaction.

AmpTools reaction amplitudes - implementation

AmpTools amplitude: in AmpTools, an Amplitude is an abstract class (i.e. a class with pure virtual methods that can't be instantiated directly).

The user has to derive (in C++ sense!) his own amplitudes starting from the Amplitude base class. *Practically, he has to "fill the gaps" in the Amplitude base class, defining the specific behavior of his amplitudes.*

The user has to specify how the Amplitude is calculated, starting from the 4-vectors of the involved particles

The method that has to be "filled": virtual complex< GDouble > calcAmplitude (GDouble **pKin) const

4-vectors (2D-array of doubles)

```
29  complex< GDouble >
30  BreitWigner::calcAmplitude( GDouble** pKin ) const {
31
32    TLorentzVector P1(pKin[m_daughter1-1][1], pKin[m_daughter1-1][2],
33                      pKin[m_daughter1-1][3], pKin[m_daughter1-1][0]);
34
35    TLorentzVector P2(pKin[m_daughter2-1][1], pKin[m_daughter2-1][2],
36                      pKin[m_daughter2-1][3], pKin[m_daughter2-1][0]);
37
38    return  complex<GDouble>(1.0,0.0) /
39            complex<GDouble>((P1+P2).M2() - m_mass*m_mass, m_mass*m_width);
40
41  }
```

Example from the Dalitz Tutorial, Breit-Wigner amplitude

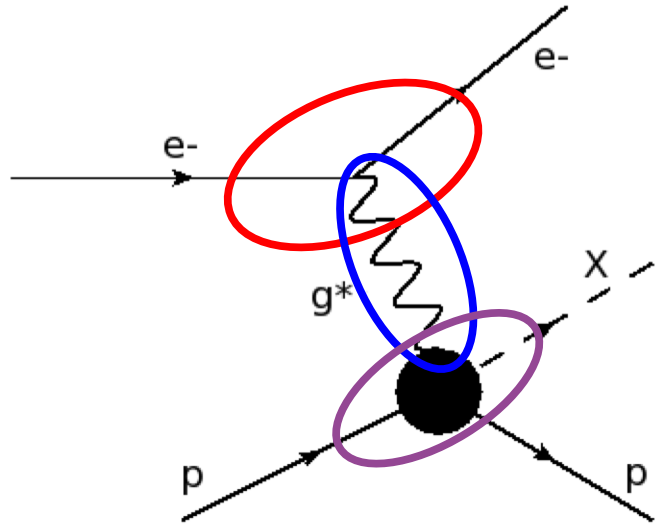
AmpTools reaction amplitudes for MesonEx

We want to use AmpTools with photo-production amplitudes. In this way, we have a practical method to communicate both with other experiments (GlueX) and with theorists.

The quasi-real photoproduction process has to be included in the framework.

The amplitude $\mathcal{M} = -\frac{e^2}{q^2} \sum_{\lambda=-1}^{1} (\varepsilon_\mu^*(k,\lambda)\overline{u}\gamma^\mu u)(\varepsilon_\nu(k,\lambda)J^\nu)$

The intensity $I(\tau) = \sum_i |\sum_j V_{i,j} A_{i,j}(\tau)|^2$



Solution: embed the electron-scattering term and the sum over "i" in the framework, in a transparent way for the user

Use a new base class derived from the Amplitude class: CLAS12PhotonsAmplitude

```
complex <GDouble> CalcAmplitude(GDouble **pKin){

return ∑λ   (calcHelicityAmplitude(λ,pKin) *
                    ElectronScattering(λ,pKin) )

}
```

Example: single π^0 production

Goal: study single π^0 production in CLAS12, describing the reaction through the photoproduction amplitudes developed by JPAC (V. Mathieu)

Discussed here: how to generate Montecarlo events starting from the photoproduction amplitude and how to check the Beam Spin Asymmetry (BSA)

Not included:

- Project on the detector (GEMC)
- Reconstruct simulated data (CLAS12 reconstruction framework)
- Analyze reconstructed data

Example: single π^0 production – how to generate MC events

AmpTools for event generation: since no physics is embedded within the framework, AmpTools doesn't have any specific method to generate MC events. This is done by combining two main ingredients:

- PhaseSpace event generator
- Intensity calculator (from the amplitudes provided by the user)

Method: "hit-or-miss"

- Generate N phase-space events
- Compute the intensity of each of them, take the maximum intensity I_{max}
- For each event, generate a random number I_{rand} between 0 and I_{max}
  - If the intensity of the event is greater than I_{rand}, keep it
  - Otherwise, skip it

The advantage of this method is that it is agnostic regarding the physics. Disadvantage is that it is computational expensive.

Example: single π^0 production – simplified model

Goal: use a very simplified reaction model and extract the BSA. Compare with the analytical prediction.

Electron-scattering cross-section: $$\frac{d\sigma}{dE' d\Omega_E d\Omega^*} = \frac{1}{\Gamma_\gamma} \frac{d\sigma_\nu}{d\Omega^*}$$

Ω_γ: the angle between the hadronic and the leptonic plane.

Expansion via response functions, ignoring ε_L: $$\frac{d\sigma_\nu}{d\Omega^*} = (R_T^0 + \varepsilon R_{TT}^0 \cos(2\phi))$$

Response functions expressed via hadronic amplitudes:

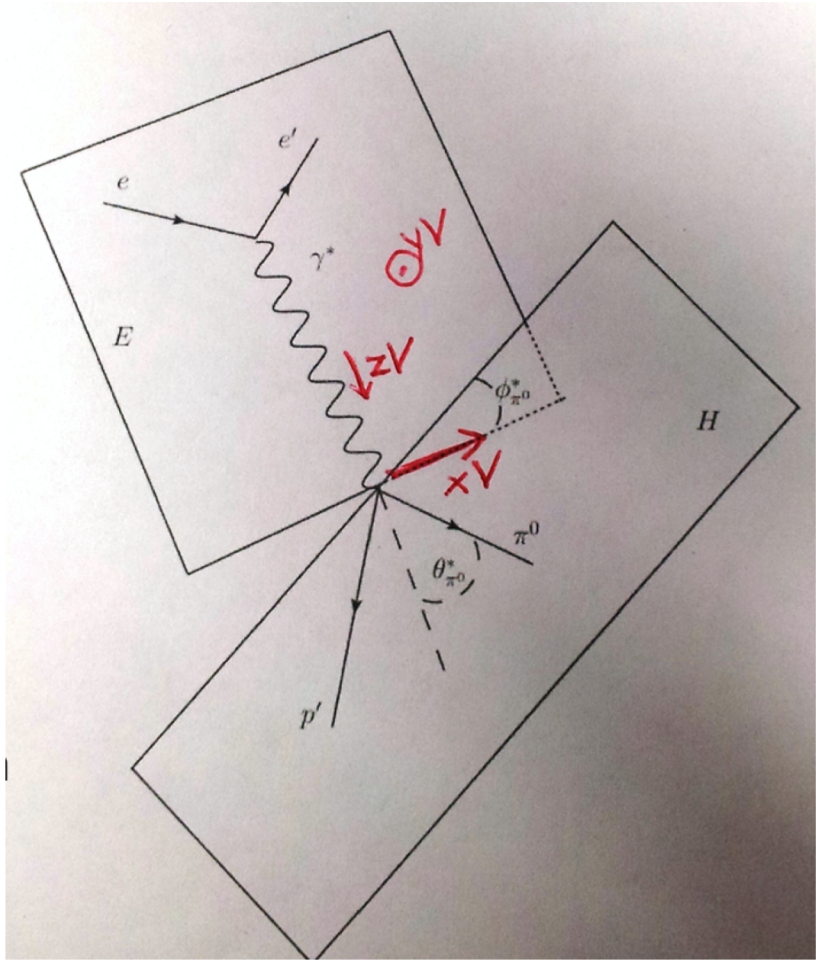
$$R_T^0 = \frac{1}{2}(|H(++,1)|^2 + |H(--,1)|^2 + |H(-+,1)|^2 + |H(+-,1)|^2)$$

$$R_{TT}^0 = Re(-H_1^* H_4 + H_2^* H_3)$$

Parity conservation relations:

$$H(\pm,\pm,1) = H(\mp,\mp,-1)$$

$$H(\pm,\mp,1) = -H(\pm,\mp,-1)$$



Example: single π^0 production – simplified model results

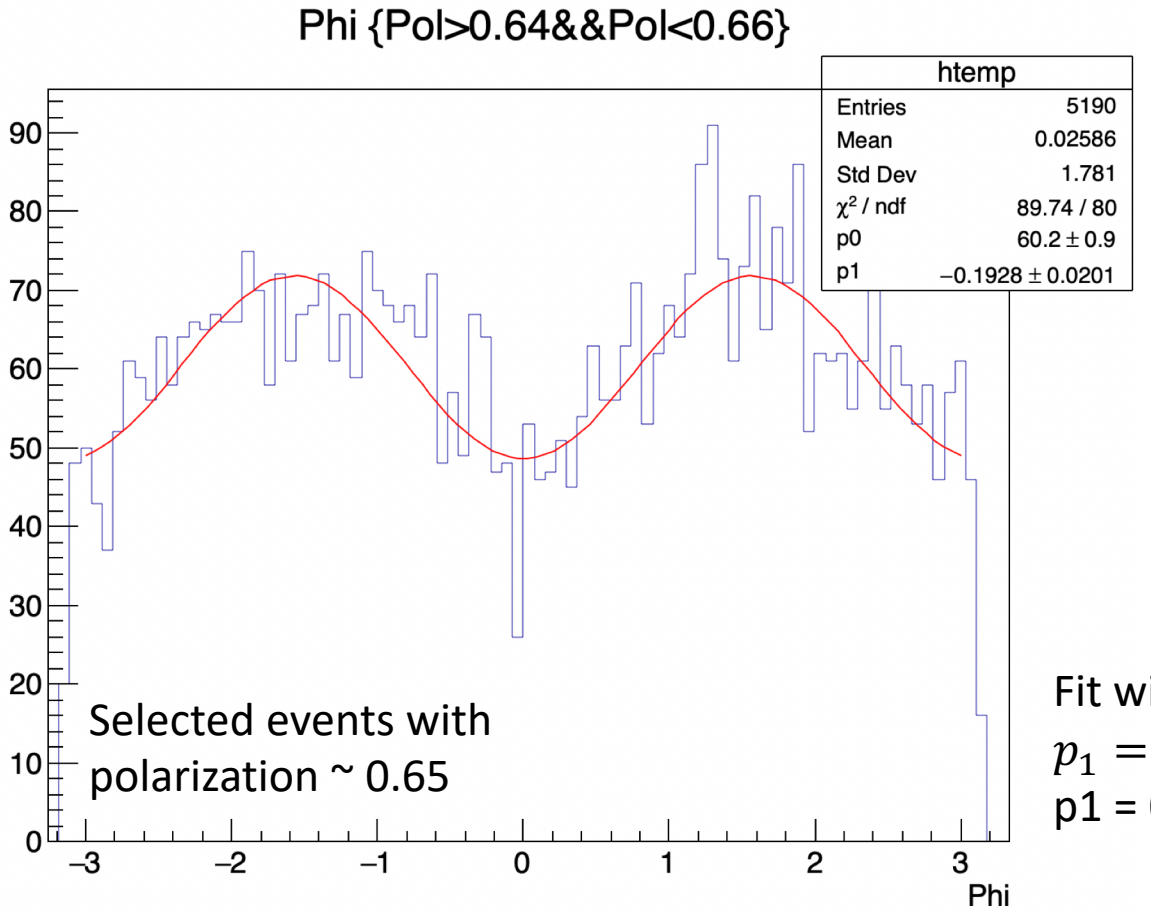
Simplified model:

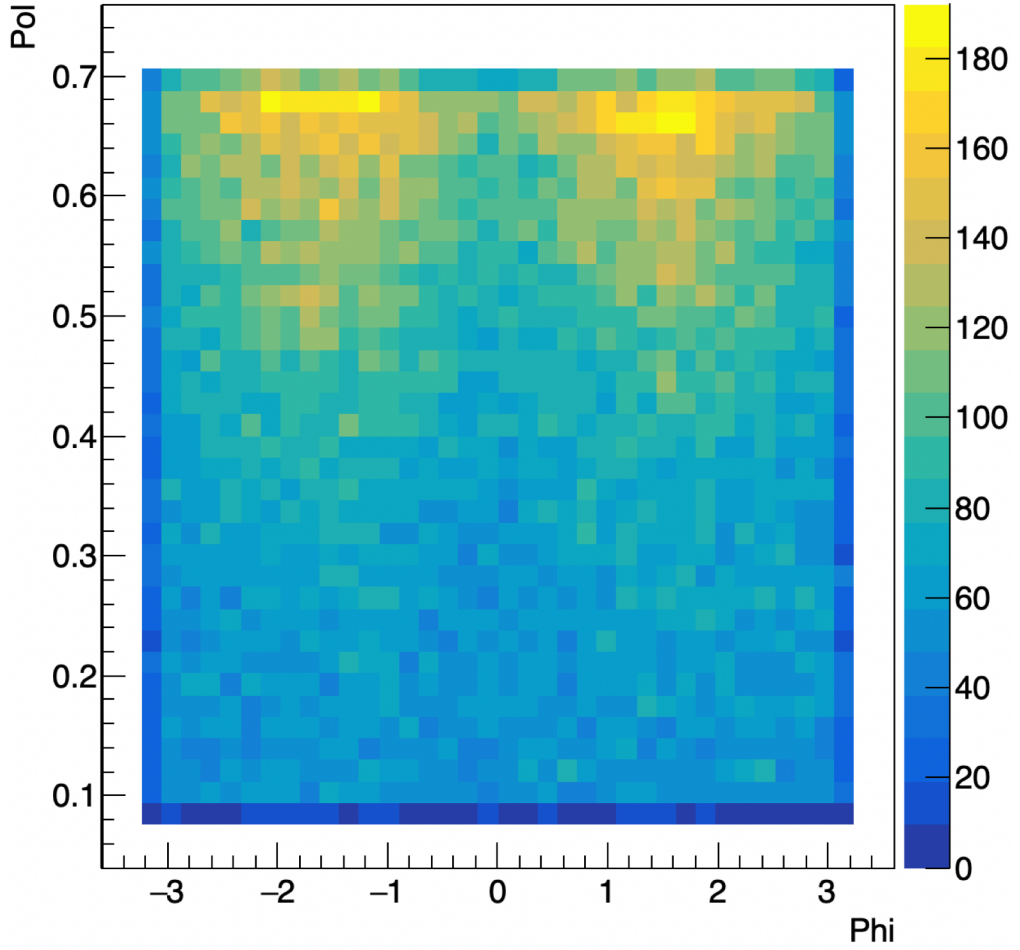
$$H(++,1) = 1. \quad H(-+,1) = \alpha$$
$$H(--,1) = 1. \quad H(+-,1) = 1.$$

Expected result:

$$\frac{d\sigma_\nu}{d\Omega^*} \propto (1 + \varepsilon \frac{2(\alpha-1)}{3+\alpha^2} \cos(2\phi))$$

= - 0.3





Fit with $f(\phi) = p_0(1 + p_1 \cos 2\phi)$

$p_1 = -0.193$

p1 = 0.65 * (-0.3) **as expected from before!**

Example: single π^0 production – a real reaction amplitude

The reaction amplitude is written by creating a new class deriving from Clas12PhotonsAmplitude

```
complex<GDouble> Pi0Regge::calcHelicityAmplitude(int helicity, GDouble** pKin) const {

    const int ibeam = 0;
    const int iscattered_e = 1;
    const int itarget = 2;
    const int iscattered_proton = 3;
    const int ipi0 = 4;

    //impinging e-
    TLorentzVector beam(pKin[ibeam][1], pKin[ibeam][2], pKin[ibeam][3], pKin[ibeam][0]);

    //scattered e-
    TLorentzVector eprime(pKin[iscattered_e][1], pKin[iscattered_e][2], pKin[iscattered_e][3], pKin[iscattered_e][0]);

    //proton target
    TLorentzVector target(pKin[itarget][1], pKin[itarget][2], pKin[itarget][3], pKin[itarget][0]);

    //recoil proton
    TLorentzVector recoil(pKin[iscattered_proton][1], pKin[iscattered_proton][2], pKin[iscattered_proton][3], pKin[iscattered_proton][0]);

    //pi0
    TLorentzVector pi0(pKin[ipi0][1], pKin[ipi0][2], pKin[ipi0][3], pKin[ipi0][0]);

    /*now I need to return the amplitude for the quasi-real photoproduction process*/
    //Virtual photon
    double vphot[4];
    vphot[0] = (beam - eprime).E();
    vphot[1] = (beam - eprime).Px();
    vphot[2] = (beam - eprime).Py();
    vphot[3] = (beam - eprime).Pz();

    int hel[4]; //array of polarizations of: photon, target-proton, pi0,recoil-proton (2x)

    hel[0] = 2 * helicity;
    hel[1] = m_helicity_target;
    hel[2] = 0; //pi0
    hel[3] = m_helicity_recoil;

    //pKin is already ordered - particle by particle - as E,Px,Py,Pz.
    //Pi0PhotAmpS needs: photon, target proton, pi0 (not the recoil proton, this is computed by 4-momentum conservation)
    if (hel[0] == -2) {
        hel[0] = -hel[0];
        hel[1] = -hel[1];
        hel[3] = -hel[3];
        return (1.*hel[1]) * (1.*hel[3]) * Pi0PhotAmpS(vphot, pKin[itarget], pKin[ipi0], hel);

    }
    else{
        return Pi0PhotAmpS(vphot, pKin[itarget], pKin[ipi0], hel);
    }

}
```

```
//the order of the particles is supposed to be:
//0: e- beam
//1: e- scattered
//2: target
//3..n-2: all the other particles
```

V. Mathieu

Conclusions

BACKUP

Low Q^2 electron scattering: equivalent photon flux

Equivalent photon-flux approximation:

$$d\sigma(\theta_e, E_e) = d\Gamma(\theta_e, E_e) \cdot \sigma_\gamma(\nu, Q^2 = 0)$$

In the Forward Tagger kinematic range:

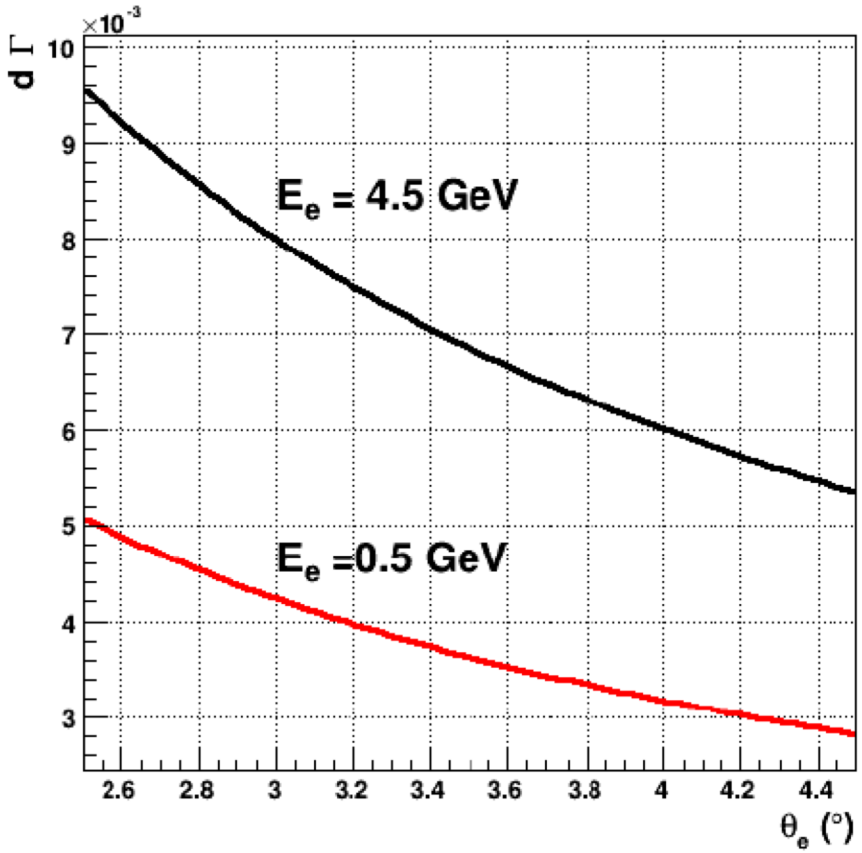
- Luminosity: 10^{35} cm^{-2} s^{-1}
- $\sigma^{TOT}_{\gamma} \sim 100$ μbarn

Expected event rate: 7 kHz

Equivalent photon flux:

- On a 40 cm long LH_2 target

$$\Phi_\gamma^{equiv} \simeq 5 \cdot 10^7 \gamma/s$$



Low Q^2 electron scattering is competitive and complementary to real photo-production.

V.M. Budnev, et al., Physics Reports, Volume 15, Issue 4, 1975