Using NNMFit for Cross Section Measurement

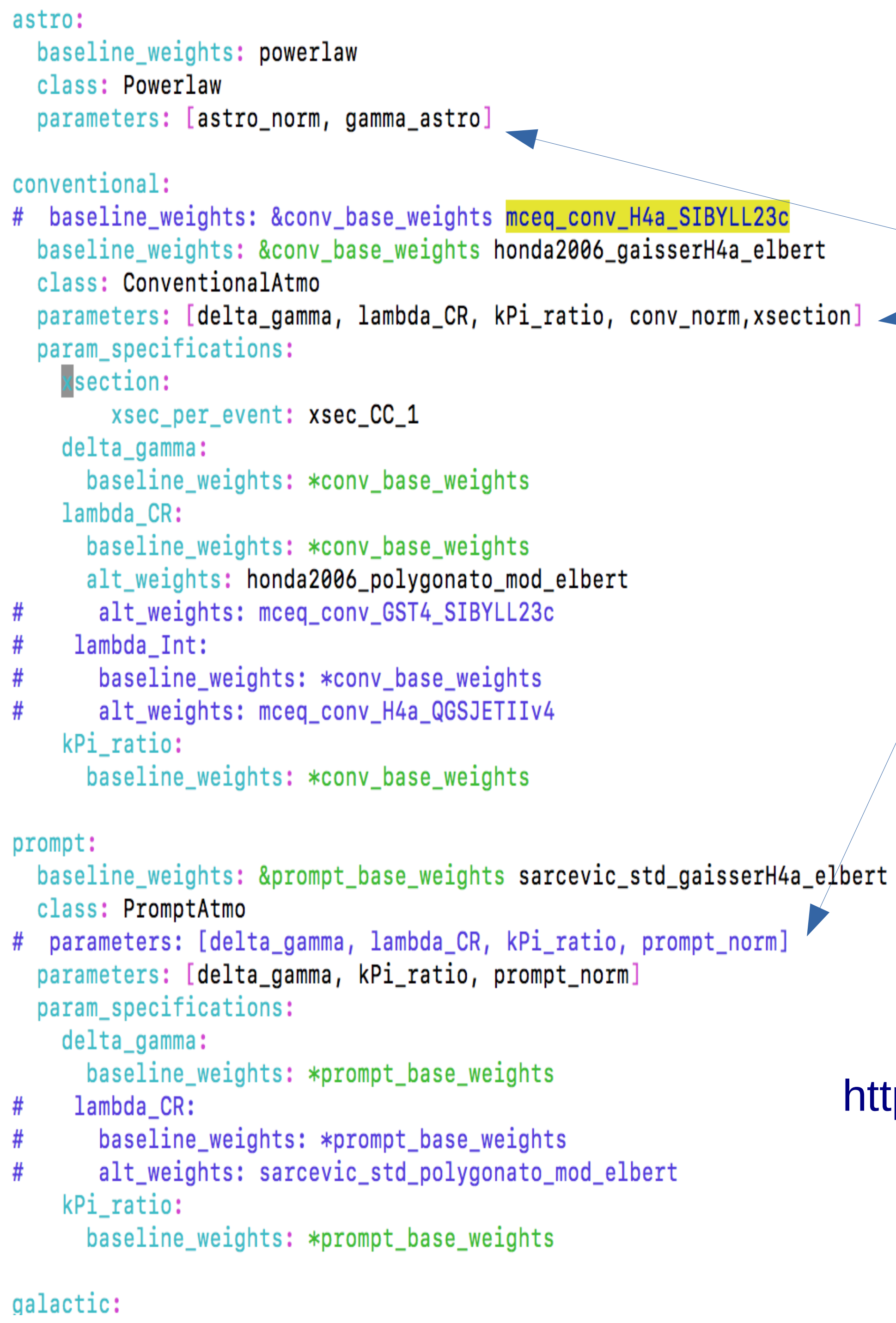
Sally Robertson

UC Berkeley / LBNL

Aim:

- Extend the analysis for 8 years using using updated tools for improved statistics and systematics.
  - Sandy's used one year
  - Using 8 year we have approximately 300,000 events. Rough calculation for statistical improvement using Sandy's results gives 8-10% improvement
  - Include energy bins, which will compare to the HESE cross section
  - Test both CC and NC cross section.
  - BSM; fixed NC with floating CC looking for a large factor of neutrinos interacting indicating another physical processes
- Have this analysis mirror the diffuse NuMu analysis as closely as possible.
  - NNMFit

https://docs.google.com/document/d/1ijpf5A5YgUmvIg8qYUKg5Ms1QEBJ56QcNloF5LU1wYM/edit?usp=sharing



```
astro:
  baseline_weights: powerlaw
  class: Powerlaw
  parameters: [astro_norm, gamma_astro]

conventional:
#  baseline_weights: &conv_base_weights mceq_conv_H4a_SIBYLL23c
  baseline_weights: &conv_base_weights honda2006_gaisserH4a_elbert
  class: ConventionalAtmo
  parameters: [delta_gamma, lambda_CR, kPi_ratio, conv_norm,xsection]
  param_specifications:
    xsection:
        xsec_per_event: xsec_CC_1
    delta_gamma:
      baseline_weights: *conv_base_weights
    lambda_CR:
      baseline_weights: *conv_base_weights
      alt_weights: honda2006_polygonato_mod_elbert
#      alt_weights: mceq_conv_GST4_SIBYLL23c
#    lambda_Int:
#      baseline_weights: *conv_base_weights
#      alt_weights: mceq_conv_H4a_QGSJETIIv4
    kPi_ratio:
      baseline_weights: *conv_base_weights

prompt:
  baseline_weights: &prompt_base_weights sarcevic_std_gaisserH4a_elbert
  class: PromptAtmo
#  parameters: [delta_gamma, lambda_CR, kPi_ratio, prompt_norm]
  parameters: [delta_gamma, kPi_ratio, prompt_norm]
  param_specifications:
    delta_gamma:
      baseline_weights: *prompt_base_weights
#    lambda_CR:
#      baseline_weights: *prompt_base_weights
#      alt_weights: sarcevic_std_polygonato_mod_elbert
    kPi_ratio:
      baseline_weights: *prompt_base_weights

galactic:
```

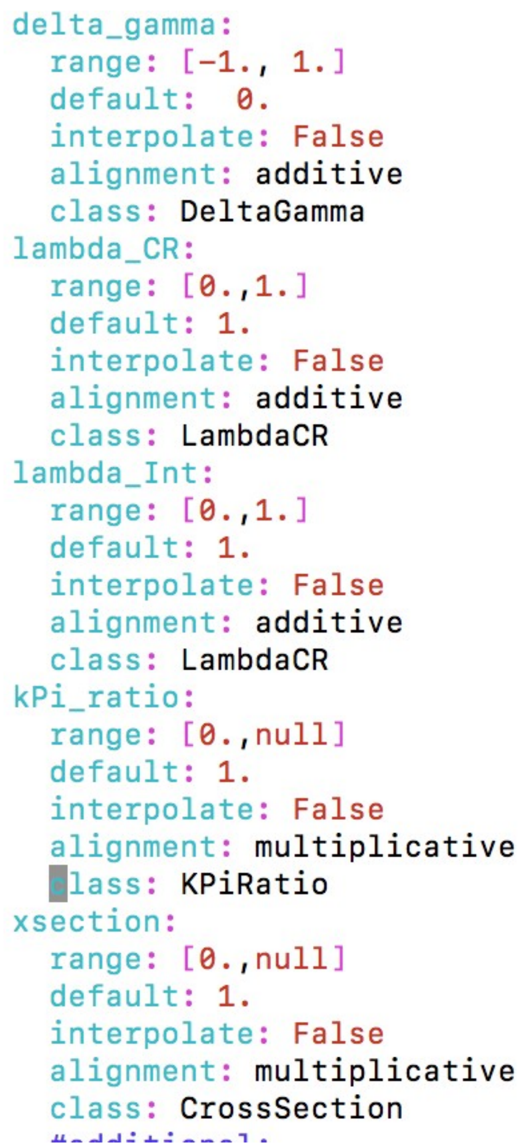
NNMFit has a file components.yaml which lets you add the parameters

Add xsection as a parameter in all three fluxes

In theory you can add any parameter
NNMFit evaluates using theano

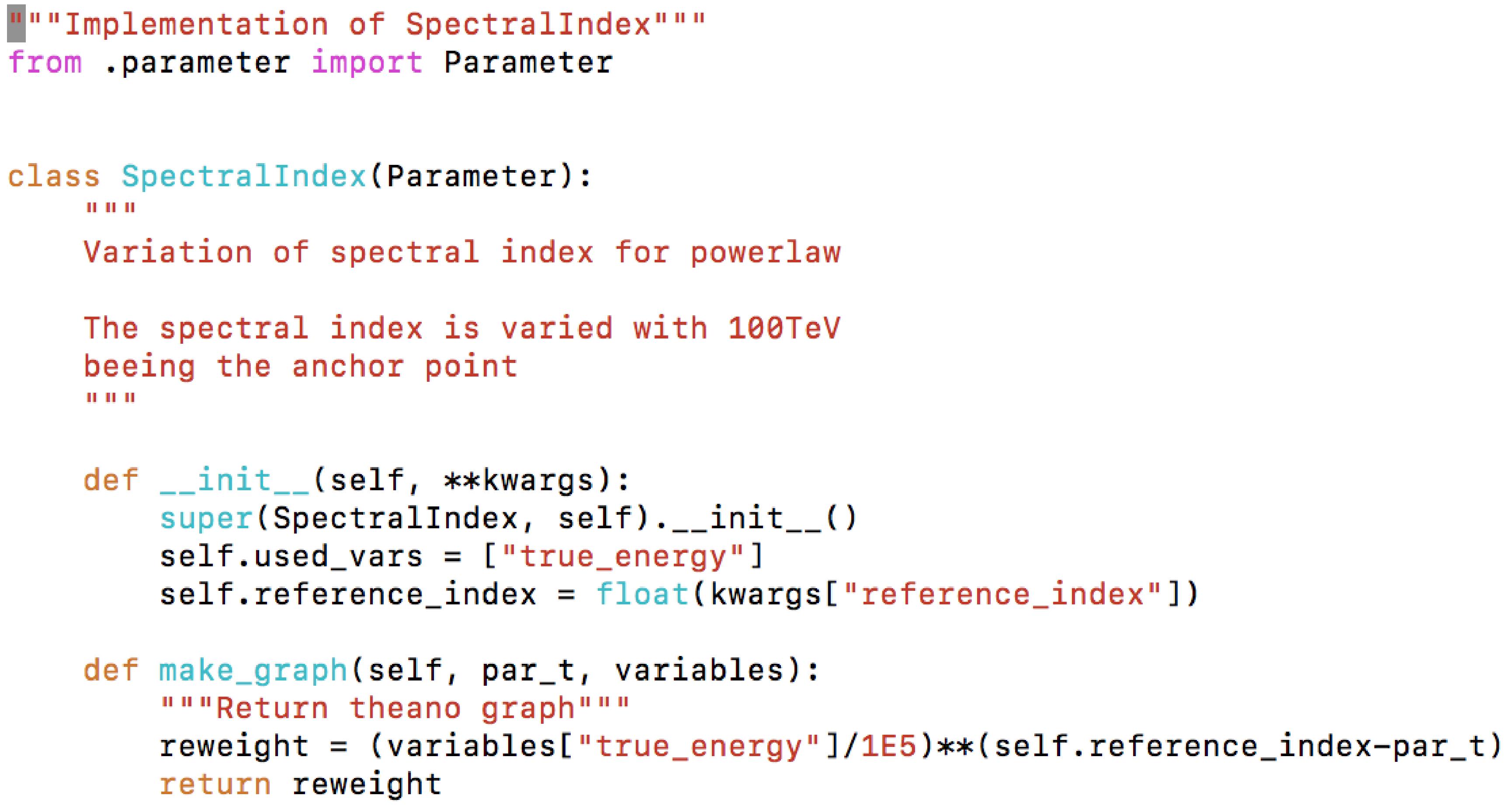
https://git.rwth-aachen.de/christian.haack/NNMFit

Add new parameter to parameter.yaml



```yaml
delta_gamma:
  range: [-1., 1.]
  default:  0.
  interpolate: False
  alignment: additive
  class: DeltaGamma
lambda_CR:
  range: [0.,1.]
  default: 1.
  interpolate: False
  alignment: additive
  class: LambdaCR
lambda_Int:
  range: [0.,1.]
  default: 1.
  interpolate: False
  alignment: additive
  class: LambdaCR
kPi_ratio:
  range: [0.,null]
  default: 1.
  interpolate: False
  alignment: multiplicative
  class: KPiRatio
xsection:
  range: [0.,null]
  default: 1.
  interpolate: False
  alignment: multiplicative
  class: CrossSection
  #additional:
```

Add a implementation to */fluxes/parameters/*



```python
"""Implementation of SpectralIndex"""
from .parameter import Parameter


class SpectralIndex(Parameter):
    """
    Variation of spectral index for powerlaw

    The spectral index is varied with 100TeV
    beeing the anchor point
    """

    def __init__(self, **kwargs):
        super(SpectralIndex, self).__init__()
        self.used_vars = ["true_energy"]
        self.reference_index = float(kwargs["reference_index"])

    def make_graph(self, par_t, variables):
        """Return theano graph"""
        reweight = (variables["true_energy"]/1E5)**(self.reference_index-par_t)
        return reweight
```

Every parameter needs to be defined for the theano graph, gradients needs to be defined

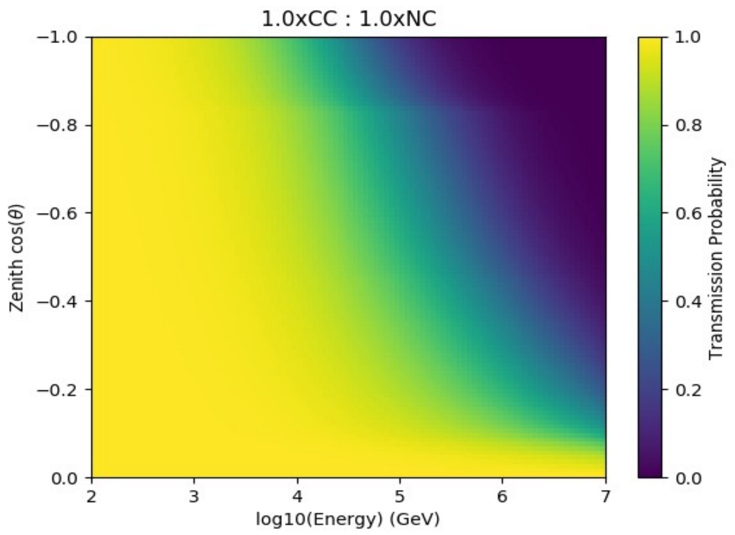
Either an expression
    or something that is recognized and differentiable.

make_datasets.py

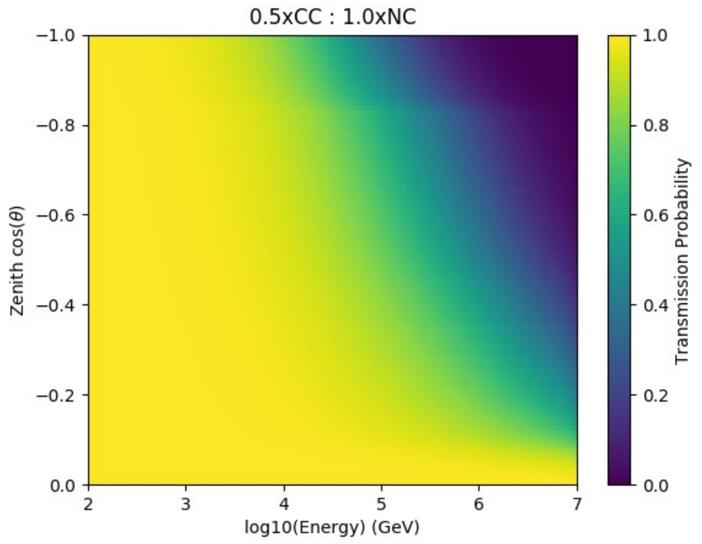
It collects all the events (MC/data) and attaches to each the relevant information

All inputs are defined in datasets.cfg

For cross section each event has a transmission probability given its energy and zenith angle

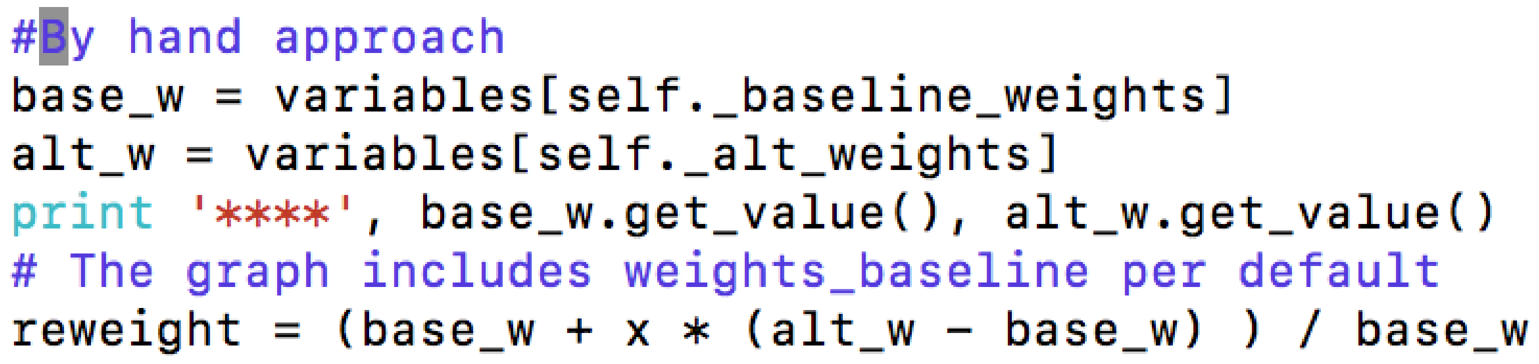


Baseline

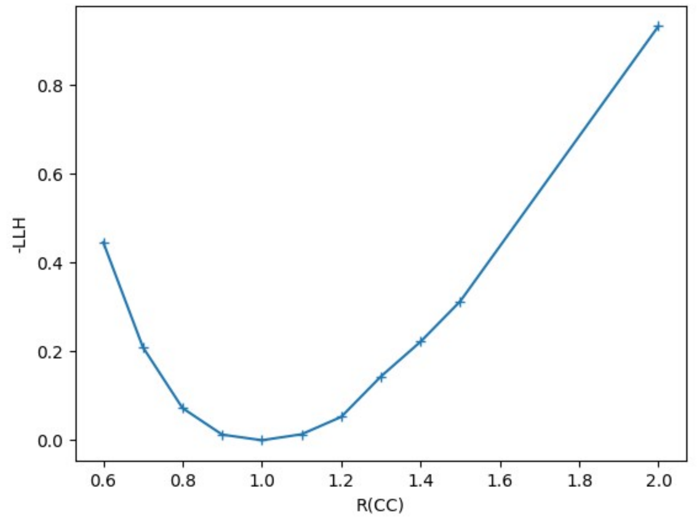


Alternative

- I can do a ‘manual’ fit for multiple of cross section.
- Fix x so it is not fitting the cross section, but comparing the baseline to alternate
- This is slow and not ideal

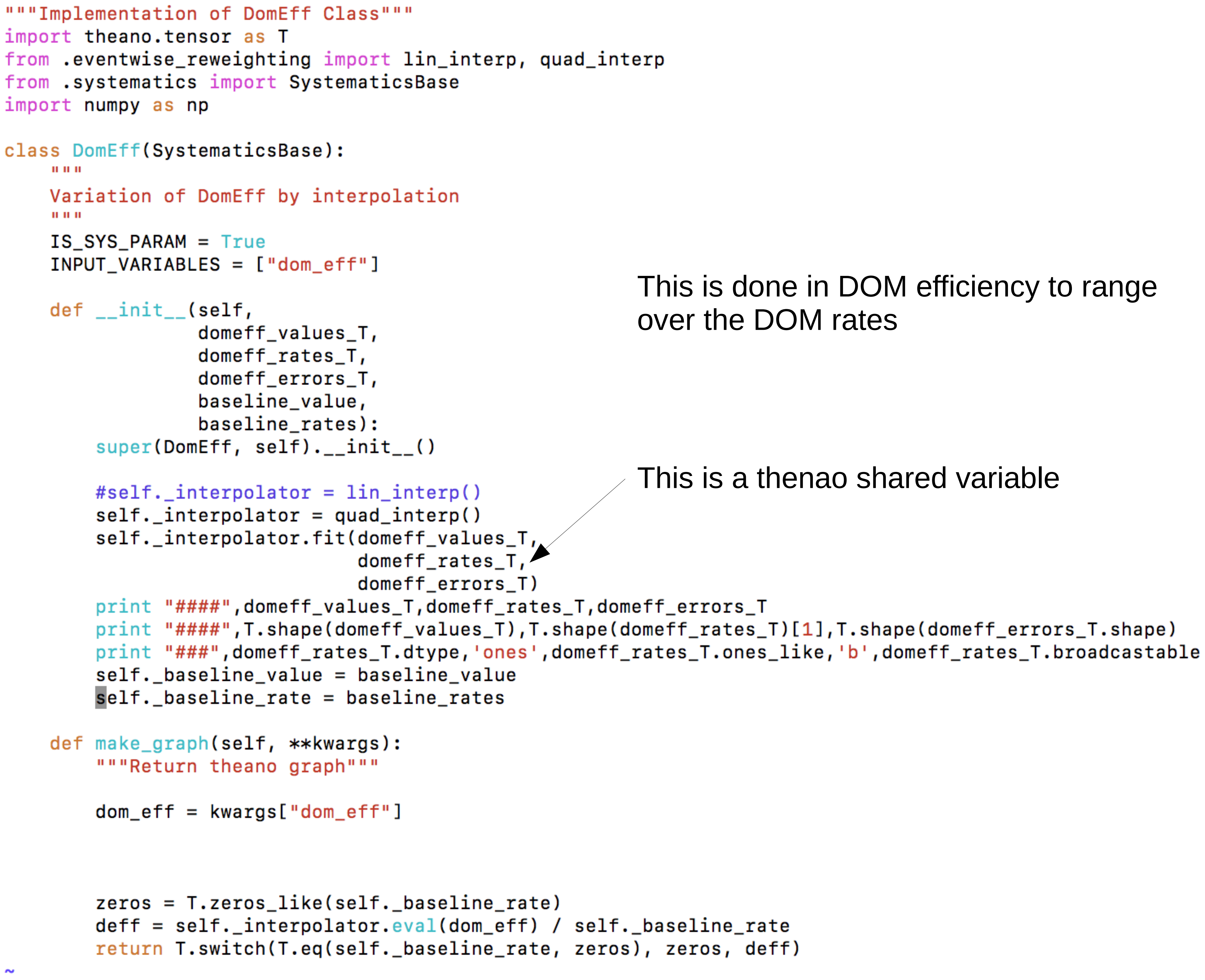


```
#By hand approach
base_w = variables[self._baseline_weights]
alt_w = variables[self._alt_weights]
print '****', base_w.get_value(), alt_w.get_value()
# The graph includes weights_baseline per default
reweight = (base_w + x * (alt_w - base_w) ) / base_w
```



Currently:

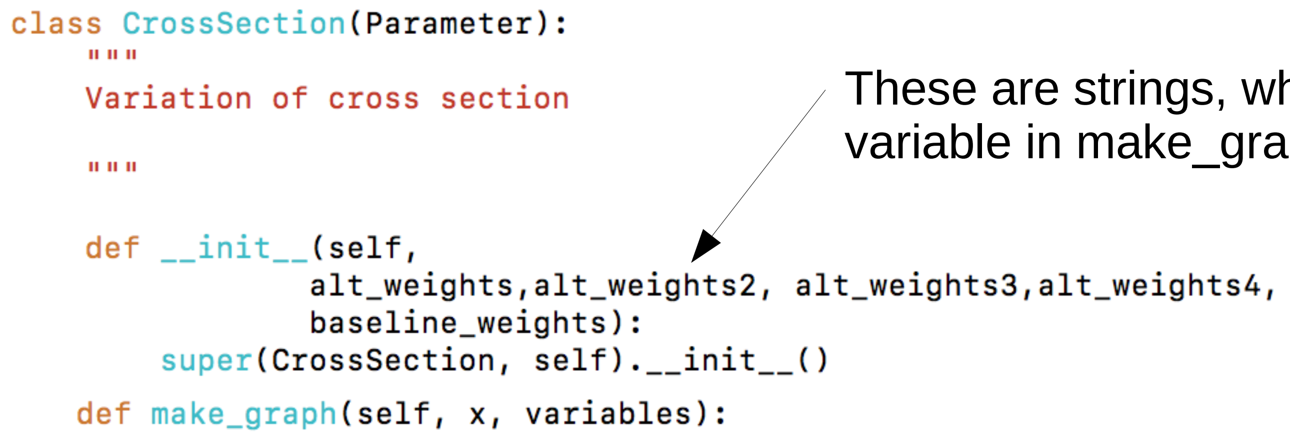
- NNM fit cannot do a fit with reference to a table (or spline) of values for Transmission Prob(zenith, energy, R(CC))
- I’m working on using interpolate inside the theano framework between different cross section propagations.



```
"""Implementation of DomEff Class"""
import theano.tensor as T
from .eventwise_reweighting import lin_interp, quad_interp
from .systematics import SystematicsBase
import numpy as np

class DomEff(SystematicsBase):
    """
    Variation of DomEff by interpolation
    """
    IS_SYS_PARAM = True
    INPUT_VARIABLES = ["dom_eff"]

    def __init__(self,
                 domeff_values_T,
                 domeff_rates_T,
                 domeff_errors_T,
                 baseline_value,
                 baseline_rates):
        super(DomEff, self).__init__()

        #self._interpolator = lin_interp()
        self._interpolator = quad_interp()
        self._interpolator.fit(domeff_values_T,
                               domeff_rates_T,
                               domeff_errors_T)
        print "####",domeff_values_T,domeff_rates_T,domeff_errors_T
        print "####",T.shape(domeff_values_T),T.shape(domeff_rates_T)[1],T.shape(domeff_errors_T.shape)
        print "###",domeff_rates_T.dtype,'ones',domeff_rates_T.ones_like,'b',domeff_rates_T.broadcastable
        self._baseline_value = baseline_value
        self._baseline_rate = baseline_rates

    def make_graph(self, **kwargs):
        """Return theano graph"""

        dom_eff = kwargs["dom_eff"]



        zeros = T.zeros_like(self._baseline_rate)
        deff = self._interpolator.eval(dom_eff) / self._baseline_rate
        return T.switch(T.eq(self._baseline_rate, zeros), zeros, deff)
~
```

This is done in DOM efficiency to range over the DOM rates

This is a thenao shared variable



```
class CrossSection(Parameter):
    """
    Variation of cross section

    """

    def __init__(self,
                alt_weights,alt_weights2, alt_weights3,alt_weights4,
                baseline_weights):
        super(CrossSection, self).__init__()

   def make_graph(self, x, variables):
```

These are strings, which are used to grab variable in make_graph

Trying to interpolate in here results in theano constantly unfolding

Cross section parameter is currently not working

Hackaton Topic

- Try to make interpolator idea work
  - Change how the cross section are read in
  - Edit interpolator

- Implement photospline in theano