nusystematics and systematicstools

Luke Pickering

2021-10-11

MICHIGAN STATE UNIVERSITY

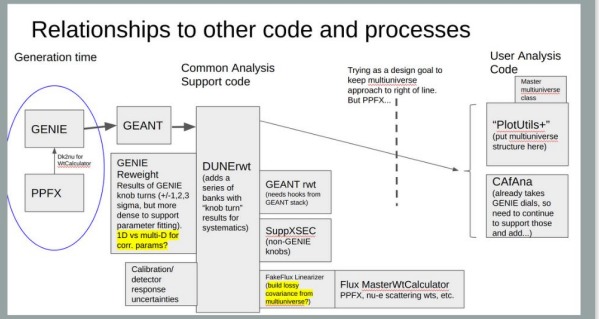
DUNE DEEP UNDERGROUND NEUTRINO EXPERIMENT

Motivation

- Charge of the Interaction Systematics group in 2018

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`).



MICHIGAN STATE UNIVERSITY

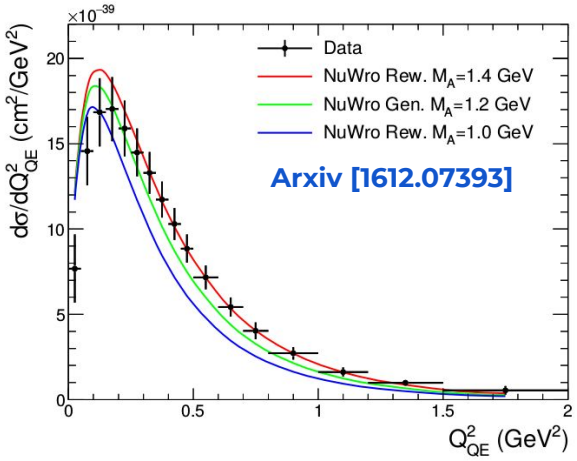
DEEP UNDERGROUND NEUTRINO EXPERIMENT

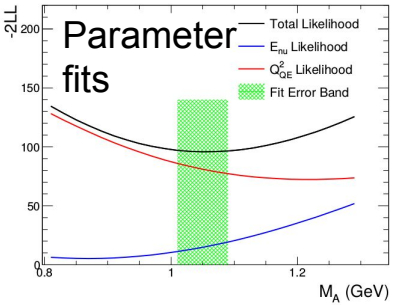
MICHIGAN STATE UNIVERSITY

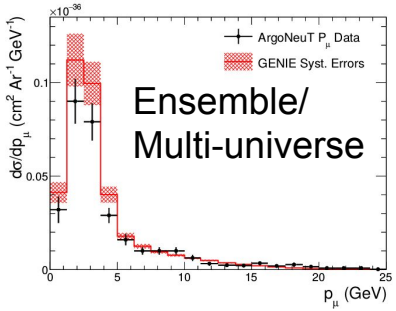
DEEP UNDERGROUND NEUTRINO EXPERIMENT

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 distribution of goodness-of-fits for a range of parameter variations.
- Extract errors by mapping out a goodness-of-fit as a function of θ, or from an ensemble of randomly thrown, varied parameter values.
- Parameters can be discrete or continuous.

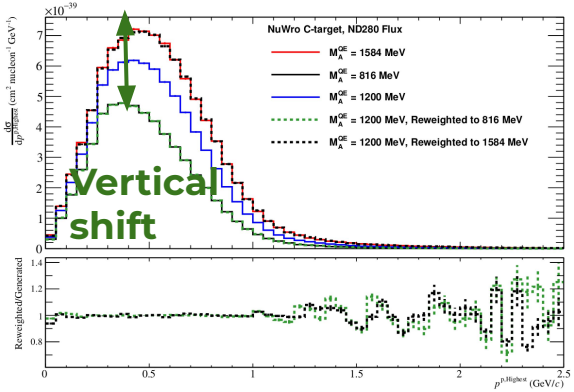


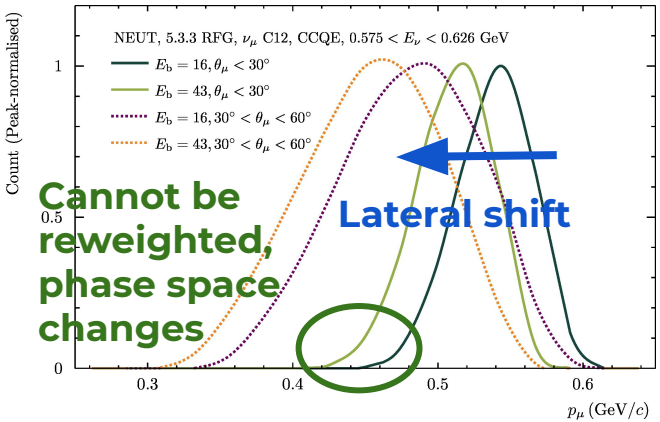




Parameter Variation Responses

- A **response** can be calculated 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 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 when we wanted a generic 'response' framework.

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 enforce it.
  - Provide tools for building parameterized response functions: Splines, polynomials, …
- **Key goal --** Do not 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.
  - Can be used for: Flux, Interaction, and GEANT4-level uncertainties.
  - Could be used for: Calibrations, detector systematics.

Tool Overview

- Two new packages were written to meet the charge and design goals
  - `systematicstools`: a generic systematic parameter framework providing facilities for parameter metadata interrogation and a plugin architecture for event weighters/variers
    - No physics whatsoever.
  - `nusystematics`: a package built on top of `systematicstools` that provides an interface to GENIE ReWeight as well as custom systematic event weighting implementations
- Live in https://github.com/LArSoft
- Dependencies:
  - `ROOT`
  - Can be built with or without dependence on `ART`
  - `nusystematics` depends on GENIE+ReWeight
- Other:
  - Can interface with NUISANCE for prediction/error comparisons with published xsec data
  - Can be built against GENIE v2 or v3:
    - GENIE v3 interface needs validation

Some (too many) Details

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

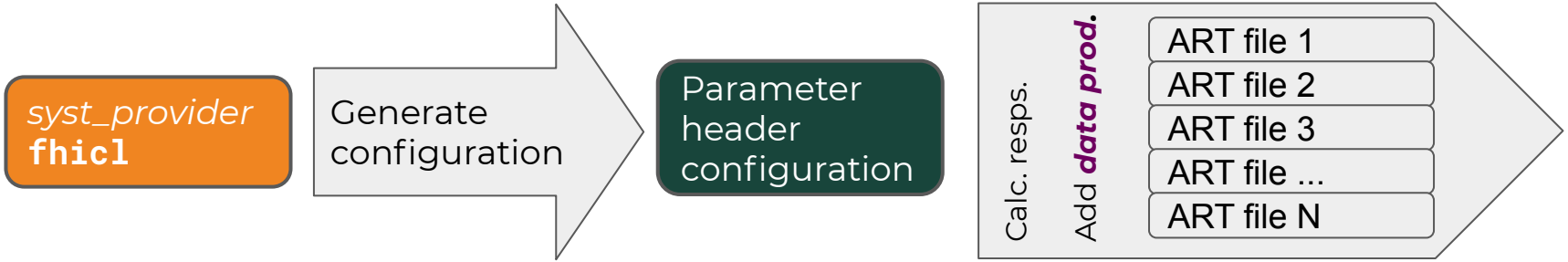
- The data product used in **nusystematics** is very similar:
  - `std::vector< struct { paramId_t, std::vector<double> } >`
  - `paramId_t` is an `unsigned int`
- Outer `std::vector` contains '*event unit*'s:
  - Generalized sub-unit of an `art::event`: often will correspond to a single true neutrino interactions within an event trigger.
  - 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, so we 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 (**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 -2***σ*** to 2***σ*** at 0.25***σ*** 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 Gener

`-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
}
```

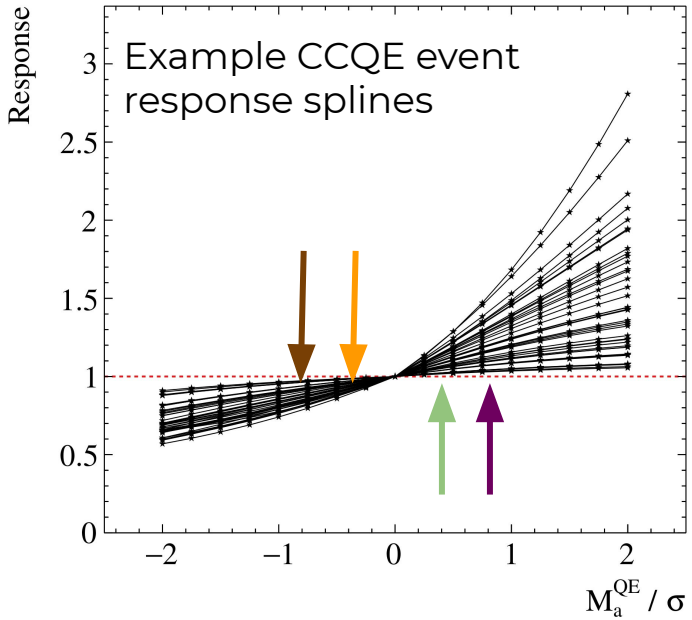
- An MD5 hash of the configuration FHiCL is used as the data product instance name to ensure that the correct metadata is used to interpret responses.

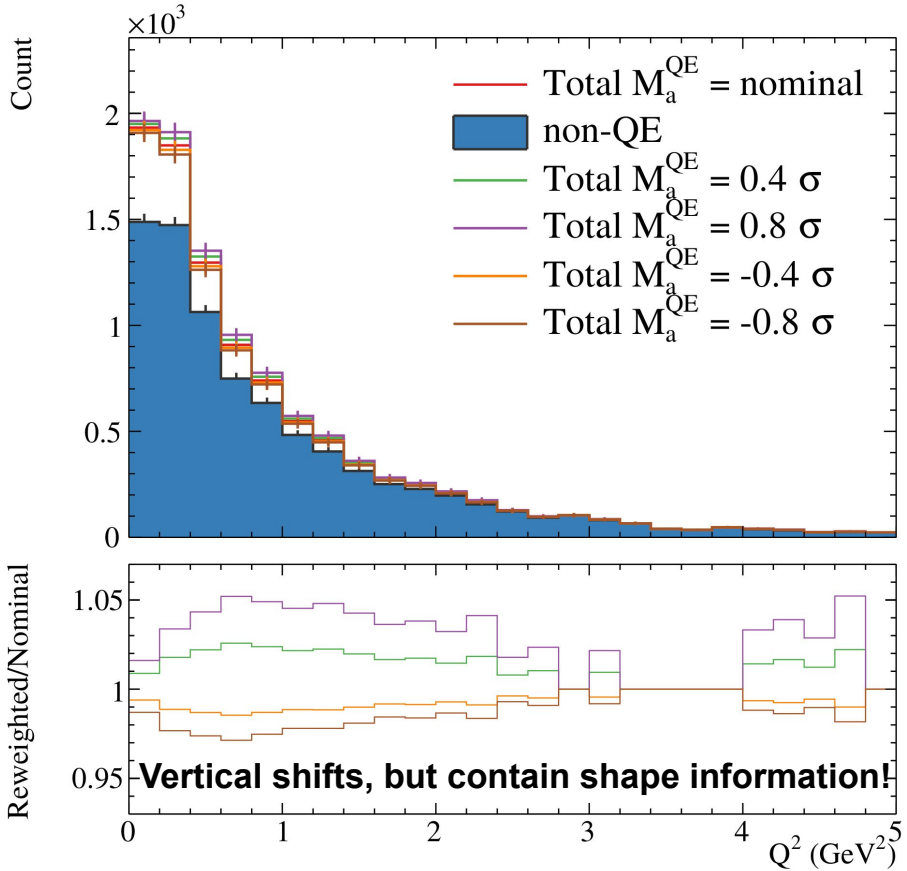
Interpreting Responses: Pre-fab tools

- 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

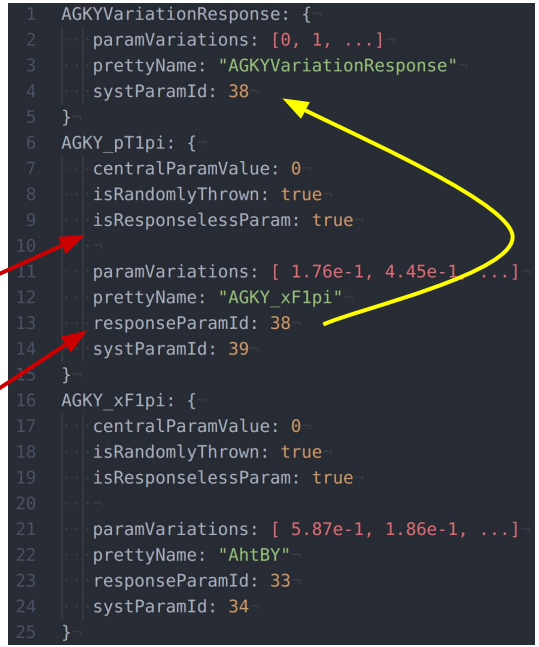
- Can spline calculated responses to allow approximated continuous parameter evaluation between limits.





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.



Parameter Headers

- Use parameter unique Id to look up parameter meta-data.
- This meta-data is used to configure systematic providers as well as interpret their responses.
- Generic format used for all providers and must be fully sufficient to interpret responses.
- *e.g.* spline-able MaCCQE responses, where:
  - responses[3] corresponds to MaCCQE=-1.5σ

```
generated_systematic_provider_configuration: {
  GENIEReweight_DUNE_TDR_Splines: {
    MaCCQE: {
      isSplineable: true
      paramVariations: [-3,-2.5,-2,-1.5,-1,-0.5,0,0.5,1,1.5,2,2.5,3]
      prettyName: MaCCQE
      systParamId: 0
    }
    instance_name: DUNE_TDR_Splines
    parameter_headers: [MaCCQE]
    tool_type: GENIEReweight
  }
  syst_providers: [
    GENIEReweight_DUNE_TDR_Splines
  ]
}
```

Tool Config

- However, it would be nice if an arbitrary systematic response provider could be configured in a less verbose format.
- The *Tool Config* is completely provider specific and is the FHiCL that should be written by end users.
- Each provider must be able to generate compliant *Parameter Headers* from input *Tool Config*.
- E.g. The *Tool Config* that generates the previous example.

```
GENIEReWeight_Tool_Config: {
    tool_type: "GENIEReWeight"
    instance_name: "DUNE_TDR_Splines"
    ################################ CCQE Parameters
    MaCCQE_variation_descriptor: "(-3,3,0.5)"
}

syst_providers: [GENIEReWeight_Tool_Config]
```

Thanks for listening