NUISANCE Tutorial

Adding Samples

Patrick Stowell, Luke Pickering,
Clarence Wret, Callum Wilkinson

06/09/17

Introduction

- Going to focus on adding a sample into NUISANCE today.

- End result will be a sample implementation that you can load from a card file, using the same method we saw in the nuiscomp examples last week.

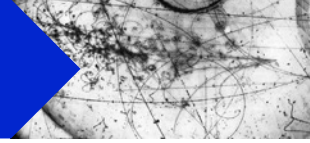
- **Will try to cover**
  1. Core Sample Structure
  2. Actually Adding a sample

Why do I care?

- Adding data into our framework gives people another way to easily make comparisons with your measurement.

- Rack up those citation counts!

- We also use the same framework for model comparisons/testing on T2K, so very useful if you ever find yourself having to implement a generator model /reweight dial.



Processing Structure

Building NUISANCE

- **NUISANCE requires a few external dependencies to build**
  - ROOT
  - Any generator and its dependencies.

- So if we want GENIE we have to build GENIESupport aswell...

- Some build notes are on our site:

```
https://nuisance.hepforge.org/nuisanceinstallation.html
```

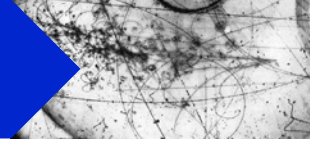
Building NUISANCE (2)

- If you are on a FNAL gpvm you can use /cvmfs/

```
git clone http://nuisance.hepforge.org/git/nuisance.git
cd nuisance_tutorial/cvmfsbuild/
source nuisance-dependencies.sh
source nuisance-checkout.sh
```

- Should work for everyone on a FNAL gpvm, let me know if it fails!

- If you are not on a gpvm you will have to already have your own installation of NUISANCE on your computer.



Processing Structure

Requesting a sample

```
<nuisance>
  <!-- List of Samples -->
  <sample name="MINERvA_CCQE_XSec_1DQ2_nu"
          input="GENIE:genie/gntp.Default.MINERvA_fhc_numu.CH.2500000.root" />
</nuisance>
```

- Saw last week that we can request different data comparisons in NUISANCE by specifying the sample name in a card file.

- What is actually happening when we put a sample here?

Routines Classes

- The card file you read in gets passed to a routines class for the given application

Application	Routines Class
nuiscomp	src/Routines/ComparisonRoutines.cxx
nuismin	src/Routines/MinimiserRoutines.cxx
nuissyst	src/Routines/SystematicRoutines.cxx

- Routines parse your options and handle the overall running of whatever you are trying to do.

- First thing they usually do is setup a JointFCN.

JointFCN

- JointFCN is just a list of all samples/priors loaded that allows you to easily process all of them as efficiently as possible.

- When created it checks the configuration for a list of sample XML entries and tries to create a sample for each one.

src/FCN/JointFCN.cxx :: LoadSamples

```
MeasurementBase* NewLoadedSample = SampleUtils::CreateSample(key);
```

- Where an ugly string comparison list instantiates the class.

src/FCN/SampleList.cxx :: CreateSample

```
...
 } else if (!name.compare("MINERvA_CCOpi_XSec_1DThetae_nue")) {
   return (new MINERvA_CCOpi_XSec_1DThetae_nue(samplekey));
...
```

Event Loop

- When JointFCN->ReconfigureSamples() is called, the JointFCN loops over all the samples it has loaded and processes their events to produce the MC curves.

```
[LOG Reconf]:--- Starting Reconfigure iter. 0
[LOG Reconf]:--- Event Manager Reconfigure
[LOG Reconf]:--- MINERvA_CC1pip1p_XSec_1DTpi_nu : Processed 0 events. [M, W] = [13, 1]
[LOG Reconf]:--- MINERvA_CC1pip1p_XSec_1DTpi_nu : Processed 20000 events. [M, W] = [13, 1]
[LOG Reconf]:--- MINERvA_CC1pip1p_XSec_1DTpi_nu : Processed 40000 events. [M, W] = [13, 1]
[LOG Reconf]:--- MINERvA_CC1pip1p_XSec_1DTpi_nu : Processed 60000 events. [M, W] = [26, 1]
[LOG Reconf]:--- MINERvA_CC1pip1p_XSec_1DTpi_nu : Processed 80000 events. [M, W] = [1, 1]
[LOG Reconf]:--- Filled 4104 signal events.
[LOG Reconf]:--- Time taken ReconfigureUsingManager() : 21
```

- Sample class implementations contain all information required to figure out which events are signal and what should be filled.

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

This is not the actual code, but gives an idea of what functions are called inside the Measurement classes

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

Reset all MC histograms

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Gets all weights for parameters you gave to NUISANCE

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Calculates any kinematics inside the sample ready to be filled

FillEventVariables called first so that extra background plots can also be created if needed.

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Checks if the standard histograms should actually be filled

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Actually fill the variables we calculated before into the histograms

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Take the weighted event spectrum and calculate a predicted cross-section.

Simplified Event Loop

- Every time Reconfigure is called the following steps usually happen for each sample.

```
MeasurementBase::ResetAll();

for (event in MeasurementBase::GetInput()){

    weight = FitWeight::CalcWeight(event);

    MeasurementBase::FillEventVariables(event);

    if (MeasurementBase::IsSignal(event)){
        MeasurementBase::FillHistograms(rwweight);
    }
}

MeasurementBase::ConvertEventRates()

Chi2 = MeasurementBase::GetLikelihood();
```

Compare MC to data using covariance, etc.

Adding Data

- **To add your data into the existing processing loop you have to add your own a C++ sample class implementation**
- Main design focus is differential cross-sections in 1D and 2D.
- Lots of existing samples for rare cases in the code, and we are constantly thinking of ways to make handling them easier.
- If you have issues, or you think your sample is more complicated let us know and we can help with suggestions on how to implement it.

Adding a Sample

Problems

- Produced a fake data distribution which can be downloaded from the following repo.

```
$ git clone https://github.com/NUISANCEMC/nuisance_tutorial.git
```

- Data files are located in

```
nuisance_tutorial/tutorial050917_addingsamples/data/
```

- Only one problem to do today

Add MINERvA CC1pi1p 1DTpi data

- Methods to add 2D data are very similar. More complex things like ratios/cross-correlations/smearing are also possible, but not covered today sorry.

Templates

- Sample templates are contained inside the folder

```
nuisance_tutorial/tutorial050917_addingsamples/templates/
```

- 3 different types:
  - **templated1D_bare** : Just the required functions, completely empty. Write everything manually.
  - **templated1D_brief** : Brief notes on the required order to be followed in each function.
  - **templated1D_verb** : Verbose notes on what each stage of the file is actually doing.

- Will post verbose solution to following directory afterwards

```
nuisance_tutorial/tutorial050917_addingsamples/solutions/
```

Cheat Codes

- Data is fake, and has been generated from genie Default.
- Errors are statistical, from running over 50000 events

- Correlation matrix is just diagonal to keep things simple.
- If you had a non-diagonal covariance the functionality is exactly the same.

- **Fake data definition means you can check your data with GENIE Default first to see that you get a chi2 of ~0.0.**
- Then compare other models and higher stats to it for extra validation.

MINERvA CC1pip1p 1DTpi

- Data distribution given as a text file:

Bin Edges	Central Value	Bin Error
0.0	5.5742e-42	3.0825e-43
50.0	8.3869e-42	3.7811e-43
100.0	6.5629e-42	3.3448e-43
150.0	5.2418e-42	2.1137e-43
250.0	3.6650e-42	2.4995e-43
300.0	2.9490e-42	2.2421e-43
350.0	1.5512e-42	1.6261e-43
400.0	1.2785e-42	1.4763e-43
450.0	5.7958e-43	9.9397e-44
500.0	0.0	0.0

Last line says the top bin edge

Alongside our data we need a list of cuts to apply to MC

#	Cut
1	Initial State Muon Neutrino
2	Final State Muon (CC)
3	Final State pi+ or pi-
4	>0 Final State Protons
5	Muon Theta Angle < 20 degrees
6	Wexp < 1.4 GeV

MINERvA CC1pip1p 1DTpi

- Have a corresponding diagonal correlation matrix

1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

- Also included information on target/flux. Should already have some MC files from last week so we will use those instead.

```
genie/gntp.Default.MINERvA_fhc_numu.CH.2500000.root
```

Checklist

Requirement	Finished
Class File	
CMakeLists.txt	
Constructor	
FillEventVariables	
IsSignal	
SampleList.cxx	

- First thing we have to do is add a new class

Template Files

- We need to make a new C++ file inside the src directory.
- To avoid writing from scratch, copy the files inside your tutorial folder to the nuisance src directory.
- Rename the class files to something sensible at the same time.

```
$ cp Template1D_verbose.h   $NUISANCE/src/MINERvA/MINERvA_CC1pip1p_XSec_1DTpi_nu.h
$ cp Template1D_verbose.cxx $NUISANCE/src/MINERvA/MINERvA_CC1pip1p_XSec_1DTpi_nu.cxx
```

NAMING FORMAT: EXPERIMENT_CHANNEL_TYPE_DISTRIBUION_EXTRAIDs

Renaming

- Then need to rename the functions inside the template class to our new class name.

- Can do manually in each file or just use **sed**

```
$ sed -e "s/tutorial1D/MINERvA_CC1pip1p_XSec_1DTpi_nu/g" ./MINERvA_CC1pip1p_XSec_1DTpi*
```

- Should end up with something like the file below

MINERvA_CC1pip1p_XSec_1DTpi_nu.h

```
#ifndef MINERvA_CC1pip1p_XSec_1DTpi_nu_H_SEEN
#define MINERvA_CC1pip1p_XSec_1DTpi_nu_H_SEEN
#include "Measurement1D.h"
class MINERvA_CC1pip1p_XSec_1DTpi_nu : public Measurement1D {
    public:
    // Main Constructor
    MINERvA_CC1pip1p_XSec_1DTpi_nu(nuiskey samplekey);
...
```

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	
Constructor	
FillEventVariables	
IsSignal	
SampleList.cxx	

- Now we need to actually register the class with CMAKE.

CMakeLists

- If you renamed the file correctly, it should be buildable.

- To include it in the build we have to manually add it to the CMakeLists file in this folder.

$NUISANCE/src/MINERvA/CMakeLists.txt

```
set(IMPLFILES
MINERvA_CC1pip1p_XSec_1DTpi_nu.cxx

MINERvA_CCQE_XSec_1DQ2_antinu.cxx
…

set(HEADERFILES
MINERvA_CC1pip1p_XSec_1DTpi_nu.h

MINERvA_CCQE_XSec_1DQ2_antinu.h
…
```

Have to add the .cxx file to the IMPLFILES list and the .h file to the HEADERFILES list.

Rebuilding

- With the CMakeLists.txt file updated, now go into the NUISANCE build directory and make the code again.

This directory is wherever you built nuisance using cmake, so could be different to mine.

```
cd $NUISANCE/builds/genie2126-nuwrorw/
make install

...
Scanning dependencies of target exp
[ 28%] Building CXX object
src/MINERvA/CMakeFiles/expMINERvA.dir/MINERvA_CC1pip1p_XSec_1DTpi_nu.cxx.o
MINERvALinking CXX static library libexpMINERvA.a
...
[100%] Built target SignalDefTests
```

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	👍
Constructor	
FillEventVariables	
IsSignal	
SampleList.cxx	

- If it built successfully, great! But the template is just a placeholder, we need to add data and cuts to it.

Functions

- If we look inside the header file we can see the 3 functions we have to define.

```
// Main Constructor where we define what data we need to setup
// and sort out scaling factors.
MINERvA_CC1pip1p_XSec_1DTpi_nu(nuiskey samplekey);

// Function to calculate event kinematics we need when binning.
// Called for ALL events
void FillEventVariables(FitEvent *event);

// Function to figure out whether an event
// is a signal event for this sample.
bool isSignal(FitEvent *event);
```

- All samples need at least these 3 functions defined to work.

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	👍
Constructor	
FillEventVariables	
IsSignal	
SampleList.cxx	

- Lets start with the constructor

- Constructors are a little bit awkward to setup until you are familiar with the structure.

Constructor

- Has 5 main sections that need to go in order.

1. **LoadSampleSettings**	Reads the XML entry for this sample from the card file
2. **FinaliseSampleSettings**	Sets any defaults that weren't provided in the card file
3. **ScalingFactor**	Calculates factor needed to get a cross-section out
4. **PlotSetup**	Initialise data/covariance histograms.
5. **Finalise**	Final setup, clones data histogram to make standard MC histograms.

Load Sample Settings

- When a sample is created the XML key you used to create it is passed to the sample so that it can read all the options.

- Options accessible at any time in the sample by reading the fSettings object (src/FitBase/SampleSettings.cxx)

- E.g. Add some extra flag in a specific sample

src/MINERvA/MINERvA_CC1pip1p_Xsec_1DTpi_nu.cxx

```
fWCutValue = fSettings.GetD("WCut");
```

- Which can be set in the card file

XML cardfile

```
<sample name="MINERvA_CC1pip1p_Xsec_1DTpi_nu" WCut="1.4" ... />
```

Load Sample Settings

- LoadSampleSettings should be called before any proper settings are setup.

- Put this at the top of our constructor

```
std::string descrip = "MINERvA_CC1pip1p_XSec_1DTpi_nu" \
                      "\n Target: CH" \
                      "\n Flux: MINERvA FHC numu" \
                      "\n Signal: CC1pi+/- 1p W<1.4 theta<20deg";

fSettings = LoadSampleSettings(samplekey);
fSettings.SetDescription(descrip);
```

Sample Settings

- Users can pass different inputs to SampleSettings but its annoying if they have to pass them everytime they write a cardfile

- Better if we setup some defaults aswell, so if a specific option isn't provided in the card, NUISANCE knows what to use.

```
fSettings = LoadSampleSettings(samplekey);

fSettings.SetDescription(descrip);
fSettings.SetTitle("MINERvA_CC1pip1p_XSec_1DTpi_nu");
fSettings.SetXTitle("T_{#pi} (MeV)");
fSettings.SetYTitle("d#sigma/dT_{#pi} (cm^{2}/MeV/nucleon)");
fSettings.SetEnuRange(0.0, 100.0);
fSettings.DefineAllowedTargets("C,H");
fSettings.DefineAllowedSpecies("numu");
fSettings.SetAllowedTypes("FIX,FREE,SHAPE/DIAG,FULL", "FIX/FULL");

fSettings.SetDataInput( FitPar::GetDataBase()+"/MINERvA/CC1pip1p/Tpi/data.csv");
fSettings.SetCovarInput( FitPar::GetDataBase()+"/MINERvA/CC1pip1p/Tpi/correlation.csv");

FinaliseSampleSettings();
```

Put all your defaults between Load and FinaliseSampleSettings.

The ones shown here are the usual ones included.

SampleSettings : SetTitle

- Pretty self explanatory, set the Title, X, Y, and Z titles for the standard data and MC histograms.

- In our case these will be:

```
fSettings.SetTitle("MINERvA_CC1pip1p_XSec_1DTpi_nu");
fSettings.SetXTitle("T_{#pi} (MeV)");
fSettings.SetYTitle("d#sigma/dT_{#pi} (cm^{2}/MeV/nucleon)");
```

SampleSettings : EnuRange

- EnuCuts have to be handled specially by NUISANCE because they can affect the scaling procedure.

- If your analysis has a TRUE beam energy cut then place the cut ranges in GeV inside `SetEnuRange(low,high)`

- If bounded on one side, or you have no EnuCut then use your experimental range as the limit. (MINERvA ~0.0-100.0GeV)

```
fSettings.SetEnuRange(1.5,  10.0);
```
1.5 < Enu/GeV < 10.0

```
fSettings.SetEnuRange(1.5, 100.0);
```
Enu > 1.5 GeV

```
fSettings.SetEnuRange(0.0, 100.0);
```
No Enu Cut

SampleSettings: Target/Beam

- Define allowed targets/beam functions list things that are allowed to be passed as inputs.
- We eventually plan to have automated checks, but at the moment these don't actually do much...
- Good to add them anyway as it's an extra guide for people using the class to make comparisons.

```
fSettings.DefineAllowedTargets("C,H");
fSettings.DefineAllowedSpecies("numu");
```

Allowed strings can be found inside src/Utils/BeamUtils.cxx and src/Utils/TargetUtils.cxx

SampleSettings: Types

- Saw last week that we could request different ways to handle the likelihood in the fit (e.g. SHAPE or FREE)

- SampleSettings::SetAllowedTypes defines what is actually allowed to be included in the type field.

```
SampleSettings::SetAllowedTypes(allowed_types,
                                default_types);
```

- List all allowed seperated by a slash (/). The second field defines what you want NUISANCE to use if "type" not given.

```
fSettings.SetAllowedTypes("FIX,FREE,SHAPE/DIAG,FULL", "FIX/FULL");
```

This is the usual one to use, default is fixed with a full covariance.

Conflicting Types

- Some types are conflicting, e.g. trying to run a SHAPE only fit with a FREE floating normalization is useless.
- **If you want to make NUISANCE check for this, separate all conflicting types with a comma (,) instead of a slash (/).**

- e.g. Allow FIX, FREE, and SHAPE fits, but don't allow both to be given at the same time.

Constructor

```
fSettings.SetAllowedTypes("FIX,FREE,SHAPE", "FIX");
```

nuiscomp output

```
[LOG Sample]:-- Finalising Sample Settings: MINERvA_CC1pip_XSec_1DTpi_nu
[ERR FATAL ]: ERROR: Conflicting fit options provided: SHAPE/FREE
You should only supply one of these options in card file.
Aborted
```

SampleSettings: SetInput

- `SetDataInput` and `SetCovarInput` tell NUISANCE where the data sources are located.
- In our case these should point to a text or ROOT file inside the NUISANCE data directory `$NUISANCE/data/MINERvA/`
- No data is read at this point, we are just setting up the paths during SampleSettings::SetDataInput().
- **`FitPar::GetDataBase()` returns data folder for you.**

SampleSettings: Input

- First we should copy our data files to the NUISANCE database

```
$ cd nuisance_tutorial/tutorial050917_addingsamples/data/MINERvA_CC1pip1p_XSec_1DTpi_nu
$ mkdir $NUISANCE/data/MINERvA/CC1pi1p/
$ mkdir $NUISANCE/data/MINERvA/CC1pi1p/Tpi/
$ cp ./* $NUISANCE/data/MINERvA/CC1pi1p/Tpi/
```

- Now our data inputs in the SampleSettings are just

```
std::string base = FitPar::GetDataBase();
fSettings.SetDataInput( base+"/MINERvA/CC1pi1p/Tpi/data.csv");
fSettings.SetCovarInput(base+"/MINERvA/CC1pi1p/Tpi/correlation.csv");
```

Finalise Sample Settings

- Finalise sample settings goes through and takes all the XML settings already setup and sets a few different internal flags and objects inside the Measurement class.

```
FinaliseSampleSettings();
```

- **Importantly, it also creates the MC InputHandler at this point.**

- Anything handling the flux/rate/N-events has to come after this point if it requires access to the InputHandler.

Settings Implementation

- So far our SampleSettings section should look something like

```
std::string descrip = "MINERvA_CC1pip1p_XSec_1DTpi_nu" \
                      "\n Target: CH" \
                        "\n Flux: MINERvA FHC numu" \
                        "\n Signal: CC1pi+/- 1p W<1.4 theta<20deg";

fSettings = LoadSampleSettings(samplekey);

fSettings.SetDescription(descrip);
fSettings.SetTitle("MINERvA_CC1pip1p_XSec_1DTpi_nu");
fSettings.SetXTitle("T_{#pi} (MeV)");
fSettings.SetYTitle("d#sigma/dT_{#pi} (cm^{2}/MeV/nucleon)");
fSettings.SetEnuRange(0.0, 100.0);
fSettings.DefineAllowedTargets("C,H");
fSettings.DefineAllowedSpecies("numu");
fSettings.SetAllowedTypes("FIX,FREE,SHAPE/DIAG,FULL", "FIX/FULL");

std::string base = FitPar::GetDataBase();
fSettings.SetDataInput( base+"/MINERvA/CC1pi1p/Tpi/data.csv");
fSettings.SetCovarInput(base+"/MINERvA/CC1pi1p/Tpi/correlation.csv");

FinaliseSampleSettings();
```

Scaling Factor

- As mentioned last week NUISANCE requires the flux and event rate histograms to properly normalise events.

$$R(E_\nu) = \Phi(E_\nu) \times \sigma(E_\nu) \times T$$

N-Targets

Predicted rate given the flux

Flux

Total Xsec spline

- How does this work?

Scaling Factor Method

- **Method is as follows:**

1. Bin up the signal events in required distribution
2. Divide by **total** events to get a fractional spectrum
3. Multiply by Total Rate to get Rate for signal selection
4. Divide by flux integral, just like a normal cross-section.

- Binning signal handled by the event loop, the rest can be contained into a single scaling factor.

$$F = \frac{\int R(E_\nu)}{N \int_{E_{low}}^{E_{high}} \Phi(E_\nu)}$$

Scaling Factor

- Equivalent Scaling factor in NUISANCE is as follows

```
fScaleFactor = GetEventHistogram()->Integral("width") \
               / double(fNEvents)
               / TotalIntegratedFlux("width");
```

- The EventRate histogram is kept in units of

$$(10^{-38}\ \mathrm{cm}^2/\mathrm{nucleon})\ \mathrm{x}\ (\mathrm{Flux\text{-}Histogram\text{-}Units})$$

- So dividing the the flux integral away gets you

$$(10^{-38}\ \mathrm{cm}^2/\mathrm{nucleon})$$

- If you need something else (e.g. /neutron) include extra factors in the fScaleFactor term.

Scaling Factor

- We want just cm^2/nucleon so our factor is this.

```
// Scaling Setup -------------------------------------------------
// ScaleFactor setup for DiffXSec/cm2/Nucleon
fScaleFactor = GetEventHistogram()->Integral("width")
             * double(1E-38)
             / double(fNEvents)
             / TotalIntegratedFlux("width");
```

fNEvents also setup from the input handler during the FinalseSampleSettings() call.

GetEventHistogram() and GetFluxHistogram() return TH1D's from the InputHandler

TotalIntegratedFlux gets you the integral between your Enu cut limits.

Constructor

- So now we should have the following constructor

```
std::string descrip = "MINERvA_CC1pip1p_XSec_1DTpi_nu" \
                      "\n Target: CH" \
                      "\n Flux: MINERvA FHC numu" \
                      "\n Signal: CC1pi+/- 1p W<1.4 theta<20deg";

fSettings = LoadSampleSettings(samplekey);

fSettings.SetDescription(descrip);
fSettings.SetTitle("MINERvA_CC1pip1p_XSec_1DTpi_nu");
fSettings.SetXTitle("T_{#pi} (MeV)");
fSettings.SetYTitle("d#sigma/dT_{#pi} (cm^{2}/MeV/nucleon)");
fSettings.SetEnuRange(0.0, 100.0);
fSettings.DefineAllowedTargets("C,H");
fSettings.DefineAllowedSpecies("numu");
fSettings.SetAllowedTypes("FIX,FREE,SHAPE/DIAG,FULL", "FIX/FULL");

std::string base = FitPar::GetDataBase();
fSettings.SetDataInput( base+"/MINERvA/CC1pi1p/Tpi/data.csv");
fSettings.SetCovarInput(base+"/MINERvA/CC1pi1p/Tpi/correlation.csv");

FinaliseSampleSettings();

// Scaling Setup ---------------------------------------------------
fScaleFactor = GetEventHistogram()->Integral("width")
             * double(1E-38)
             / double(fNEvents)
             / TotalIntegratedFlux("width");
```

Data Setup

- To setup the data we now need to initialise the histogram

```
Measurement1D::SetDataFromTextFile( std::string path_to_file );
```

- Lots of helper functions to support this. Look inside src/FitBase/Measurement1D.h for some.

```
virtual void SetDataFromTextFile(std::string datafile);
virtual void SetDataFromRootFile(std::string inhistfile, std::string histname);
virtual void SetPoissonErrors();

virtual void ScaleData(double scale);
virtual void ScaleDataErrors(double scale);
```

Covariance Setup

- Similar functions to load in correlation matrices.

```
Measurement1D::SetCorrelationFromTextFile( std::string path_to_file );
```

- **In general I recommend using the SetCorrelation functions.**
- The other SetCovar functions assume units of 1E-76 on the entries, so it gets a bit confusing sometimes. Working to fix this

```
virtual void SetCovarFromTextFile(std::string covfile, int dim = -1);
virtual void SetCovarFromMultipleTextFiles(std::string covfiles, int dim = -1);
virtual void SetCovarFromRootFile(std::string covfile, std::string histname="");

virtual void SetCorrelationFromTextFile(std::string covfile, int dim = -1);
virtual void SetCorrelationFromRootFile(std::string covfile, std::string histname="");

virtual void ScaleCovar(double scale);
```

Data/Covar Setup

- Data text file and correlations are already in the format NUISANCE assumes and we have set them in fSampleSettings.
- Just pass the SetDataFromTextFile function our data input and it will automatically load in the data histogram for us.
- SetCorrelation function will take the errors on our newly created data histogram and multiply them by the correlations to make a covariance.

```
// Plot Setup ------------------------------------------------------
SetDataFromTextFile( fSettings.GetDataInput() );
SetCorrelationFromTextFile( fSettings.GetCovarInput() );
```

FinaliseMeasurement

- At the end of each constructor we have to call FinaliseMeasurement.

```
 // Final setup  ----------------------------------------------
FinaliseMeasurement();
```

- This runs a load of extra checks on what has been setup.
- **Importantly it clones the data histogram in `fDataHist` and uses it create the MC Histogram `fMCHist`.**

Our Constructor

```
MINERvA_CC1pip1p_XSec_1DTpi_nu::MINERvA_CC1pip1p_XSec_1DTpi_nu(nuiskey samplekey) {
      // -----------------------------------------
      std::string descrip = "MINERvA_CC1pip1p_XSec_1DTpi_nu" \
                            "\n Target: CH" \
                            "\n Flux: MINERvA FHC numu" \
                            "\n Signal: CC1pi+/- 1p W<1.4 theta<20deg";
      fSettings = LoadSampleSettings(samplekey);
      fSettings.SetDescription(descrip);
      fSettings.SetTitle("MINERvA_CC1pip1p_XSec_1DTpi_nu");
      fSettings.SetXTitle("T_{#pi} (MeV)");
      fSettings.SetYTitle("d#sigma/dT_{#pi} (cm^{2}/MeV/nucleon)");
      fSettings.SetEnuRange(0.0, 100.0);
      fSettings.DefineAllowedTargets("C,H");
      fSettings.DefineAllowedSpecies("numu");
      fSettings.SetAllowedTypes("FIX,FREE,SHAPE/DIAG,FULL", "FIX/FULL");
      fSettings.SetDataInput( FitPar::GetDataBase() + "/MINERvA/CC1pip1p/Tpi/data.csv");
      fSettings.SetCovarInput( FitPar::GetDataBase() + "/MINERvA/CC1pip1p/Tpi/correlation.csv");
      FinaliseSampleSettings();
      // Scaling Setup ---------------------------------------------------
      fScaleFactor = GetEventHistogram()->Integral("width")
                   * double(1E-38)
                   / double(fNEvents)
                   / TotalIntegratedFlux("width");
      // Plot Setup ------------------------------------------------------
      SetDataFromTextFile( fSettings.GetDataInput() );
      SetCorrelationFromTextFile( fSettings.GetCovarInput() );
       // Final setup  ---------------------------------------------------
      FinaliseMeasurement();
}
```

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	👍
Constructor	👍
FillEventVariables	
IsSignal	
SampleList.cxx	

- Constructor should now be finished.
- If you want, you can go back to your build folder and make install to check it builds.

FillEventVariables

- Now have to define what we actually want to fill in our histogram.
- FitEvent class is passed to FillEventVariables, so the definition should pull variables out of that event class.
- Data distribution is the final state pion kinetic energy so we need a way to calculate that from the particle stack.
- **Warning: FillEventVariables is called for all events not just signal, so we need to first explicitly check we have a pion.**

FitEvent

- FitEvent class is a common interface that contains a list of all particles (initial, intermediate, final) in the event.
- **For most analyses this is all you should need.**
- It is possible to get the generator event if needed, but you can figure that out on your own as it is strongly discouraged!
- To look at possible helper functions, lets look inside src/InputHandler/FitEvent.h

FitEvent Functions

- Number of functions in the header. Mostly self explanatory.

```
…
inline bool HasFSNuMuon        (void) const { return HasFSParticle(14);   };
…
inline int NumFSProton        (void) const { return NumFSParticle(2212); };
…
inline FitParticle* GetHMFSPiPlus     (void) { return GetHMFSParticle(211);  };
…
```

- Few things to point out :
  - HM means highest momentum.
  - FS means Final State
  - IS means Initial State

- We want the only pion in the event, so will need the functions

```
FitEvent::NumFSParticle(int pdg)
FitEvent::GetHMFSParticle(int pdg)
```

FitParticle

- FitEvent can return FitParticle object pointers.
- If a particle is not found by the helper functions (e.g. GetHMFS) then you will get a NULL pointer.

- **FitParticle objects simply contain:**
  - fP : 4-momentum,
  - fPID : Pdg code
  - fStatus : Status of the particle.
- Most important status codes are (kInitialState, kFinalState)

- So to get the TLorentzVector we can do

```
TLorentzVector ppi = event->GetHMFSParticle(211)->fP;
```

Particle Groups

- Some of the helper functions also let you pass in multiple PDG's and it will group them.

- e.g. Get Total FS Proton **and FS** Neutron count

```
int nuclPDG[] = {2212, 2112};
int nnucl = event->NumFSParticle(nuclPDG);
```

- e.g. Get Highest Momentum FS Charged Pion

```
int piPDG[] = {211, -211};
FitParticle* hmfspion = event->GetHMFSParticle(piPDG);
```

Initial Checks

- Can't calculate Tpi if there is no pion, so first we need to check that the pion exists.

- If it doesn't we just return the function.

```
void MINERvA_CC1pip_XSec_1DTpi_nu::FillEventVariables(FitEvent *event) {

     int piPDG[] = {211, -211};

     // Check we have a pion to get TPi
     if (event->NumFSParticle(piPDG) == 0) return;

     // Assign X
     fXVar = 0.0;

     return;
};
```

Assigning X

- Can calculate Tpi from the LorentzVector

```
TLorentzVector Ppip = event->GetHMFSParticle(piPDG)->fP;
double Tpi = Ppip.E() - Ppip.Mag();
```

- Once we have Tpi we need to explicitly tell the sample to treat Tpi as the X-variable to be binned.

```
// Assign Tpi to be binned in X
fXVar = Tpi;
```

- Similar `fYVar` must also be set for 2D samples.

FillEventVariables

- Using the functions shown before our function is now:

```
void MINERvA_CC1pip_XSec_1DTpi_nu::FillEventVariables(FitEvent *event) {

     int piPDG[] = {211, -211};

     // Check we have a pion to get TPi
     if (event->NumFSParticle(piPDG) == 0) return;

     // Get Tpi Calculated
     TLorentzVector Ppip = event->GetHMFSParticle(piPDG)->fP;
     double Tpi = Ppip.E() - Ppip.Mag();

     // Assign Tpi to be binned in X
     fXVar = Tpi;

     return;
};
```

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	👍
Constructor	👍
FillEventVariables	👍
IsSignal	
SampleList.cxx	

- Over half way through!
- If you want, you can an go back to your build folder and make install to check it builds.

IsSignal

- Signal definition should use similar functions to those used in FillEventVariables.
- Extract everything you need from the FitEvent.

- Aim of the function is to return False if an event doesn't pass our given signal selection.

#	Cut
1	Initial State Muon Neutrino
2	Final State Muon (CC)
3	Final State pi+ or pi-
4	>0 Final State Protons
5	Muon Theta Angle < 20 degrees
6	Wexp < 1.4 GeV

Standard Signals

- A few standard signal cuts already defined in src/Utils/SignalDef.cxx

```
bool isCCINC(FitEvent *event, int nuPDG, double EnuMin = 0, double EnuMax = 0);
bool isNCINC(FitEvent *event, int nuPDG, double EnuMin = 0, double EnuMax = 0);
bool isCC0pi(FitEvent *event, int nuPDG, double EnuMin = 0, double EnuMax = 0);
...
```

- Ones there are usually measurement independent so look there for things like CC0pi/CCINC/etc.

- Some experimental folders have their own too `src/MINERvA/MINERvA_SignalDef.cxx`

```
bool isCC1pip_MINERvA(FitEvent *event, double EnuMin, double EnuMax, bool isRestricted=false);
bool isCCNpip_MINERvA(FitEvent *event, double EnuMin, double EnuMax, bool isRestricted=false);
...
```

isCCINC

- SignalDef::isCCINC is relevant for a lot of CC samples

```
bool SignalDef::isCCINC(FitEvent *event,
                        int nuPDG,
                        double EnuMin,
                        double EnuMax)
```

- Applies a few cuts at once:
  - Is the event Charged Current
  - Is Neutrino the correct PDG
  - Is the Neutrino Energy within our Enu range

- **Handles cuts 1 and 2 in our selection ☺**

Enu ranges accesible in the class with these objects

```
if (!SignalDef::isCCINC(event, 14, EnuMin, EnuMax)) return false;
```

Number of Particles

- NumFSParticle functions returns final state particle counts.
- **Code below covers cuts 2, 3, and 4**

```
if (event->NumFSLeptons() != 1) return false;

int piPDG[] = {211, -211};
if (event->NumFSParticle(piPDG) != 1) return false;

if (event->NumFSProton() < 1) return false;
```

Kinematic Cuts : Theta

- Try to put cuts that require calculation at the end after the simpler multiplicity cuts to improve efficiency.
- **Need to calculate theta and W to cut on them.**
- Phase space cut just requires muon and neutrino. So can grab their vectors and calculate the angle quite easily.
- **Following code handles cut 5**

```
TLorentzVector pnu = event->GetHMISParticle(14)->fP;
TLorentzVector pmu = event->GetHMFSParticle(13)->fP;

double th_nu_mu = pmu.Vect().Angle(pnu.Vect()) * 180. / M_PI;
if (th_nu_mu >= 20) return false;
```

Kinematic Cuts : W

- Some cuts may be more complicated, but can usually be defined from initial and final state particles.
- **If they require extra generator information, they are suspcious!**

- Functions to calculate some other kinematic quantities can be found inside src/Utils/FitUtils.cxx

```
double Wrec(TLorentzVector pnu, TLorentzVector pmu);
double th(TLorentzVector part, TLorentzVector part2);
double Q2QErec(double pl, double costh, double binding, bool neutrin);
double MpPi(TLorentzVector pp, TLorentzVector ppi);
```

- **FitUtils::Wrec calculates us the experimental W for the event which we can use in for cut 6**

```
double hadMass = FitUtils::Wrec(pnu, pmu);
if (hadMass > 1400.0) return false;
```

A final Signal Cut

- Piecing all that together gives us:

```
bool MINERvA_CC1pip1p_XSec_1DTpi_nu::isSignal(FitEvent *event){

// First, make sure it's CCINC
if (!SignalDef::isCCINC(event, 14, EnuMin, EnuMax)) return false;

// Check only one FS lepton
if (event->NumFSLeptons() != 1) return false;

// Check for only one pi+ or pi-
int piPDG[] = {211, -211};
if (event->NumFSParticle(piPDG) != 1) return false;

// Require at least one proton
if (event->NumFSProton() < 1) return false;

// Restricted angle theta_mu < 20 degrees
TLorentzVector pnu = event->GetHMISParticle(14)->fP;
TLorentzVector pmu = event->GetHMFSParticle(13)->fP;
double th_nu_mu = FitUtils::th(pmu, pnu) * 180. / M_PI;
if (th_nu_mu >= 20) return false;

// W experimental < 1400.0
double hadMass = FitUtils::Wrec(pnu, pmu);
if (hadMass > 1400.0) return false;

}
```

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	👍
Constructor	👍
FillEventVariables	👍
IsSignal	👍
SampleList.cxx	

- If you want, you can go back to your build folder and make install to check it builds.

FCN

- Now that our sample is finished and builds successfully we have to add it the SampleList.cxx file.

- First include our header file at the top of SampleList.cxx

```
#include "SampleList.h"
#include "MINERvA_CC1pip1p_XSec_1DTpi_nu.h"
```

- Then slot in another ugly if statement entry like follows

```
} else if (!name.compare("MINERvA_CC1pip1p_XSec_1DTpi_nu")){
   return (new MINERvA_CC1pip1p_XSec_1DTpi_nu(samplekey));
```

- Remember to make sure your name doesn't conflict with another samples name!

Checklist

Requirement	Finished
Class File	👍
CMakeLists.txt	👍
Constructor	👍
FillEventVariables	👍
IsSignal	👍
SampleList.cxx	👍

- Now we definitely have to go back to our build folder and run "make install" again to build the comparisons application.

Ready to run

- MC files can be found here (non-MINERvA people see backup)

```
$ ls /minerva/data/users/jstowell/NUISTUTORIAL/MC/
```

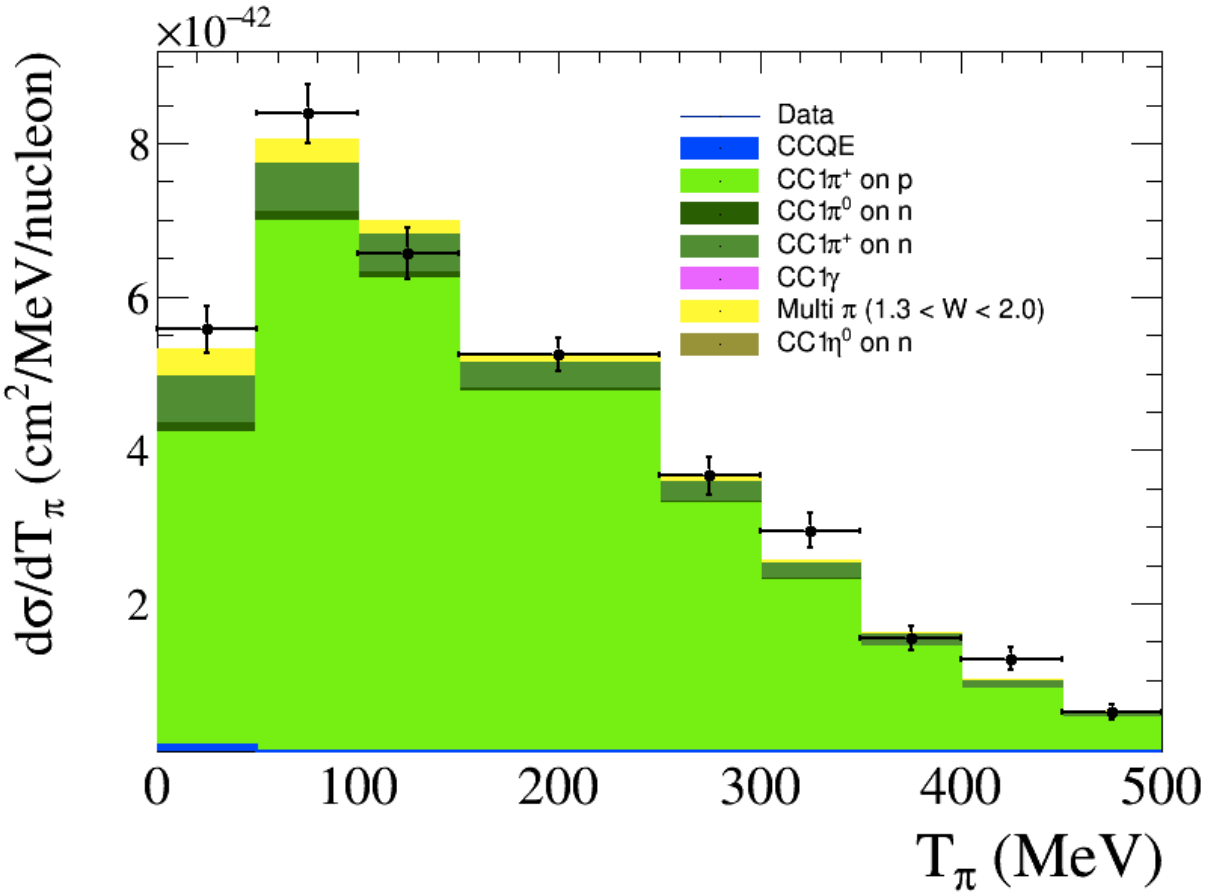
- Edit a card file so the sample name corresponds to whatever we put in the ugly if statement in the SampleList.cxx file.

```
<nuisance>
    <sample name="MINERvA_CC1pip1p_XSec_1DTpi_nu"
            input="GENIE:genie/gntp.Default.MINERvA_fhc_numu.CH.2500000.root"
</nuisance>
```

- Then run it as normal.

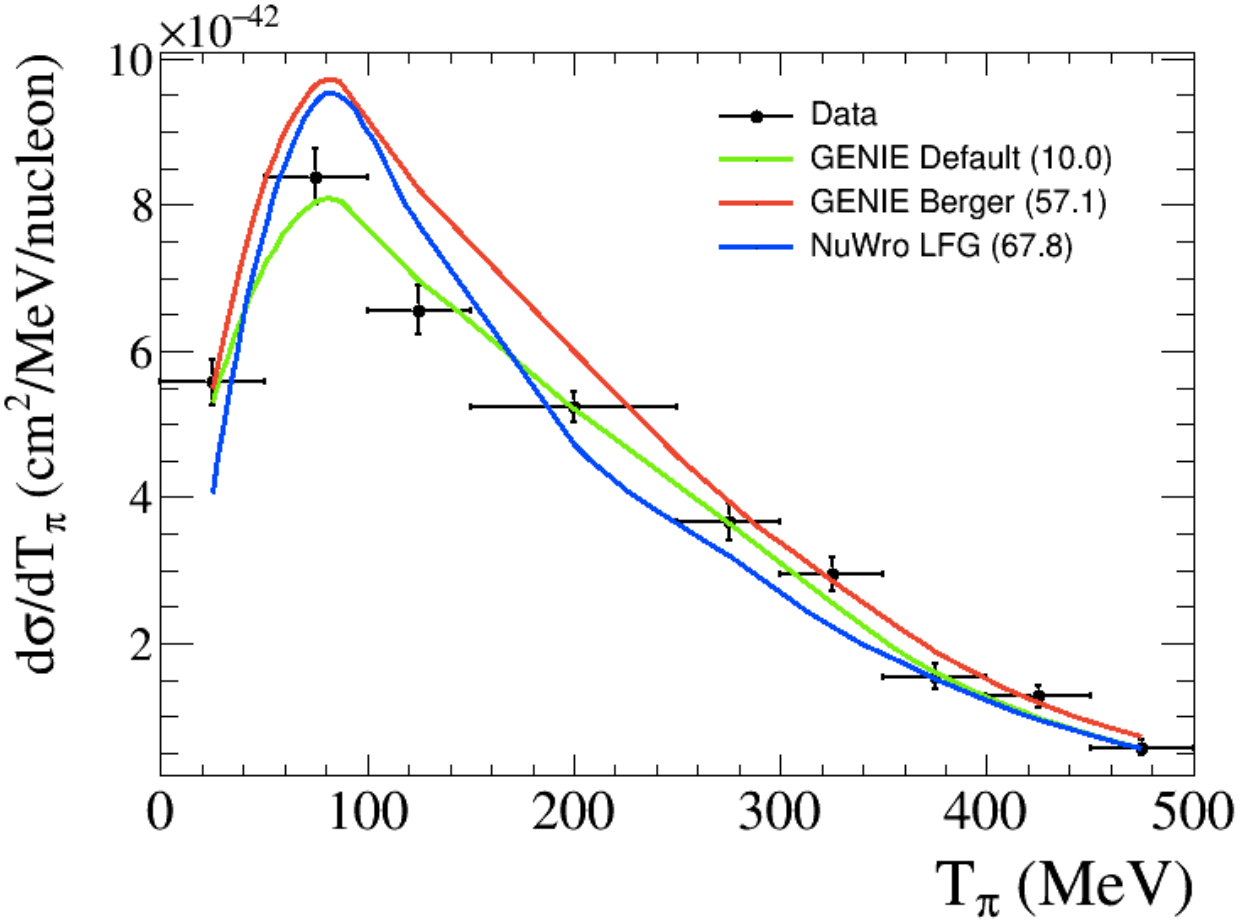
Final MC

- Should produce you a nice MC plot similar to the one below.



Generator Comparisons

- Class should work with any generator NUISANCE supports



Usual Issues

- **Nuiscomp says no sample found:**
  - Make sure you have put the correct name in your cardfile and you rebuilt nuisance after you edited SampleList.cxx
- **My data histograms are empty (or data file is Zombie):**
  - Did you make sure to put the data files in the database?
- **My Likelihood calculations are crazy:**
  - Check the covariance is in the correct units and has the same dimensions as the data.
  - This can also be an issue with the covariance itself...

Usual Issues

- **My MC Histograms are empty:**
  - First add debug logging statements to IsSignal to check that some events make it to the return true stage.
  - If none make it to this stage then your signal definitions could be incorrect.

```
...
if (hadMass > 1400.0) return false;

// DEBUGGING
ERR(WRN) << "Found a good signal event!" << std::endl;

return true;
}
```

Usual Issues

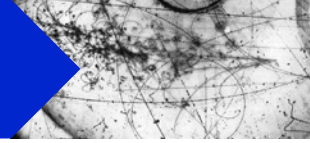
- **My MC Histograms are empty:**
  - Second add debug statements to FillEventVariables to check fXVar is set correctly.
  - This usually happens if fXVar is not in the correct units (e.g. GeV instead of MeV)

```
// Assign Tpi to be binned in X
fXVar = Tpi;

// LOGGING
ERR(WRN) << "fXVar has been set to : " << fXVar << std::endl;
```

Summary

- Hopefully if you have reached this point you got a working sample.
- There are further examples inside the other folders in the nuisance_tutorial repo, alongside various examples throughout the code.
- If you get to a point where you have a finalized signal selection consider putting it into nuisance to make lots of nice MC plots.
- We want as much data as possible, so email us if you ever get stuck trying to add a new sample!



Backup

Download MC Files

- Can download MC files by running script inside git repo

```
$ cd nuisance_tutorial/mc_files/
$ source downloadfiles.sh ALL
```

Multiple Samples

```
<nuisance>
  <!-- List of Samples -->
  <sample name="MINERvA_CC1pip_XSec_1DTpi_nu"
          input="GENIE:genie/gntp.Default.MINERvA_fhc_numu.CH.2500000.root" />
  <sample name="MINERvA_CC1pip_XSec_1Dth_nu"
          input="GENIE:genie/gntp.Default.MINERvA_fhc_numu.CH.2500000.root" />
</nuisance>
```

- NUISANCE reads events from disk, and then distributes them to relevant sample classes.
- Minimal extra overhead when loading a number of different distributions or datasets from one MC file.

- Just add an extra sample xml entry for every dataset you care about and NUISANCE will load them all at once.

Multiple Sample Output

- Run this joint sample in a similar fashion.

```
$ nuiscomp -c samplecc1pip.xml -o samplecc1pip.root -n 100000
```

- Likelihoods for both samples added uncorrelated, to form a joint total likelihood for the comparison.

```
[LOG Minmzr]:- Getting likelihoods...                    : -2logL
[LOG Minmzr]:- -> MINERvA_CC1pip_XSec_1DTpi_nu           : 44.6792/7
[LOG Minmzr]:- -> MINERvA_CC1pip_XSec_1Dth_nu            : 260.352/13
[LOG Fitter]: Likelihood for JointFCN:                    305.031
```

- Two sets of histograms also now contained in the output file.

gevgen_nuisance

- Custom gevgen app can be used to generate GENIE events specially for NUISANCE.
- Automatically saves required histograms into output file.
- Can also run with both combined targets and combined beams (i.e. nue+nuebar)

```
gevgen_nuisance [-h]
          [-r run#]
           -n nev
           -e energy (or energy range)
         -p neutrino_pdg
         -t target_pdg       [DIFFERENT TO GENIE's]
         -f flux_description [DIFFERENT TO GENIE's]
        [-o outfile_name]
        [-w]
        [--seed random_number_seed]
        [--cross-sections xml_file]
        [--event-generator-list list_name]
        [--message-thresholds xml_file]
        [--unphysical-event-mask mask]
        [--event-record-print-level level]
        [--mc-job-status-refresh-rate  rate]
        [--cache-file root_file]
```

gevgen_nuisance

- Options are similar to the standard gevgen app, but target and flux are different (and easier!)

```
gevgen_nuisance -f MINERvA_fhc_numu -t CH  <other arguments>
```

- Only works with GENIE 2.12 and later!

- To build this application, build NUISANCE with the following flags

```
cmake -DUSE_GENIE=1 -DBUILD_GEVGEN=1
```

gevgen_nuisance (2)

- Possible to generate events with the standard gevgen application and "prepare" them for NUISANCE if needed.

- Example generate MINERvA CH events and prepare them.

```
gevgen -f minerva_flux.root,numu -e 0.0,100.0
                -t 1000060120[0.9231],1000010010[0.0769]
                -r 1 -n 2500000 --cross-sections gxspl.gz
```

- Once the sample is made, need to prepare it using our PrepareGENIE application (note the target def is different!)

```
PrepareGENIE -i gntp.ghep.1.root -f minerva_flux.root,numu
             -t 1000060120,1000010010
```

Target is not fractional! For CH2 use
`1000060120,1000010010,1000010010`

Have to pass in the same flux again too..

PrepareNuWro

- NuWro events generated with nuwro-reweight (our special branch) automatically saves the information needed.

- Otherwise, there is another PrepareNuWro app.

```
[USAGE]: PrepareNuwro [-h]
                      [-f]
                      [-F <FluxRootFile>,<FluxHistName>]
                      [-o output.root]
                      inputfile.root [file2.root ...]

-h : Print this message.
-f : Pass -f argument to '$ hadd' invocation.
-F : Read input flux from input descriptor.
-o : Write full output to a new file.
```

Configs

- NUISANCE keeps a global configuration list accessible throughout the code.
- Defaults kept in `$NUISANCE/parameters/config.xml`
- These can be overriden at run time in the card file or on the command line
- Card File

```
<config NAME="OVERRIDE_VALUE" />
```

- Command Line (all applications support –q)

```
nuiscomp -c cardfile.xml -o output.root -q NAME=OVERRIDE_VALUE
```