Changes and bugs in GENIE 2.12.x

Igor D. Kakorin, Konstantin S. Kuzmin, and Vadim A. Naumov

October 25, 2018

Abstract

GENIE is a suite of products (Generator, Comparisons, Tuning) for the experimental neutrino community [1]. The long-term goal of the GENIE project is to develop a 'canonical' neutrino event Monte Carlo generator whose validity will extend to all nuclear targets and neutrino flavors over a wide spectrum of energies ranging from ~ 1 MeV to ~ 1 PeV [2,3]. The present document describes the bugs found in the GENIE code and suggests several improvements and additions. Currently the document concerns versions 2.12.x of the project [3]. This is a living document which we will continue to update, following the changes in future versions of the GENIE project and progress in our own work.

JINR NOνA Group Internal report

Dubna

Contents

List of Figures

List of Tables

Chapter 1

Implementation of additional models and optimization of existing ones

1.1 Implementation of RFG model by Smith and Moniz.

Status: implemented in GENIE 3.0.0

Files: src/Utils/SmithMonizUtils.h,
src/Utils/SmithMonizUtils.cxx,
src/LlewellynSmith/SmithMonizQELCCPXSec.h,
src/LlewellynSmith/SmithMonizQELCCXSec.cxx,
src/LlewellynSmith/SmithMonizQELCCXSec.h,
src/LlewellynSmith/SmithMonizQELCCXSec.cxx,
src/LlewellynSmith/LinkDef.h,
src/QEL/QELEventGeneratorSM.h,
src/QEL/QELEventGeneratorSM.cxx,
src/QEL/LinkDef.h,
src/Conventions/KinePhaseSpace.h,
src/GHEP/GHepRecord.cxx,
src/Conventions/KineVar.h,
config/UserPhysicsOptions.xml,
config/SmithMonizQELCCPXSec.xml,
config/SmithMonizQELCCXSec.xml,
config/SmithMonizUtils.xml,
config/QELEventGeneratorSM.xml,
config/master_config.xml,
config/Messenger.xml,...

The RFG model by Smith and Moniz [4] is well described in the paper [5] [1]. The implementation of this model in GENIE is based on the four classes:

1. `class SmithMonizUtils` is auxiliary. The most important class methods are:
   - `double E_nu_thr_SM(void) const` – returns energy threshold for reaction;
   - `Range1D_t Q2QES_SM_lim(void) const` – returns range of kinematically allowed Q^2 for given energy;
   - `Range1D_t vQES_SM_lim(double Q2) const` – returns range of kinematically allowed ν – energy transfer for given Q^2;
   - `Range1D_t kFQES_SM_lim(double nu, double Q2) const` – returns range of kinematically allowed Fermi momentum of nucleon in nucleus for given Q^2 and ν.
2. `class SmithMonizQELCCPXSec` contains methods for calculation of the differential cross sections. It contains the following methods:
   - `double d3sQES_dQ2dvdkF_SM (const Interaction * interaction) const` – returns differential cross section $\frac{d\sigma}{dQ^2 d\nu dk_F}$;
   - `double d2sQES_dQ2dv_SM(const Interaction * i) const` – returns differential cross section $\frac{d\sigma}{dQ^2 d\nu}$ which is a result of integration of $\frac{d\sigma}{dQ^2 d\nu dk_F}$ using Gauss quadrature;

[1] The paper contains some minor typos.

- `double dsQES_dQ2_SM(const Interaction * interaction) const` – returns differential cross section $\frac{d\sigma}{dQ^2}$ calculated by the well-known "ABC-formula" [5] for the free nucleon target (also used for the deuterium targets after applying the correction factor which takes into account nuclear effects according to model by Singh and Arenhövel [6].[2]

Depending on the phase space passed to the method `double SmithMonizQELCCPXSec::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const` the appropriate method for calculation differential cross section is chosen. If phase space is **kPSQ2fE** then the method `double dsQES_dQ2_SM(const Interaction * interaction) const` is called, if phase space is **kPSQ2vfE** then the method `double d2sQES_dQ2dv_SM(const Interaction * i) const` is called. New phase space **kPSQ2vfE** is defined in file KinePhaseSpace.h.

3. `class SmithMonizQELCCXSec` implements methods to integrate differential cross sections. The appropriate integrator is selected according to target mass A. If $A < 3$ then 1D integrator implemented in `class utils::gsl::dXSec_dQ2_E` is called, if $A \geq 3$ then 2D integrator implemented in `class genie::utils::gsl::d2Xsec_dQ2dv::d2Xsec_dQ2dv` is called.
The `class genie::utils::gsl::d2Xsec_dQ2dv::d2Xsec_dQ2dv` implements a new 2D integrator for to integrate the cross section $\frac{d\sigma}{dQ^2 d\nu dk_F}$.

4. `class QELEventGeneratorSM` is responsible for generating events according to the Smith-Moniz model. It picks Q^2 and ν and calculates momenta of incoming and outgoing particles from these kinematic variables. The kinematic 2D region (Q^2,ν) is divided into two parts: $Q^2 \leq 2$ GeV^2 and $Q^2 > 2$ GeV^2 because the distribution $\frac{d\sigma}{dQ^2 d\nu}$ depended on Q^2 and ν has the pronounced peak in the region $Q^2 < 2$ GeV^2. The kinematic variables is pick either from the first or the second part accordingly to relative probability of the event generation in each part. Such division notably speeds up the event generation. The maximum values of the distribution $\frac{d\sigma}{dQ^2 d\nu}$ in these regions are calculated and cached. The maximum value in the first region is calculated by the method `double ComputeMaxXSec(const Interaction * in) const` and cached by the method `void CacheMaxXSec(const Interaction * in, double xsec) const`. The methods `double ComputeMaxXSec2(const Interaction * in) const` and `void CacheMaxXSec2(const Interaction * in, double xsec) const` do the same in the second region ($Q^2 > 2$ GeV^2).

   Because the maximums are located at the beginning of each region they are sought on the logarithmic grid. This allows us to find with guarantee the maximum value on a not-too shallow grid through a small number of steps. After the kinematic variable Q^2 is pick, another variable ν is pick at given Q^2 according to distribution
   `Range1D_t vQES_SM_lim(double Q2) const`. The maximum value of the distribution is calculated in the method
   `double ComputeMaxDiffv (const Interaction * in) const` and cached in the method
   `void CacheMaxDiffv (const Interaction * in, double xsec) const`.

We don't place the code here because it is too bulk. For more details, see the **smith_moniz** branch of GENIE issue tracker.

A comparison between the double-differential CCQE cross sections calculated with the three interaction models [3] incorporated in GENIE (see Table 1.1) and MiniBooNE data [7, 8] is shown in Fig. 1.1 by histograms. The red solid curves in the Figure represent calculations performed with our own program using the Smith-Moniz RFG model with the same input parameters as in GENIE.

Table 1.1: Three different models used in Fig. 1.1 to compare with the MiniBooNE dataset.

Plot label:	QEL CC model	MEC CC model	RFG model
$M_A^{\rm run}$	SmithMonizQELCCPXSec	—	RFG by Smith & Moniz
Empirical MEC	LwlynSmithQELCCPXSec	EmpiricalMECPXSec2015	RFG by Bodek & Ritchie
Nieves et al., LFG	NievesQELCCPXSec	NievesSimoVacasMECPXSec2016	LFG by GENIE

[2] More specifically, to account for the nuclear effects, we adopt the closure approximation over the dinucleon states following Ref. [6], where the MEC contributions were estimated using single-pion exchange diagrams in the static limit. In this version of the code, the Reid hard-core potential and Hulthen wave function for the deuteron are adopted, as providing the best description of the νd data.

[3] To make calculation with model labeled "Nieves et al., LFG" somy changes to the `class QELEventGeneratot` were made because now it is under construction.

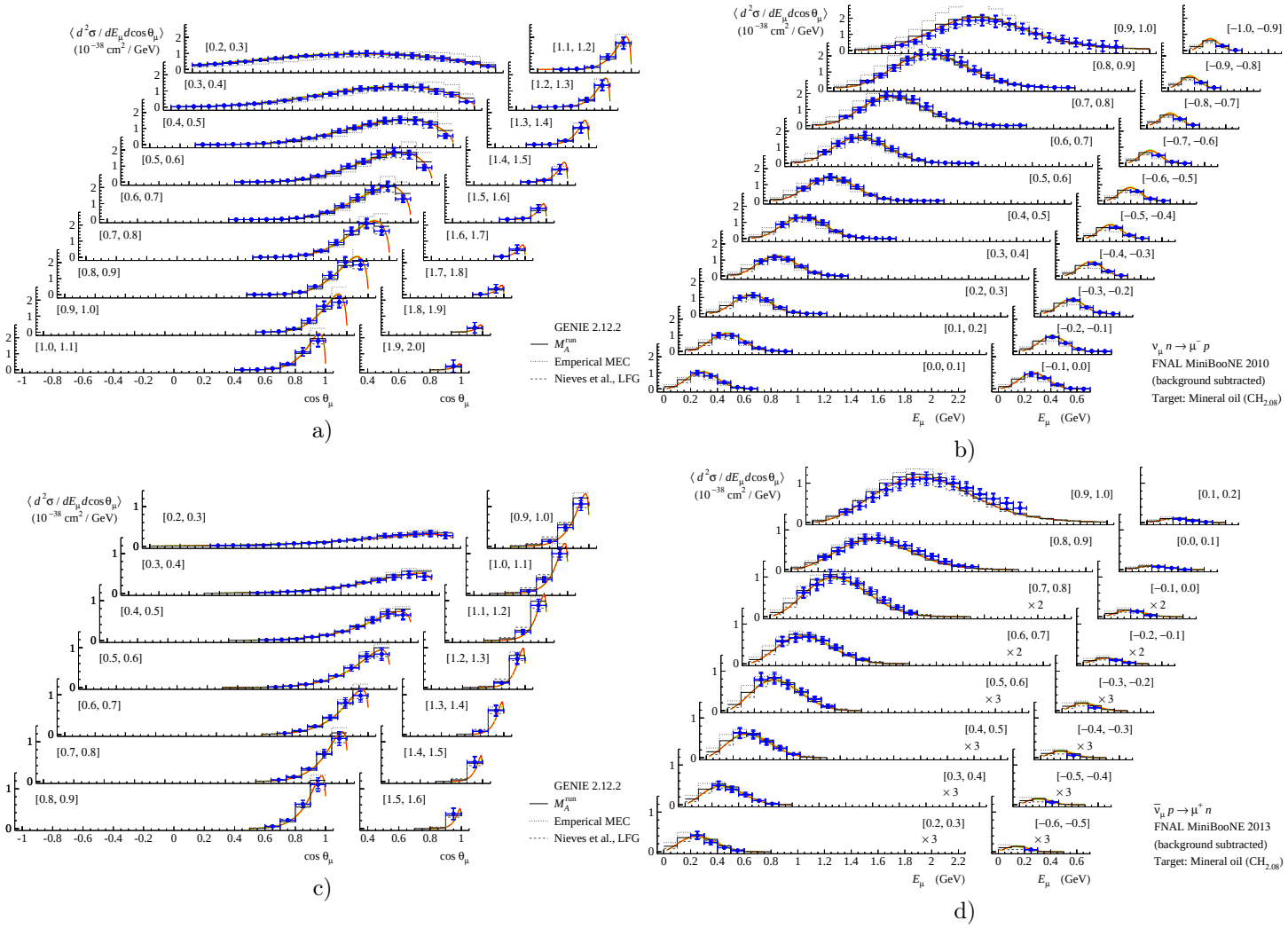

Figure 1.1: Double-differential cross sections of ν_μ and $\overline{\nu}_\mu$ CCQE interactions with hydrocarbon averaged over the (anti)neutrino energy spectrum. The data points of the MiniBooNE experiment are taken from Refs. [7, 8].

1.2 Implementation of the BBBA07 model for the Sachs electromagnetic form factors.

Status: implemented in GENIE 3.0.0

Files: src/ElFF/BBA07ELFormFactorsModel.h,
src/ElFF/BBA07ELFormFactorsModel.cxx,
config/UserPhysicsOptions.xml,
config/BBA07ELFormFactorsModel.xml

The latest and the most comprehensive parametrization of the electric and magnetic form factors of the nucleon, $G_E^{p,n}$ and $G_M^{p,n}$, developed by Bodek *et al.* [9] (BBBA07) is recommended by the authors as a preferred. This model is already implemented into GENIE since version 2.6.0 but cannot be used because it is considered now as "experimental feature" and not polished. So we suggest to include our polished and tested implementation of the model BBBA07 in GENIE. For details, please see the source files in the **smith_moniz** branch of GENIE issue tracker.

1.3 A new class for fast adaptive resonance integrators.

Status: implemented in GENIE 3.0.0

Files: src/ReinSehgal/ReinSehgalRESXSecWithCacheFast.h,
src/ReinSehgal/ReinSehgalRESXSecWithCacheFast.cxx,
src/ReinSehgal/ReinSehgalRESXSecFast.h,
src/ReinSehgal/ReinSehgalRESXSecFast.cxx,
src/ReinSehgal/LinkDef.h,
config/master_config.xml config/ReinSehgalRESXSecFast.xml,

Very important problem concerns the integration speed for calculating the resonance production cross sections. The faster integrator implemented in ROOT and used by GENIE is the adaptive integrator (see Tables 1.2 and 1.3) based on the Genz-Malik adaptive algorithm [10]. However there is a caution while using it. This type of integrator does not work properly with a discontinuous integrand. For calculating resonance cross sections at a given neutrino energy one needs to integrate the double-differential cross section $d^2\sigma/dWdQ^2$ over the kinematically allowed region, where W and Q^2 are the integration variables. In GENIE, the function responsible for the calculation of double-differential cross section simply returns zero, when the kinematic variables lie out of the allowed region. Thus the integrand is in general discontinuous, that is why the calculation of the integrals like

$$\int \mathrm{d}W\mathrm{d}Q^2 \frac{d^2\sigma}{dWdQ^2}(W,Q^2)$$

by the adaptive method may yield (and in fact yields) erroneous results. To make it functional, one needs to make a change of variables in order to integrate the double-differential cross sections exactly over the kinematically allowed region, where the integrand is definitely continuous.

To implement the aforesaid two new integrators (with and without caching) are added: `class ReinSehgalRESXSecWithCacheFast` and `class ReinSehgalRESXSecFast`. For more detail see in the **smith_moniz** branch of GENIE issue tracker.

Table 1.2: A comparison of efficiencies of four different integration methods used in GENIE.

Integration method:	PLAIN	MISER	VEGAS	ADAPTIVE
average relative error	5.70×10^{-2}	1.55×10^{-3}	2.55×10^{-5}	1.51×10^{-5}
real	0m42.186s	0m46.957s	3m8.819s	0m2.970s
user	0m41.084s	0m45.868s	3m7.140s	0m1.916s
sys	0m0.068s	0m0.048s	0m0.084s	0m0.044s

Table 1.3: A comparison of VEGAS and ADAPTIVE integrators.

Method:	VEGAS *gsl-ncalls*=10000, *ESplineMax*=400, *is_FastReinSehgal*=false	ADAPTIVE *gsl-ncalls*=1000000000, *gsl-relative-tolerance*=10^{-6}, *ESplineMax*=400, *is_FastReinSehgal*=true
real	336m2.043s	7m54.403s
user	3337m35.096s	60m18.866s
sys	0m38.722s	0m0.268s

Table 1.2 shows a comparison between the computing times needed for calculation of the $\nu_\mu + p \to \Delta^{++} + \mu$ cross section (in the Rein-Sehgal model) by different methods. The Monte-Carlo integrators with the default settings and fast adaptive integrator have been compared by the command:

```
time gmkspl -p 14 -t 1000010010 -n 100 -e 200 --event-generator-list CCRES
```

on processor Intel® Core™ i7-3610QM CPU @ 2.30GHz. The relative bug of the ADAPTIVE integrator has been set to the average relative error of the VEGAS integrator.

In addition, the execution time of the command

```
gmkspl -p -16,-14,-12,12,14,16 -t 1000010010,1000000010 -e 400.1 -n 100 --event-generator-list RES
```

using the Berger-Sehgal model has been determined for the VEGAS and ADAPTIVE integrators on a 12-core processor Intel® Xeon® CPU X5650 @ 2.67GHz (all 18 resonances have been included); the result of this comparison is shown in Table 1.3.

In the above tests, each projectile–target pair was calculated on a separate processor core.

1.4 Pauli blocking for resonance single pion production models models.

Status: implemented in GENIE 3.0.0

Files: src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx,
src/ReinSehgal/BSKLNBaseRESPXSec2014.h,
src/ReinSehgal/ReinSehgalRESPXSec.cxx,
src/ReinSehgal/ReinSehgalRESPXSec.h,
src/ReinSehgal/ReinSehgalRESXSec.h,
src/ReinSehgal/ReinSehgalRESXSec.cxx,
src/ReinSehgal/ReinSehgalRESXSecFast.h,
src/ReinSehgal/ReinSehgalRESXSecFast.cxx,
config/BergerSehgalRESPXSec2014.xml,
config/KuzminLyubushkinNaumovRESPXSec2014.xml,
config/ReinSehgalRESPXSec.xml,
config/UserPhysicsOptions.xml

The cross sections of resonance single pion production models are calculated on the bare nucleon in GENIE. We suggest to implement a simple model to account for the Pauli blocking [11–13]. For more detail see mentioned above files in the **smith_moniz** branch of GENIE issue tracker.

Chapter 2

Future plans

NOTE: Some of the suggested new options and improvements not interesting for the GENIE Collaboration will be implemented only to our local copy of the packet used in the NOνA near detector experiments and in the JUNO experiment.

General options

1. Inclusion of an option for calculating the final lepton polarization in the quasielastic scattering, (anti)neutrino-induced single pion production, and DIS.
2. Improvement of the default GENIE parametrization for the mean charged-secondary-hadron multiplicity in full phase space for the charged-current neutrino and antineutrino scattering on hydrogen and deuterium, and maybe (at least, in the not so near future) on heavier nuclear targets according to Ref. [14].

CCQE

1. Adjustment of the parameters of the running axial mass for the Bodek-Ritchie RFG model [15, 16] (the default RFG model in GENIE).
2. An improvement of the empirical MEC model on the basis of the running axial mass model.
3. Update of the atomic/nuclear masses using the latest AME data [17].
4. Implementation of the most accurate current models for the electromagnetic form factors of the nucleon.
5. Account of the non-leading contributions into the hadronic current which take into account the proton-neutron mass difference (+ exact kinematics) for the CCQE scattering on free nucleons [18] (an extension of the Strumia-Vissani results [19]).
6. Inclusion of the second-class weak currents (scalar and tensor form factors).[1]
7. Account of the radiative corrections and contributions of weak magnetism and neutron recoil to next-to-leading order in the expansion in inverse powers of the nucleon mass to the IBD reactions.[2]
8. Inclusion of the improved nuclear correction models for the $\overline{\nu}d$ and νd reactions.
9. Inclusion of the option for calculating the cross sections for the light strange hyperon quasielastic production (CCQE with hypercharge change $\Delta Y = 1$) on the basis of the model of Ref. [20].

Resonance single pion neutrinoproduction within the RS-KLN-BS model(s)

1. Account of the interferences between the neighboring resonances which have the same spin and πN orbital angular momentum.
2. Implementation of different models for the non-resonance background (NRB).
3. Inclusion of the option for calculation of the angular distribution (triple-differential cross sections).
4. Inclusion of the improved default input parameters (axial mass, cutoffs, NRB, etc.) obtained from the new global fit to the past (already used in the analysis [18]) and current data on single-pion neutrinoproduction.[3]
5. Inclusion of the Smith-Moniz RFG model [4] for Δ-resonance neutrinoproduction.

[1] For application to the experiments looking for the Standard Model violating effects in the neutrino sector.
[2] For the reactor antineutrino experiments and SBL experiments with artificial low-energy $\overline{\nu}_e$ radioactive sources.
[3] The analysis is in progress.

Any suggestions and comments are welcome.

Chapter 3

Historical Matters: Already fixed bugs.

3.1 Bug: Incorrect calculation of the empirical MEC-NC contribution.

Status: fixed in GENIE 2.12.0
Files: src/MEC/EmpiricalMECPXSec2015.cxx
Method: `double EmpiricalMECPXSec2015::Integral(const Interaction * interaction) const`

The cross section calculated by the following operator

```
double xsec = 0.5*(Z*fXSecAlgNCQE->Integral(inp) + N*fXSecAlgNCQE->Integral(inn));
```

systematically overestimates the desired MEC contribution for the scattering off an isoscalar nucleus, because the methods

`fXSecAlgNCQE->Integral(inp)` and `fXSecAlgNCQE->Integral(inn)`

actually return the cross sections for the scattering off a *nucleus* rather than these for the scattering off a *bound nucleon*. To correct this typo, the above line should be replaced by the following one:

```
double xsec = (Z*fXSecAlgNCQE->Integral(inp) + N*fXSecAlgNCQE->Integral(inn))/(Z+N);
```

3.2 Bug: Incorrect definition of mathematical constants.

Status: fixed in GENIE 2.12.0
Files: src/Conventions/Constants.h

Two literals were set incorrectly:

```
static const double kSqrt2_7   =  0.28571429;
...
static const double kSqrt6_35  =  0.17142857;
```

3.3 Bug: Wrong sign in the KLN extension of the Rein-Sehgal model for the resonance single pion neutrinoproduction.

Status: fixed in GENIE 2.10.0
Files: src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx
Method: `double BSKLNBaseRESPXSec2014.cxx::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const`

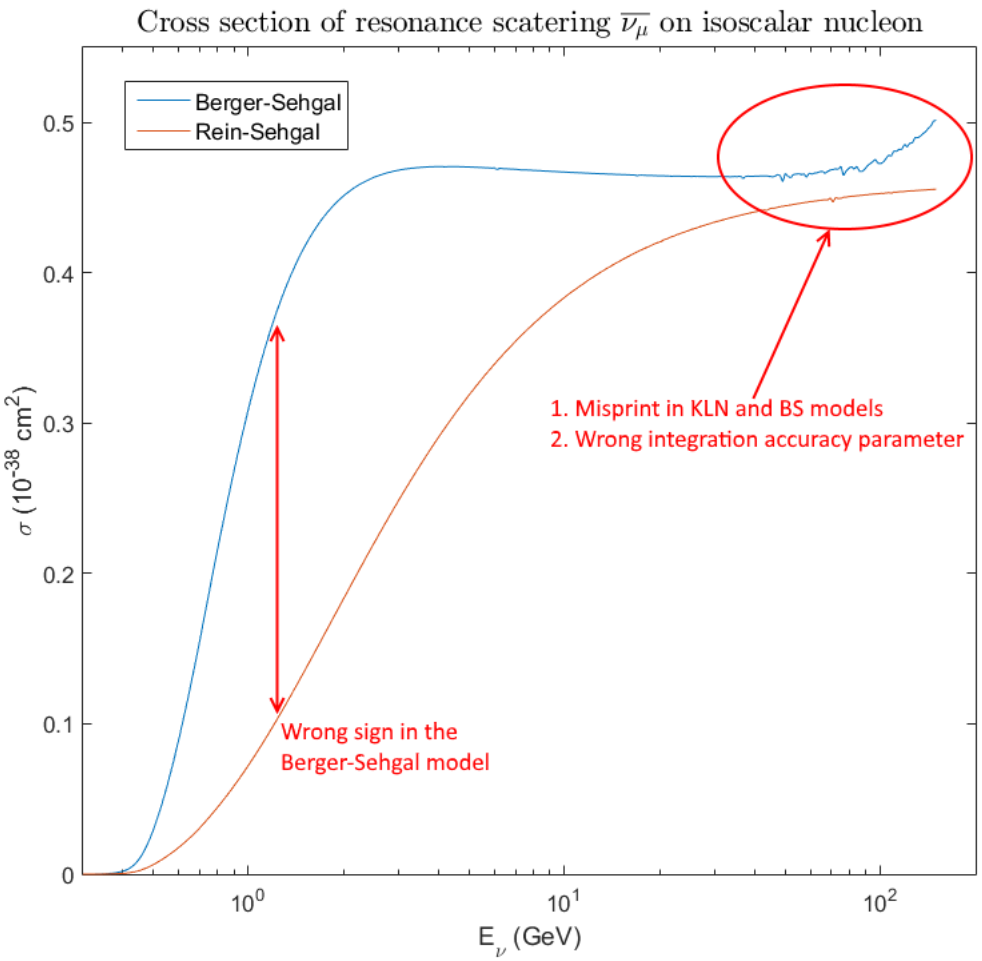

Figure 3.1: Cross sections of $\overline{\nu}_\mu$ resonance scattering on the isoscalar nucleon vs. energy. The curves are calculated with the Berger-Sehgal and Rein-Sehgal models and illustrate the bugs and misprints in the Berger-Sehgal code.

According the formulas for antineutrino [21]

$$\left[c_L^\lambda\right]_{\overline{\nu}} = \lambda \left[c_R^{-\lambda}\right]_\nu, \quad \left[c_R^\lambda\right]_{\overline{\nu}} = \lambda \left[c_L^{-\lambda}\right]_\nu, \quad \left[c_S^\lambda\right]_{\overline{\nu}} = -\lambda \left[c_S^{-\lambda}\right]_\nu,$$

the following lines

```
if (is_nubar  is_lplus) {
     KNL_cL_plus  =  1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_minus - KNL_jy_minus);
     KNL_cL_minus = -1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_plus  - KNL_jy_plus);

     KNL_cR_plus  =  1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_minus + KNL_jy_minus);
     KNL_cR_minus = -1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_plus  + KNL_jy_plus);
}
```

should be replaced by the following ones:

```
if (is_nubar  is_lplus) {
     KNL_cL_plus  =  1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_minus  + KNL_jy_minus);
     KNL_cL_minus = -1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_plus   + KNL_jy_plus);
```

```
      KNL_cR_plus  =  1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_minus  - KNL_jy_minus);
      KNL_cR_minus = -1 * TMath::Sqrt(0.5)* KNL_K * (KNL_jx_plus   - KNL_jy_plus);
}
```

3.4 Bug and improvement: Wrong integration accuracy parameter for the Monte-Carlo integrator.

Status: fixed in GENIE 2.11.0 (obsolete)

Files: src/Base/XSecIntegratorI.h,
src/ReinSehgal/ReinSehgalRESXSecWithCache.cxx,
src/ReinSehgal/ReinSehgalRESXSec.cxx,
config/ReinSehgalRESXSec.xml

The accuracy parameter *gsl-relative-tolerance* is not actually used by Monte-Carlo integrators, like PLAIN, MISER and VEGAS. It is therefore suggested to introduce three new parameters for tuning the integration accuracy (apply only to the MC integration methods). The first parameter, *gsl-ncalls*, sets the maximum number of integrand calls, the second one, *gsl-threshold*, sets the threshold above which *gsl-ncalls* is multiplied by the third parameter *gsl-ncalls-factor*. For this purpose the following members shoud be added to `class XSecIntegratorI` in the file XSecIntegratorI.h:

```
int fGSLNCalls;           ///< GSL number of function calls (apply only to the MC integratioon methods)
double fGSLThreshold;     ///< the threshold for the neutrino energy, above which the initial number of the integrand calls,
    gsl-ncalls,
                              is multiplied by fGSLNCallsFactor
double fGSLNCallsFactor;  ///< factor that is multiplied by the initial value of gsl-ncalls, when the threshold is reached
```

The method `ReinSehgalRESXSec::LoadConfig(void)` in file ReinSehgalRESXSec.cxx should be replaced by the following

```
void ReinSehgalRESXSec::LoadConfig(void)
{
  AlgConfigPool * confp = AlgConfigPool::Instance();
  const Registry * gc = confp->GlobalParameterList();

  // Get GSL integration type
  fGSLIntgType = fConfig->GetStringDef("gsl-integration-type",  "vegas");
  fGSLNCalls   = fConfig->GetIntDef("gsl-ncalls", 100000);
  fGSLThreshold= fConfig->GetDoubleDef("gsl-ncalls-threshold", 50);
  fGSLNCallsFactor= fConfig->GetDoubleDef("gsl-ncalls-factor", 1);


  // Get upper E limit on res xsec spline (=f(E)) before assuming xsec=const
  fEMax = fConfig->GetDoubleDef("ESplineMax", 100);
  fEMax = TMath::Max(fEMax,20.); // don't accept user Emax if less than 20 GeV

  // Create the baryon resonance list specified in the config.
  fResList.Clear();
  string resonances = fConfig->GetStringDef(
                  "resonance-name-list", gc->GetString("ResonanceNameList"));
  fResList.DecodeFromNameList(resonances);
}
```

The analogous method `ReinSehgalSPPXSec::LoadConfig(void)` in the file ReinSehgalSPPXSec.cxx should be replaced by the following

```
void ReinSehgalSPPXSec::LoadConfig(void)
{
  AlgConfigPool * confp = AlgConfigPool::Instance();
  const Registry * gc = confp->GlobalParameterList();

  // Get GSL integration type & relative tolerance
  fGSLIntgType = fConfig->GetStringDef("gsl-integration-type",  "adaptive");
  fGSLNCalls   = fConfig->GetIntDef("gsl-ncalls", 100000);
  fGSLThreshold= fConfig->GetDoubleDef("gsl-threshold", 50);
  fGSLNCallsFactor= fConfig->GetDoubleDef("gsl-ncalls-factor", 1);

  // get upper E limit on res xsec spline (=f(E)) before assuming xsec=const
  fEMax = fConfig->GetDoubleDef("ESplineMax", 100);
  fEMax = TMath::Max(fEMax,20.); // don't accept user Emax if less than 20 GeV

  // create the baryon resonance list specified in the config.
  fResList.Clear();
  string resonances = fConfig->GetStringDef("ResonanceNameList", gc->GetString("ResonanceNameList"));
  fResList.DecodeFromNameList(resonances);
}
```

The lines

```
ROOT::Math::IntegratorMultiDim ig(ig_type);
ig.SetRelTolerance(fGSLRelTol);
```

in the methods

```
void ReinSehgalRESXSecWithCache::CacheResExcitationXSec(const Interaction * in) const  and
double ReinSehgalRESXSec::Integrate(const XSecAlgorithmI * model, const Interaction * interaction) const
```

in the files ReinSehgalRESXSecWithCache.cxx and ReinSehgalRESXSec.cxx should be replaced by the following line

```
ROOT::Math::IntegratorMultiDim ig(ig_type,0,fGSLRelTol,Ev<fGSLThreshold  ?  fGSLNCalls : fGSLNCalls*fGSLNCallsFactor);
```

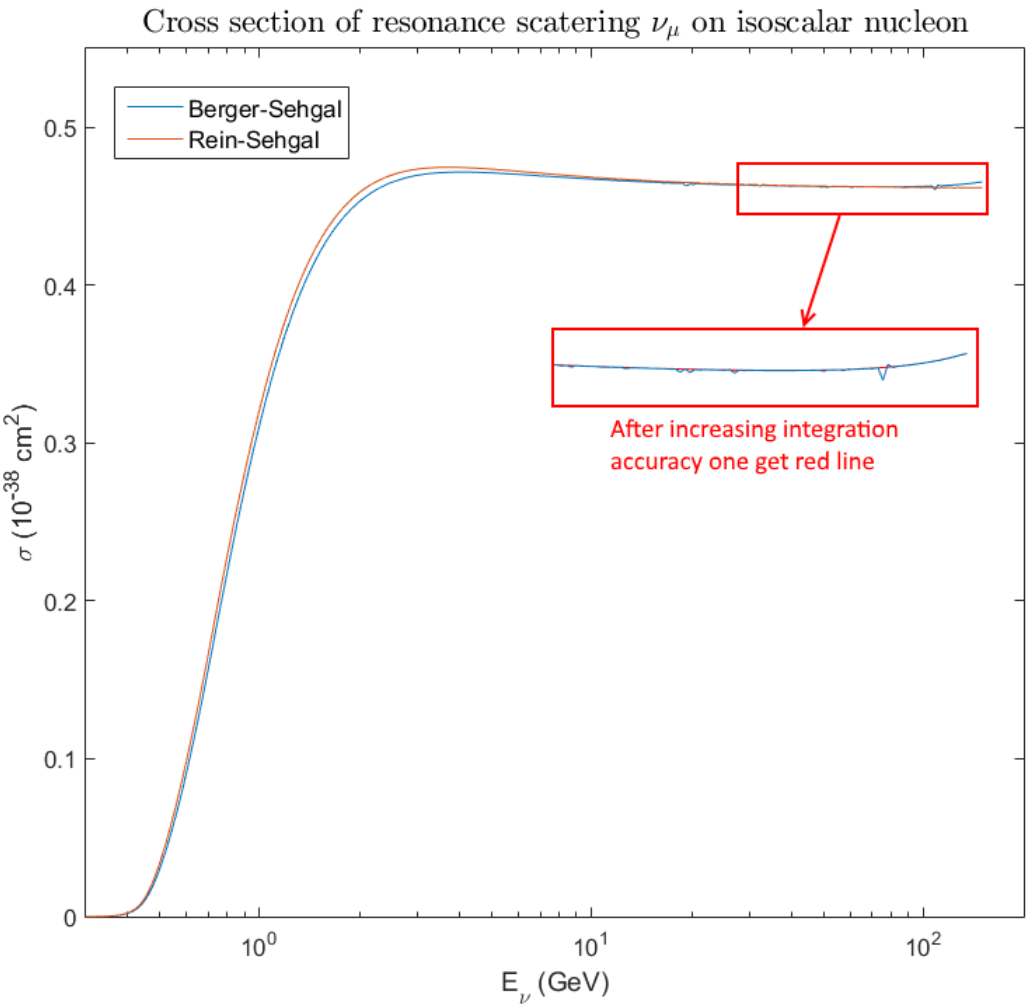

Figure 3.2: Cross section of ν_μ vs energy. These graphs are calculated by the Berger-Sehgal and the Rein-Sehgal models and illustrate the effect of increasing integration accuracy after the code modification described in Section 3.4.

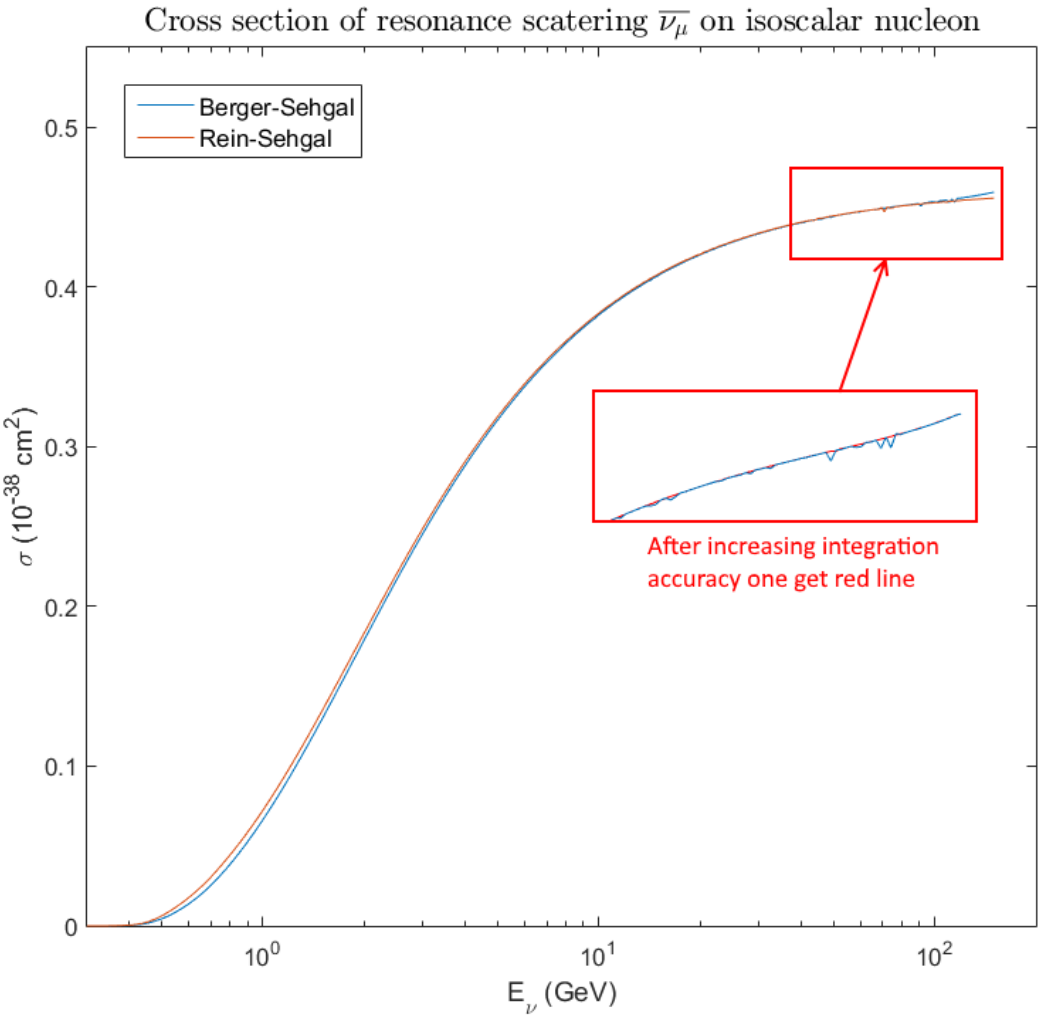

Figure 3.3: Cross section of $\overline{\nu}_\mu$ vs energy. These graphs are calculated by the Berger-Sehgal and Rein-Sehgal models and illustrate the effect of increasing integration accuracy after code modification described in Section 3.4.

3.5 Bug: Misprints in the Berger-Sehgal model.

Status: fixed in GENIE 2.11.0
Files: src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx
Method: `double BSKLNBaseRESPXSec2014::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const`

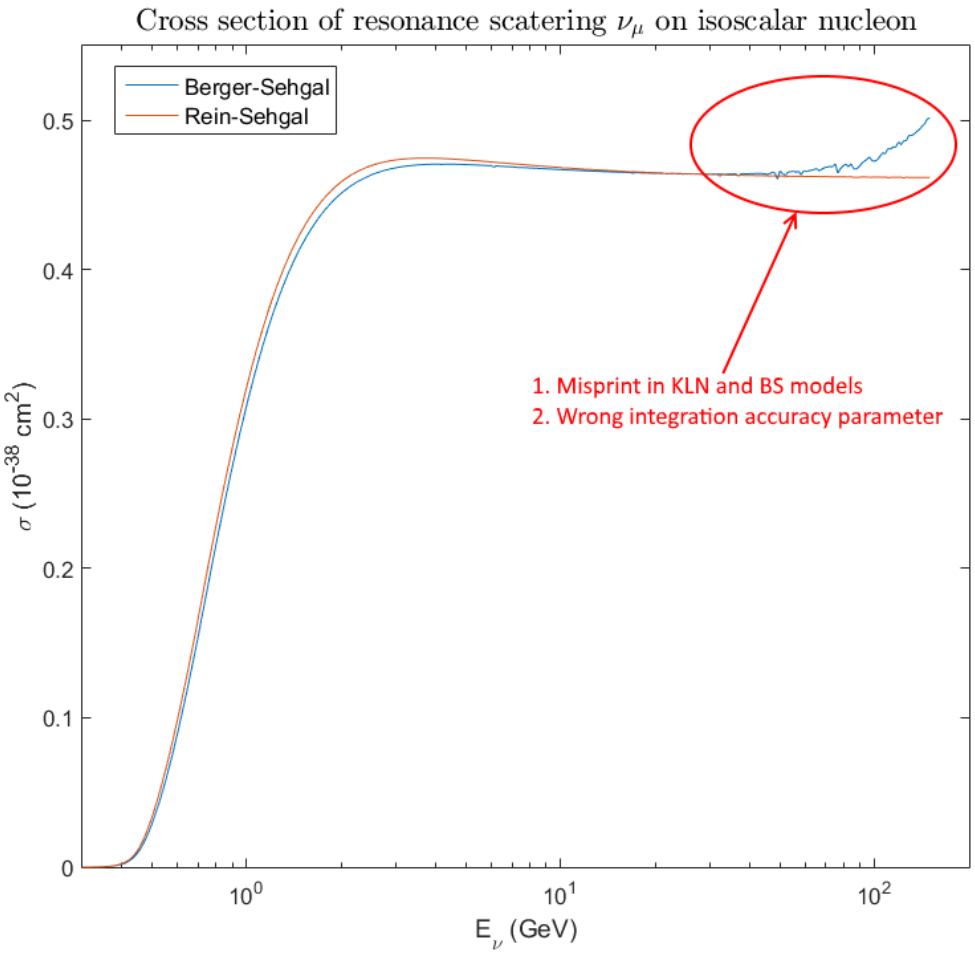

Figure 3.4: Cross sections of ν_μ resonance scattering on the isoscalar nucleon vs. energy. The curves are calculated with the Berger-Sehgal and Rein-Sehgal models and illustrate the bugs and misprints in the Berger-Sehgal code.

All occurrences of the following code

```
fFKR.S = KNL_S_minus;
fFKR.B = KNL_B_minus;
fFKR.S = KNL_C_minus;
```

should be replaced by the following one

```
fFKR.S = KNL_S_minus;
fFKR.B = KNL_B_minus;
fFKR.C = KNL_C_minus;
```

and all occurrences of the following code

```
fFKR.S = BRS_S_minus;
fFKR.B = BRS_B_minus;
fFKR.S = BRS_C_minus;
```

should be replaced by the following one

```
fFKR.S = BRS_S_minus;
fFKR.B = BRS_B_minus;
fFKR.C = BRS_C_minus;
```

3.6 Bug and improvement: Calculation of Breit-Wigner normalization factor "on-the-fly".

Status: fixed in GENIE 2.11.0

Files: src/ReinSehgal/ReinSehgalRESPXSec.cxx,
src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx,
src/BaryonResonance/BaryonResUtils.h,
src/BaryonResonance/BaryonResUtils.cxx

The normalization factor $\eta_{BW}^{(\nu)}(W)$ in Eq. (2.33) of Ref. [22] should be calculated as

$$N_\nu = \int_{W_{min}}^{\infty} dW \, \frac{\Gamma_\nu}{2\pi} \, \frac{1}{(W - M_\nu)^2 + \Gamma_\nu^2/4}, \tag{3.1}$$

where

$$\Gamma_\nu = \Gamma_\nu^0 \left[\frac{q_\pi(W)}{q_\pi(M_\nu)} \right]^{2L+1} \quad \text{and} \quad q_\pi(W) = \frac{1}{2W} \sqrt{\left(W^2 - m_N^2 - m_\pi^2\right)^2 - 4 m_N^2 m_\pi^2}.$$

To calculate the integral (3.1) numerically, it is needed to use a cut off, that is some definite upper limit in the integration variable W. The following code is responsible for calculation of the factor N_ν in GENIE 2.10.x:

```
double NR  = utils::res::BWNorm (resonance);
// Following NeuGEN, avoid problems with underlying unphysical
// model assumptions by restricting the allowed W phase space
// around the resonance peak
if      (W > MR + fN0ResMaxNWidths * WR && IR==0) return 0.;
else if (W > MR + fN2ResMaxNWidths * WR && IR==2) return 0.;
else if (W > MR + fGnResMaxNWidths * WR) return 0.;
```

However there is a problem here. The factor NR is calculated with the fixed values of the parameters

```
MR, WR, N0ResMaxNWidths, N2ResMaxNWidths,  and GnResMaxNWidths,
```

while in fact they are subject of change since, there is no any physical reasons for using the currently adopted "recipe". So it seems reasonable to realize the calculation of NR "on-the-fly". Any case, the user should keep this ambiguity in mind. In order to implement the mentioned improvements, the following code

```
double BWNorm (Resonance_t res);  ///< breit-wigner normalization factor
```

should be removed from the file BaryonResUtils.h and the following one

```
#include <map>
......
double BWNorm (Resonance_t res, double N0ResMaxNWidths=6, double N2ResMaxNWidths=2,
                                                          double GnResMaxNWidths=4);  ///< breit-wigner normalization factor
struct CacheBWNorm {
   CacheBWNorm()
   {
        cache[kP33_1232] = 0.0;
        cache[kS11_1535] = 0.0;
        cache[kD13_1520] = 0.0;
        cache[kS11_1650] = 0.0;
        cache[kD13_1700] = 0.0;
        cache[kD15_1675] = 0.0;
        cache[kS31_1620] = 0.0;
        cache[kD33_1700] = 0.0;
        cache[kP11_1440] = 0.0;
        cache[kP33_1600] = 0.0;
        cache[kP13_1720] = 0.0;
        cache[kF15_1680] = 0.0;
        cache[kP31_1910] = 0.0;
        cache[kP33_1920] = 0.0;
        cache[kF35_1905] = 0.0;
        cache[kF37_1950] = 0.0;
        cache[kP11_1710] = 0.0;
        cache[kF17_1970] = 0.0;
   }
   std::map <genie::EResonance, double> cache;
 };
```

should be inserted instead. The function

```
double genie::utils::res::BWNorm(Resonance_t res)
{
  switch(res) {
    case kP33_1232  : return 0.916744 ; break;
    case kS11_1535  : return 0.934833 ; break;
    case kD13_1520  : return 0.920773 ; break;
    case kS11_1650  : return 0.933974 ; break;
    case kD13_1700  : return 0.954652 ; break;
    case kD15_1675  : return 0.924617 ; break;
    case kS31_1620  : return 0.934125 ; break;
    case kD33_1700  : return 0.747914 ; break;
    case kP11_1440  : return 0.762679 ; break;
    case kP33_1600  : return 0.852495 ; break;
    case kP13_1720  : return 0.857668 ; break;
    case kF15_1680  : return 0.789636 ; break;
    case kP31_1910  : return 0.838467 ; break;
    case kP33_1920  : return 0.850868 ; break;
    case kF35_1905  : return 0.622885 ; break;
    case kF37_1950  : return 0.677866 ; break;
    case kP11_1710  : return 0.872677 ; break;
    case kF17_1970  : return 0.950270 ; break;
    default: break;
  }
  return -1;
}
```

should be deleted from the file BaryonResUtils.cxx and the following code

```
#include <map>
#include <TMath.h>
#include "Utils/BWFunc.h"
......
double genie::utils::res::BWNorm(Resonance_t res, double NOResMaxNWidths, double N2ResMaxNWidths, double GnResMaxNWidths)
{
   static genie::utils::res::CacheBWNorm cbwn;
   if (res==kNoResonance)      return -1;
   if (cbwn.cache[res]!=0)   return cbwn.cache[res];

  // Get baryon resonance parameters
  int    IR  = utils::res::ResonanceIndex    (res);
  int    LR  = utils::res::OrbitalAngularMom (res);
  double MR  = utils::res::Mass              (res);
  double WR  = utils::res::Width             (res);

  // imported part of code from src/contrib/misc/bwnorm.C

  double NW = GnResMaxNWidths;
  if(IR==2) NW = N2ResMaxNWidths;
  if(IR==0) NW = NOResMaxNWidths;

  double Wmin = 0.001;
  double Wmax = MR + NW*WR;
  int N = 1000* TMath::Nint( (Wmax-Wmin)/WR );
  if(N%2==0) N++;

  double dW = (Wmax-Wmin)/(N-1);

  double sum = 0.5 * (genie::utils::bwfunc::BreitWignerL(Wmin,LR,MR,WR,1.0) +
      genie::utils::bwfunc::BreitWignerL(Wmax,LR,MR,WR,1.0));

  for(int i=1; i<N-1; i++) {
    double W = Wmin + i*dW;
    sum += ( genie::utils::bwfunc::BreitWignerL(W,LR,MR,WR,1.0) * (i%2+1) );
  }

  sum *= (2.*dW/3.);
  cbwn.cache[res]=sum;
  return cbwn.cache[res];
}
```

should be inserted. Next, instead of the lines

```
double NR  = utils::res::BWNorm (resonance);
......
if      (W > MR + fNOResMaxNWidths * WR && IR==0) return 0.;
else if (W > MR + fN2ResMaxNWidths * WR && IR==2) return 0.;
else if (W > MR + fGnResMaxNWidths * WR) return 0.;
```

in the methods

```
double ReinSehgalRESPXSec::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const and
double BSKLNBaseRESPXSec2014::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const
```

in the files ReinSehgalRESPXSec.cxx and BSKLNBaseRESPXSec2014.cxx, the lines

```
double NR  = utils::res::BWNorm (resonance,fN0ResMaxNWidths,fN2ResMaxNWidths,fGnResMaxNWidths);
......
if (W > MR + fN0ResMaxNWidths * WR && IR==0) return 0.;
else if (W > MR + fN2ResMaxNWidths * WR && IR==2) return 0.;
else if (W > MR + fGnResMaxNWidths * WR) return 0.;
```

should be included.

3.7 Bugs and shortcomings in the single-pion production model.

Status: fixed in GENIE 2.11.0
Files: src/ReinSehgal/ReinSehgalSPPPXSec.cxx

The resonance single pion-production model has some bugs and shortcomings

1. Single-pion production total cross section must be calculated as $\sum_{i=1}^{N} \sigma_i \cdot (Branching\ ratio)_i \cdot (Isospin\ Clebsch - Gordon\ coefficient)_i$, but really it is calculated as $\sum_{i=1}^{N} \sigma_i \cdot (Branching\ ratio)_i^2 \cdot (Isospin\ Clebsch - Gordon\ coefficient)_i^2$, where N is number of all resonances taken into account.
2. Annoying debugging information is printed at each cycle of calculation of the double-differential cross-section.
3. List of resonances to be taken into account is not read from the default GENIE physics and tuning parameters.

To fixed these bugs the following code should be placed instead of the erroneous one into the file ReinSehgalSPPPXSec.cxx

```
//____________________________________________________________________________
/*
 Copyright (c) 2003-2015, GENIE Neutrino MC Generator Collaboration
 For the full text of the license visit http://copyright.genie-mc.org
 or see $GENIE/LICENSE

 Author: Costas Andreopoulos <costas.andreopoulos \at stfc.ac.uk>
         University of Liverpool & STFC Rutherford Appleton Lab - November 22, 2004

 For the class documentation see the corresponding header file.

 Important revisions after version 2.0.0 :

*/
//____________________________________________________________________________

#include "Algorithm/AlgConfigPool.h"
#include "Algorithm/AlgFactory.h"
#include "Base/XSecIntegratorI.h"
#include "BaryonResonance/BaryonResUtils.h"
#include "Conventions/Constants.h"
#include "Conventions/KineVar.h"
#include "Interaction/SppChannel.h"
#include "Messenger/Messenger.h"
#include "ReinSehgal/ReinSehgalSPPPXSec.h"
#include "Utils/KineUtils.h"
#include "Utils/MathUtils.h"

using namespace genie;
using namespace genie::constants;

//____________________________________________________________________________
ReinSehgalSPPPXSec::ReinSehgalSPPPXSec() :
XSecAlgorithmI("genie::ReinSehgalSPPPXSec")
{

}
//____________________________________________________________________________
ReinSehgalSPPPXSec::ReinSehgalSPPPXSec(string config) :
XSecAlgorithmI("genie::ReinSehgalSPPPXSec", config)
{

}
//____________________________________________________________________________
ReinSehgalSPPPXSec::~ReinSehgalSPPPXSec()
{

}
//____________________________________________________________________________
double ReinSehgalSPPPXSec::XSec(
                const Interaction * interaction, KinePhaseSpace_t kps) const
{
  if(! this -> ValidProcess    (interaction) ) return 0.;
  if(! this -> ValidKinematics (interaction) ) return 0.;
```

```
#ifdef __GENIE_LOW_LEVEL_MESG_ENABLED__
  LOG("ReinSehgalSpp", pDEBUG)
                << "Computing a cross section for " << *interaction;
#endif
  //-- Check whether a resonance has been specified
  //   If yes, compute only the contribution of this resonance at the
  //   specified exclusive state

  Resonance_t inpres = interaction->ExclTag().Resonance();
  if(inpres != kNoResonance) {
#ifdef __GENIE_LOW_LEVEL_MESG_ENABLED__
        LOG("ReinSehgalSpp", pDEBUG)
               << "Computing only the contribution from: " << utils::res::AsString(inpres);
#endif
    if(!fResList.Find(inpres)) {
       LOG("ReinSehgalSpp", pWARN)
           << "Resonance: " << utils::res::AsString(inpres) << " was not found in my list";
       return 0;
    }
    //-- Compute the contribution of this resonance
    //-- Get the Breit-Wigner weighted xsec for exciting the resonance

    return fSingleResXSecModel->XSec(interaction,kps);
  }

  //-- Loop over the specified list of baryon resonances and compute
  //   the cross section for the input exclusive channel

  return this->XSecNRES(interaction,kps);
}
//____________________________________________________________________________
double ReinSehgalSPPPXSec::XSecNRES(
                const Interaction * interaction, KinePhaseSpace_t kps) const
{
// computes the 1pi cros section taking into account the contribution of all
// specified baryon resonances

  unsigned int nres = fResList.NResonances();
#ifdef __GENIE_LOW_LEVEL_MESG_ENABLED__
  LOG("ReinSehgalSpp", pDEBUG)
    << "Computing SPP cross section using " << nres << " resonances";
#endif

  //-- Get 1pi exclusive channel
  SppChannel_t spp_channel = SppChannel::FromInteraction(interaction);
#ifdef __GENIE_LOW_LEVEL_MESG_ENABLED__
  LOG("ReinSehgalSpp", pDEBUG)
              << "SPP channel " << SppChannel::AsString(spp_channel);
#endif

  double xsec = 0;
  for(unsigned int ires = 0; ires < nres; ires++) {

     //-- Get next resonance from the resonance list
     Resonance_t res = fResList.ResonanceId(ires);

     //-- Set current resonance to interaction object
     interaction->ExclTagPtr()->SetResonance(res);

     //-- Get the BR for the (resonance) -> (exclusive final state)
         double br = SppChannel::BranchingRatio(spp_channel, res);

         //-- Get the Isospin Glebsch-Gordon coefficient for the given resonance
         //   and exclusive final state
         double igg = SppChannel::IsospinWeight(spp_channel, res);

         //-- Compute the weighted xsec
         //  (total weight = Breit-Wigner * BR * isospin Glebsch-Gordon)
         double res_xsec_contrib = fSingleResXSecModel->XSec(interaction,kps)*br*igg;
#ifdef __GENIE_LOW_LEVEL_MESG_ENABLED__
         LOG("ReinSehgalSpp", pDEBUG)
     << "Contrib. from [" << utils::res::AsString(res) << "] = "
     << "<Glebsch-Gordon = " << igg
     << "> * <BR(->1pi) = " << br
     << "> * <Breit-Wigner * d^nxsec/dK^n = " << rxsec
     << "> = " << res_xsec_contrib;
#endif

     //-- Add contribution of this resonance to the cross section
```

```
    xsec += res_xsec_contrib;
  }

  //-- delete the resonance from the input interaction
  interaction->ExclTagPtr()->SetResonance(kNoResonance);

  return xsec;
}
//____________________________________________________________________________
double ReinSehgalSPPPXSec::Integral(const Interaction * interaction) const
{
  return fXSecIntegrator->Integrate(this,interaction);
}
//____________________________________________________________________________
bool ReinSehgalSPPPXSec::ValidProcess(const Interaction * interaction) const
{
  if(interaction->TestBit(kISkipProcessChk)) return true;

  //-- Get the requested SPP channel
  SppChannel_t spp_channel = SppChannel::FromInteraction(interaction);
  if( spp_channel == kSppNull ) {
    LOG("ReinSehgalSpp", pERROR)
            << "\n *** Insufficient SPP exclusive final state information!";
    return false;
  }
#ifdef __GENIE_LOW_LEVEL_MESG_ENABLED__
  LOG("ReinSehgalSpp", pDEBUG)
                       << "Reaction: " << SppChannel::AsString(spp_channel);
#endif
  return true;
}
//____________________________________________________________________________
void ReinSehgalSPPPXSec::Configure(const Registry & config)
{
  Algorithm::Configure(config);
  this->LoadConfig();
}
//____________________________________________________________________________
void ReinSehgalSPPPXSec::Configure(string config)
{
  Algorithm::Configure(config);
  this->LoadConfig();
}
//____________________________________________________________________________
void ReinSehgalSPPPXSec::LoadConfig(void)
{
// load the single resonance cross section algorithm specified in the config.

  fSingleResXSecModel =
   dynamic_cast<const XSecAlgorithmI *> (this->SubAlg("SingleRESDiffXSecAlg"));
  assert(fSingleResXSecModel);

  //-- Create a BaryonResList by decoding the resonance list from
  //   the XML input
  //   The list of resonances can be specified as a string with
  //   comma separated resonance names (eg "P33(1233),S11(1535),D13(1520)")
  //   The BaryonResList can also decode lists of pdg-codes or
  //   resonance-ids (Resonance_t enumerations).
  //   Support for this will be added here as well.

  fResList.Clear();
  AlgConfigPool * confp = AlgConfigPool::Instance();
  const Registry * gc = confp->GlobalParameterList();
  string resonances = fConfig->GetStringDef(
                    "ResonanceNameList", gc->GetString("ResonanceNameList"));
  fResList.DecodeFromNameList(resonances);

  //-- load the differential cross section integrator
  fXSecIntegrator =
      dynamic_cast<const XSecIntegratorI *> (this->SubAlg("XSec-Integrator"));
  assert(fXSecIntegrator);
}
//____________________________________________________________________________
```

3.8 Bug: Misspells in the code.

Status: fixed in GENIE 2.11.0

Files: config/LwlynSmithQELCCPXSec.xml,
config/P33PaschosLalakulichPXSec.xml,
config/StrumiaVissaniIBDPXSec.xml,
config/UserPhysicsOptions.xml,
src/LlewellynSmith/LwlynSmithQELCCPXSec.cxx,
src/LlewellynSmith/LwlynSmithQELCCPXSec.h,
src/ReinSehgal/ReinSehgalSPPPXSec.cxx,
etc...

1. In the code: Cabbibo. Should be: Cabibbo
2. In the code: Glebsch. Should be: Clebsch

3.9 Remark: the obsolete resonance parameters.

Status: fixed in GENIE 2.11.0

Files: src/Conventions/Constants.h,
src/Conventions/Units.h,
src/BaryonResonance/BaryonResUtils.cxx

In the file BaryonResUtils.cxx, the masses and widths of the resonances are also set according to PDG-2016 [23]:

```
double genie::utils::res::Mass(Resonance_t res)
{
  switch(res) {
    case kP33_1232  : return 1.232 * units::GeV ; break;
    case kS11_1535  : return 1.535 * units::GeV ; break;
    case kD13_1520  : return 1.515 * units::GeV ; break;
    case kS11_1650  : return 1.655 * units::GeV ; break;
    case kD13_1700  : return 1.700 * units::GeV ; break;
    case kD15_1675  : return 1.675 * units::GeV ; break;
    case kS31_1620  : return 1.630 * units::GeV ; break;
    case kD33_1700  : return 1.700 * units::GeV ; break;
    case kP11_1440  : return 1.430 * units::GeV ; break;
    case kP33_1600  : return 1.600 * units::GeV ; break;
    case kP13_1720  : return 1.720 * units::GeV ; break;
    case kF15_1680  : return 1.685 * units::GeV ; break;
    case kP31_1910  : return 1.890 * units::GeV ; break;
    case kP33_1920  : return 1.920 * units::GeV ; break;
    case kF35_1905  : return 1.880 * units::GeV ; break;
    case kF37_1950  : return 1.930 * units::GeV ; break;
    case kP11_1710  : return 1.710 * units::GeV ; break;
    case kF17_1970  : return 2.190 * units::GeV ; break;
    default: break;
  }
  return -1;
}

double genie::utils::res::Width(Resonance_t res)
{
  switch(res) {
    case kP33_1232  : return 0.117 * units::GeV ; break;
    case kS11_1535  : return 0.150 * units::GeV ; break;
    case kD13_1520  : return 0.115 * units::GeV ; break;
    case kS11_1650  : return 0.140 * units::GeV ; break;
    case kD13_1700  : return 0.150 * units::GeV ; break;
    case kD15_1675  : return 0.150 * units::GeV ; break;
    case kS31_1620  : return 0.140 * units::GeV ; break;
    case kD33_1700  : return 0.300 * units::GeV ; break;
    case kP11_1440  : return 0.350 * units::GeV ; break;
    case kP33_1600  : return 0.320 * units::GeV ; break;
    case kP13_1720  : return 0.250 * units::GeV ; break;
    case kF15_1680  : return 0.130 * units::GeV ; break;
    case kP31_1910  : return 0.280 * units::GeV ; break;
    case kP33_1920  : return 0.260 * units::GeV ; break;
    case kF35_1905  : return 0.330 * units::GeV ; break;
    case kF37_1950  : return 0.285 * units::GeV ; break;
    case kP11_1710  : return 0.100 * units::GeV ; break;
    case kF17_1970  : return 0.500 * units::GeV ; break;
    default: break;
  }
  return -1;
}
```

3.10 Bug: A mistaken calculation in the KLN and BS extensions of the Rein-Sehgal model.

Status: fixed in GENIE 2.11.0
Files: src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx
Method: `double BSKLNBaseRESPXSec2014.cxx::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const`

Incorrect manipulations with the variable *costh* lead to a bend in the plot of the cross section vs. neutrino energy shown in Fig. 3.5.

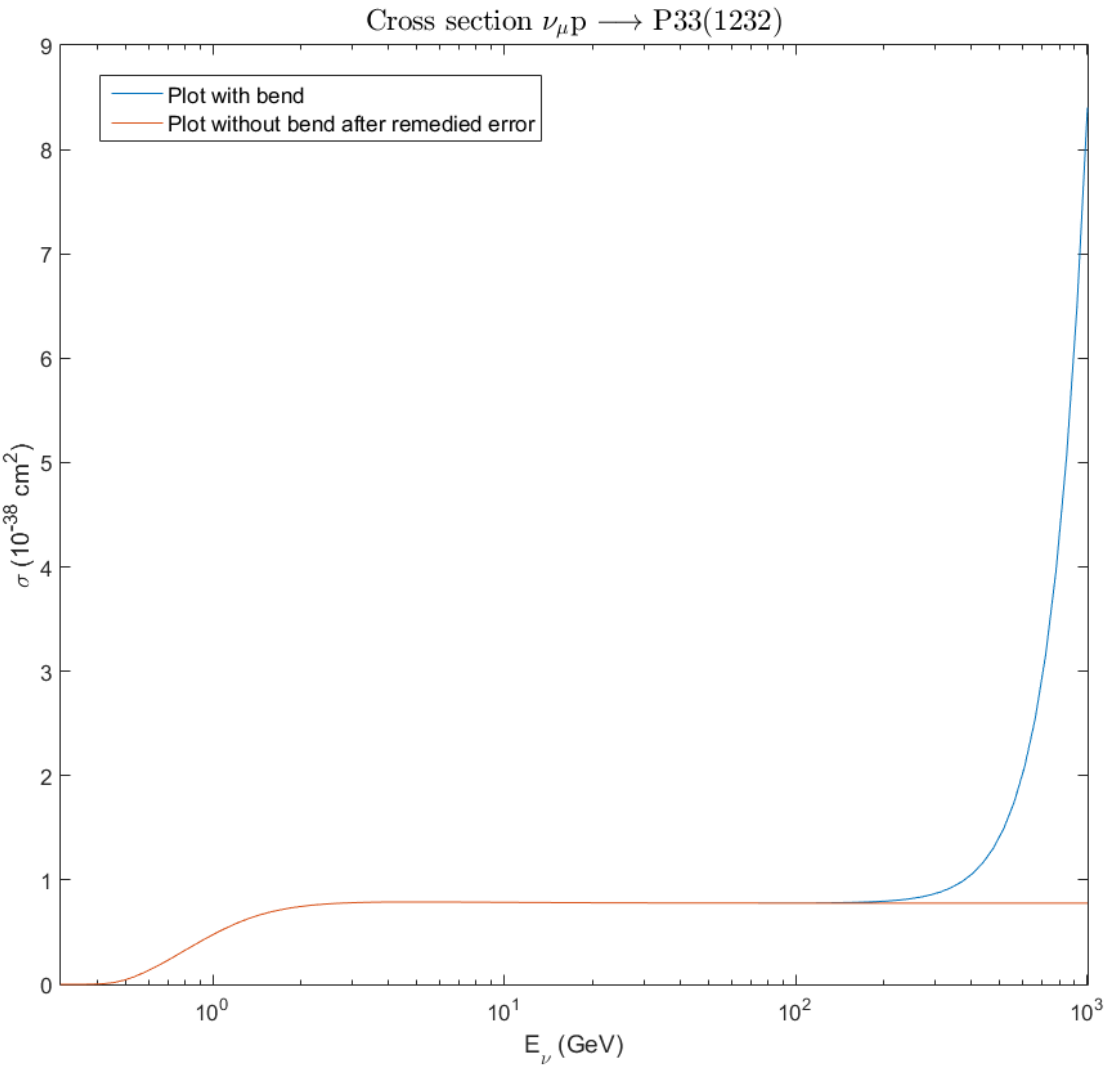

Figure 3.5: Cross section of reaction $\nu_\mu p \to$ P33(1232) vs. neutrino energy.

The following lines

```
Pl = TMath::Sqrt(Eprime*Eprime - ml*ml);
if(costh <= -1. + 1e-7) {
  LOG("BSKLNBaseRESPXSec2014", pDEBUG)
        << "Changing costh = " << costh << " to -1";
  costh = -1 + 1e-6;
}
if(costh >= 1. - 1e-7){
  LOG("BSKLNBaseRESPXSec2014", pDEBUG)
        << "Changing costh = " << costh << " to 1";
  costh = 1 - 1e-6;
}
vstar = (Mnuc*v + q2)/W;//missing W
Qstar = TMath::Sqrt(-q2 + vstar*vstar);
```

should be deleted to remedy the bend.

3.11 Improvement: Implementation of the running axial mass for the CCQE cross sections.

Status: implemented in GENIE 2.11.0

Files: src/LlewellynSmith/LwlynSmithFFCC.h,
src/LlewellynSmith/LwlynSmithFFCC.cxx,
config/LwlynSmithFFCC.xml,
config/UserPhysicsOptions.xml

It has been suggested recently[1] that the nontrivial (beyond the standard relativistic Fermi-gas model, RFG) nuclear effects can be effectively described by so-called "running" axial nucleon mass that is by replacement of the standard (constant) value M_A by a decreasing function of energy, $M_A^{\rm run}$. A detailed statistical analysis to all available accelerator data on the total, differential and double differential cross sections and q^2 distributions for the CCQE interaction with various nuclear targets yields the following simple *ansatz*:

$$M_A^{\rm run} = M_0(1 + E_0/E_\nu). \tag{3.2}$$

It provides a good description of the data with the following set of the parameters:

$$M_0 = 1.006 \pm 0.025 \text{ GeV}, \quad E_0 = 0.334^{+0.058}_{-0.054} \text{ GeV}. \tag{3.3}$$

A comparison of the calculated total CCQE ν_μ and $\overline{\nu}_\mu$ cross sections with the current data is shown in Fig. 3.6. The parameter M_0 can be identified with the conventional axial mass M_A extracted from the deuterium experiments.

The violet lines in the figures show the calculations with GENIE supplemented by the running axial mass (3.2) with the parameters (3.3). The differences with our own calculations are caused by many reasons, in particular, we use the Smith-Minitz RFG model [4] with some small improvements described in Ref. [5], while GENIE uses a bit simplified approach; our electromagnetic form-factor model are also slightly different from that used by GENIE.

To implement CCQE interaction model with the running axial mass ("CCQE+MArun") in GENIE the following lines should be added to the file LwlynSmithFFCC.h

```
protected:
  virtual void LoadConfig (void);

  double fE0;  ///< E0 for calculating running axial mass: Ma*(1+E0/Enu)
  bool fRunMa; ///< Is running axial mass?
```

and the method `double LwlynSmithFFCC::FA(const Interaction * interaction) const` in the file LwlynSmithFFCC.cxx should be replaced by the following

```
 double LwlynSmithFFCC::FA(const Interaction * interaction) const
{

  const InitialState & init_state = interaction->InitState();
  const Target &       target     = init_state.Tgt();

  // get scattering parameters
  const Kinematics & kine = interaction->Kine();
  double q2 = kine.q2();

  // calculate FA(q2)
  double dn;
  if (fRunMa && target.A()>2)
  {
    const InitialState & init_state = interaction -> InitState();
    double E  = init_state.ProbeE(kRfLab);
    dn = TMath::Power(1.-q2/TMath::Power(fMa*(1+fE0/E), 2), 2);
  }
  else
        dn = TMath::Power(1.-q2/fMa2, 2);
  double FA = fFA0/dn;
  return FA;
}
```

and the following method should be added

```
void LwlynSmithFFCC::LoadConfig(void)
{
```

[1] The details will be published elsewhere soon [24]; see also Ref. [25] for a short summary, and Refs. [26, 27] for more details.

```
// Load configuration data from its configuration Registry (or global defaults)
// to private data members
  AlgConfigPool * confp = AlgConfigPool::Instance();
  Registry * gc = confp->GlobalParameterList();
  fE0    = fConfig->GetDoubleDef("E0" , gc->GetDouble("CCQE-E0"));      // E0 for calculating running axial mass:
      Ma*(1+E0/Enu)
  fRunMa = fConfig->GetBoolDef("Run_Ma", gc->GetBool("CCQE-Run_Ma"));   // Is running axial mass?
  return LwlynSmithFF::LoadConfig();
}
```

The configuration file LwlynSmithFFCC.xml should be replaced by the following

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<alg_conf>

<!--
Configuration sets for Llewellyn-Smith QEL CC form factors

Configurable Parameters:
.....................................................................................................
Name                      Type     Optional   Comment                    Default
.....................................................................................................
Ma                        double   Yes        Axial  Mass                GPL: QEl-Ma
FA0                       double   Yes        FA(q2=0)                   GPL: QEL-FA0
E0                        double   Yes        Parameter for Ma_run       GPL: CCQE-E0
Run_Ma                    bool     Yes        Is running axial mass?     GPL: CCQE-Run_Ma
ElasticFormFactorsModel   alg      No         Elastic form factors model GPL: ElasticFormFactorsModel
WeinbergAngle             double   Yes        Weinberg angle             GPL: WeinbergAngle

-->

  <param_set name="Default">
  </param_set>

  <param_set name="Dipole">
     <param type="alg" name="ElasticFormFactorsModel"> genie::DipoleELFormFactorsModel/Default </param>
  </param_set>

</alg_conf>
```

and the lines

```
  <param type="double" name="QEL-Ma"> 0.990 </param>
  <param type="double" name="QEL-Mv"> 0.840 </param>
```

in the global configuration file UserPhysicsOptions.xml should be replaced by the following ones

```
  <param type="double" name="QEL-Ma"> 1.006 </param>
  <param type="double" name="QEL-Mv"> 0.840 </param>
  <param type="double" name="CCQE-E0"> 0.343 </param>
  <param type="bool"   name="CCQE-Run_Ma"> true </param>
```

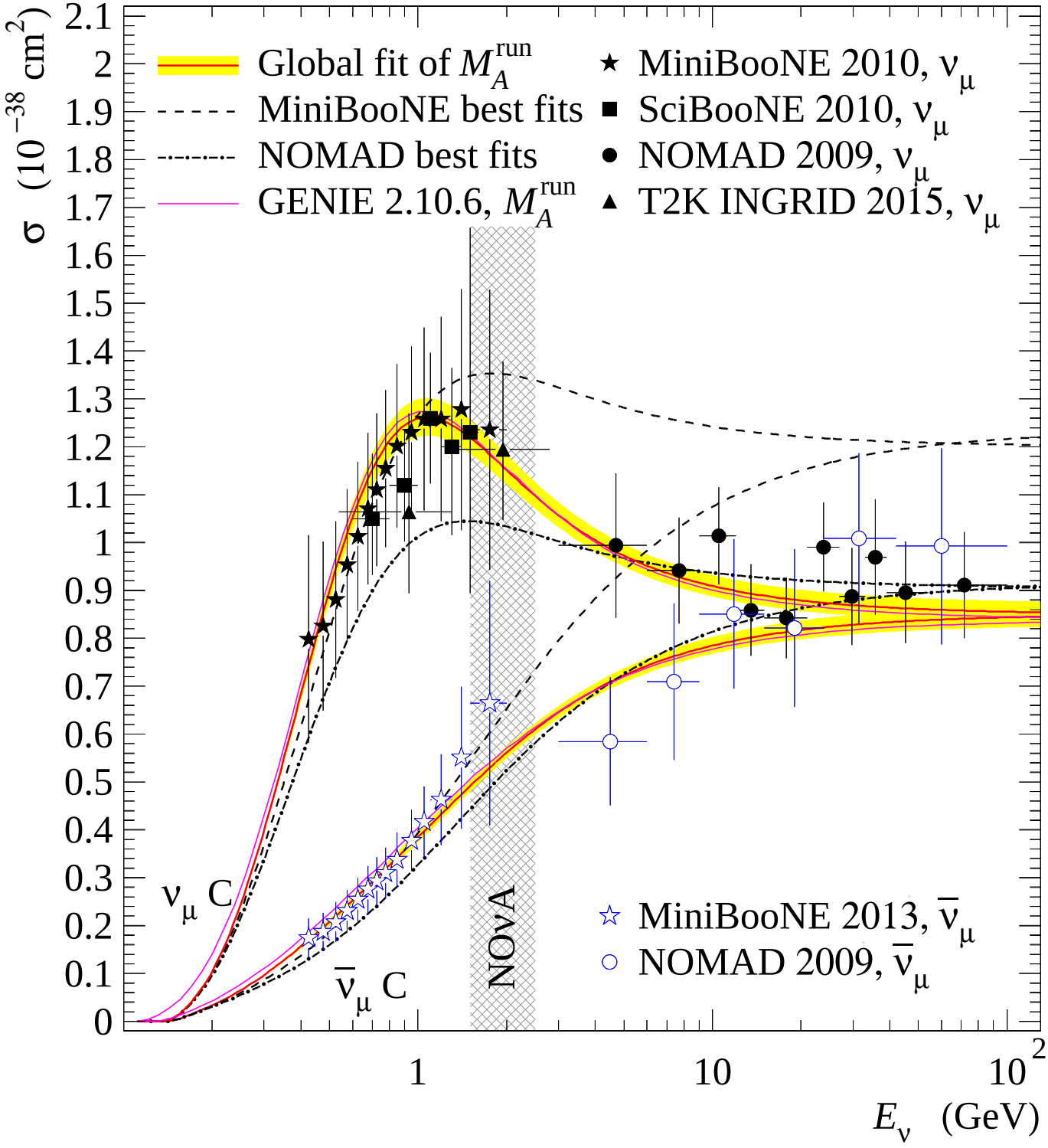

Figure 3.6: Total cross sections for the ν_μ and $\overline{\nu}_\mu$ CCQE interactions with carbon vs. energy. The data points of the NOMAD, MiniBooNE, SciBooNE , and T2K experiments are taken from Refs. [7, 8, 28–34]

3.12 Bug: Memory leaks.

Status: fixed in GENIE 2.11.0
Files: src/MEC/MECGenerator.cxx,
src/Nuclear/FGMBodekRitchie.cxx,
src/HadronTransport/HAIntranuke.cxx

The listed below methods suffer from memory leaks. This is due to the fact that the allocated memory does not released because different bugs. A way to correct these problems is proposed bellow.

In the first case the problem is in call of the method `GetP4()` of the class `GHepParticle` which allocates memory with operator new in it. To fix the problem in the method

```
void MECGenerator::RecoilNucleonCluster(GHepRecord * event) const
```

defined in the file MECGenerator.cxx the line

```
TLorentzVector p4cluster(*nucleon_cluster->GetP4());
```

should be replaced by the following ones

```
TLorentzVector * tmp = nucleon_cluster->GetP4();
TLorentzVector p4cluster(*tmp);
delete tmp;
```

In the second case the problem is due to a misprint in the destructor

```
FGMBodekRitchie::~FGMBodekRitchie()
```

defined in the file FGMBodekRitchie.cxx. To fix it the line

```
for( ; iter != fProbDistroMap.begin(); ++iter) {
```

should be replaced by the following one

```
for( ; iter != fProbDistroMap.end(); ++iter) {
```

In the third case in the method

```
void HAIntranuke::InelasticHA(GHepRecord* ev, GHepParticle* p, INukeFateHA_t fate) const
```

defined in the file HAIntranuke.cxx allocated areas of memory are not released if some conditions are not satisfied. To fix it the lines

```
GHepParticle * t = new GHepParticle(*p);
t->SetPdgCode(tcode);

// set up fermi target
Target target(ev->TargetNucleus()->Pdg());
double tM = t->Mass();

// handle fermi momentum
if(fDoFermi)
{
  target.SetHitNucPdg(tcode);
  fNuclmodel->GenerateNucleon(target);
  TVector3 tP3 = fFermiFac * fNuclmodel->Momentum3();
  double tE = TMath::Sqrt(tP3.Mag2()+ tM*tM);
  t->SetMomentum(TLorentzVector(tP3,tE));
}
else
{
  t->SetMomentum(0,0,0,tM);
}

GHepParticle * cl = new GHepParticle(*p); // clone particle, to run IntBounce at proper energy
                                                                    // calculate energy and momentum using
                                                                        invariant mass
double pM  = p->Mass();
double E_p = ((*p->P4() + *t->P4()).Mag2() - tM*tM - pM*pM)/(2.0*tM);
double P_p = TMath::Sqrt(E_p*E_p - pM*pM);
cl->SetMomentum(TLorentzVector(P_p,0,0,E_p));
                          // momentum doesn't have to be in right direction, only magnitude
double C3CM = fHadroData->IntBounce(cl,tcode,scode,h_fate);
delete cl;
```

```
if (C3CM<-1.)   // hope this doesn't occur too often - unphysical but we just pass it on
{
  LOG("HAIntranuke", pWARN) << "unphysical angle chosen in InelasicHA - put particle outside nucleus";
  p->SetStatus(kIStStableFinalState);
  ev->AddParticle(*p);
  delete t;
  return;
}
double KE1L = p->KinE();
double KE2L = t->KinE();
LOG("HAIntranuke",pINFO)
  <<  "  KE1L = " << KE1L << "   " << KE1L << "  KE2L = " << KE2L;
GHepParticle * cl1 = new GHepParticle(*p);
GHepParticle * cl2 = new GHepParticle(*t);
bool success = utils::intranuke::TwoBodyCollision(ev,pcode,tcode,scode,s2code,C3CM,
                                        cl1,cl2,fRemnA,fRemnZ,fRemnP4,kIMdHA);
if(success)
{
  double P3L = TMath::Sqrt(cl1->Px()*cl1->Px() + cl1->Py()*cl1->Py() + cl1->Pz()*cl1->Pz());
  double P4L = TMath::Sqrt(cl2->Px()*cl2->Px() + cl2->Py()*cl2->Py() + cl2->Pz()*cl2->Pz());
  double E3L = cl1->KinE();
  double E4L = cl2->KinE();
  LOG ("HAIntranuke",pINFO) << "Successful quasielastic scattering or charge exchange";
  LOG("HAIntranuke",pINFO)
<< "C3CM = " << C3CM << "\n  P3L, E3L = "
<< P3L << "   " << E3L << "  P4L, E4L = "<< P4L << "   " << E4L ;
  if(ev->Probe() ) { LOG("HAIntranuke",pINFO)
  << "P4L = " << P4L << " ;E4L=  " << E4L << "\n probe KE = " << ev->Probe()->KinE() << "\n";
if (ev->Probe() && (E3L>ev->Probe()->E()E4L>ev->Probe()->E()))  //is this redundant?
  {
        exceptions::INukeException exception;
        exception.SetReason("TwoBodyCollison gives KE> probe KE in hA simulation");
        throw exception;
  }
ev->AddParticle(*cl1);
ev->AddParticle(*cl2);
delete cl1;
delete cl2;

LOG("HAIntranuke", pDEBUG) << "Nucleus : (A,Z) = ("<<fRemnA<<','<<fRemnZ<<')';
  }

} else
{
  exceptions::INukeException exception;
  exception.SetReason("TwoBodyCollison failed in hA simulation");
  throw exception;
}

delete t;
```

should be replaced by the following ones

```
GHepParticle t(*p);
t.SetPdgCode(tcode);

// set up fermi target
Target target(ev->TargetNucleus()->Pdg());
double tM = t.Mass();

// handle fermi momentum
if(fDoFermi)
{
  target.SetHitNucPdg(tcode);
  fNuclmodel->GenerateNucleon(target);
  TVector3 tP3 = fFermiFac * fNuclmodel->Momentum3();
  double tE = TMath::Sqrt(tP3.Mag2()+ tM*tM);
  t.SetMomentum(TLorentzVector(tP3,tE));
}
else
{
  t.SetMomentum(0,0,0,tM);
}

GHepParticle * cl = new GHepParticle(*p); // clone particle, to run IntBounce at proper energy
                                                                    // calculate energy and momentum using invariant
                                                                        mass
double pM  = p->Mass();
double E_p = ((*p->P4() + *t.P4()).Mag2() - tM*tM - pM*pM)/(2.0*tM);
double P_p = TMath::Sqrt(E_p*E_p - pM*pM);
```

```
cl->SetMomentum(TLorentzVector(P_p,0,0,E_p));
                          // momentum doesn't have to be in right direction, only magnitude
double C3CM = fHadroData->IntBounce(cl,tcode,scode,h_fate);
delete cl;
if (C3CM<-1.)   // hope this doesn't occur too often - unphysical but we just pass it on
{
  LOG("HAIntranuke", pWARN) << "unphysical angle chosen in InelasicHA - put particle outside nucleus";
  p->SetStatus(kIStStableFinalState);
  ev->AddParticle(*p);

  return;
}
double KE1L = p->KinE();
double KE2L = t.KinE();
LOG("HAIntranuke",pINFO)
  <<  "  KE1L = " << KE1L << "    " << KE1L << "  KE2L = " << KE2L;
GHepParticle cl1(*p);
GHepParticle cl2(t);
bool success = utils::intranuke::TwoBodyCollision(ev,pcode,tcode,scode,s2code,C3CM,
                                        &cl1,&cl2,fRemnA,fRemnZ,fRemnP4,kIMdHA);
if(success)
{
  double P3L = TMath::Sqrt(cl1.Px()*cl1.Px() + cl1.Py()*cl1.Py() + cl1.Pz()*cl1.Pz());
  double P4L = TMath::Sqrt(cl2.Px()*cl2.Px() + cl2.Py()*cl2.Py() + cl2.Pz()*cl2.Pz());
  double E3L = cl1.KinE();
  double E4L = cl2.KinE();
  LOG ("HAIntranuke",pINFO) << "Successful quasielastic scattering or charge exchange";
  LOG("HAIntranuke",pINFO)
<< "C3CM = " << C3CM << "\n  P3L, E3L = "
<< P3L << "    " << E3L << "  P4L, E4L = "<< P4L << "    " << E4L ;
  if(ev->Probe() ) { LOG("HAIntranuke",pINFO)
  << "P4L = " << P4L << " ;E4L=  " << E4L << "\n probe KE = " << ev->Probe()->KinE() << "\n";
if (ev->Probe() && (E3L>ev->Probe()->E()E4L>ev->Probe()->E()))  //is this redundant?
  {
        exceptions::INukeException exception;
        exception.SetReason("TwoBodyCollison gives KE> probe KE in hA simulation");
        throw exception;
  }
ev->AddParticle(cl1);
ev->AddParticle(cl2);



LOG("HAIntranuke", pDEBUG) << "Nucleus : (A,Z) = ("<<fRemnA<<','<<fRemnZ<<')';
  }

} else
{
  exceptions::INukeException exception;
  exception.SetReason("TwoBodyCollison failed in hA simulation");
  throw exception;
}
```

The next case is analogous to the previous one. To fix bugs in the method

```
void HAIntranuke::Inelastic(GHepRecord* ev, GHepParticle* p, INukeFateHA_t fate) const
```

defined in the file HAIntranuke.cxx the lines

```
GHepParticle* s1 = new GHepParticle(*p);
GHepParticle* s2 = new GHepParticle(*p);
GHepParticle* s3 = new GHepParticle(*p);

bool success = utils::intranuke::PionProduction(
 ev,p,s1,s2,s3,fRemnA,fRemnZ,fRemnP4, fDoFermi,fFermiFac,fFermiMomentum,fNuclmodel);

if (success){
LOG ("HAIntranuke",pINFO) << " successful pion production fate";
// set status of particles and conserve charge/baryon number
s1->SetStatus(kIStStableFinalState);  //should be redundant
//          if (pdg::IsPion(s2->Pdg())) s2->SetStatus(kIStHadronInTheNucleus);
s2->SetStatus(kIStStableFinalState);
//          if (pdg::IsPion(s3->Pdg())) s3->SetStatus(kIStHadronInTheNucleus);
s3->SetStatus(kIStStableFinalState);

ev->AddParticle(*s1);
ev->AddParticle(*s2);
ev->AddParticle(*s3);
```

```
delete s1;
delete s2;
delete s3;
```

should be replaced by the following ones

```
GHepParticle s1(*p);
GHepParticle s2(*p);
GHepParticle s3(*p);

bool success = utils::intranuke::PionProduction(
 ev,p,&s1,&s2,&s3,fRemnA,fRemnZ,fRemnP4, fDoFermi,fFermiFac,fFermiMomentum,fNuclmodel);

if (success){
LOG ("HAIntranuke",pINFO) << " successful pion production fate";
// set status of particles and conserve charge/baryon number
s1.SetStatus(kIStStableFinalState);  //should be redundant
//          if (pdg::IsPion(s2->Pdg())) s2->SetStatus(kIStHadronInTheNucleus);
s2.SetStatus(kIStStableFinalState);
//          if (pdg::IsPion(s3->Pdg())) s3->SetStatus(kIStHadronInTheNucleus);
s3.SetStatus(kIStStableFinalState);

ev->AddParticle(s1);
ev->AddParticle(s2);
ev->AddParticle(s3);
```

The memory leaks in above mentioned methods are critical, because these methods are called many times during even generation. Table 3.1 shows number of lost bytes in each of these methods during generation of 25000 events. The number of lost bytes were obtained by Valgrind — tools that can automatically detect many memory management and threading bugs.

Table 3.1: Number of lost bytes in some critical methods during generation of 25000 events.

Method	Number of lost bytes
`void MECGenerator::RecoilNucleonCluster(GHepRecord * event) const`	679488
`FGMBodekRitchie::~FGMBodekRitchie()`	34176
`void HAIntranuke::InelasticHA(GHepRecord* ev, GHepParticle* p, INukeFateHA_t fate) const`	727776
`void HAIntranuke::Inelastic(GHepRecord* ev, GHepParticle* p, INukeFateHA_t fate) const`	66528

3.13 Bug: Incorrect calculation of the empirical MEC-EM contribution.

Status: fixed in GENIE 2.12.4
Files: src/MEC/EmpiricalMECPXSec2015.cxx
Method: `double EmpiricalMECPXSec2015::Integral(const Interaction * interaction) const`

The cross section calculated by the following operator

```
double xsec = 0.5*(Z*fXSecAlgEMQE->Integral(inp) + N*fXSecAlgEMQE->Integral(inn));
```

systematically overestimates the desired MEC contribution for the scattering off an isoscalar nucleus, because the methods

`fXSecAlgEMQE->Integral(inp)` and `fXSecAlgEMQE->Integral(inn)`

actually return the cross sections for the scattering off a *nucleus* rather than these for the scattering off a *bound nucleon*. To correct this typo, the above line should be replaced by the following one:

```
double xsec = (Z*fXSecAlgEMQE->Integral(inp) + N*fXSecAlgEMQE->Integral(inn))/A;
```

3.14 Bug and improvement: "Double counting" of the Fermi momenta for proton and neutron. Suggesting simple parameterizations for the Fermi momenta and binding (separation) energies.

Status: fixed in GENIE 3.0.0

Files: src/Utils/NuclearUtils.h,
src/Utils/NuclearUtils.cxx,
src/Nuclear/FGMBodekRitchie.h,
src/Nuclear/FGMBodekRitchie.cxx,
config/UserPhysicsOptions.xml,
config/FGMBodekRitchie.xml

Let's remind that the values of Fermi momenta for proton (p_F^p) and neutron (p_F^n) bound in a nucleus can be obtained from the Fermi momentum for the isoscalar nucleon (p_F) by the well known relations

$$p_F^p = (2Z/A)^{1/3} p_F \quad \text{and} \quad p_F^n = (2N/A)^{1/3} p_F \qquad (N = A - Z). \tag{3.4}$$

(see, e.g., Refs. [4, 35]). These relations are based on the simplest assumption that the density of nuclear matter is approximately constant irrespective of the proton-to-neutron ratio Z/N. By using Eqs. (3.4) and the data presented in Ref. [35] one obtains the values listed in Table 3.2.

Table 3.2: Nuclear Fermi momenta for the 'isoscalar nucleon' (p_F), proton (p_F^p) and neutron (p_F^n)

.

Nucleus	p_F (MeV)	p_F^p (MeV)	p_F^n (MeV)
${}_3\mathrm{Li}^6$	169	169	169
${}_6\mathrm{C}^{12}$	221	221	221
${}_{12}\mathrm{Mg}^{25}$	235	235	235
${}_{20}\mathrm{Ca}^{40}$	251	251	251
${}_{28}\mathrm{Ni}^{58.7}$	260	257	263
${}_{39}\mathrm{Y}^{89}$	254	243	264
${}_{50}\mathrm{Sn}^{118.7}$	260	245	274
${}_{73}\mathrm{Ta}^{181}$	265	247	281
${}_{82}\mathrm{Pb}^{208}$	265	245	283

The GENIE code operates with the table FermiMomentumTables.xml which includes the pre-calculated values of p_F^p and p_F^n; these values exactly coincide with those from Table 3.2. The following code fragment in the file FGMBodekRitchie.cxx

```
//-- look up the Fermi momentum for this Target
  FermiMomentumTablePool * kftp = FermiMomentumTablePool::Instance();
  const FermiMomentumTable * kft  = kftp->GetTable(fKFTable);
  double KF = kft->FindClosestKF(target_pdgc, nucleon_pdgc);
  LOG("BodekRitchie", pNOTICE) << "KF = " << KF;
```

looks up the appropriate Fermi momenta for p or n in the table FermiMomentumTables.xml. Then, however, the obtained momentum is **again recalculated according to the same formulas** (3.4) in the following part of the code:

```
//-- correct the Fermi momentum for the struck nucleon
  assert(target.HitNucIsSet());
  bool is_p = pdg::IsProton(nucleon_pdgc);
  if(is_p) KF *= TMath::Power( 2*Z/A, 1./3.);
  else     KF *= TMath::Power( 2*N/A, 1./3.);
  LOG("BodekRitchie", pINFO) << "Corrected KF = " << KF;
```

This is an obvious bug because the table FermiMomentumTables.xml already contains the values of p_F^p and p_F^n! This bug produces a null effect in the case of isoscalar nuclei, while for other nuclei it becomes quite essential. To correct the bug, the code fragment above should be simply deleted.

There are two more (though less essential) drawbacks in the present GENIE scheme for calculation of the Fermi momenta and binding (separation) energies:

- If the requested binding (separation) energy cannot be found in the configuration file UserPhysicsOptions.xml then the average binding energy is computed from Wapstra's semi-empirical formula which is not quite applicable to the RFG binding energies.

- If the requested Fermi momentum cannot be found in the configuration file FermiMomentumTables.xml then the code uses the Fermi momentum for the nucleus closest by charge to the requested nucleus. This simplification is also too rough.

Both these small drawbacks can be easily avoided. Namely, let us suggest, *as an option*, our interpolation formulas for the A-dependence of the isoscalar nucleon Fermi momenta, p_F, and ξ-dependence of the binding (separation) energies, E_b:

$$\begin{aligned} p_F &= 270[1 - 4.2/A + (6.0/A)^2 - (5.3/A)^3], \\ E_b &= 50.4\left[1 - 2.26/\xi + (1.73/\xi)^2 - (1.21/\xi)^3\right], \quad \xi = Z/A^{1/3}. \end{aligned} \tag{3.5}$$

These formulas were obtained from the data on electron-nucleus scattering presented in Refs. [36, 37] (note that the data from Ref. [35] partially obsolete, after the updated analysis published in Ref. [36]). Figure 3.7 shows a comparison of the parameterizations (3.5) with the data from Refs. [35–37], the values of p_F and E_b used in GENIE, and with some inputs used by other authors.

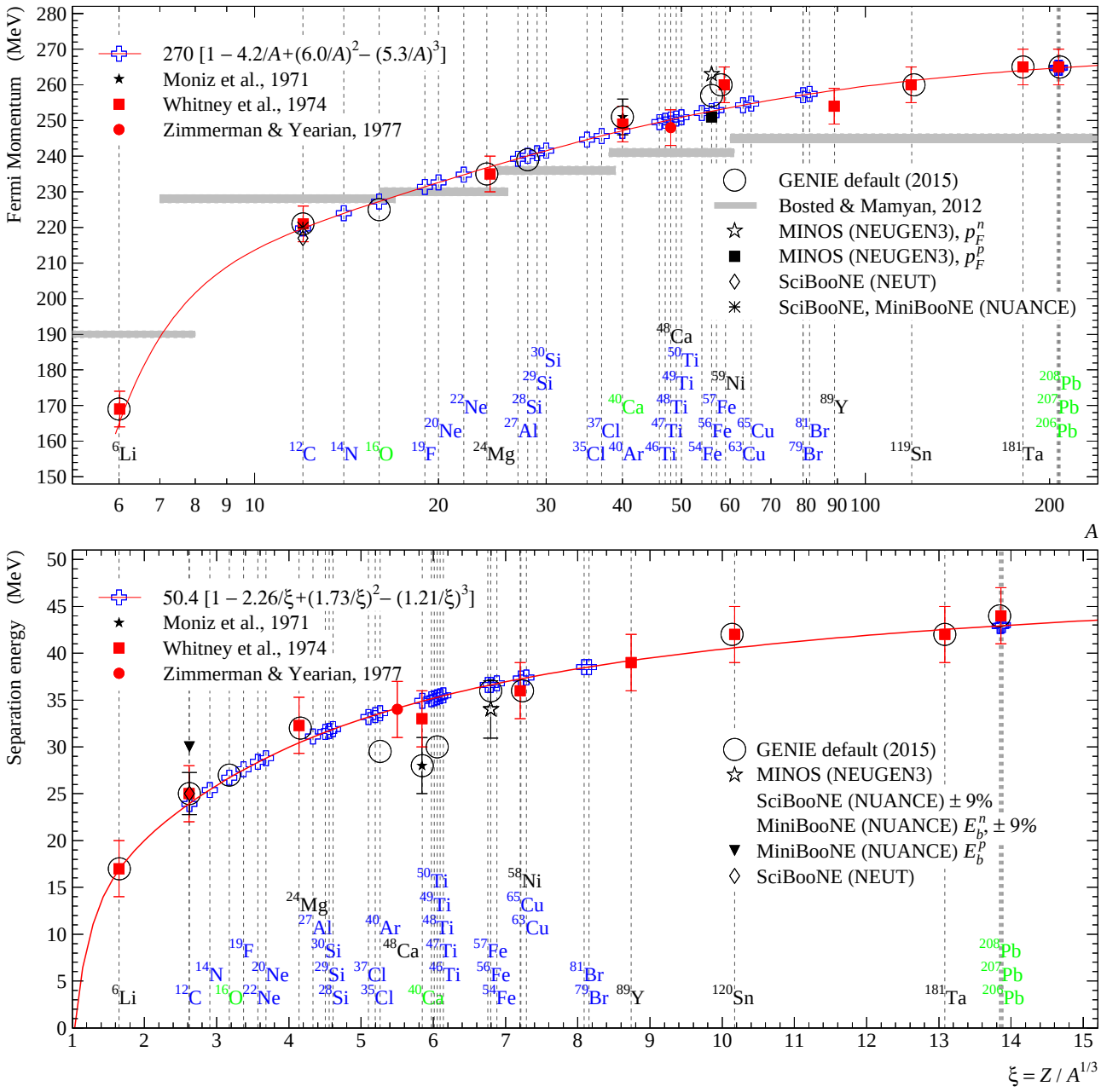

Figure 3.7: The data points shown by red solid symbols were used in the interpolation procedure. The blue empty crosses just indicate the nuclei involved into the detector targets used in the past or current neutrino experiments; the corresponding values were computed according to Eqs. (3.5). The other symbols are explained in the legends.

IMPORTANT NOTE: The parameterizations (3.5) do not work for $A < 6$. Therefore the user who (by unknown reasons) wishes to apply the RFG model to the very light nuclei (e.g., helium or deuterium) must use either the input values from FermiMomentumTables.xml or his/her own inputs.

To implement this scheme, the following private member should be added to the class `class FGMBodekRitchie` in the file FGMBodekRitchie.h

```
bool fUseFGParametrization;
```

the following line should be added to the method `void FGMBodekRitchie::LoadConfig(void)` in the file FGMBodekRitchie.cxx

```
fUseFGParametrization = fConfig->GetBoolDef ("UseFGParametrization",
                                    gc->GetBool("RFG-UseFGParametrization"));
```

the following lines

```
//-- set removal energy
//
int Z = target.Z();
map<int,double>::const_iterator it = fNucRmvE.find(Z);
if(it != fNucRmvE.end()) fCurrRemovalEnergy = it->second;
else fCurrRemovalEnergy = nuclear::BindEnergyPerNucleon(target);
```

of the method `bool FGMBodekRitchie::GenerateNucleon(const Target & target) const` in the file FGMBodekRitchie.cxx should be replaced by the following ones

```
//-- set removal energy
//
if (target.A()<6 || !fUseFGParametrization)
{
        int Z = target.Z();
        map<int,double>::const_iterator it = fNucRmvE.find(Z);
        if(it != fNucRmvE.end()) fCurrRemovalEnergy = it->second;
        else fCurrRemovalEnergy = nuclear::BindEnergyPerNucleon(target);
}
else
        fCurrRemovalEnergy = nuclear::BindEnergyPerNucleonParametrization(target);
```

the following lines

```
//-- look up the Fermi momentum for this Target
FermiMomentumTablePool * kftp = FermiMomentumTablePool::Instance();
const FermiMomentumTable * kft  = kftp->GetTable(fKFTable);
double KF = kft->FindClosestKF(target_pdgc, nucleon_pdgc);
LOG("BodekRitchie", pNOTICE) << "KF = " << KF;

//-- correct the Fermi momentum for the struck nucleon
assert(target.HitNucIsSet());
bool is_p = pdg::IsProton(nucleon_pdgc);
if(is_p) KF *= TMath::Power( 2*Z/A, 1./3.);
else     KF *= TMath::Power( 2*N/A, 1./3.);
LOG("BodekRitchie", pINFO) << "Corrected KF = " << KF;
```

of the method `TH1D * FGMBodekRitchie::ProbDistro(const Target & target) const` in the file FGMBodekRitchie.cxx should be replaced by the following ones

```
double KF;
if (A<6 || !fUseFGParametrization)
{
        //-- look up the Fermi momentum for this Target
        FermiMomentumTablePool * kftp = FermiMomentumTablePool::Instance();
        const FermiMomentumTable * kft  = kftp->GetTable(fKFTable);
        KF = kft->FindClosestKF(target_pdgc, nucleon_pdgc);
        LOG("BodekRitchie", pNOTICE) << "KF = " << KF;
}
else
{
        //-- define the Fermi momentum for this Target
        //
        KF = nuclear::FermiMomentumForIsoscalarNucleonParametrization(target);
        //-- correct the Fermi momentum for the struck nucleon
        assert(target.HitNucIsSet());
        bool is_p = pdg::IsProton(nucleon_pdgc);
        if(is_p) KF *= TMath::Power( 2*Z/A, 1./3.);
        else     KF *= TMath::Power( 2*N/A, 1./3.);
        LOG("BodekRitchie", pINFO) << "Corrected KF = " << KF;
}
```

the following lines

```
//____________________________________________________________________________
double genie::utils::nuclear::BindEnergyPerNucleonParametrization(const Target & target);
double genie::utils::nuclear::FermiMomentumForIsoscalarNucleonParametrization(const Target & target);
```

should be added in the file NuclearUtils.h and the following lines

```
//____________________________________________________________________________
double genie::utils::nuclear::BindEnergyPerNucleonParametrization(const Target & target)
{
// Compute the average binding energy per nucleon (in GeV)

  if(!target.IsNucleus()) return 0;
  double x = TMath::Power(target.A(),1/3.0) / target.Z();

 return (0.05042772591+x*(-0.11377355795+x*(0.15159890400-0.08825307197*x)));
}
//____________________________________________________________________________
double genie::utils::nuclear::FermiMomentumForIsoscalarNucleonParametrization(const Target & target)
{
// Compute Fermi momentum for isoscalar nucleon (in GeV)

  if(!target.IsNucleus()) return 0;
  double x = 1.0 / target.A();

  return (0.27+x*(-1.12887857491+x*(9.72670908033-39.53095724456*x)));
}
```

should be added to the file NuclearUtils.cxx.
The following parameter should be added

```
<param type="bool"   name="RFG-UseFGParametrization">    false  </param>
```

in the configuration file UserPhysicsOptions.xml and the following lines should be inserted in the comment section of configuration file FGMBodekRitchie.xml

```
UseFGParametrization bool     Yes       use parametrization for Fermi momentum and     GPL RFG-UseFGParametrization
                                        binging energy
```

3.15 Bug: incorrect calculation of the suppression factor in the Bodek-Ritchie RFG model.

Status: fixed in GENIE 3.0.0
Files: src/Nuclear/FGMBodekRitchie.cxx, src/Utils/NuclearUtils.cxx

The suppression factor $R(Q^2)$ for the RFG model by Bodek&Ritchie [15] is calculated in the method `TH1D * FGMBodekRitchie::ProbDistro(const Target & target) const` in the file FGMBodekRitchie.cxx:

```
double R  = 1. / (1.- KF/4.);
```

and in the function `double genie::utils::nuclear::RQEFG_generic (double q2, double Mn, double kFi, double kFf, double pmax)` in the file NuclearUtils.cxx:

```
double rkf   = 1./(1. - kFi/4.);
```

The problem is that the distribution (6) in Ref. [15] is normalized at a constant momentum upper bound ($p_{\max}$ from hereafter) which is originally set to 4 GeV in Ref. [15]. However, $p_{\max}$ becomes a variable quantity in the GENIE code and is set to 1 GeV by default. This is a mistake since the value of $p_{\max}$ must be consistent with the sewing and normalization conditions. To remove this mistake, the line in the method `TH1D * FGMBodekRitchie::ProbDistro(const Target & target) const` in the file FGMBodekRitchie.cxx:

```
double R  = 1. / (1.- KF/4.);
```

must be replaced by the follwoing ones:

```
double R  = 1. / (1.- KF/fPmax);
```

But it is not quite clear, how to account this fact and change in proper way the code in the function `double genie::utils::nuclear::RQEFG_generic (double q2, double Mn, double kFi, double kFf, double pmax)` in the file NuclearUtils.cxx:

```
double rkf   = 1./(1. - kFi/4.);
```

We have no good idea about origin of the current default value $p_{\max} = \texttt{fPmax} = 1$ GeV. If it is "random" choice, we suggest the simplest solution: to set it back to the original Bodek-Ritchie value $p_{\max} = \texttt{fPmax} = 4$ GeV. Any case, it is necessary to keep in mind this nuance since ignoring it may lead to an uncontrolled loss of accuracy.

3.16 Bug: Incorrect reading of the configuration file.

Status: fixed in GENIE 3.0.0
Files: src/LlewellynSmith/KuzminNaumov2016AxialFormFactorModel.cxx

The lines in the method `void KuzminNaumov2016AxialFormFactorModel::LoadConfig(void)` in the file KuzminNaumov2016AxialFormFactorModel.cxx:

```
fMa  = fConfig->GetDoubleDef("QEL-Ma", gc->GetDouble("QEL-Ma"));
fFA0 = fConfig->GetDoubleDef("QEL-FA0", gc->GetDouble("QEL-FA0"));
```

should be replaced by the following ones:

```
fMa  = fConfig->GetDoubleDef("Ma", gc->GetDouble("QEL-Ma"));
fFA0 = fConfig->GetDoubleDef("FA0", gc->GetDouble("QEL-FA0"));
```

3.17 Bug: Incorrect working with cache for KineGenerator.

Status: fixed in GENIE 3.0.0
Files: src/EVGModules/KineGeneratorWithCache.cxx

The spline is not recreated if cached energy is greater than the maximum abscissa of the spline, because of the bug in the method `void KineGeneratorWithCache::CacheMaxXSec(const Interaction * interaction, double max_xsec) const` in the file KineGeneratorWithCache.cxx:

```
if( E < cb->Spl()->XMin() && E < cb->Spl()->XMax() ) {
        cb->CreateSpline();
}
```

To fix the bug we suggest to replace these lines by the following ones:

```
if( E < cb->Spl()->XMin() || E > cb->Spl()->XMax() ) {
        cb->CreateSpline();
}
```

Further, if the energy is too close to the ends of the spline (the distance between the energy and the end of the spline is less than 0.2 GeV) then the spline is not used for evaluation of cached maximum at this energy, but it is calculated again and stored in the C++ associative containers—map that leads to an essential increase of the memory used each time when KineGenerator meets such energy and to spending the CPU time. This behavior is because of the bug in the method `double KineGeneratorWithCache::FindMaxXSec(const Interaction * interaction) const` in the file KineGeneratorWithCache.cxx:

```
if( E > cb->Spl()->XMin()+0.2 && E < cb->Spl()->XMax()-0.2) {
   double spl_max_xsec = cb->Spl()->Evaluate(E);
   LOG("Kinematics", pINFO)
          << "\nInterpolated: max xsec (E=" << E << ") = " << spl_max_xsec;
   return spl_max_xsec;
}
```

There are several solutions to this problem.[2] To the moment, let us use the simplest one, namely, we suggest to replace the above lines by the following ones:

```
if( E >= cb->Spl()->XMin() && E <= cb->Spl()->XMax()) {
   double spl_max_xsec = cb->Spl()->Evaluate(E);
   LOG("Kinematics", pINFO)
          << "\nInterpolated: max xsec (E=" << E << ") = " << spl_max_xsec;
   return spl_max_xsec;
 }
This solution has been tested in several examples and it does work satisfactory, saving the memory and CPU time.
However, we will try to find a more general solution in future update.
```

[2] The most radical one is to use another spline, more appropriate for this case.

3.18 Improvement: excluding the obsolete parameters for the integrator used in the Rein-Sehgal based models for resonance production. Correcting the input parameters for the 1D integrator.

Status: fixed in GENIE 3.0.0

Files: src/Base/XSecIntegratorI.h,
src/CrossSections/QELXSec.cxx,
src/ReinSehgal/ReinSehgalRESXSecWithCache.cxx,
src/ReinSehgal/ReinSehgalRESXSec.cxx,
src/ReinSehgal/ReinSehgalSPPXSec.cxx,
config/ReinSehgalRESXSec.xml,
config/ReinSehgalSPPXSec.xml,
config/QELXSec.xml

The configuration files for the integrators ReinSehgalRESXSec.xml and ReinSehgalSPPXSec.xml are incorrect, because they contain unused parameters and some parameters in these files become obsolete after the new fast adaptive integrator has been implemented as independent class (Section 1.3). The corrected configuration files are presented bellow.

The configuration file ReinSehgalRESXSec.xml is[3]

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<alg_conf>

<!--
Configuration for the Rein-Seghal RES excitation xsec algorithm.
This algorithm offers a fast alternative to RESXSec algorithm as it precomputes and
caches free nucleon resonance neutrinoproduction cross sections.

Algorithm Configurable Parameters:
.....................................................................................................
Name                        Type    Opt  Comment                                                 Default
.....................................................................................................
gsl-integration-type        string  Yes  name of GSL numerical integrator                       adaptive
                                                                                  (adaptive, vegas, miser, plain)
gsl-max-eval                int     Yes  GSL number of function calls                          100000
gsl-relative-tolerance      double  Yes  relative tolerance of integration                    0.01
resonance-name-list         string  Yes  list of resonances to be taken into account          GPL: ResonanceNameList
ESplineMax                  double  Yes  Emax in RES splines, xsec(E>Emax)=xsec(E=Emax)                 100

-->
 Misprints in the nucleon and baryon resonance parameters
<!--
  In this configuration set we include all the resonances defined in GlobalParameterList
-->
  <param_set name="Default">
    <param type="double" name ="ESplineMax">                  100  </param>
    <param type="string" name ="gsl-integration-type">       vegas  </param>
    <param type="int"    name ="gsl-max-eval" >            5000000  </param>
    <param type="double" name ="gsl-relative-tolerance">    0.0001  </param>
  </param_set>

<!--
  In this configuration set we include all resonances known to GENIE including the the ambiguous P33(1600), F17(1970)
-->
  <param_set name="AllResonances">
    <param type="string" name="ResonanceNameList">
                  P33(1232),S11(1535),D13(1520),S11(1650),D13(1700),D15(1675),S31(1620),
                  D33(1700),P11(1440),P33(1600),P13(1720),F15(1680),P31(1910),P33(1920),
                  F35(1905),F37(1950),P11(1710),F17(1970)
    </param>
    <param type="double" name ="ESplineMax">                  100  </param>
    <param type="string" name ="gsl-integration-type">       vegas  </param>
    <param type="int"    name ="gsl-max-eval" >            5000000  </param>
    <param type="double" name ="gsl-relative-tolerance"  >  0.0001  </param>
  </param_set>
```

[3] Note that the resonances P33(1600) and F17(1970) mentioned in this file as "ambiguous" are in fact quite robust after correctiong the misprints as is described in Section 3.24.

```
</alg_conf>
```

The configuration file ReinSehgalSPPXSec.xml is

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<alg_conf>

<!--
Configuration for the Rein-Sehgal exclusive SPP (1pi) xsec algorithm.
This algorithm offers a fast alternative to RESXSec algorithm as it precomputes and
caches free nucleon resonance neutrinoproduction cross sections.

Algorithm Configurable Parameters:
.....................................................................................................
Name                        Type    Opt  Comment                                          Default
.....................................................................................................
SingleRESDiffXSecAlg        alg     No   algorithm used for computing res excitation xsec
ResonanceNameList           string  Yes  list of resonances to be taken into account        GPL: ResonanceNameList
UseDRJoinScheme             bool    Yes  Use DIS/RES joining scheme?                      GPL: UseDRJoinScheme
Wcut                        double  Yes  Param used in DIS/RES joining                    GPL: Wcut
gsl-integration-type        string  Yes  name of GSL numerical integrator                adaptive
                                                                  (adaptive, vegas, miser, plain)
gsl-max-eval                int     Yes  GSL number of function calls                    100000
gsl-relative-tolerance      double  Yes  relative tolerance of integration              0.01
ESplineMax                  double  No   Emax in RES splines, xsec(E>Emax)=xsec(E=Emax)
-->

<!--
  In this configuration set we include only the first two resonances
-->
  <param_set name="Fast">
     <param type="alg"    name="SingleRESDiffXSecAlg">   genie::ReinSehgalRESPXSec/Default                    </param>
     <!-- <param type="alg"    name="SingleRESDiffXSecAlg">   genie::BergerSehgalRESPXSec2014/Default              </param>
         -->
     <param type="double" name="ESplineMax">             20                                </param>
     <param type="string" name ="gsl-integration-type">       vegas  </param>
     <param type="int"    name ="gsl-max-eval" >            5000000  </param>
     <param type="double" name ="gsl-relative-tolerance"  >  0.0001  </param>
     <param type="string" name="ResonanceNameList">      P33(1232),P11(1440)             </param>
  </param_set>


<!--
  In this configuration set we include all the resonances defined in GlobalParameterList
-->
  <param_set name="Default">
     <param type="alg"    name="SingleRESDiffXSecAlg">   genie::ReinSehgalRESPXSec/Default                    </param>
     <!-- <param type="alg"    name="SingleRESDiffXSecAlg">   genie::BergerSehgalRESPXSec2014/Default              </param>
         -->
     <param type="string" name ="gsl-integration-type">       vegas  </param>
     <param type="int"    name ="gsl-max-eval" >            5000000  </param>
     <param type="double" name ="gsl-relative-tolerance"  >  0.0001  </param>
     <param type="double" name="ESplineMax">             20                              </param>
  </param_set>

<!--
  In this configuration set we include all resonances known to GENIE including the the ambiguous P33(1600), F17(1970)
-->
  <param_set name="AllResonances">
     <param type="alg"    name="SingleRESDiffXSecAlg">   genie::ReinSehgalRESPXSec/Default                    </param>
     <!-- <param type="alg"    name="SingleRESDiffXSecAlg">   genie::BergerSehgalRESPXSec2014/Default              </param>
         -->
     <param type="string" name ="gsl-integration-type">       vegas  </param>
     <param type="int"    name ="gsl-max-eval" >            5000000  </param>
     <param type="double" name ="gsl-relative-tolerance"  >  0.0001  </param>
     <param type="string" name="ResonanceNameList">
                  P33(1232),S11(1535),D13(1520),S11(1650),D13(1700),D15(1675),S31(1620),
                  D33(1700),P11(1440),P33(1600),P13(1720),F15(1680),P31(1910),P33(1920),
                  F35(1905),F37(1950),P11(1710),F17(1970)
      <param type="double" name="ESplineMax">             20                                </param>
     </param>
  </param_set>

</alg_conf>
```

Some unused parameters can be safely deleted from the integrator code; these are: *UseDRJoinScheme, Wcut, Kine-Wmin, Kine-Wmax, Kine-Q2min*, and *Kine-Q2max*.

So there is a need to make some changes in the code of these integrators to account for the above-mentioned changes. Replace the lines in the file ReinSehgalRESXSec.cxx in the method

```
double ReinSehgalRESXSec::Integrate(const XSecAlgorithmI * model, const Interaction * interaction) const :
```

```
ROOT::Math::IntegratorMultiDim ig(ig_type);
ig.SetRelTolerance(fGSLRelTol);
```

by the following lines:

```
ROOT::Math::IntegratorMultiDim ig(ig_type,0,fGSLRelTol,fGSLMaxEval);
```

Replace the lines in the method `void ReinSehgalRESXSec::LoadConfig(void)` in the file ReinSehgalRESXSec.cxx:

```
fGSLNCalls    = fConfig->GetIntDef("gsl-ncalls", 100000);
fGSLThreshold= fConfig->GetDoubleDef("gsl-threshold", 50);
fGSLNCallsFactor= fConfig->GetDoubleDef("gsl-ncalls-factor", 1);
```

by the following lines:

```
fGSLMaxEval  = fConfig->GetIntDef("gsl-max-eval", 100000);
```

Replace the lines in the method `void ReinSehgalSPPXSec::LoadConfig(void)` in the file ReinSehgalSPPXSec.cxx:

```
fGSLNCalls    = fConfig->GetIntDef("gsl-ncalls", 100000);
fGSLThreshold= fConfig->GetDoubleDef("gsl-threshold", 50);
fGSLNCallsFactor= fConfig->GetDoubleDef("gsl-ncalls-factor", 1);
```

by the following lines:

```
fGSLMaxEval  = fConfig->GetIntDef("gsl-max-eval", 100000);
```

GENIE call 1D integrator in ROOT with incorrect parameters. To fix this one need to make the following changes: Replace the lines in the class `XSecIntegratorI` (file XSecIntegratorI.h):

```
int    fGSLNCalls;   ///< GSL number of function calls (apply only to MC integratioon methods)
double fGSLThreshold;///< the threshold for the neutrino energy, above which the initial number of function multiply by
    fGSLNCallsFactor
double fGSLNCallsFactor;///< factor that is multiplied by the initial number of function calls when the threshold is reached
```

by the following ones:

```
unsigned int fGSLMaxSizeOfSubintervals;  ///< GSL maximum number of sub-intervals for 1D integrator
unsigned int fGSLRule;                   ///< GSL Gauss-Kronrod integration rule (only for GSL 1D adaptive type)
```

Replace the lines in the method `double QELXSec::Integrate(const XSecAlgorithmI * model, const Interaction * in) const` in the file QELXSec.cxx:

```
double abstol = 1; //We mostly care about relative tolerance
ROOT::Math::Integrator ig(*func,ig_type,abstol,fGSLRelTol,fGSLMaxEval);
```

by the following ones:

```
double abstol = 0; //We mostly care about relative tolerance
ROOT::Math::Integrator ig(*func,ig_type,abstol,fGSLRelTol,fGSLMaxSizeOfSubintervals, fGSLRule);
```

Replace the lines in the method `void QELXSec::LoadConfig(void)` in the file QELXSec.cxx:

```
fGSLRelTol    = fConfig->GetDoubleDef("gsl-relative-tolerance", 0.01);
fGSLMaxEval   = (unsigned int) fConfig->GetIntDef ("gsl-max-eval", 100000);
```

by the following ones:

```
fGSLRelTol    = fConfig->GetDoubleDef("gsl-relative-tolerance", 0.001);
fGSLMaxSizeOfSubintervals  = (unsigned int) fConfig->GetIntDef ("gsl-max-size-of-subintervals", 40000);
fGSLRule      = (unsigned int) fConfig->GetIntDef ("gsl-rule", 3);
if (fGSLRule>6) fGSLRule=3;
```

And finally, replace the configuration file QELXSec.xml by the following one:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<alg_conf>

<!--
Configuration for the QELXSec xsec algorithm.

Configurable Parameters:
.....................................................................................
Name                          Type     Optional   Comment                                  Default
gsl-integration-type          string   Yes        name of GSL 1D numerical integrator      adaptive
gsl-max-size-of-subintervals  int      Yes        GSL maximum number of sub-intervals      40000
```

```
                                        for 1D integrator
gsl-relative-tolerance       double   Yes       GSL max evaluations for 1D integrator       0.001
gsl-rule                     int      Yes       GSL Gauss-Kronrod integration rule          3
                                               (only for GSL 1D adaptive type)
.....................................................................................................
-->

  <param_set name="Default">
    <param type="string" name = "gsl-integration-type">          adaptive  </param>
    <param type="int"    name = "gsl-max-size-of-subintervals">     40000  </param>
    <param type="double" name = "gsl-relative-tolerance">           0.001  </param>
    <param type="int"    name = "gsl-rule">                             3  </param>
  </param_set>

</alg_conf>
```

3.19 Bug and improvement: Absence of the the CKM factor in the Rein-Sehgal model and its extensions.

Status: fixed in GENIE 3.0.0

Files: src/ReinSehgal/ReinSehgalRESPXSec.h,
src/ReinSehgal/ReinSehgalRESPXSec.cxx,
src/ReinSehgal/BSKLNBaseRESPXSec2014.h,
src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx

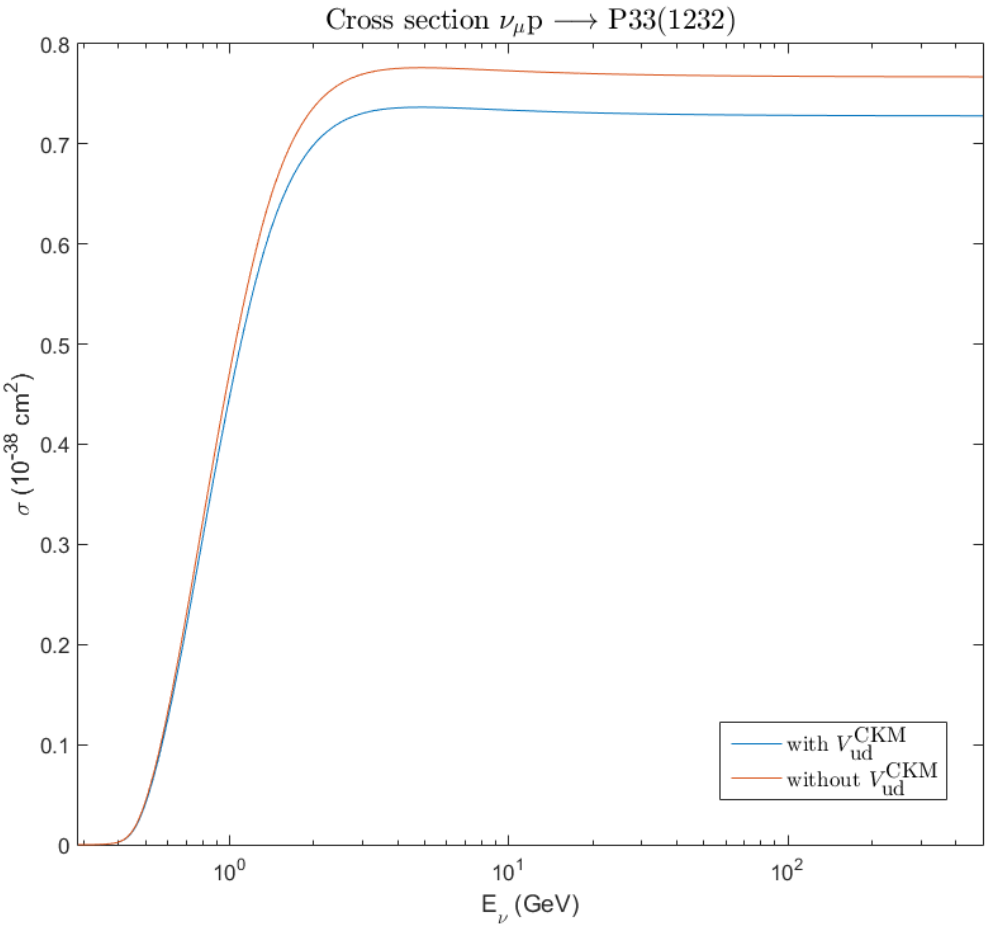

Figure 3.8: Cross section of ν_μ vs. energy.

The ud element of the CKM matrix, $V_{ud}^{\rm CKM} \approx \cos\theta_{\rm C}$, is not accounted for in the Rein-Sehgal model and its extensions (in other words, $V_{ud}^{\rm CKM}$ is put to be 1, as it was originally assumed for simplicity in the RS paper), but actually the factor $|V_{ud}^{\rm CKM}|^2$ in the resonance neutrinoproduction cross sections is very essential (see, e.g., Fig. 3.8), and **must** be accounted for. For this purpose, the member

```
double   fVud2;              ///< Vud^2(square of magnitude ud-element of CKM-matrix)
```

should be added to the classes

```
class ReinSehgalRESPXSec and
class BSKLNBaseRESPXSec2014
```

in the files
ReinSehgalRESPXSec.h and BSKLNBaseRESPXSec2014.h.
The lines

```
double Vud = fConfig->GetDoubleDef("CKM-Vud", gc->GetDouble("CKM-Vud"));
fVud2 = TMath::Power(Vud, 2);
```

should be added to the methods

```
void ReinSehgalRESPXSec::LoadConfig(void) and
void BSKLNBaseRESPXSec2014::LoadConfig(void)
```

After the line

```
double g2 = kGF2;
```

should be added the following line

```
if(is_CC) g2 = kGF2*fVud2;
```

in the methods

```
double ReinSehgalRESPXSec::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const and
double BSKLNBaseRESPXSec2014::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const
```

in the files ReinSehgalRESPXSec.cxx and BSKLNBaseRESPXSec2014.cxx

In the configuration files ReinSehgalRESPXSec.xml, KuzminLyubushkinNaumovRESPXSec2014.xml and BergerSehgalRESPXSec2014.xml the following lines should be added into the comment section:

```
<!--
......
CKM-Vud  double  Yes   Magnitude of ud-element of CKM-matrix  GPL: CKM-Vud
......
-->
```

3.20 Bug and improvement: a new parameter for switch-on/off of the Breit-Wigner distribution normalization in the Rein-Sehgal based models.

Status: fixed in GENIE 3.0.0
Files: src/ReinSehgal/ReinSehgalRESPXSec.h,
src/ReinSehgal/ReinSehgalRESPXSec.cxx,
src/ReinSehgal/BSKLNBaseRESPXSec2014.h,
src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx

The normalization factor $\eta_{BW}^{(\nu)}(W)$ in Eq. (2.33) of Ref. [22] should be calculated as

$$N_\nu = \int_{W_{min}}^{\infty} dW \, \frac{\Gamma_\nu}{2\pi} \, \frac{1}{(W - M_\nu)^2 + \Gamma_\nu^2/4}, \tag{3.6}$$

where

$$\Gamma_\nu = \Gamma_\nu^0 \left[\frac{q_\pi(W)}{q_\pi(M_\nu)} \right]^{2L+1} \quad \text{and} \quad q_\pi(W) = \frac{1}{2W} \sqrt{\left(W^2 - m_N^2 - m_\pi^2\right)^2 - 4m_N^2 m_\pi^2}.$$

To calculate the integral (3.6) numerically, it is needed to use a cut off that is some definite upper limit in W. The following code is responsible for calculation of the factor N_ν in GENIE 2.11.x:

```
double NR  = utils::res::BWNorm (resonance,fNOResMaxNWidths,fN2ResMaxNWidths,fGnResMaxNWidths);
......
if (W > MR + fNOResMaxNWidths * WR && IR==0) return 0.;
else if (W > MR + fN2ResMaxNWidths * WR && IR==2) return 0.;
else if (W > MR + fGnResMaxNWidths * WR) return 0.;
```

However there is a problem here: on the one hand, there is no any physical reason for using the currently adopted "recipe" and, on the other hand, the result of integration (3.6) strongly depends on the cutoff, especially for the S-wave resonances. Any case, the user should keep this ambiguity in mind. The simplest possibility to avoid this ambiguity is simply put $N_\nu = 1$ (the most preferable solution in our opinion), but the code must provide the opportunity for such customer's choice. In order to implement the mentioned improvement, the following member

```
bool     fNormBW;            /// < normalize resonance breit-wigner to 1?
```

should be added to the classes

```
class ReinSehgalRESPXSec and
class BSKLNBaseRESPXSec2014
```

in the files ReinSehgalRESPXSec.h and BSKLNBaseRESPXSec2014.h. Next, the following line

```
fNormBW = fConfig->GetBoolDef("BreitWignerNorm", true);
```

should be added to the methods

```
void ReinSehgalRESPXSec::LoadConfig(void) and
void BSKLNBaseRESPXSec2014::LoadConfig(void)
```

Next, instead of the lines

```
double NR  = utils::res::BWNorm (resonance,fNOResMaxNWidths,fN2ResMaxNWidths,fGnResMaxNWidths);
......
if (W > MR + fNOResMaxNWidths * WR && IR==0) return 0.;
else if (W > MR + fN2ResMaxNWidths * WR && IR==2) return 0.;
else if (W > MR + fGnResMaxNWidths * WR) return 0.;
```

in the methods

```
double ReinSehgalRESPXSec::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const and
double BSKLNBaseRESPXSec2014::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const
```

in the files ReinSehgalRESPXSec.cxx and BSKLNBaseRESPXSec2014.cxx, the lines

```
double NR  = fNormBW?utils::res::BWNorm (resonance,fNOResMaxNWidths,fN2ResMaxNWidths,fGnResMaxNWidths):1;
......
if (fNormBW) {
   if (W > MR + fNOResMaxNWidths * WR && IR==0) return 0.;
   else if (W > MR + fN2ResMaxNWidths * WR && IR==2) return 0.;
   else if (W > MR + fGnResMaxNWidths * WR) return 0.;
}
```

should be included.
In the configuration files ReinSehgalRESPXSec.xml, KuzminLyubushkinNaumovRESPXSec2014.xml and BergerSehgalRESPXSec2014.xml the lines

```
<!--
......
BreitWignerNorm   bool   Yes   Normalize breit-wigner?   true
......
-->
```

should be added into the comment section.

3.21 Improvement: Excluding the relic NeuGEN ν_τ-reduction factor.

Status: fixed in GENIE 3.0.0
Files: src/ReinSehgal/BSKLNBaseRESPXSec2014.h,
src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx

Considering that the BSKLN resonance model automatically takes into account the final lepton mass, the NeuGEN inspired reduction factor for the τ production cross section should be rejected from this model. So we suggest to exclude the corresponding parts of the code, namely:

```
// Apply NeuGEN nutau cross section reduction factors
double rf = 1.0;
Spline * spl = 0;
if (is_CC && fUsingNuTauScaling) {
  if (pdg::IsNuTau(probepdgc)) {
    spl = fNuTauRdSpl;
  }
  else if (pdg::IsAntiNuTau(probepdgc)) {
    spl = fNuTauBarRdSpl;
  }
  if(spl) {
    if(E <spl->XMax()) rf = spl->Evaluate(E);
  }
}
xsec *= rf;

.....

// NeuGEN reduction factors for nu_tau: a gross estimate of the effect of
// neglected form factors in the R/S model
fUsingNuTauScaling = fConfig->GetBoolDef("UseNuTauScalingFactors", true);
if(fUsingNuTauScaling) {
if(fNuTauRdSpl)    delete fNuTauRdSpl;
if(fNuTauBarRdSpl) delete fNuTauBarRdSpl;

assert(gSystem->Getenv("GENIE"));
string base = gSystem->Getenv("GENIE");

string filename = base + "/data/evgen/rein_sehgal/res/nutau_xsec_scaling_factors.dat";
LOG("BSKLNBaseRESPXSec2014", pINFO)
  << "Loading nu_tau xsec reduction spline from: " << filename;
fNuTauRdSpl = new Spline(filename);

filename = base + "/data/evgen/rein_sehgal/res/nutaubar_xsec_scaling_factors.dat";
LOG("BSKLNBaseRESPXSec2014", pINFO)
  << "Loading bar{nu_tau} xsec reduction spline from: " << filename;
fNuTauBarRdSpl = new Spline(filename);
}
```

3.22 Improvement: More detailed configuration files for BSKNL single pion production resonance model.

Status: fixed in GENIE 3.0.0
Files: config/ReinSehgalRESPXSec.xml,
config/BergerSehgalRESPXSec2014.xml,
config/KuzminLyubushkinNaumovRESPXSec2014.xml

The file ReinSehgalRESPXSec.xml should be as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<alg_conf>

<!--
Configuration for the Rein-Sehgal RES excitation d^xsec / dQ^2 dW xsec algorithm

Configurable Parameters:
.....................................................................................................................
Name                        Type    Optional   Comment                                                  Default
.....................................................................................................................
Zeta                        double  Yes   FKR's Z parameter                                           GPL: RS-Zeta
Omega                       double  Yes   FKR's Omega parameter                                       GPL: RS-Omega
Ma                          double  Yes   RES axial mass                                              GPL: RES-Ma
Mv                          double  Yes   RES vector mass                                             GPL: RES-Mv
BreitWignerWeight           bool    Yes   Weight xsec with breit-wigner?                             true
BreitWignerNorm             bool    Yes   Normalize breit-wigner?                                    true
weinberg-angle              double  Yes   Weinberg angle                                              GPL: WeinbergAngle
CKM-Vud                     double  Yes   Magnitude of ud-element of CKM-matrix                       GPL: CKM-Vud
UseNuTauScalingFactors      bool    Yes   Load/Use NEUGEN reduction factor splines for nutaus        true
UseDRJoinScheme             bool    Yes   Use DIS/RES joining scheme?                                GPL: UseDRJoinScheme
Wcut                        double  Yes   Param used in DIS/RES joining                               GPL: Wcut
MaxNWidthForN2Res           double  Yes   x in limiting allowed W phase space for n=2 res according to  2.0
                                          W < min{ Wmin(physical), MassRes + x * WidthRes }
MaxNWidthForN0Res           double  Yes   As above for n=0 resonances                                 6.0
MaxNWidthForGNRes           double  Yes   As above for the remaining resonances                       4.0
XSec-Integrator             alg
-->

  <param_set name="Default">
     <param type="alg"  name="XSec-Integrator">  genie::ReinSehgalRESXSecFast/Default    </param>
  </param_set>


</alg_conf>
```

The files BergerSehgalRESPXSec2014.xml and KuzminLyubushkinNaumovRESPXSec2014.xml should be as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<alg_conf>

<!--
Configuration for the Berger-Sehgal(Kuzmin-Lyubushkin-Naumov) RES excitation d^xsec / dQ^2 dW xsec algorithm

Configurable Parameters:
.....................................................................................................................
Name                        Type    Optional   Comment                                                  Default
.....................................................................................................................
Zeta                        double  Yes   FKR's Z parameter                                           GPL: RS-Zeta
Omega                       double  Yes   FKR's Omega parameter                                       GPL: RS-Omega
minibooneGA                 bool    Yes   tuning to axial form factors by MiniBoone (J. Nowak)       GPL: minibooneGA
minibooneGV                 bool    Yes   tuning to vector form factors by MiniBoone (J. Nowak)      GPL: minibooneGV
Ma                          double  Yes   RES axial mass                                              GPL: RES-Ma
Mv                          double  Yes   RES vector mass                                             GPL: RES-Mv
BreitWignerWeight           bool    Yes   Weight xsec with breit-wigner?                             true
BreitWignerNorm             bool    Yes   Normalize breit-wigner?                                    true
weinberg-angle              double  Yes   Weinberg angle                                              GPL: WeinbergAngle
CKM-Vud                     double  Yes   Magnitude of ud-element of CKM-matrix                       GPL: CKM-Vud
UseDRJoinScheme             bool    Yes   Use DIS/RES joining scheme?                                GPL: UseDRJoinScheme
Wcut                        double  Yes   Param used in DIS/RES joining                               GPL: Wcut
MaxNWidthForN2Res           double  Yes   x in limiting allowed W phase space for n=2 res according to  2.0
                                          W < min{ Wmin(physical), MassRes + x * WidthRes }
MaxNWidthForN0Res           double  Yes   As above for n=0 resonances                                 6.0
```

```
MaxNWidthForGNRes           double  Yes   As above for the remaining resonances                          4.0
XSec-Integrator             alg
-->

  <param_set name="Default">
     <param type="alg"  name="XSec-Integrator">  genie::ReinSehgalRESXSecFast/Default    </param>
  </param_set>

</alg_conf>
```

For more details, see sections 3.19–3.21.

3.23 Bug: Misprints in the Berger-Sehgal model.

Status: fixed in GENIE 3.0.0
Files: src/ReinSehgal/BSKLNBaseRESPXSec2014.cxx
Method: `double BSKLNBaseRESPXSec2014::XSec(const Interaction * interaction, KinePhaseSpace_t kps) const`

All occurrences of the following code:

```
fFKR.S = KNL_C_minus;
```

should be replaced by the following one:

```
fFKR.C = KNL_C_minus;
```

and all occurrences of the following code:

```
fFKR.S = BRS_C_minus;
```

should be replaced by the following one:

```
fFKR.C = BRS_C_minus;
```

(See Section 3.5 for detailed description).

3.24 Bug: Misprints in the nucleon and baryon resonance parameters.

Status: fixed in GENIE 3.0.0
Files: src/BaryonResonance/BaryonResUtils.cxx
Function: `int genie::utils::res::ResonanceIndex(Resonance_t res)`

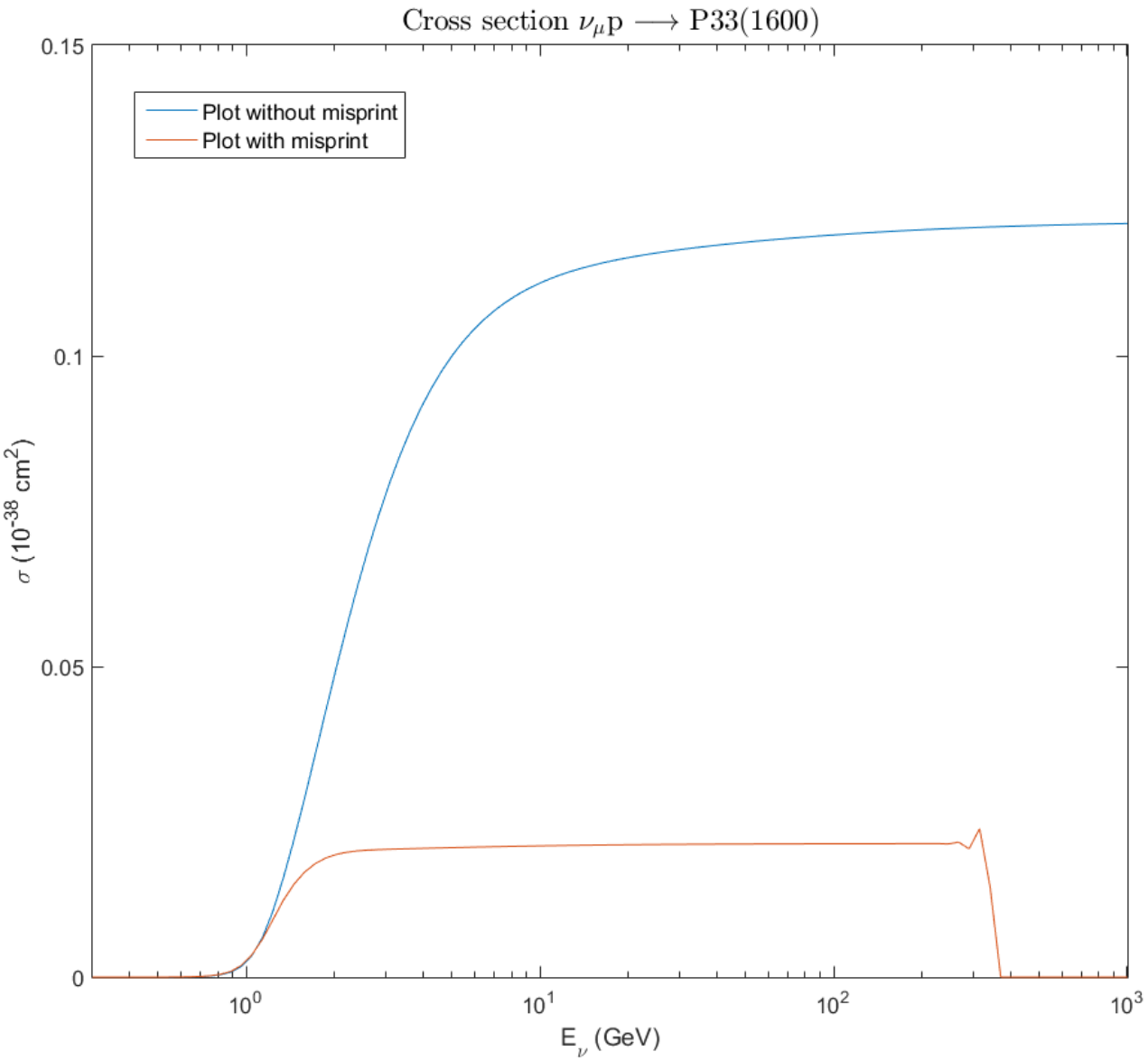

Figure 3.9: Cross section of ν_μ vs. energy.

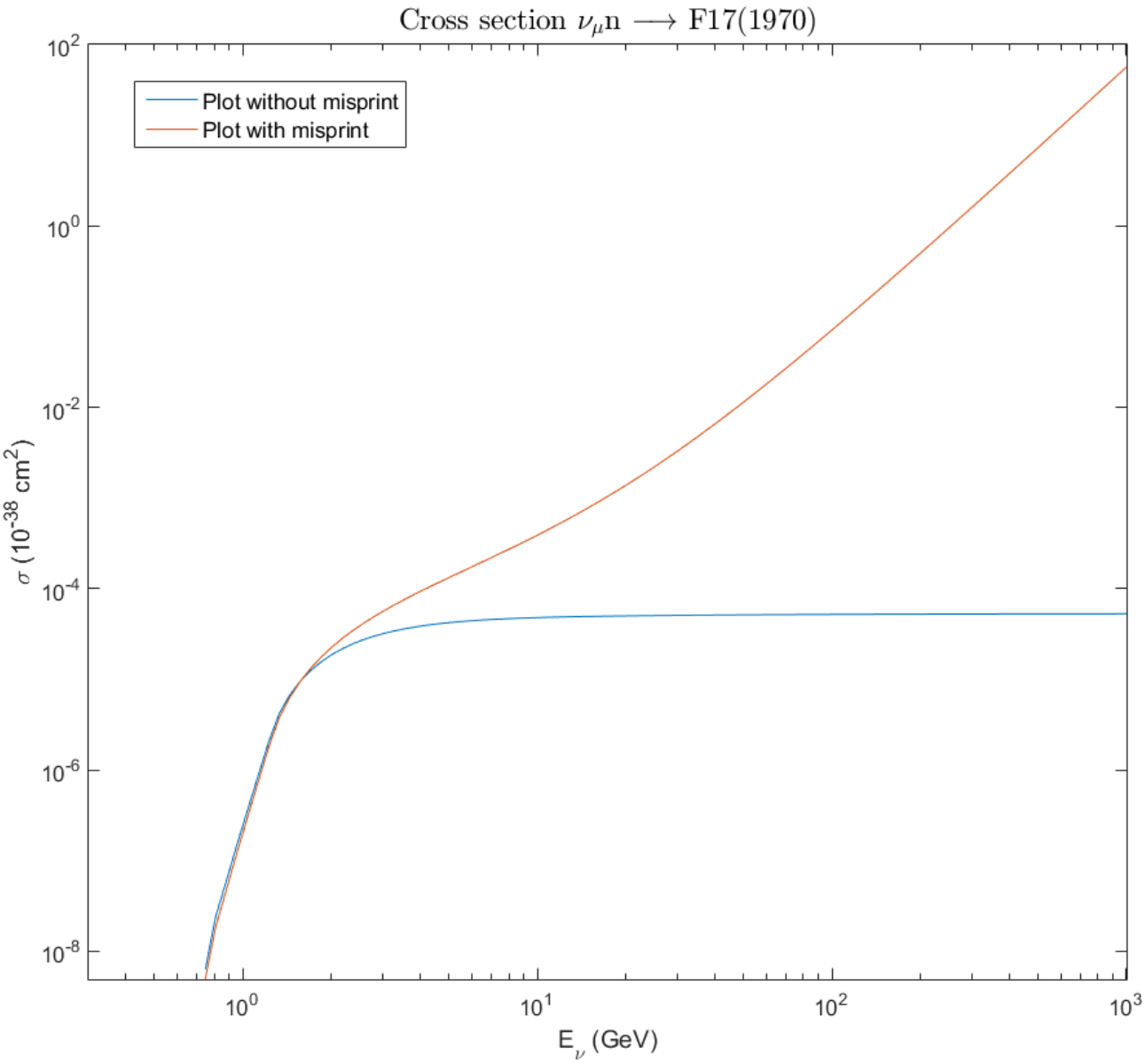

Figure 3.10: Cross section of ν_μ vs. energy.

The lines

```
......
case kP33_1600:  return 9; break;
......
case kF17_1970:  return 0; break;
......
```

should be replaced by the following ones according to Table 1 of Ref. [22]:

```
......
case kP33_1600:  return 2; break;
......
case kF17_1970:  return 2; break;
......
```

3.25 Bug: Misprints in the helicity amplitudes for the Rein-Sehgal model and its extensions.

Status: fixed in GENIE 3.0.0
Files: src/ReinSehgal/RSHelicityAmplModelCC.cxx,
src/ReinSehgal/RSHelicityAmplModelNCn.cxx

According to [22] the lines

```
case (kP13_1720) :
   {
     double L2      = TMath::Power(fkr.Lamda, 2);
     double L2Rm    = L2 * fkr.Rminus;
     double L2Rp    = L2 * fkr.Rplus;
     double LTm     = fkr.Lamda * fkr.Tminus;
     double LTp     = fkr.Lamda * fkr.Tplus;
     double a       = kSqrt3_5 * L2 * fkr.S;
     double b       = kSqrt5_3 * (L2 * fkr.C - 5 * fkr.Lamda * fkr.B);

     fAmpl.fMinus1 =  -kSqrt27_10 * LTm - kSqrt5_3 * L2Rm;
     fAmpl.fPlus1  =   kSqrt27_10 * LTm + kSqrt5_3 * L2Rp;
     fAmpl.fMinus3 =   k3_Sqrt10 * LTm;
     fAmpl.fPlus3  =  -k3_Sqrt10 * LTp;
     fAmpl.f0Minus =   a-b;
     fAmpl.f0Plus  =  -a-b;
     break;
   }
```

of the method `const RSHelicityAmpl & RSHelicityAmplModelCC::Compute(Resonance_t res, const FKR & fkr) const`
in the file RSHelicityAmplModelCC.cxx should be replaced by the following ones

```
case (kP13_1720) :
   {
     double L2      = TMath::Power(fkr.Lamda, 2);
     double L2Rm    = L2 * fkr.Rminus;
     double L2Rp    = L2 * fkr.Rplus;
     double LTm     = fkr.Lamda * fkr.Tminus;
     double LTp     = fkr.Lamda * fkr.Tplus;
     double a       = kSqrt3_5 * L2 * fkr.S;
     double b       = kSqrt5_3 * (L2 * fkr.C - 5 * fkr.Lamda * fkr.B);

     fAmpl.fMinus1 =  -kSqrt27_10 * LTm - kSqrt5_3 * L2Rm;
     fAmpl.fPlus1  =   kSqrt27_10 * LTp + kSqrt5_3 * L2Rp;
     fAmpl.fMinus3 =   k3_Sqrt10 * LTm;
     fAmpl.fPlus3  =  -k3_Sqrt10 * LTp;
     fAmpl.f0Minus =   a-b;
     fAmpl.f0Plus  =  -a-b;
     break;
   }
```

and the lines

```
case (kP33_1600) :
   {
     double xr      = 2*xi*fkr.R;
     double L2      = TMath::Power(fkr.Lamda, 2);
     double L2RmxiR = L2 * (fkr.Rminus + xr);
     double L2RpxiR = L2 * (fkr.Rplus  + xr);

     fAmpl.fMinus1 =  k1_Sqrt6 * L2RmxiR;
     fAmpl.fPlus1  = -k1_Sqrt6 * L2RmxiR;
     fAmpl.fMinus3 =  k1_Sqrt2 * L2RmxiR;
     fAmpl.fPlus3  = -k1_Sqrt2 * L2RpxiR;
     fAmpl.f0Minus = -kSqrt2_3 * (L2 * fkr.C - 2 * fkr.Lamda * fkr.B);
     fAmpl.f0Plus  =  fAmpl.f0Minus;
     break;
   }
```

of the method `const RSHelicityAmpl & RSHelicityAmplModelNCn::Compute(Resonance_t res, const FKR & fkr) const`
in the file RSHelicityAmplModelNCn.cxx should be replaced by the following ones

```
case (kP33_1600) :
   {
     double xr      = 2*xi*fkr.R;
```

```
    double L2      = TMath::Power(fkr.Lamda, 2);
    double L2RmxiR = L2 * (fkr.Rminus + xr);
    double L2RpxiR = L2 * (fkr.Rplus  + xr);

    fAmpl.fMinus1 =  k1_Sqrt6 * L2RmxiR;
    fAmpl.fPlus1  = -k1_Sqrt6 * L2RpxiR;
    fAmpl.fMinus3 =  k1_Sqrt2 * L2RmxiR;
    fAmpl.fPlus3  = -k1_Sqrt2 * L2RpxiR;
    fAmpl.f0Minus = -kSqrt2_3 * (L2 * fkr.C - 2 * fkr.Lamda * fkr.B);
    fAmpl.f0Plus  =  fAmpl.f0Minus;
    break;
  }
```

For example, the influence of the bug in the helicity amplitude on the P13(1720) resonance production cross section in the charged current induced reaction $\nu_\mu n \to$ P13(1720) is shown in Fig. 3.11.

The same effect is for another bug.

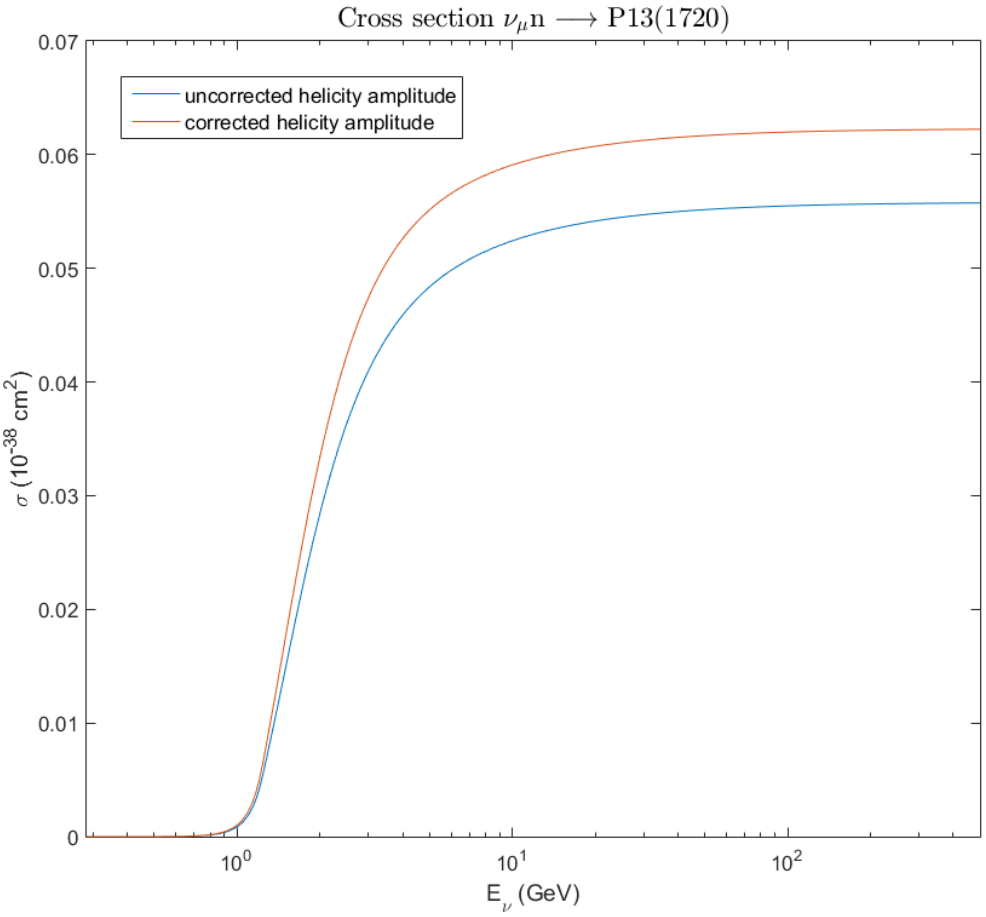

Figure 3.11: Cross section of ν_μ vs. energy.

3.26 Remark: Precision of mathematical and physical constants.

Status: fixed in GENIE 3.0.0

Files: src/Conventions/Constants.h,
src/Conventions/Units.h

It is desirable to set all mathematical constants as precisely as it allowed by the machine precision. To avoid the hard-to-detect bugs and typos in the mathematical constants, it is better to calculate them by using the standard mathematical function, rather than set them as numerical literals.

It is also better to set the physical constants to the current values according to PDG-2016 [23]. For these purposes, the relevant code in the file Constants.h should be replaced by the following one:

```
//
// pi, e,...
//
static const double kPi    = TMath::Pi();
static const double kPi2   = TMath::Power(kPi,2);
static const double kPi3   = TMath::Power(kPi,3);
static const double kPi4   = TMath::Power(kPi,4);
static const double kSqrtPi= TMath::Sqrt(kPi);
static const double ke     = TMath::E();
static const double kSqrte = TMath::Sqrt(ke);

//
// Avogadro number, Compton wavelength and such...
//
static const double kNA    = 6.022140857E+23;
static const double kLe    = 3.8615926764E-13 *units::m;
static const double kLe2   = TMath::Power(kLe,2);

//
// Coupling constants
//
static const double kAem   = 1./137.035999139;       // EM coupling const, dimensionless
static const double kAem2  = TMath::Power(kAem,2);
static const double kGF    = 1.1663787E-05;         // Fermi const from b-decay, in GeV^-2
static const double kGF2   = TMath::Power(kGF,2);

//
// Masses
//
// For simplicity, the most commonly used particle masses defined here.
// In general, however, particle masses in GENIE classes should be obtained
// through the genie::PDGLibrary as shown below:
// double mass = PDGLibrary::Instance()->Find(pdg_code)->Mass();
// For consistency, the values below must match whatever is used in PDGLibrary.
//
static const double kElectronMass   =  0.5109989461E-3;       // GeV
static const double kMuonMass       =  0.1056583745;          // GeV
static const double kTauMass        =  1.77686;               // GeV
static const double kPionMass       =  0.13957018;            // GeV
static const double kPi0Mass        =  0.1349766;             // GeV
static const double kProtonMass     =  0.938272081;           // GeV
static const double kNeutronMass    =  0.939565413;           // GeV
static const double kNucleonMass    =  (kProtonMass+kNeutronMass)/2.;
static const double kLightestChmHad =  1.870;                 // GeV ~lightest charm hadron+

static const double kElectronMass2  =  TMath::Power(kElectronMass,2); // GeV^2
static const double kMuonMass2      =  TMath::Power(kMuonMass,2);     // GeV^2
static const double kTauMass2       =  TMath::Power(kTauMass,2);      // GeV^2
static const double kPionMass2      =  TMath::Power(kPionMass,2);     // GeV^2
static const double kProtonMass2    =  TMath::Power(kProtonMass,2);   // GeV^2
static const double kNeutronMass2   =  TMath::Power(kNeutronMass,2);  // GeV^2
static const double kNucleonMass2   =  TMath::Power(kNucleonMass,2);  // GeV^2

static const double kMw             =  80.385;                        // GeV - W boson mass
static const double kMz             =  91.1876;                       // GeV - Z boson mass
static const double kMw2            =  TMath::Power(kMw,2);  // GeV^2
static const double kMz2            =  TMath::Power(kMz,2);  // GeV^2

//
// Misc constants for empirical formulas
//
```

```
static const double kNucRo      = 1.2E-15 * units::m;            // Ro in nuclear radius formula R=Ro*A^(1/3), in GeV^-1
static const double kNucDensity = 2.3E+17 * units::kg/units::m3; // Nuclear density (in nuclear core), in GeV^4

//
// Earth consts
//
static const double kREarth = 6378.1 * units::km;

//
// Sqrts frequently encountered in helicity amplitude calculations
//
static const double kSqrt2     =  TMath::Sqrt2();
static const double kSqrt3     =  TMath::Sqrt(3);
static const double kSqrt4     =  TMath::Sqrt(4);
static const double kSqrt5     =  TMath::Sqrt(5);
static const double kSqrt6     =  TMath::Sqrt(6);
static const double kSqrt7     =  TMath::Sqrt(7);
static const double kSqrt8     =  TMath::Sqrt(8);
static const double kSqrt9     =  TMath::Sqrt(9);
static const double kSqrt10    =  TMath::Sqrt(10);
static const double kSqrt12    =  TMath::Sqrt(12);
static const double kSqrt15    =  TMath::Sqrt(15);
static const double kSqrt18    =  TMath::Sqrt(18);
static const double kSqrt20    =  TMath::Sqrt(20);
static const double kSqrt24    =  TMath::Sqrt(24);
static const double kSqrt27    =  TMath::Sqrt(27);
static const double kSqrt30    =  TMath::Sqrt(30);
static const double kSqrt35    =  TMath::Sqrt(35);
static const double kSqrt40    =  TMath::Sqrt(40);
static const double kSqrt60    =  TMath::Sqrt(60);
static const double kSqrt120   =  TMath::Sqrt(120);
static const double k1_Sqrt2   =  1/TMath::Sqrt2();
static const double k1_Sqrt3   =  1/TMath::Sqrt(3);
static const double k1_Sqrt5   =  1/TMath::Sqrt(5);
static const double k1_Sqrt6   =  1/TMath::Sqrt(6);
static const double k1_Sqrt7   =  1/TMath::Sqrt(7);
static const double k1_Sqrt10  =  1/TMath::Sqrt(10);
static const double k1_Sqrt15  =  1/TMath::Sqrt(15);
static const double k1_Sqrt24  =  1/TMath::Sqrt(24);
static const double k1_Sqrt30  =  1/TMath::Sqrt(30);
static const double k1_Sqrt35  =  1/TMath::Sqrt(35);
static const double k1_Sqrt60  =  1/TMath::Sqrt(60);
static const double k1_Sqrt120 =  1/TMath::Sqrt(120);
static const double k2_Sqrt3   =  2/TMath::Sqrt(3);
static const double k2_Sqrt5   =  2/TMath::Sqrt(5);
static const double k2_Sqrt15  =  2/TMath::Sqrt(15);
static const double k2_Sqrt35  =  2/TMath::Sqrt(35);
static const double k3_Sqrt2   =  3/TMath::Sqrt2();
static const double k3_Sqrt5   =  3/TMath::Sqrt(5);
static const double k3_Sqrt10  =  3/TMath::Sqrt(10);
static const double k3_Sqrt20  =  3/TMath::Sqrt(20);
static const double k3_Sqrt40  =  3/TMath::Sqrt(40);
static const double kSqrt2_3   =  TMath::Sqrt(2.0/3);
static const double kSqrt2_5   =  TMath::Sqrt(2.0/5);
static const double kSqrt2_6   =  TMath::Sqrt(2.0/6);
static const double kSqrt2_7   =  TMath::Sqrt(2.0/7);
static const double kSqrt2_15  =  TMath::Sqrt(2.0/15);
static const double kSqrt3_2   =  TMath::Sqrt(3.0/2);
static const double kSqrt3_4   =  TMath::Sqrt(3.0/4);
static const double kSqrt3_5   =  TMath::Sqrt(3.0/5);
static const double kSqrt3_8   =  TMath::Sqrt(3.0/8);
static const double kSqrt3_10  =  TMath::Sqrt(3.0/10);
static const double kSqrt3_18  =  TMath::Sqrt(3.0/18);
static const double kSqrt3_20  =  TMath::Sqrt(3.0/20);
static const double kSqrt3_35  =  TMath::Sqrt(3.0/35);
static const double kSqrt3_40  =  TMath::Sqrt(3.0/40);
static const double kSqrt4_15  =  TMath::Sqrt(4.0/15);
static const double kSqrt5_2   =  TMath::Sqrt(5.0/2);
static const double kSqrt5_3   =  TMath::Sqrt(5.0/3);
static const double kSqrt5_8   =  TMath::Sqrt(5.0/8);
static const double kSqrt5_12  =  TMath::Sqrt(5.0/12);
static const double kSqrt6_5   =  TMath::Sqrt(6.0/5);
static const double kSqrt6_35  =  TMath::Sqrt(6.0/35);
static const double kSqrt9_10  =  TMath::Sqrt(9.0/10);
static const double kSqrt9_40  =  TMath::Sqrt(9.0/40);
static const double kSqrt18_5  =  TMath::Sqrt(18.0/5);
static const double kSqrt18_20 =  TMath::Sqrt(18.0/20);
static const double kSqrt18_35 =  TMath::Sqrt(18.0/35);
static const double kSqrt24_35 =  TMath::Sqrt(24.0/35);
```

```
static const double kSqrt27_10 =  TMath::Sqrt(27.0/10);
static const double kSqrt27_40 =  TMath::Sqrt(27.0/40);
```

The appropriate lines in the file Units.h should be replaced by the following ones:

```
static const double second   = 1.51926847E+24/GeV;
static const double meter    = second/299792458/GeV;
static const double kilogram = 5.60958864E+26*GeV;
```

The foregoing list of constants is incomplete. Some constants are not set to more precise values, some are defined in the others files, for example in UserPhysicsOptions.xml and users, therefore, must keep their eyes on precision of specified in GENIE constants.

3.27 Remark: Pion mass in the Breit-Wigner function.

Status: fixed in GENIE 3.0.0

Files: src/Utils/BWFunc.h,
src/Utils/BWFunc.cxx

While the isospin symmetry suggests to neglect the difference in the charged and neutral pion masses, phenomenologically it is not quite appropriate to use the mass of π^0 universally in calculating the Breit-Wigner function,[4] as this is done in GENIE (see the file BWFunc.cxx):

```
double genie::utils::bwfunc::BreitWignerL(
               double W, int L, double mass, double width0, double norm)
{
//Inputs:
// - W:      Invariant mass (GeV)
// - L:      Resonance orbital angular momentum
// - mass:   Resonance mass (GeV)
// - width0: Resonance width
// - norm:   Breit Wigner norm

  //-- sanity checks
  assert(mass   >  0);
  assert(width0 >  0);
  assert(norm   >  0);
  assert(W      >  0);
  assert(L      >= 0);

  //-- auxiliary parameters
  double mN    = kNucleonMass;
  double mPi   = kPi0Mass;
  double m_2   = TMath::Power(mass, 2);
  double mN_2  = TMath::Power(mN,   2);
  double mPi_2 = TMath::Power(mPi,  2);
  double W_2   = TMath::Power(W,    2);

  //-- calculate the L-dependent resonance width
  double qpW_2 = ( TMath::Power(W_2 - mN_2 - mPi_2, 2) - 4*mN_2*mPi_2 );
  double qpM_2 = ( TMath::Power(m_2 - mN_2 - mPi_2, 2) - 4*mN_2*mPi_2 );
  if(qpW_2 < 0) qpW_2 = 0;
  if(qpM_2 < 0) qpM_2 = 0;
  double qpW   = TMath::Sqrt(qpW_2) / (2*W);
  double qpM   = TMath::Sqrt(qpM_2) / (2*mass);
  double width = width0 * TMath::Power( qpW/qpM, 2*L+1 );

  //-- calculate the Breit Wigner function for the input W
  double width_2 = TMath::Power( width,  2);
  double W_m_2   = TMath::Power( W-mass, 2);

  double bw = (0.5/kPi) * (width/norm) / (W_m_2 + 0.25*width_2);
  return bw;
}
```

It seems to be more reasonable to use the mass of that pion which has been produced in the given reaction, even if the effect is very small.

[4] The same is all the more true for kinematics.

3.28 Bug: Memory leaks.

Status: fixed in GENIE 3.0.0

Files: src/HadronTransport/INukeHadroData.cxx,
src/MEC/MECHadronTensor.cxx,
src/Geo/PointGeomAnalyzer.cxx,
src/HadronTransport/HAIntranuke.cxx,
src/Decay/BaryonResonanceDecayer.cxx,
src/Fragmentation/KNOHadronization.cxx,
src/Utils/RunOpt.cxx,
src/Algorithm/AlgConfigPool.cxx,
src/Nuclear/FermiMomentumTablePool.cxx,
src/Messenger/Messenger.cxx

The methods listed below suffer from memory leaks caused by several bugs. A way to remove these bugs is proposed bellow.

To fix the problems one need:

- add the following lines in the destructor `INukeHadroData::~INukeHadroData()` in the file INukeHadroData.cxx:

```
// p/n+p/n hA x-section splines
delete fXSecPp_Tot;
delete fXSecPp_Elas;
delete fXSecPp_Reac;
delete fXSecPn_Tot;
delete fXSecPn_Elas;
delete fXSecPn_Reac;
delete fXSecNn_Tot;
delete fXSecNn_Elas;
delete fXSecNn_Reac;

// K+A x-section fraction splines
delete fFracKA_Tot;
delete fFracKA_Elas;
delete fFracKA_Inel;
delete fFracKA_Abs;
```

- add following lines in the destructor `MECHadronTensor::~MECHadronTensor()` in the file MECHadronTensor.cxx:

```
vector<int>::const_iterator it=fKnownTensors.begin();
for( ; it!=fKnownTensors.end(); ++it)
{
  MECHadronTensor::MECHadronTensorTable table = fTargetTensorTables[*it];
  for(int tensorType = 0; tensorType <= MECHadronTensor::kMHTValenciaDeltapn; ++tensorType)
  {
    vector<genie::BLI2DNonUnifGrid*> Grids = table.Table[(MECHadronTensor::MECHadronTensorType_t)tensorType];
    for (vector<genie::BLI2DNonUnifGrid*>::iterator gridit = Grids.begin() ; gridit != Grids.end(); ++gridit)
    {
      genie::BLI2DNonUnifGrid *hadTensorGrid = *gridit;
      if(hadTensorGrid)
      {
         delete hadTensorGrid;
         hadTensorGrid=0;
      }
    }
  }
}
```

- add the line in the method `void PointGeomAnalyzer::CleanUp(void)` in the file PointGeomAnalyzer.cxx:

```
if( fCurrVertex )         delete fCurrVertex;
```

- add the following lines in the method `void HAIntranuke::Inelastic(GHepRecord* ev, GHepParticle* p, INukeFateHA_t fate) const` in the file HAIntranuke.cxx:

```
delete t1;
delete t2;
```

after the lines:

```
ev->AddParticle(*t1);
ev->AddParticle(*t2);
```

- replace the line in the method `TClonesArray * BaryonResonanceDecayer::DecayExclusive(int pdg_code, TLorentzVector & p, TDecayChannel * ch) const` in the file BaryonResonanceDecayer.cxx:

  ```
  TClonesArray * particle_list = new TClonesArray("TMCParticle", 1+nd);
  ```

  by the following one:

  ```
  TClonesArray * particle_list = 0;
  ```

- add the line in the method `TClonesArray * KNOHadronization::Hadronize(const Interaction * interaction) const` in the file KNOHadronization.cxx:

  ```
  delete pdgcv;
  ```

  before the last line of the method:

  ```
  return particle_list;
  ```

- add the line in the destructor `RunOpt::~RunOpt()` in the file RunOpt.cxx:

  ```
  if( fUnphysEventMask )        delete fUnphysEventMask;
  ```

Some memory leaks arise because incorrect work with **libxml2** library. To fix them one need:

- replace the lines in the method `bool AlgConfigPool::LoadMasterConfig(void)` in the file AlgConfigPool.cxx:

  ```
  xmlNodePtr xml_root = xmlDocGetRootElement(xml_doc);
  if(xml_root==NULL) {
   SLOG("AlgConfigPool", pERROR)
                  << "The XML doc is empty! (filename : " << fMasterConfig << ")";
   xmlFree(xml_doc);
   return false;
  }
  ```

  by the following ones:

  ```
  xmlNodePtr xml_root = xmlDocGetRootElement(xml_doc);
  if(xml_root==NULL) {
   SLOG("AlgConfigPool", pERROR)
                  << "The XML doc is empty! (filename : " << fMasterConfig << ")";
   xmlFreeDoc(xml_doc);
   return false;
  }
  ```

  replace the lines:

  ```
  if( xmlStrcmp(xml_root->name, (const xmlChar *) "genie_config") ) {
   SLOG("AlgConfigPool", pERROR)
                   << "The XML doc has invalid root element! "
                                                          << "(filename : " << fMasterConfig << ")";
   return false;
  }
  ```

  by the following ones:

  ```
  xmlNodePtr xml_root = xmlDocGetRootElement(xml_doc);
  if(xml_root==NULL) {
   SLOG("AlgConfigPool", pERROR)
                  << "The XML doc is empty! (filename : " << fMasterConfig << ")";
   xmlFreeDoc(xml_doc);
   return false;
  }
  ```

  and replace the lines:

  ```
  xmlFree(xml_ac);
  xmlFree(xml_doc);
  ```

  by the following ones:

  ```
  xmlFreeNode(xml_ac);
  xmlFreeDoc(xml_doc);
  ```

- replace the lines in the method `bool AlgConfigPool::LoadRegistries(string key_prefix, string file_name, string root)` in the file AlgConfigPool.cxx:

```
xmlNodePtr xml_cur = xmlDocGetRootElement( xml_doc );
if(xml_cur==NULL) {
 SLOG("AlgConfigPool", pERROR)
                << "The XML document is empty! (filename : " << file_name << ")";
 return false;
}
if( xmlStrcmp(xml_cur->name, (const xmlChar *) root.c_str()) ) {
 SLOG("AlgConfigPool", pERROR)
                 << "The XML document has invalid root element! "
                                                                    << "(filename : " << file_name << ")";
 return false;
}
```

by the following ones:

```
xmlNodePtr xml_cur = xmlDocGetRootElement( xml_doc );
if(xml_cur==NULL) {
 SLOG("AlgConfigPool", pERROR)
                << "The XML document is empty! (filename : " << file_name << ")";
 xmlFreeDoc(xml_doc);
 return false;
}
if( xmlStrcmp(xml_cur->name, (const xmlChar *) root.c_str()) ) {
 SLOG("AlgConfigPool", pERROR)
                 << "The XML document has invalid root element! "
                                                                    << "(filename : " << file_name << ")";
 xmlFreeNode(xml_cur);
 xmlFreeDoc(xml_doc);
 return false;
}
```

and replace all occurrences of the lines:

```
xmlFree(xml_param);
xmlFree(xml_cur);
xmlFree(xml_doc);
```

by the following ones:

```
xmlFreeNode(xml_param);
xmlFreeNode(xml_cur);
xmlFreeDoc(xml_doc);
```

- replace the lines in the method `XmlParserStatus_t FermiMomentumTablePool::ParseXMLTables(string filename)` in the file FermiMomentumTablePool.cxx:

```
if(xml_root==NULL) return kXmlEmpty;
const xmlChar * xml_root_name = (const xmlChar *)"fermi_momentum_const";
if( xmlStrcmp(xml_root->name, xml_root_name) ) return kXmlInvalidRoot;
```

by the following ones:

```
if(xml_root==NULL)
{
  xmlFreeDoc(xml_doc);
  return kXmlEmpty;
}
const xmlChar * xml_root_name = (const xmlChar *)"fermi_momentum_const";
if( xmlStrcmp(xml_root->name, xml_root_name) )
{
  xmlFreeDoc(xml_doc);
  return kXmlInvalidRoot;
}
```

replace all occurrences of the lines:

```
xmlFree(xml_cur);
xmlFree(xml_kf);
xmlFree(xml_kft);
```

by the following ones:

```
xmlFreeNode(xml_cur);
xmlFreeNode(xml_kf);
xmlFreeNode(xml_kft);
```

and replace the lines:

```
xmlFree(xml_root);
xmlFree(xml_doc);
```

by the following one:

```
xmlFreeDoc(xml_doc);
```

- replace the lines in the method `bool Messenger::SetPrioritiesFromXmlFile(string filenames)` in the file Messenger.cxx:

```
xmlNodePtr xml_root = xmlDocGetRootElement(xml_doc);
if(xml_root==NULL) {
   SLOG("Messenger", pERROR)
           << "XML doc. has null root element! [file: " << filename << "]";
   return false;
}

if( xmlStrcmp(xml_root->name, (const xmlChar *) "messenger_config") ) {
   SLOG("Messenger", pERROR)
         << "XML doc. has invalid root element! [file: " << filename << "]";
   return false;
}
```

by the following one:

```
xmlNodePtr xml_root = xmlDocGetRootElement(xml_doc);
if(xml_root==NULL) {
   SLOG("Messenger", pERROR)
           << "XML doc. has null root element! [file: " << filename << "]";
   xmlFreeDoc(xml_doc);
   return false;
}

if( xmlStrcmp(xml_root->name, (const xmlChar *) "messenger_config") ) {
   SLOG("Messenger", pERROR)
         << "XML doc. has invalid root element! [file: " << filename << "]";
   xmlFreeNode(xml_root);
   xmlFreeDoc(xml_doc);
   return false;
}
```

and replace the line:

```
xmlFree(xml_msgp);
```

by the followings two:

```
xmlFreeNode(xml_msgp);
xmlFreeDoc(xml_doc);
```

Acknowledgments

We gratefully acknowledge the help of Costas Andreopoulos, Gabriel Nathan Perdue, Hugh Gallagher and Steven A. Dytman. Also we would like to acknowledge numerous useful conversations with the members of the JINR NOνA group. This work has been partially supported by the Russian Foundation for Basic Research under Grants No. 14-22-03090 and No. 16-02-01104-a.

Bibliography

[1] GENIE Neutrino Monte Carlo. http://www.genie-mc.org/.

[2] C. Andreopoulos et al. The GENIE neutrino Monte Carlo generator. *Nucl. Instrum. Meth.*, A614:87–104, 2010.

[3] C. Andreopoulos, C. Barry, S. Dytman, H. Gallagher, T. Golan, R. Hatcher, G. Perdue, and J. Yarba. The GENIE neutrino Monte Carlo generator: Physics and user manual. 2015.

[4] R. A. Smith and E. J. Moniz. Neutrino reactions on nuclear targets. *Nucl. Phys.*, B43:605–622, 1972. [Erratum *ibid.* B101, 547 (1975)].

[5] K. S. Kuzmin, V. V. Lyubushkin, and V. A. Naumov. Quasielastic axial-vector mass from experiments on neutrino-nucleus scattering. *Eur. Phys. J.*, C54:517–538, 2008.

[6] S. K. Singh and H. Arenhövel. Pion exchange current effects in $\nu_\mu + d \to \mu^- + p + p$. *Z. Phys.*, A324:347–354, 1986.

[7] A. A. Aguilar-Arevalo et al. First measurement of the muon neutrino charged current quasielastic double differential cross section. *Phys. Rev.*, D81:092005, 2010.

[8] A. A. Aguilar-Arevalo et al. First measurement of the muon antineutrino double-differential charged-current quasielastic cross section. *Phys. Rev.*, D88(3):032001, 2013.

[9] A. Bodek, S. Avvakumov, R. Bradford, and Howard Scott Budd. Vector and Axial Nucleon Form Factors:A Duality Constrained Parameterization. *Eur. Phys. J.*, C53:349–354, 2008.

[10] A. C. Genz and A. A. Malik. Remarks on algorithm 006: An adaptive algorithm for numerical integration over an n-dimensional rectangular region. *J. Comput. Appl. Math.*, 6(4):295–302, 1980.

[11] S. L. Adler, S. Nussinov, and E. A. Paschos. Nuclear Charge Exchange Corrections to Leptonic Pion Production in the (3,3) Resonance Region. *Phys. Rev.*, D9:2125–2143, 1974. [Erratum Phys. Rev. D 10 (1974) 1669]].

[12] J. Y. Yu. *Neutrino interactions and nuclear effects in oscillation experiments and the nonperturbative dispersive sector in strong (quasi-)Abelian fields.* PhD thesis, Dortmund U., 2002.

[13] E. A. Paschos, J. Y. Yu, and M. Sakuda. Neutrino production of resonances. *Phys. Rev.*, D69:014013, 2004.

[14] K. S. Kuzmin and V. A. Naumov. Mean charged multiplicities in charged-current neutrino scattering on hydrogen and deuterium. *Phys. Rev.*, C88:065501, 2013.

[15] A. Bodek and J. L. Ritchie. Fermi motion effects in deep inelastic lepton scattering from nuclear targets. *Phys. Rev.*, D23:1070–1091, 1981.

[16] A. Bodek and J. L. Ritchie. Further studies of Fermi motion effects in lepton scattering from nuclear targets. *Phys. Rev.*, D24:1400–1402, 1981.

[17] G. Audi et al. The 2012 atomic mass evaluation and the mass tables. *Nucl. Data Sheets*, 120:1–5, 2014.

[18] K. S. Kuzmin, V. V. Lyubushkin, and V. A. Naumov. Axial masses in quasielastic neutrino scattering and single-pion neutrinoproduction on nucleons and nuclei. *Acta Phys. Polon.*, B37:2337–2348, 2006.

[19] A. Strumia and F. Vissani. Precise quasielastic neutrino/nucleon cross section. *Phys. Lett.*, B564:42–54, 2003.

[20] K. S. Kuzmin and V. A. Naumov. Axial mass in reactions of quasielastic antineutrino-nucleon scattering with strange hyperon production. *Phys. Atom. Nucl.*, 72:1501–1512, 2009. [*Yad. Fiz.* 72:1555–1566, (2009)].

[21] K. S. Kuzmin, V. V. Lyubushkin, and V. A. Naumov. Lepton polarization in neutrino nucleon interactions. *Mod. Phys. Lett.*, A19:2815–2829, 2004. [Phys. Part. Nucl. 35, S133 (2004)].

[22] D. Rein and L. M. Sehgal. Neutrino excitation of baryon resonances and single pion production. *Annals Phys.*, 133:79–153, 1981.

[23] C. Patrignani. Review of particle physics. *Chin. Phys.*, C40(10):100001, 2016.

[24] K. S. Kuzmin and V. A. Naumov. Running axial-vector mass of the nucleon for a precise evaluation of the quasielastic (anti)neutrino–nucleus cross section. 2016. in progress.

[25] L. D. Kolupaeva, K. S. Kuzmin, O. N. Petrova, and I. M. Shandrov. Some uncertainties of neutrino oscillation effect in the NOνA experiment. *Mod. Phys. Lett.*, A31(12):1650077, 2016.

[26] I. D. Kakorin, K. S. Kuzmin, and V. A. Naumov. Running axial mass for quasielastic neutrino-nucleus scattering. In *International Worikshop on Global Fit to Neutrino Scattering Data and Generator Tuning 'NuTune2016', UK, University of Liverpool, July 11–12, 2016*. https://indico.fnal.gov/contributionDisplay.py?contribId=14&sessionId=18&confId=11610 (unpublished).

[27] I. D. Kakorin, K. S. Kuzmin, and V. A. Naumov. Running axial mass for CCQE neutrino-nucleus scattering. In *NOvA Collaboration Meeting, ND Physics Working Groups (FNAL, Batavia, IL, October 20–23, 2016*. http://nova-docdb.fnal.gov:8080/cgi-bin/DisplayMeeting?conferenceid=2959 (NOVA Document 16335-v5).

[28] J. L. Alcaraz-Aunion and J. Walding. Measurement of the ν_μ-CCQE cross section in the SciBooNE experiment. *AIP Conf. Proc.*, 1189:145–150, 2009.

[29] A. Martinez de la Ossa Romero. *Study of accelerator neutrino interactions in a liquid argon TPC*. PhD thesis, Granada U., Theor. Phys. Astrophys., 2007.

[30] V.V. Lyubushkin et al. A study of quasi-elastic muon neutrino and antineutrino scattering in the NOMAD experiment. *Eur. Phys. J.*, C63:355–381, 2009.

[31] T. Kikawa (on behalf of the T2K Collaboration). Recent results from T2K on neutrino interaction measurements. In *XXIV Workshop on Weak Interactions and Neutrinos 'WIN13', Natal, Brazil, September 16—21, 2013*. http://www.t2k.org/docs/talk/170/win13 (unpublished).

[32] D. Hadley (on behalf of the T2K Collaboration). Measurement of the ν_μ CCQE cross section with the ND280 detector at T2K. *PoS(EPS-HEP 2013)008*, 2013.

[33] K. Abe et al. Measurement of the inclusive ν_μ charged current cross section on carbon in the near detector of the T2K experiment. *Phys. Rev.*, D87(9):092003, 2013.

[34] K. Abe et al. Measurement of the ν_μ charged-current quasielastic cross section on carbon with the ND280 detector at T2K. *Phys. Rev.*, D92(11):112003, 2015.

[35] E. J. Moniz, I. Sick, R. R. Whitney, J. R. Ficenec, R. D. Kephart, and W. P. Trower. Nuclear Fermi momenta from quasielastic electron scattering. *Phys. Rev. Lett.*, 26:445–448, 1971.

[36] R. R. Whitney, I. Sick, J. R. Ficenec, R. D. Kephart, and W. P. Trower. Quasielastic electron scattering. *Phys. Rev.*, C9:2230–2235, 1974.

[37] P. D. Zimmerman and M. R. Yearian. Fermi momenta and separation energies obtained from the quasi-elastic scattering of electrons from ^{48}Ca and ^{40}Ca. *Z. Phys.*, A278:291–293, 1976.