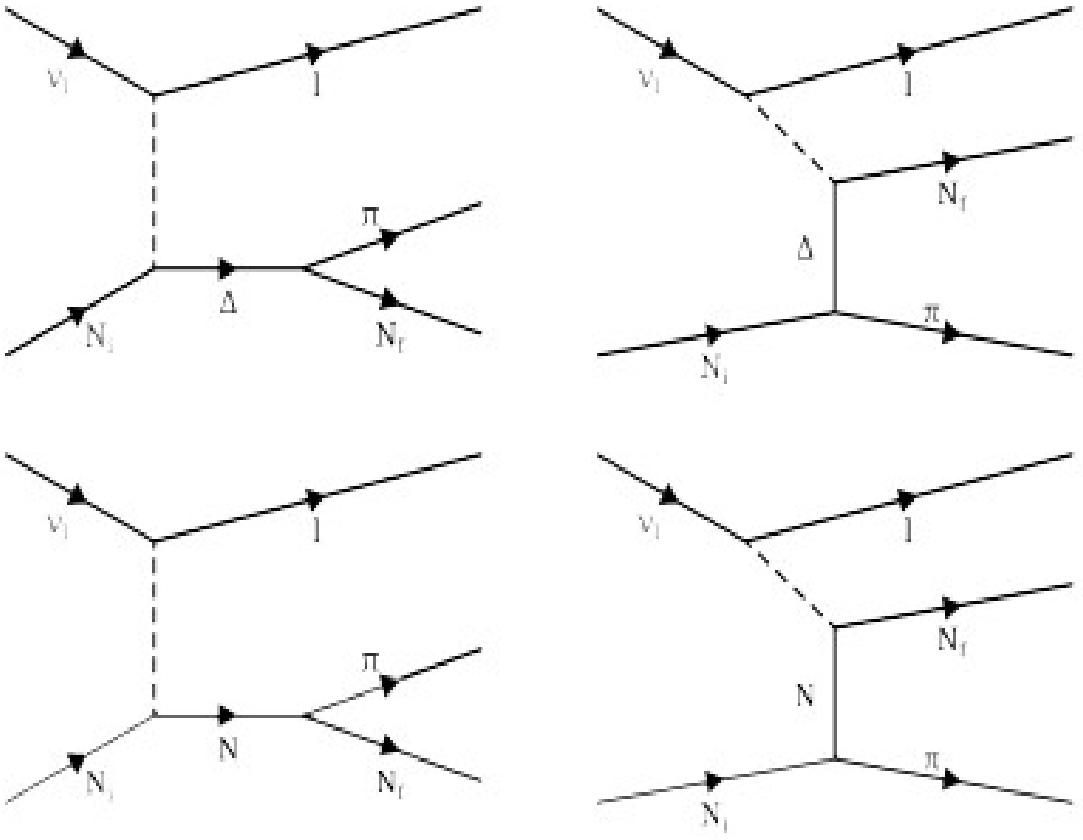
Implementation of the Alvarez-Ruso microscopic coherent model

Steve Boyd , Dan Scully , Steve Dennis

- Brief model description
- Implementation with lookup tables
- Implementation without lookup tables

THE UNIVERSITY OF WARWICK

Microscopic models



- Delta/Nucleon production
- Final state nucleon in the same state as initial nucleon
- A number of models on the market
  - Alvarez-Ruso
  - Nieves
  - Hernandez

Models apply below approx. 5 GeV
None (up to now) have been included in the generators

Model

$$\frac{d^5\sigma}{dE_l\, d\Omega_l\, d\Omega_\pi} = \frac{1}{8(2\pi)^5}\frac{|p_l||p_\pi|}{|p_\nu|}\left|\overline{M}^2\right|$$

Leptonic current is standard : $j_l^\mu = \overline{u_l}\,\gamma^\mu\left(1-\gamma_5\right)u_\nu$

Hadronic current is more complicted :

$$j_{p\,\pi}^\mu = \frac{f^*}{m_\pi}\, p_\pi^\alpha\, F\left(p_\Delta\right)\overline{u_{N,f}}\, D\left(p_\Delta\right)\Lambda_{\alpha\beta}\, A^{\beta\mu}\, u_{N,i}$$

Model

$$j^{\mu}_{p\,\pi} = \frac{f^{*}}{m_{\pi}}\, p^{\alpha}_{\pi}\, F\left(p_{\Delta}\right) \overline{u_{N,f}}\, D\left(p_{\Delta}\right) \Lambda_{\alpha\beta}\, A^{\beta\mu}\, u_{N,i}$$

f* : $\Delta \rightarrow \pi$ N decay constant

$P_{\{\pi,\Delta\}}$: pion, Delta momenta

$F(p_{\Delta})$: Δ–πN vertex form factor

$D(p_{\Delta})$: In-medium delta propagator

$\Lambda_{\alpha\beta}$: spin 3/2 projection operator

$A^{\beta\mu}$: N-Δ coupling

$$A^{\mu\alpha} = \left\{ \frac{C_3^V}{m_N}\left(g^{\mu\alpha}\gamma_{\delta}q^{\delta} - q^{\mu}\gamma^{\alpha}\right) + \frac{C_4^V}{m_N^2}\left(g^{\mu\alpha}q\cdot P_{\Delta} - q^{\mu}P_{\Delta}{}^{\alpha}\right) + \frac{C_5^V}{m_N^2}\left(g^{\mu\alpha}q\cdot P_{\mathrm{N,i}} - q^{\mu}P_{\mathrm{N,i}}{}^{\alpha}\right)\right\}\gamma_5$$

$$+\left\{ \frac{C_3^A}{m_N}\left(g^{\mu\alpha}\gamma_{\delta}q^{\delta} - q^{\mu}\gamma^{\alpha}\right) + \frac{C_4^A}{m_N^2}\left(g^{\mu\alpha}q\cdot P_{\Delta} - q^{\mu}P_{\Delta}{}^{\alpha}\right) + C_5^A g^{\mu\alpha} + \frac{C_6^A}{m_N^2}q^{\mu}q^{\alpha}\right\}$$

Model

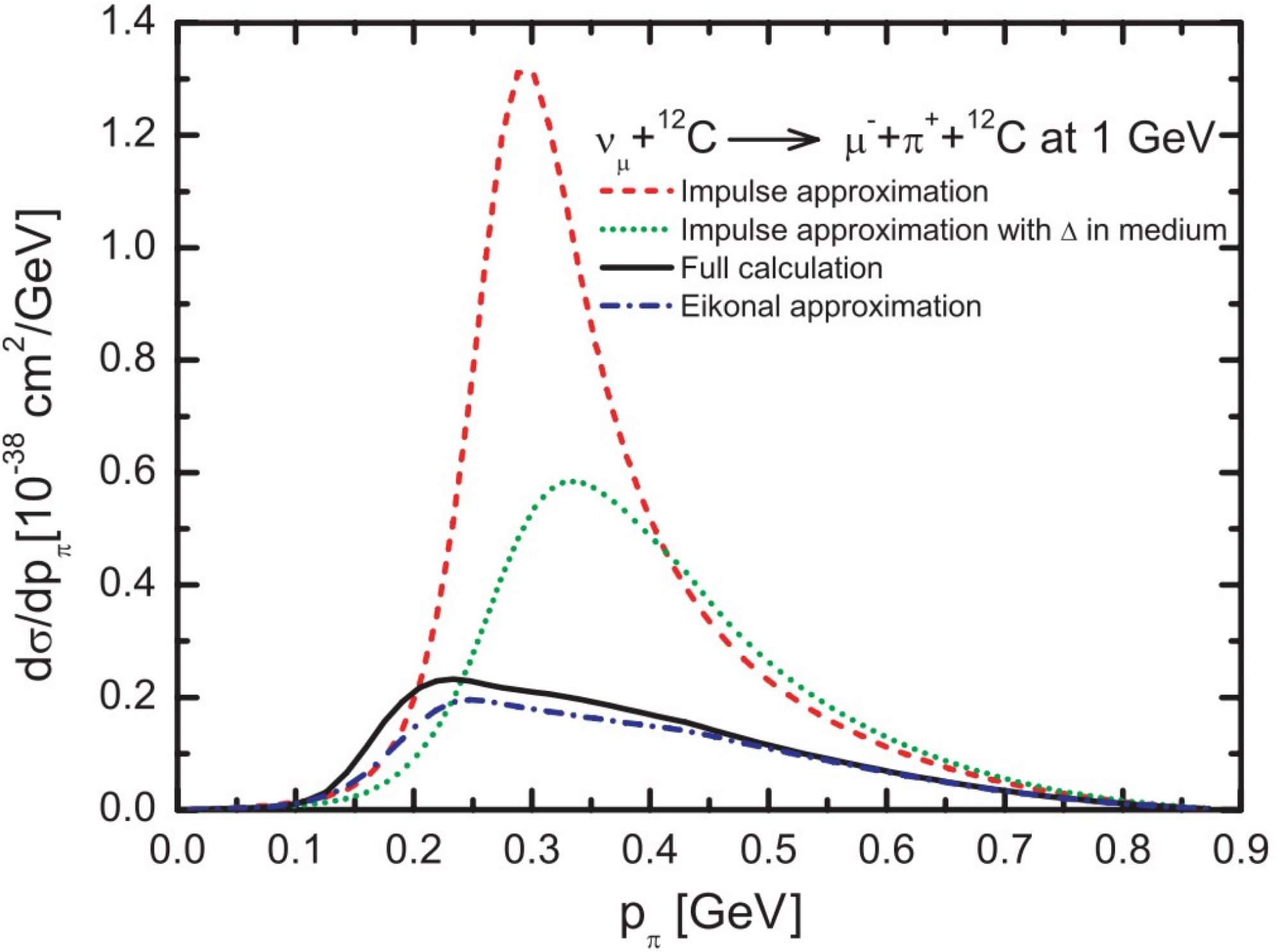
Nuclear hadronic current using a plane wave for the pion wavefunction

$$J_{A\pi}{}^{\mu} = \frac{-i}{2}\int e^{i\left(\vec{q} - p_{\pi}\right)\cdot r}\left(\rho_p(r) + \frac{\rho_n(r)}{3}\right)\frac{f^*}{m_{\pi}} P_{\pi}{}^{\alpha} F\left(P_{\Delta}\right) D\left(P_{\Delta}\right) \mathrm{Tr}\left\{\bar{u}_{\mathrm{N},f}\Lambda_{\alpha\beta}A^{\beta\mu}u_{\mathrm{N},i}\right\} \mathrm{d}^3 r$$

More realistic description modified the pion wavefunction using the *eikonal approximation*

$$\varphi^{*}_{\mathrm{eikonal}} = e^{-ip_{\pi}\cdot r}\exp\left[-i\int_{r}^{\infty}\frac{\Pi(r')}{2p_{\pi}}\mathrm{d}\,r'\right]$$

Pion self energy


$\nu_\mu + {}^{12}C \longrightarrow \mu^- + \pi^+ + {}^{12}C$ at 1 GeV
Impulse approximation
Impulse approximation with Δ in medium
Full calculation
Eikonal approximation
$d\sigma/dp_\pi$ [10^{-38} cm^2/GeV]
p_π [GeV]
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

Implementation

- Model was copied from Luis' FORTRAN calcuation
- Standalone implementation was used to verify the translation to C++
- GENIE Nuclear utils are used
- One of the polar angles is free : so can reduce the 5-D integral to a 4-D integral

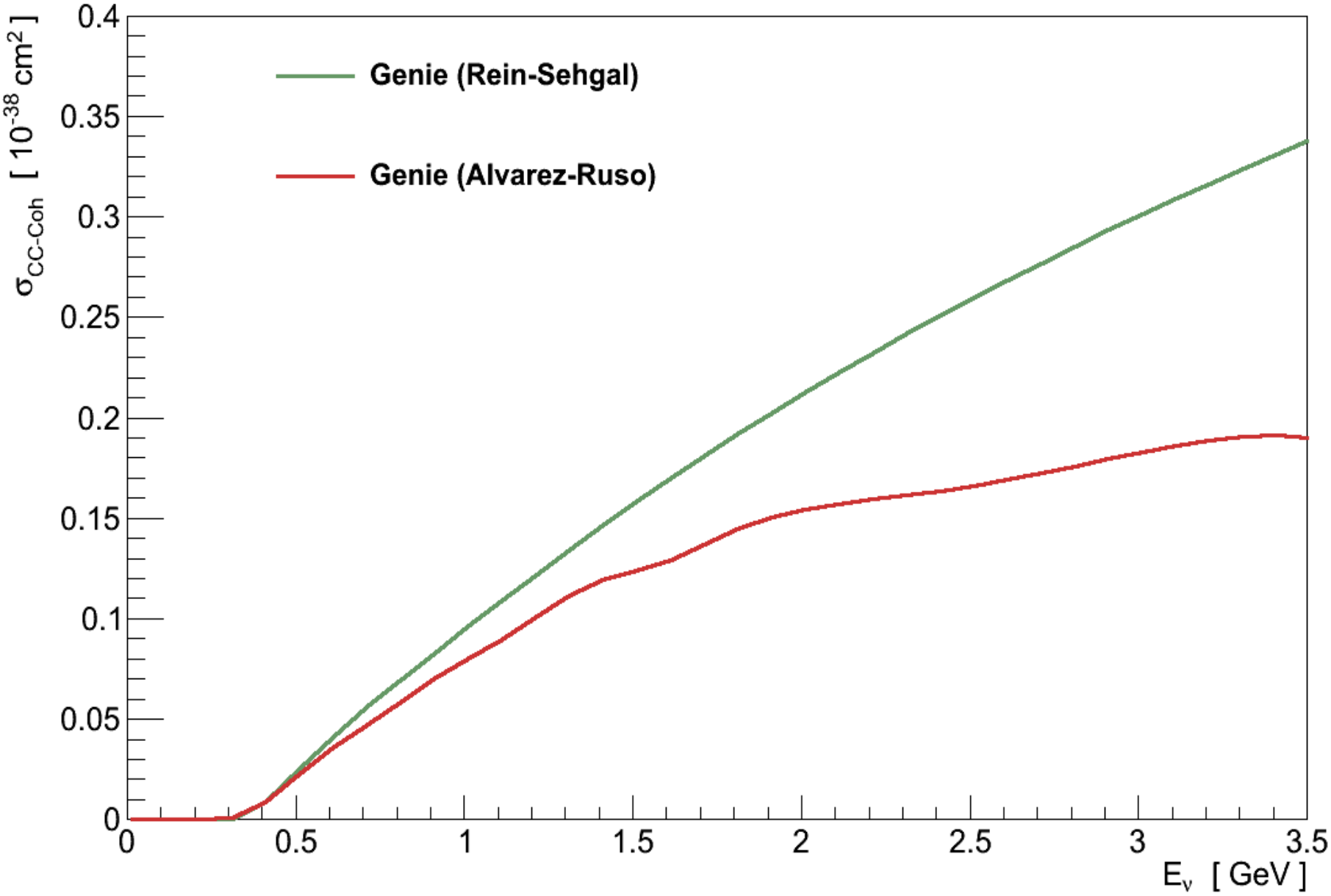
$$\frac{d^5\sigma}{dE_l\, d\Omega_l\, d\Omega_\pi} \rightarrow 2\pi \frac{d^4\sigma}{dE_l\, d\Omega_l\, d\theta_\pi}$$

- Calculation currently cuts off at 5 GeV

Implementation 1 : Lookup tables

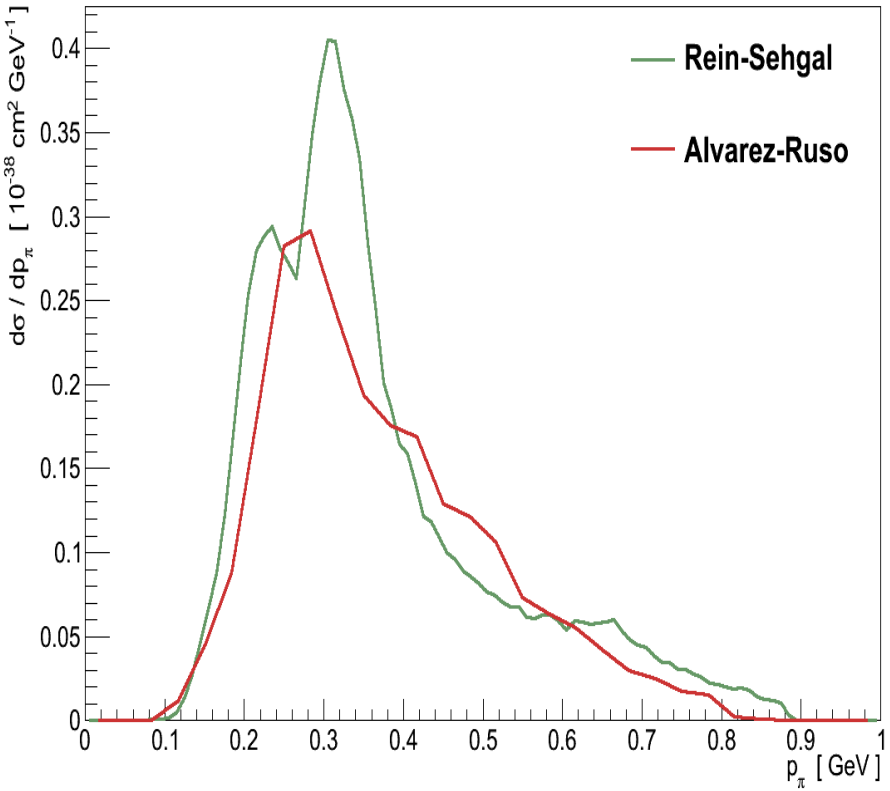
- For Dan's thesis we needed the AR model to be relatively fast
- The only way to do this was to pre-calculate the pion wavefunctions and store them in look-up tables
- One look-up table per (neutrino, target) combination

  - Approximately 0.05 s per cross section evaluation
  - Takes approximately 10 hours to generate a cross-section spline
  - Size of each table is about 25 Mb. Tables are only text and this could be optimised a lot.

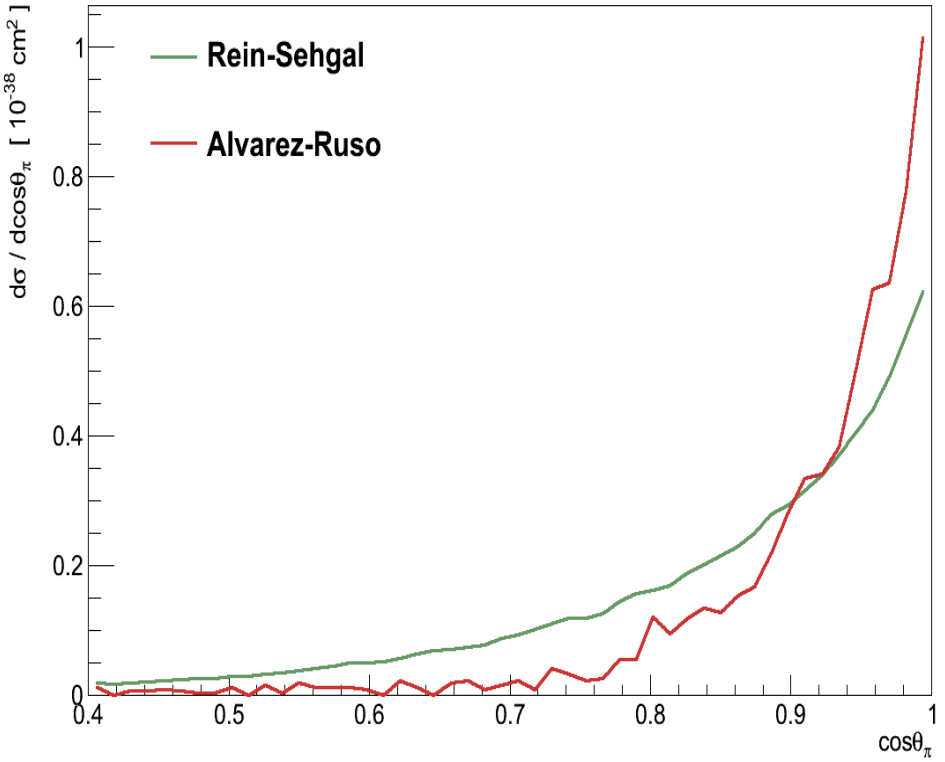

Genie (Rein-Sehgal)
Genie (Alvarez-Ruso)
$\sigma_{CC\text{-}Coh}$ [10^{-38} cm^2]
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
0.5
1
1.5
2
2.5
3
3.5
E_ν [GeV]

ν_μ on C14

Some kinematic distributions



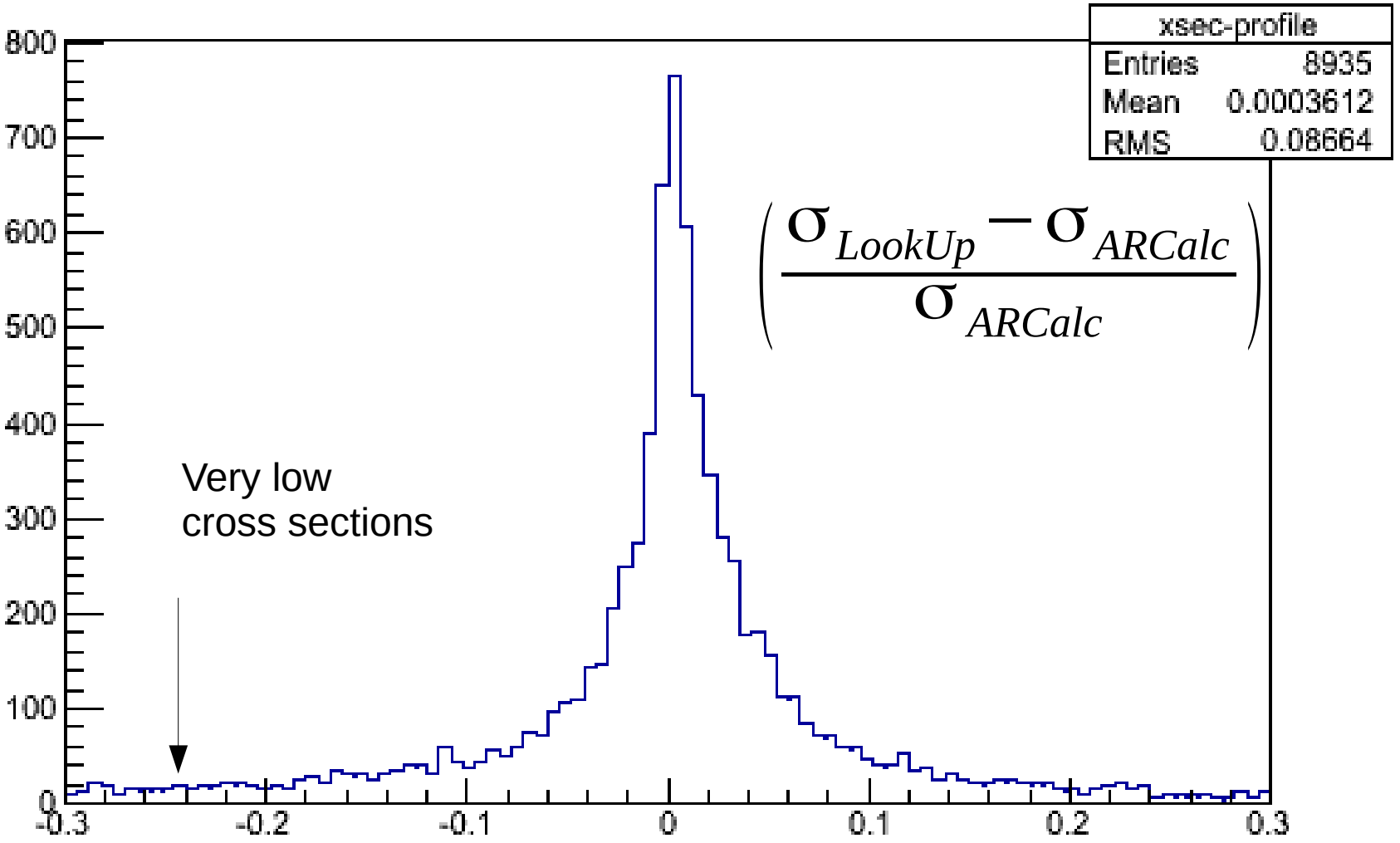
p_π



$\cos\theta_\pi$

ν_μ on C12 @ E_ν = 1.0 GeV

But is it accurate?



Implementation 2 : Using wavefunction calculation

- Moving from lookup tables to wave function calculation on an event by event basis slows down the code by about a factor of 10 : one evaluation takes about 0.5 seconds now.

- Still, for event generation this is probably endurable.

- The largest problem we have is processing time in the spline calculation.

- We've implemented this by splitting the integral. The wavefn is only dependent on E_π so only needs to be calculated when E_π changes

Implementation 2

$$\sigma(E_\nu) = 2\pi \int \frac{d\sigma}{dE_l} \left(\int \int \int \frac{d^3\sigma}{d\Omega_\mu \, d\theta_\pi} d\Omega_\mu \, d\theta_\pi \right) dE_l$$

This may have sped up the spline calculation.

I can't tell because we are still running the first spline test after 1.5 days

Incl.	Self	Called	Function	Location
99.99	0.00	3	gsl_integration_qk	libgsl.so.0.16.0
99.99	0.00	48	ROOT::Math::GSLFunctionAdapter<ROOT::Math:...	libMathMore.so.5
99.99	0.00	48	ROOT::Math::WrappedFunction<genie::utils::gsl:...	libGCrossSections
99.99	0.00	48	genie::utils::gsl::wrap::dXSec_dElep_AR::DoEval(...	libGCrossSections
99.99	0.00	48	ROOT::Math::GSLMCIntegrator::Integral(double ...	libMathMore.so.5
99.99	0.00	48	gsl_monte_miser_integrate	libgsl.so.0.16.0
99.98	0.00	477 651	ROOT::Math::GSLMonteFunctionAdapter<ROOT...	libMathMore.so.5
99.98	0.00	477 651	genie::utils::gsl::wrap::d3Xsec_dOmegaldThetap...	libGCrossSections
99.89	0.00	477 370	genie::AlvarezRusoCOHXSec::XSec(genie::Intera...	libGAlvarezRuso-2
99.83	0.00	477 370	alvarezruso::Coh::DXSec(double, double, double...	libGAlvarezRuso-2
99.12	35.00	477 370	alvarezruso::Coh::NuclearCurrent(TLorentzVect...	libGAlvarezRuso-2
89.58	0.00	646	gsl_monte_miser_integrate'2	libgsl.so.0.16.0
33.20	2.94	1 527 582 405	alvarezruso::Coh::DeltaCouplingInMed(TLorentz...	libGAlvarezRuso-2
12.20	2.85	7 412 010 985	TLorentzVector::TLorentzVector(TLorentzVecto...	libPhysics.so.5.34
10.83	10.83	29 801 607 225	__muldc3	libgcc_s.so.1
10.37	0.97	564 516 003	alvarezruso::Coh::DeltaWidthPauliBlocked(TLor...	libGAlvarezRuso-2
10.24	2.39	7 984 165 486	TLorentzVector::~TLorentzVector()	libPhysics.so.5.34
9.56	9.56	2 305 111 343	0x00000000000173d0	libm-2.11.3.so
9.18	1.54	564 516 002	alvarezruso::Coh::DeltaSelfEnergyIm(double)	libGAlvarezRuso-2
7.43	7.43	16 548 144 859	TObject::~TObject()	libCore.so.5.34
7.13	3.47	8 557 752 073	TVector3::TVector3(TVector3 const&)	libPhysics.so.5.34
6.83	6.83	15 970 240 428	TObject::TObject(TObject const&)	libCore.so.5.34
4.57	0.73	8 563 481 090	TVector3::~TVector3()	libPhysics.so.5.34
3.22	0.79	566 425 482	alvarezruso::Coh::PiDecayVertex(TLorentzVecto...	libGAlvarezRuso-2
3.09	0.75	564 516 003	alvarezruso::Coh::DeltaWidthFree(TLorentzVect...	libGAlvarezRuso-2
3.05	0.56	764 175 202	cexp	libm-2.11.3.so
2.97	1.83	1 527 582 404	alvarezruso::Coh::NucleonPropagator(TLorentz...	libGAlvarezRuso-2
2.19	0.95	1 527 582 405	csqrt	libm-2.11.3.so
1.78	1.78	3 085 884 808	__divdc3	libgcc_s.so.1
1.76	1.76	4 583 515 214	__fpclassify	libm-2.11.3.so
1.59	1.59	769 951 226	0x00000000000190c0	libm-2.11.3.so
1.23	1.18	1 129 032 005	alvarezruso::Coh::PionMomentumCM(TLorentzV...	libGAlvarezRuso-2
0.89	0.23	571 677 395	TLorentzVector::TLorentzVector(TVector3 const...	libPhysics.so.5.34
0.86	0.78	19 572 149	alvarezruso::integrationTools::RG202D(double, ...	libGAlvarezRuso-2
0.63	0.00	48	alvarezruso::Coh::SolveWavefunctions()	libGAlvarezRuso-2
0.63	0.01	384 000	alvarezruso::EikonalSolution::Element(double, d...	libGAlvarezRuso-2
0.48	0.48	2 822 580 010	alvarezruso::Coh::DeltaSelfEnergyConstant(dou...	libGAlvarezRuso-2
0.42	0.42	9 931 947 102	alvarezruso::Constants::NucleonMass()	libGAlvarezRuso-2
0.37	0.01	7 684 800	alvarezruso::Nucl::DensitiesR(double, double&, ...	libGAlvarezRuso-2
0.37	0.01	15 369 600	genie::utils::nuclear::Density(double, int, double)	libGUtils-2.8.0.so
0.36	0.02	15 369 600	genie::utils::nuclear::DensityGaus(double, doubl...	libGUtils-2.8.0.so
0.28	0.01	19 669 491	genie::Messenger::operator()(char const*)	libGMessenger-2.

Code Profile with cachegrind

← Hadronic current (35%)

← gcc multiplication (11%)

← exponentiation (10%)

Assorted ROOT overhead (18%)

Other math functions (5%)

Why is the hadronic current taking so much time?

Pros & Cons

Of lookup table approach:

PRO:

- (Relatively) fast
- (Relatively) accurate

CON:

- Lookup-table needed for each neutrino/target combination
- Limited by cut-off in E_π

Pros & Cons

Of full calculation:

PRO:

- Exact at each point in phase space
- Valid over any neutrino energy range

CON:

- Slow to generate events and
- Glacially slow to generate splines (not necessarily a problem if this is not done often)
- Valid over any neutrino range

Summary & Questions

- Alvarez-Ruso microscopic model has been implemented in GENIE
- and even used in Dan's analysis
- Full calculation at each cross section evaluation is slow which could inhibit spline calculation
- We are in the process of trying to optimise it.
- Alternate lookup-table is in place. This is faster, but less accurate

Questions

- Do we want to keep both calculation methods? Which should be used by default?
- Need to finish optimising the full calculation before we can tell if it is really feasible
- How do we deal with the high energy behaviour of the model? The pion wavefunction starts oscillating rapidly at high E_π – the integration of the hadronic current will need to be done carefully
- Do we try to merge AR model and Rein-Seghal. If so how, and over what energy range?