How to do a diffuse fit

Austin Schneider

What is a diffuse fit?

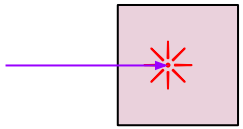
“Diffuse fit” is IceCube jargon for measuring the spectral properties of an isotropic astrophysical neutrino flux

“Isotropic” meaning that the neutrinos are equally likely to come from any direction

This scenario is not just a measurement we can make, but also corresponds to models where we cannot resolve individual sources of neutrinos

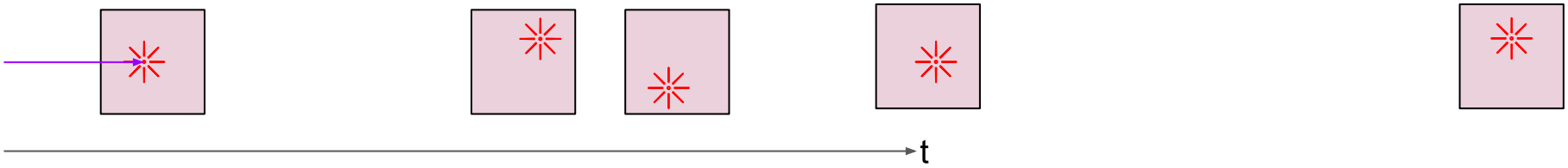
Statistics of a detector - Counting

Consider a detector that counts particles

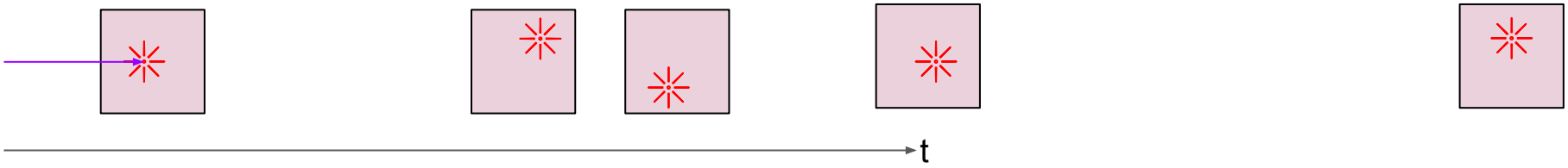


Every time a particle hits the detector we receive a signal

If we leave the detector on for a long time, we will receive many signals which we can count



Statistics of a detector - Poisson



The number of particles we detect in a time **t** is a random variable

This is a poisson process, so the count is poisson distributed

The probability of getting **k** counts is $$P(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Where **λ** is the number of expected counts in time **t**

Statistics of a detector - Likelihood

If we know the true mean expected number of counts **λ** then we can describe the probability of getting a certain outcome **k** when running the experiment $P(k|\lambda)$

However we often do not know the truth **λ** and only have the observation from our detector **k**

We can instead talk about the likelihood of truth **λ** given observation **k**

$$\mathcal{L}(\lambda|k) = P(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Same formula, different concept

Statistics of a detector - Maximum likelihood

Defining the likelihood allows us to make an estimate of the parameter \lambda by finding the most likely value of \lambda, “Maximum Likelihood”

$$\hat{\lambda} = \arg\max_{\lambda} \mathcal{L}(\lambda|k)$$

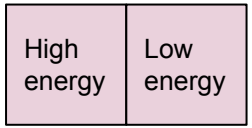
For the simple poisson case there is one easy solution

$$\hat{\lambda} = \arg\max_{\lambda} \mathcal{L}(\lambda|k) = \arg\max_{\lambda} \frac{\lambda^k e^{-\lambda}}{k!} = k$$

Statistics of a detector - A simple model

We can split our data into more than one bin

Consider 2 energy bins, we can count the data in each bin to get two measurements



Consider a flux model where the counts in bin 1 are 2x the counts in bin 2

We can construct a likelihood for this scenario that has a single parameter, in which the joint likelihood is a product of the likelihood for each bin

$$\lambda_1 = 2\lambda, \lambda_2 = \lambda \qquad \mathcal{L}(\lambda|\vec{k}) = \frac{(2\lambda)^{k_1} e^{-2\lambda}}{k_1!} \cdot \frac{(\lambda)^{k_2} e^{-\lambda}}{k_2!}$$

Statistics of a detector - General models

This approach works in general for any number of model parameters and any number of bins

$\mathcal{L}(\vec{\theta}|\vec{d})$ likelihood of model parameters $\vec{\theta}$ given data $\vec{d}$

Diffuse fit - Observables

To measure the diffuse astrophysical neutrino flux and the atmospheric neutrino flux, we care about energy and zenith

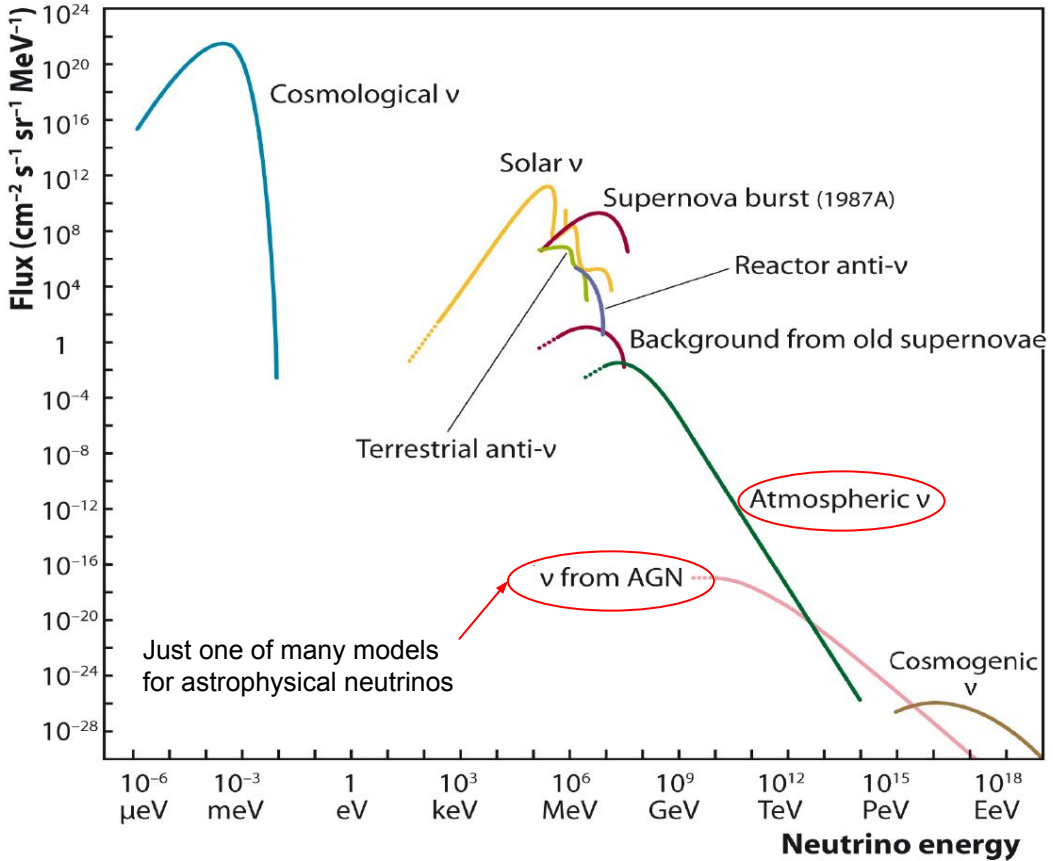
In IceCube, energy is reconstructed from the light observed in the detector

Generally more light ⇒ more energy, but we have reasonably sophisticated reconstruction techniques

Direction can also be reconstructed from the light signature

Observables - Energy

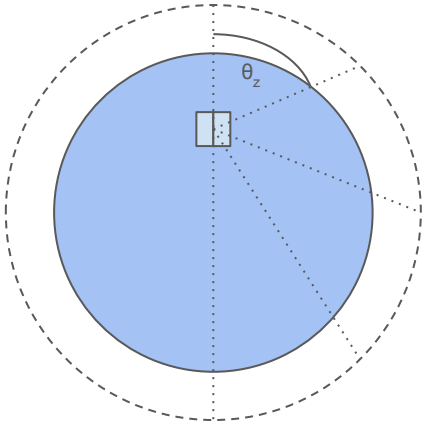
Energy is important because we expect different spectra from the atmosphere and a diffuse astrophysical flux



Observables - Zenith

Zenith is important because the atmosphere and the earth affect what we observe

Different amounts of atmosphere, ice, rock to pass through at each angle



Observables - Zenith

This effect is different for atmospheric and astrophysical neutrinos for 3 reasons

Location - atmospheric neutrinos are produced in the atmosphere

Spectrum - the spectrum is different for the two fluxes and the effect is energy dependent

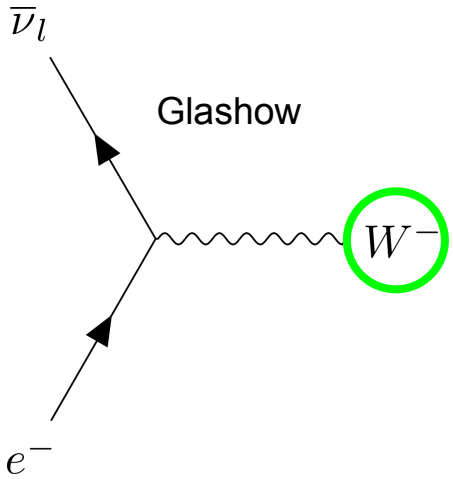
Accompanying muons - atmospheric neutrinos that do not pass through the earth often have a sister muon that also hits the detector, while neutrinos from the other side of the earth have their sister muons blocked

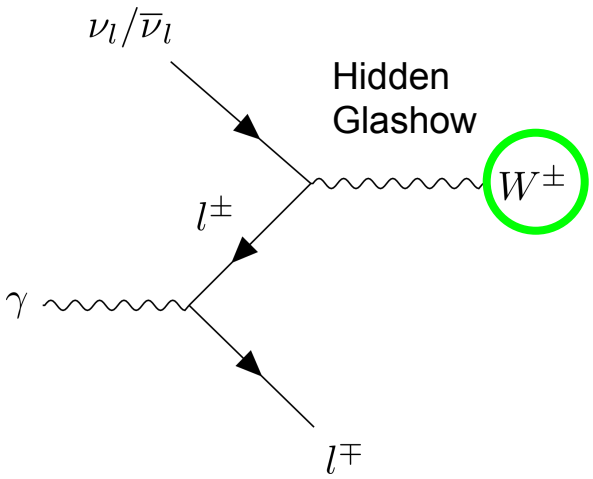
Neutrino Interactions

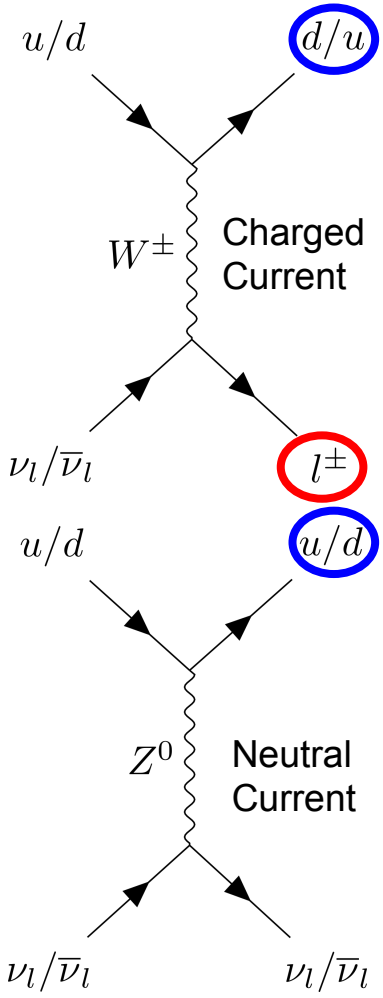
Detectable through the weak interaction

Small cross section → challenging to detect

O(100TeV) neutrinos have an interaction length O(R_{earth})





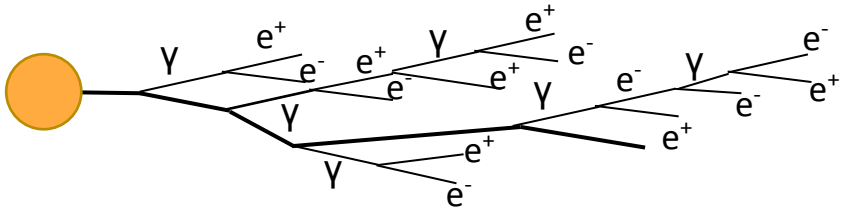


Observables - Topology

CC

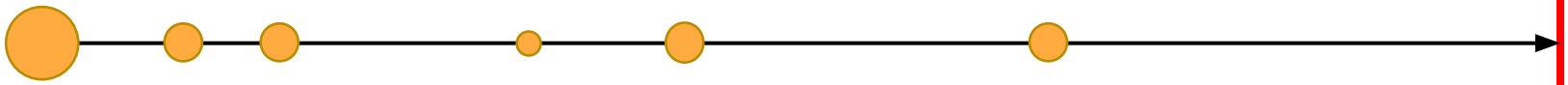
GR

e^-/e^+



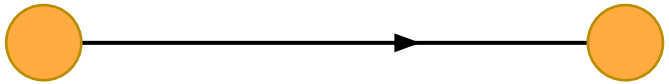
Electron travels O(1m). Starts an electromagnetic cascade of gammas, electrons, and positrons.

μ^-/μ^+



Muon traverses O(1km) in the detector. Constantly loses energy. Has large stochastic losses.

τ^-/τ^+

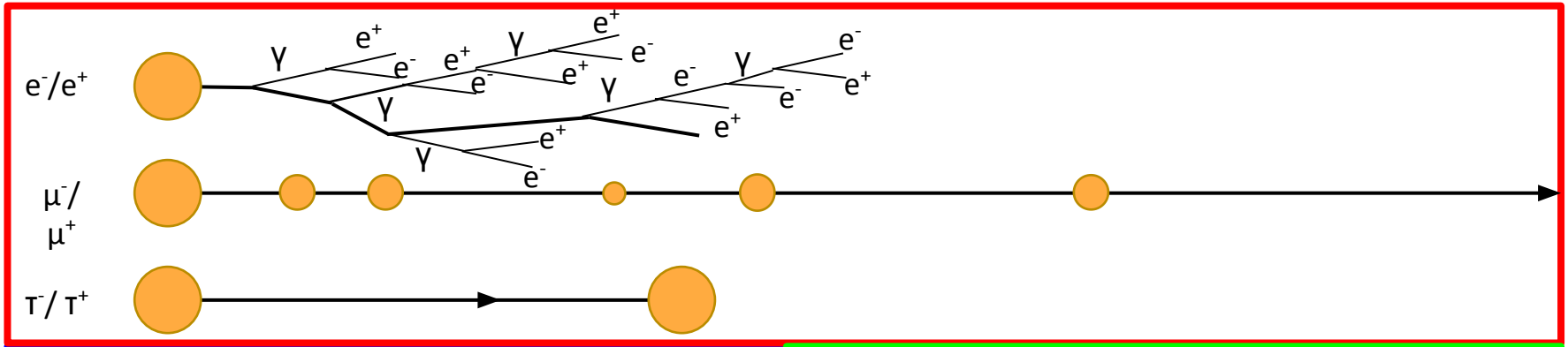


Tau decays on average after 50m per PeV

Observables - Topology

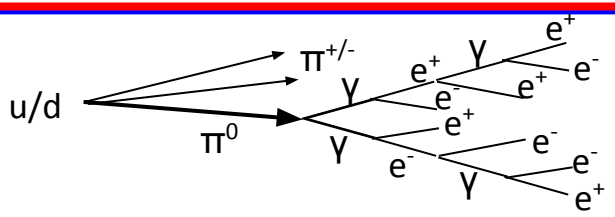
CC

GR

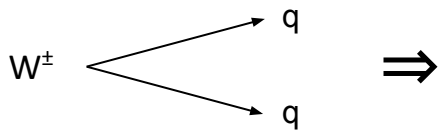


NC

CC



Momentum transfer to quark creates hadronic shower
Charged particles look like an EM cascade

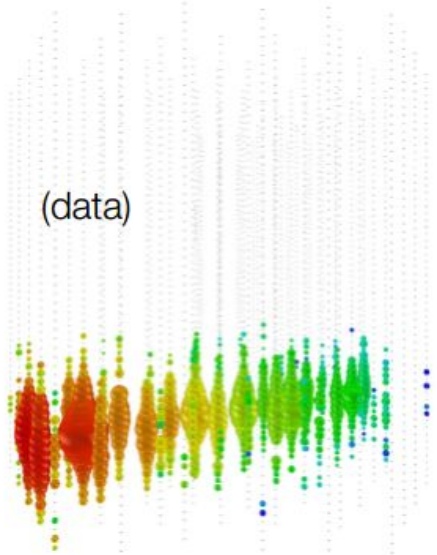


GR

W decays to quark/anti-quark pair (67.6% BR)
Results in a hadronic shower
Other decay channels are neutrino-lepton pairs (10.8% BR per flavor)

Event Detection And Topology

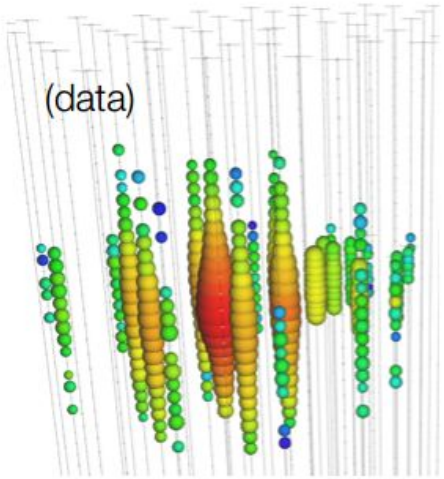
Charged-current ν_μ



Up-going track

Factor of ~2 energy resolution
< 1 degree angular resolution

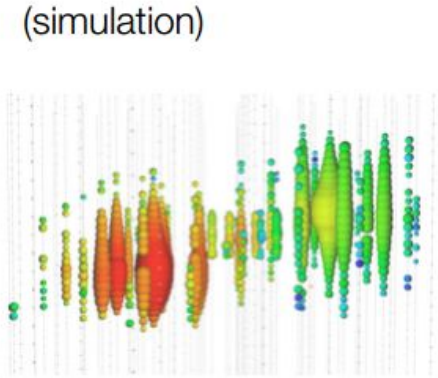
Neutral-current / ν_e



Isolated energy deposition (cascade) with no track

15% deposited energy resolution
10 degree angular resolution (above 100 TeV)

Charged-current ν_τ



"Double-bang"

(none observed yet: τ decay length is 50 m/PeV)

Early — Late

Constructing the binning

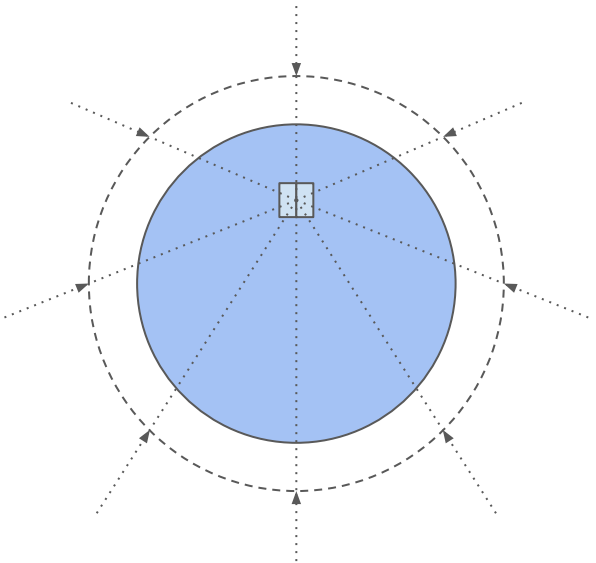
So because both energy and zenith are important, we bin our data in reconstructed zenith and reconstructed energy to obtain a grid

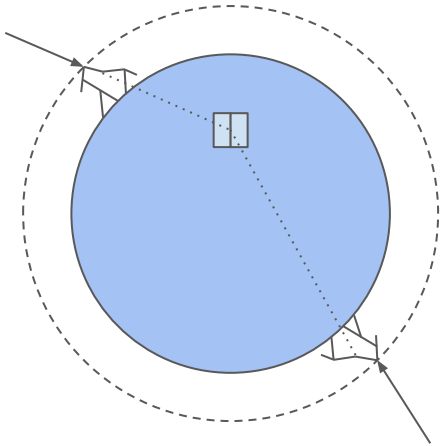
How do we determine the expectation for each bin based on our model parameters?

The answer is simulation! (also called Monte Carlo)

Monte Carlo

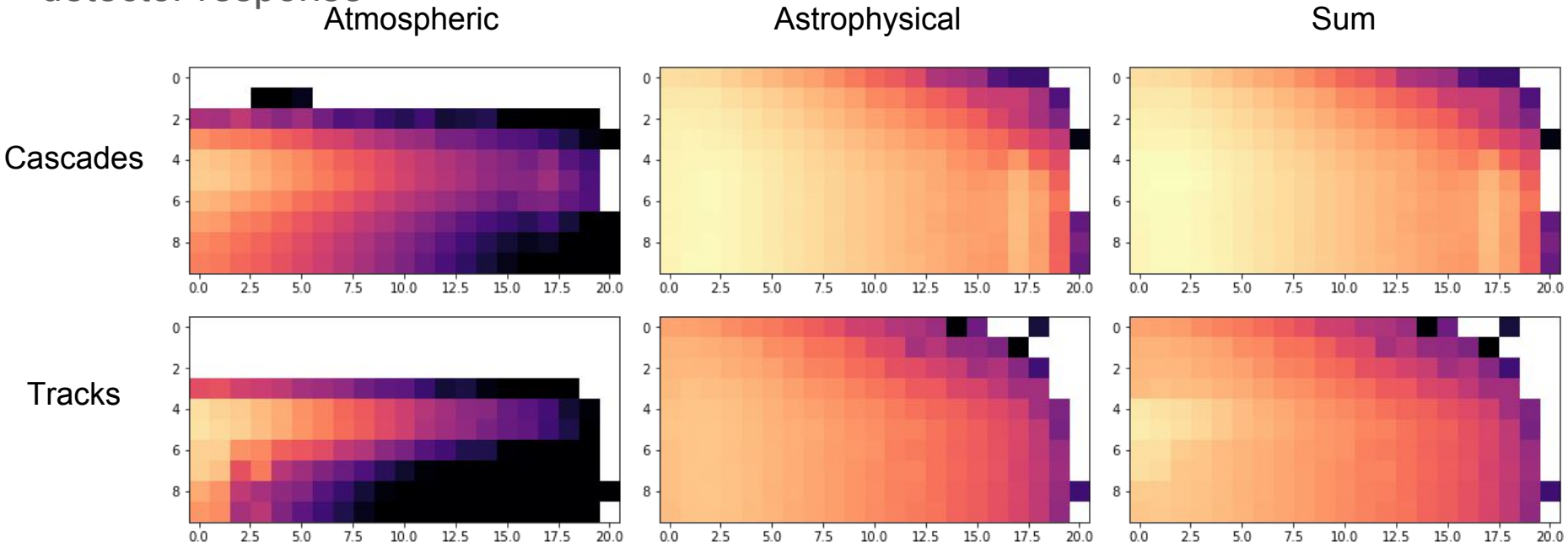
We can simulate astrophysical neutrinos or atmospheric events to see what the detector response is.





Monte Carlo

The simulation can be binned just like our data to get an expectation of our detector response



Monte Carlo - Weighting

So that covers one hypothesis (or set of values for \theta)

What if we want to test another hypothesis?

We don’t want to rerun the simulation because that’s expensive…

So we reweight the simulation!

Weighting

Consider model A where we expect one event with some range of properties, after considering detector effects

We know that the flux of events like that in model B, is half that in model A

That means if we want to test model B, when we make our histogram, we should use an event weight of 1/2

Weighting

Effectively for each event $$w_i = \frac{\Phi_{\text{test}}(E_i, \theta_i)}{\Phi_{\text{generated}}(E_i, \theta_i)}$$

The weight is the model flux divided by the flux used to generate the event

It is important to note that the fluxes we are referring to here are the neutrino flux before any detector or earth effects, the inputs to the simulation, so we use the true neutrino energy and zenith

Recall

We have a likelihood that is a product of poisson terms $\mathcal{L}(\vec{\theta}|\vec{d}) = \prod_i \frac{(\lambda_i)^{k_i} e^{-\lambda_i}}{k_i!}$

Where each λ_i is the expectation in a bin, which is a function of the nuisance parameters

$$\mathcal{L}(\vec{\theta}|\vec{d}) = \prod_i \frac{(\lambda_i(\vec{\theta}))^{k_i} e^{-\lambda_i(\vec{\theta})}}{k_i!}$$

If we consider that our expectation comes from simulation then we know

$$\mathcal{L}(\vec{\theta}|\vec{d}) = \prod_i \frac{(\sum_j w_j(\vec{\theta}))^{k_i} e^{-\sum_j w_j(\vec{\theta})}}{k_i!} \qquad \lambda_i(\vec{\theta}) = \sum_j w_j(\vec{\theta})$$

Which we maximize to obtain an estimate for $\vec{\theta}$

To the code!

Load the MC into events

Load the data

Define event weighting

Define bins

Define likelihood function

Minimize -logL w.r.t. parameters

DONE!

Code - Imports

```
import  as np
import  as sp
import
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import
```

Code - Load MC

```
f = open(... , "r")
json_contents = json.load(...)
```

```
print type(json_contents)
```

```
print json_contents.keys()
```

```
simulation_mapping = json_contents["mapping"]
simulation_events = np.array(json_contents["events"])
del json_contents
```

```
print type(simulation_mapping)
```

```
print simulation_mapping.keys()
```

```
sim_map = sorted(simulation_mapping.items(), key=lambda x: x[1])
sim_map
```

```
print simulation_events[:2]
[[(k, e[i]) for k, i in sim_map] for e in simulation_events[:2]]
```

Code - Load Data

```
json_contents =
data_mapping =
data_events =
del json_contents
```

```
data_map = sorted(data_mapping.items(), key=lambda x: x[1])
data_map
```

```
print data_events[:2]
[[(k, e[i]) for k, i in data_map] for e in data_events[:2]]
```

Code - Event weighting

```
def weight_event(event, atmo_norm, astro_norm, astro_gamma):
    astroWeight = event[...,0]

    return weight
```

Code - Binning

```
energy_bins = ...
zenith_bins = ...
topology_bins = ...
```

Code - Bin masks

```
def make_bin_masks(energies, zeniths, topologies,
                   energy_bins=energy_bins, zenith_bins=zenith_bins, to
pology_bins=topology_bins):

    energy_mapping = np.digitize(...) - 1
    zenith_mapping =
    topology_mapping =
    bin_masks = []
    for i in range(n_topology_bins):
        for j in range(n_zenith_bins):
            for k in range(n_energy_bins):
                pass
    return bin_masks
```

Code - Bin masks

```
bin_masks =
data_masks =
```

Code - Expectation

```
def get_expectation(events, masks, weighting):
    all_weights =
    weights = []
    for mask in masks:
        pass
    return np.array(weights)
```

Code - Likelihood

```
def llh(data, simulation_events, atmo_norm, astro_norm, astro_gamma):
    expect =
    l =
    l[np.logical_and(data == 0, expect == 0)] = 0
    l = np.sum(l).real
    return l
```

```
# Test it
print llh(get_expectation(data_events, data_masks, lambda x: np.ones(x.
shape[0])), simulation_events, 1.0, 6.0, 2.9)
```

Code - Binned data

```
data = np.sum(np.array(data_masks), axis=1)
print data
```

Code - Minimize

```
res = sp.optimize.minimize(lambda x: llh(...), [1.0, 8.0, 2.5], method=
'L-BFGS-B', bounds=[[0,None],[0,None],[None,None]])
```

```
print res
```

```
print 'Atmospheric Normalization = ', res.x[0]
print 'Astrophysical Normalization = ', res.x[1]
print 'Astrophysical Gamma = ', res.x[2]
```