LArSystematics:
A systematic shift framework for LArSoft

Luke Pickering, K. McFarland, K. Mahn, K. Bays,
D. Ruterbories, C. Wret

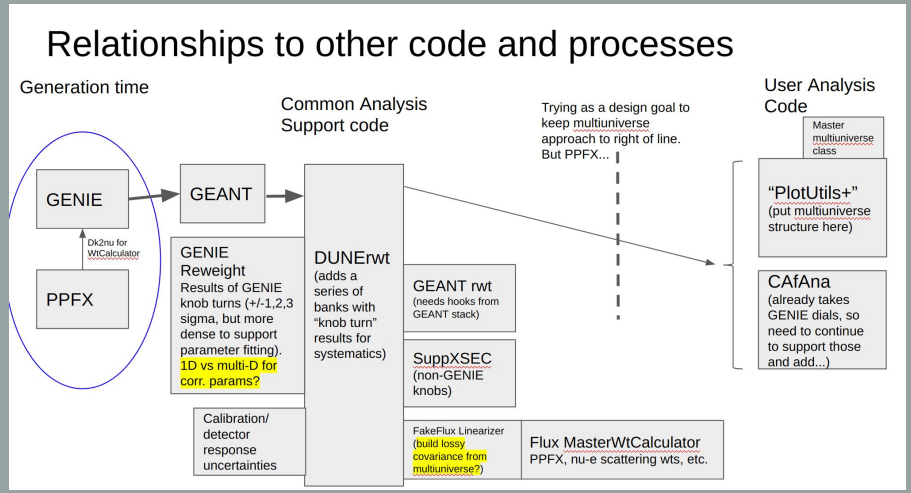
LArSoft Coordination Meeting
2018-07-17

MICHIGAN STATE UNIVERSITY

DEEP UNDERGROUND NEUTRINO EXPERIMENT

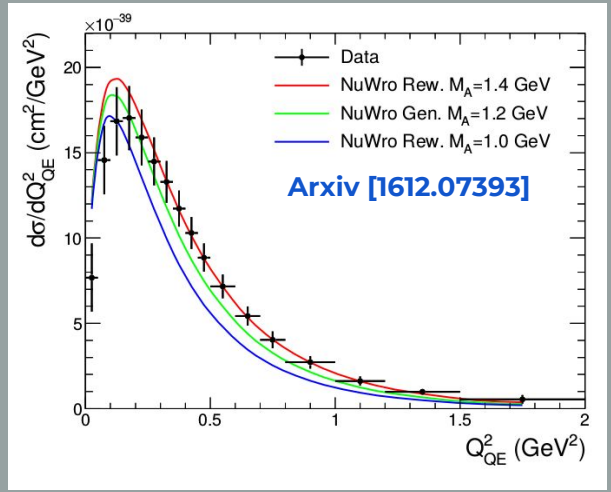
The Original Charge

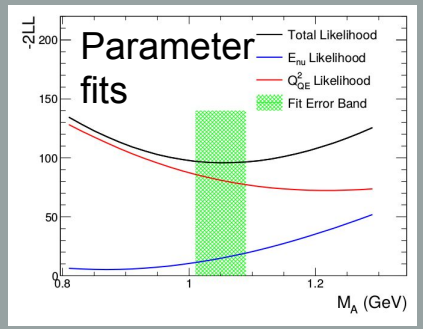
- Build/adapt neutrino interaction systematic propagation software for use in DUNE TDR sensitivity studies.
- Initial experience from MINERvA and T2K, since roped in NOvA experience too!
- Needs to be used by both the near detector analysis (Currently `EDepSim`) and the far detector analysis (`LArSoft`).

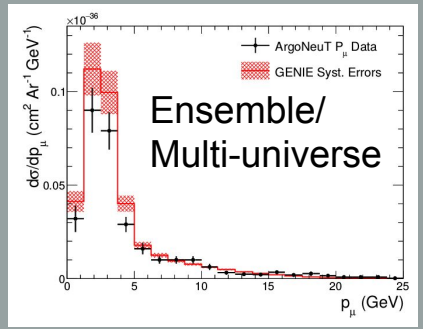


Error propagation

- General technique:
  - Systematic parameter, θ (*e.g.* MACCQE), gets varied, predictions of observations *respond*.
  - Does the new prediction look more or less like observations?
  - Build a distribution of goodness-of-fits-to-data for a range of parameter variations.
- Error propagation can be performed by mapping out the goodness-of-fit as a function of θ, or extracted from an ensemble of randomly thrown, varied parameter values.
- Parameters can be discrete or continuous.

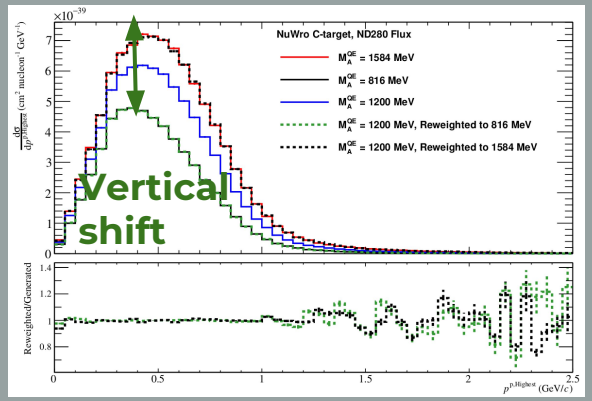


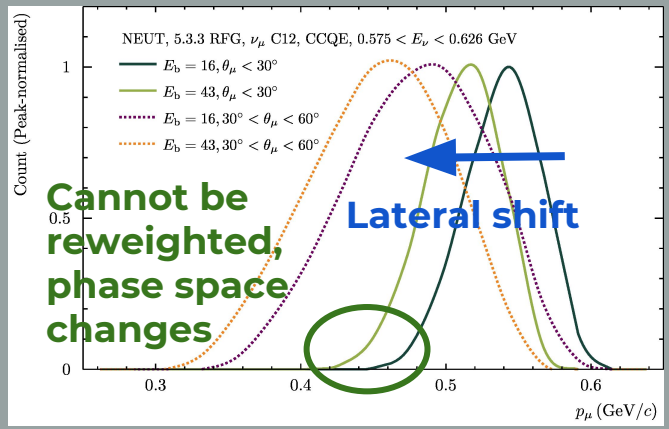




Parameter Variation Responses

- The response to a varied parameter may be determined in a number of ways.
- Full regeneration: Throw new events with a different physics model.
  - Very slow, requires re-run of det-sim, reco, ...
- **Exact reweight**: Calculate relative probability to have thrown the same event properties under a varied physics model.
  - Not all parameters are exactly r/w-able.
- **ad-hoc reweight**: Use full regeneration to calculate approximate weights as a function of some specific event properties.
  - Often not predictive in other kinematic projections.
- **Kinematic shifts**: Determine shifts in true or observed particle kinematics that characterise the change in physics model.
  - Inclusion post-reconstruction is approximate.





What Exists in LArSoft

- In `LArSim/EventWeight` there is a framework for producing EventWeights from ART events:
  - Produces `std::map<std::string, std::vector<double>>`
  - Map key corresponds to parameter name.
- LArSim Producer module doesn't use dynamic plugin framework so cannot instantiate WeightCalculators in experiment-specific codebases.
  - Uboonecode has producer module that allows compile-time linking of MicroBooNE-specific weighters.
- Semantics issues with 'weight' included in package/module/type names.
- I did not want to alter interfaces actively in-use by MicroBooNE analyzers.

The (Re-)Design Goals

- Basic unit of systematic error propagation: parameter variation ➡ response.
- **Key goal --** Flexibility of response use:
  - ‘Vertical’ (*e.g.* xsec weight) and ‘lateral’ (*e.g.* FS lepton momentum shift) responses
  - Support Multi-universe/systematic throw paradigm, but not require it.
  - Provide tools for building parameterized response functions: Splines, fit polynomials, …
- **Key goal --** Try not to force when responses should be calculated:
  - Can run at production time, or analyzers can run on their selected events.
  - This is free in the ART event framework.
- **Key goal --** Keep scope of code as wide as possible (and no wider):
  - Try to provide an extensible solution, but don’t get bogged down trying to solve the general problem.
  - Should be used for: Flux, Interaction, and GEANT4-level uncertainties.
  - Could be used for: Calibrations, detector systematics.

The ISystProvider

- Responses are calculated by implementations of the ISystProvider ABC, declares something like:
  - `std::unique_ptr<EventResponse_t> GetResponse(art::event const &)=0;`
- Must be run-time configurable to calculate and stash deterministic event responses:
  - `void Configure(fhicl::ParameterSet const &)=0;`
- Must provide information about the number and details of systematic response parameters that it provides:
  - `SystMetaData GetMetaData();`

The EventResponse

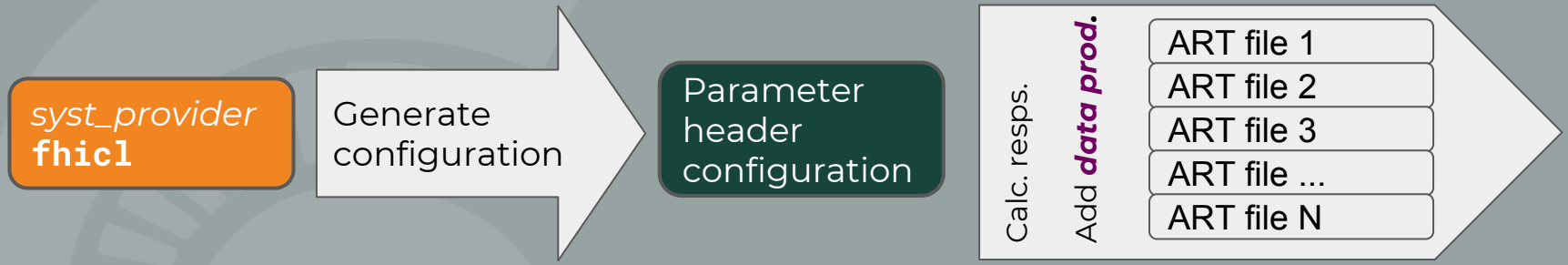
- The data product used in `LArSystematics` is very similar:
  - `std::vector< struct { paramId_t, std::vector<double> } >`
  - `paramId_t` is an `unsigned int` typedef.
- Outer `std::vector` contains '*event unit*'s:
  - Generalized sub-unit of an `art::event`: often will correspond to true neutrino interactions within a beam spill.
  - However, could be MIP-like tracks in an event responding to some reweight of GEANT rescattering parameters.
- Inner `std::vector` holds all calculated event responses to a given parameter identified by `paramId_t`.
- Correct use of responses requires extra parameter metadata:
  - Parameter name, Parameter central value, varied values, vertical/lateral shift, ...

The Metadata: Parameter Header

- As responses are generalized, need to have tools for interpreting them.
- Event responses **must** be fully interpretable from the '*Parameter Header*' configuration.
  - Format allows most to be generically interpreted.
  - For some applications, the parameter name might give the consumer a hint: *e.g.* **`EbFSMuMomShift`**
  - Arbitrary string options can also be used to pass information to interpreters: *e.g.* **PolyOrder4** might signify that responses correspond to fitted response function coefficients rather than vertical/lateral event shifts.

```cpp
namespace larsyst {
struct SystParamHeader {
  std::string prettyName;
  size_t systParamId;

  bool isWeightSystematicVariation;

  std::array<double, 2> paramValidityRange;

  bool isSplineable;
  bool isRandomlyThrown;

  std::vector<double> paramVariations;

  std::vector<std::string> opts;
};
} // namespace larsyst
```

High-level Overview



- A set of `ISystProviders` is configured by specific `FHiCL` (**user written**)
- Configurations are expanded to a common ***'parameter header'*** format by each `ISystProvider`:
  - Used to deterministically add relevant event response ***data product***s to each event.
  - Currently the generated metadata is **`FHiCL`**, but can be a bit clumsy for large sets of parameter throws -- however, it is not designed to be frequently human-written.
- This configuration is then given to ART jobs that calculate and stash responses to all configured parameter variations for each input file.

A Concrete Example: NuSystematics

- Currently one dependent package containing `ISystProviders` that handles neutrino interaction systematic uncertainties:
  - Depends on `nutools` for `simb` -> `GHep` conversion.
  - Links to `GENIE`
- At the time of writing there are three `ISystProviders`:
  - `GENIEReWeight`: GENIE ReWeight wrapper, similar to the one in nutools but to avoid more needless levels of abstraction, it interactions with GENIE directly.
  - `MKEnuq0q3Weighting`: Provides template weighting for single pion production events to move between GENIE default model and the updated MK model.
  - `MINERvAq0q3Weighting`: R. Gran RPA and Nieves 2p2h enhancement tunes and systematic uncertainties.
- All declared as ART tools that are instantiated through `art::make_tool` -- no experiment-specific Producer Modules required.
- Expect one or two more to follow in build up to DUNE TDR.

Generating Configurations

- A user will generally write simple, SystProvider-specific configuration `FHiCL`.
- The `ISystProvider` implementation must know how to translate that into common Parameter Header metadata that can be used to re-configure an instance at response-calculation time.
- *e.g.* GENIEReWeight configuring an MaCCQE spline generation job.
  - "`(-2_2:0.25)`" is translated to parameter values at -2σ to 2σ at 0.25σ steps by `GENIEReWeight_tool`

```
GENIEFFCCQEProvider_dipole_spline: {
  tool_type: "GENIEReWeight"
  uniqueName: "dipole_spline"

  MaCCQETweakDefinition: "(-2_2:0.25)"

}
syst_providers: [GENIEFFCCQEProvider_dipole_spline]
```

```
generated_systematic_provider_configuration: {
  GENIEReWeight_dipole_spline: {
    MaCCQE: {
      centralParamValue: 0
      isSplineable: true
      paramVariations: [
        -2, -1.75, -1.5, -1.25, -1, -7.5e-1, -5e-1, -2.5e-1, 0, 2.5e-1,
        5e-1, 7.5e-1, 1, 1.25, 1.5, 1.75, 2
      ]
      prettyName: "MaCCQE"
      systParamId: 0
    }
    parameterHeaders: [
      "MaCCQE"
    ]
    tool_type: "GENIEReWeight"
    uniqueName: "dipole_spline"
  }
  syst_providers: [
    "GENIEReWeight_dipole_spline"
  ]
}
```

Generated from above by `GenerateSystProviderConfig`

```
-c bla.fcl
```

Running ART Jobs

- The generated configuration can be given to the `LArSystematics` Producer module to instantiate and configure the required `ISystProviders`.
- Responses data products are then calculated.
- The configuration is human-readable/editable, but it is expected that standard sets of responses to calculate will be provided with the `ISystProviders`.

```
#include "ExampleGeneratedMetaData_GENIEFFCCQE.fcl"
ExampleLArSystProducer: {
  module_type: "LArSystEventResponse"

  generated_systematic_provider_configuration:
  @local::generated_systematic_provider_configuration
}
```

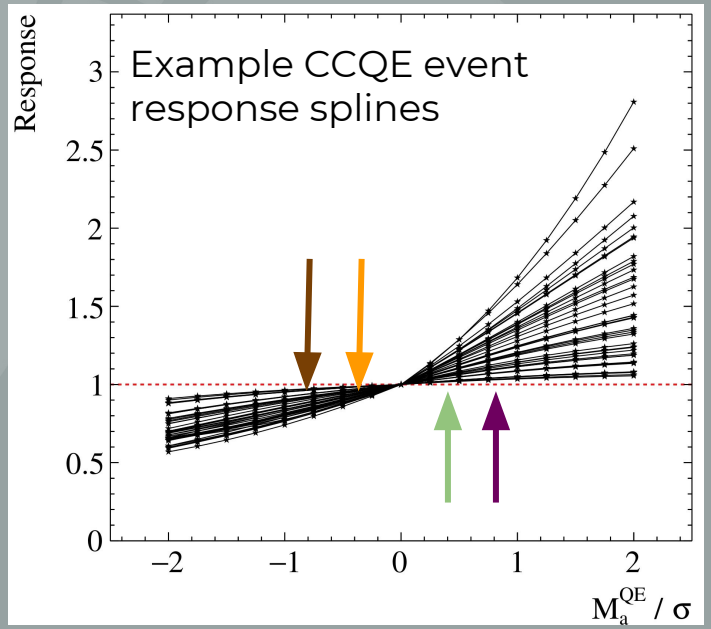
- An MD5 hash of the configuration FHiCL is used as the data product instance name to ensure that the correct metadata is used to interpret responses.

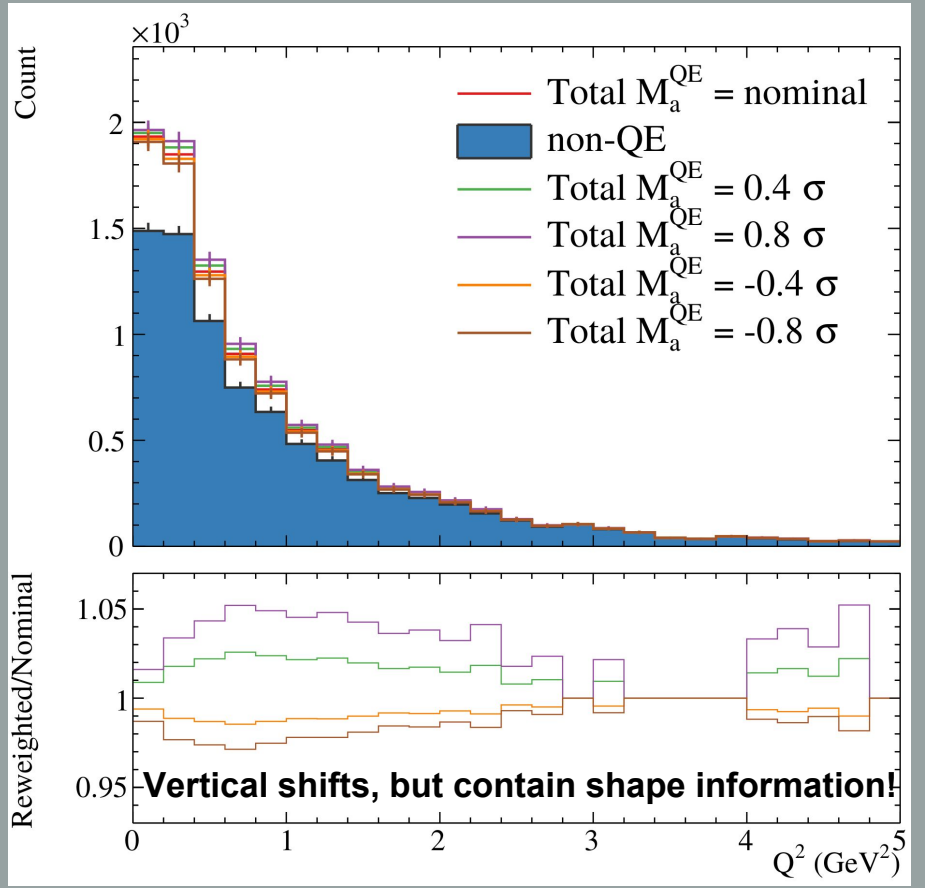
Interpreting Responses: Pre-fab tools

- The generated FHiCL configuration contains all the information required to interpret the data product responses.
  - The response interpretation could be written directly into an analysis to take advantage of any efficiency improvements, but generic tools are provided.
- Provided tools depend only on the LArSystematic interface headers and are completely detached from ART.
  - `ParameterHeaderHelper`: Provides methods to interact with objectified Parameter Header metadata and instantiate and evaluate `TSpline3` instances.
  - `EventSplineCacheHelper`: Template for caching analysis events in memory alongside the calculated parameter responses:
    - Provides various helper methods: *e.g.* to get total event weights given sets of parameter values.

Interpreting responses Example: GENIE ReWeight

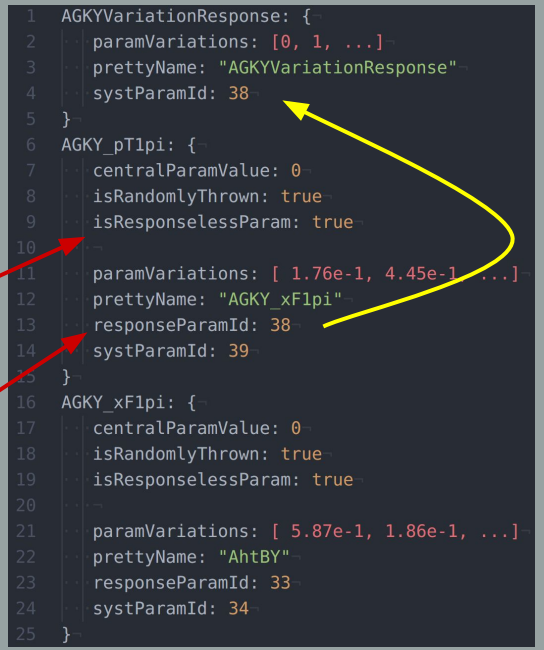
- Can spline calculated responses to allow approximated continuous parameter evaluation between limits.





Dependent Parameters

- Some response calculations depend on multiple parameters and cannot be factorized to 1D response functions.
  - e.g. Neutrino-induced single pion production models depend on 2-3+ parameters.
- Two ways forward:
  - Ignore correlations, treat as *effective* parameters and use N * 1D response parameters.
  - Only allow simultaneous ‘multi-sim’ throws of sets of parameters:
    - Introduce ‘Responseless parameter’: Not all parameters induce responses themselves but instead specify varied parameter values and a ‘response parameter’ identifier.
    - E.g. MARes, CA5 in SPP model respond through SPPResponseParameter.



Outside of ART

- For the DUNE TDR oscillation sensitivity studies, there is not time to translate the near detector simulation and analysis effectively to ART.
  - But must be able to interface with most of the same systematic uncertainties.
- Main dependencies of SystProvider configuration and response calculation code:
  - `FHiCL` language bindings
  - `art::make_tool`
  - `GENIE`
- Current solutions:
  - Standard FHiCL build depends on CET: Implemented a new, mostly standard compliant dependency-free FHiCL binding
    - If the reference implementation could be made dependency free, that would be great, but I couldn't expect it on the timescale that we needed this.
  - Only NuSystematics is to be used: hardcoded 'dynamic' instantiator written
    - `if(ps.get<std::string>("toolname") == "a" ) {} elseif…`
- Uses `#ifndef NOART` to hide ART-dependent code blocks, builds with a simple, standalone CMake-configured build.
- No duplication of response calculation code.

Still //TODO

- More inline documentation needed, but most important interfaces are documented ready for doxygen autodocs.
- User documentation needed.
- It has been written quite quickly:
  - Have some code tidy-up tasks this week, but expect more as the code gets more use over the next few months.
- Moving from my GitHub to shiny FNAL-hosted repositories.
- Standardized uncertainty fhicl generation.
- Extensive physics validation…
- (Some interface unit tests would be lovely…)

Summary

- Written framework for developing event-by-event response calculators to systematic parameters.
  - Intended to replace and expand functionality provided by LArSim/EventWeight.
- Responses are generic and interpreted through associated parameter metadata.
  - Tools are provided to aid interpretation by analysers.
- ART Producer module allows simple and automated response calculation, but physics code can also be built outside of ART:
  - So far, most analysis use has been in ARTless mode, expect ramp-up of ARTful use over the next month.
- Work originally begun for DUNE TDR, but the hope is that experience from designing and using similar tools on T2K, MINERvA, and NOvA can inform a LArSoft-level toolkit for future use.

Thanks for listening