ReMU - Response Matrix Utilities

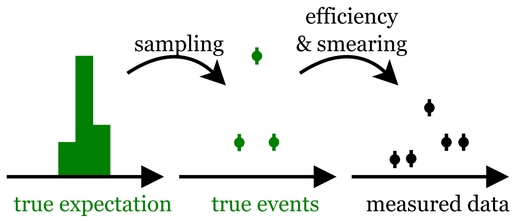
A Python framework for likelihood calculations and hypothesis testing using binned events and response matrices.

license MIT | DOI 10.5281/zenodo.1217572 | docs https://remu.readthedocs.io

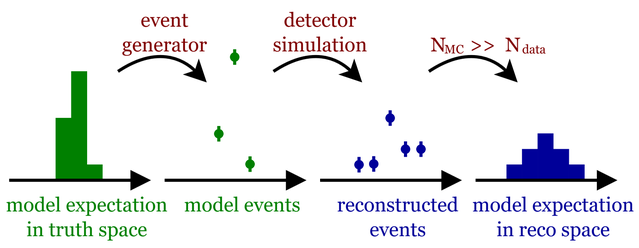
Lukas Koch, STFC Rutherford Appleton Laboratory, lukas.koch@stfc.ac.uk

Forward Folding

Conducting an experiment:
- Sample from a true distribution
- Detector adds efficiency and smearing effects



To compare with models:
- Sample from model distribution
- Simulate detector effects
- High statistics limit yields expectation values to be compared with data

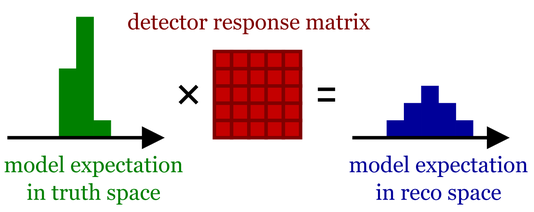


Only feasible for limited number of models because:
- Full detector simulation takes lots of computing time
- Requires expert knowledge about detector
- Detector simulation software often only available within a collaboration

Find universal linear relation between true (μ_j) and reconstructed (ν_j) distributions:

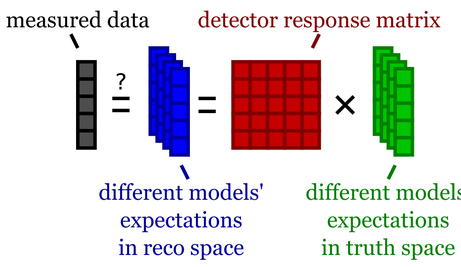
$$\nu_i = \sum_j P(\text{reco bin} = i \,|\, \text{truth bin} = j) \cdot \mu_j$$

Replace full simulation with response matrix R_{ij}:

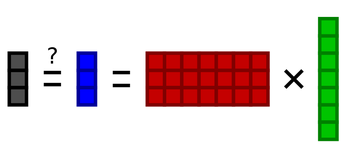


$$R_{ij} = P(\text{reco bin} = i \,|\, \text{truth bin} = j)$$

Matrix multiplication is computationally simple task that does not require expert knowledge and allows fast tests of many theoretical models:



Linear relationship between truth and reco must be universal, i.e. independent of tested models, so truth binning will be finer (and in more variables) than reco binning:



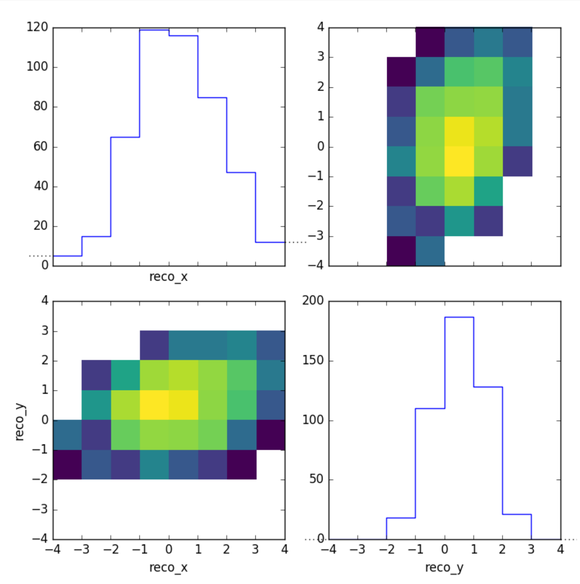
Under-constrained truth not an issue when only folding forward, from truth to reco.

Implementation in ReMU

ReMU implements classes for n-dimensional binning:

```
with open("reco-binning.yml", 'r') as f:
    reco_binning = binning.yaml.load(f)

reco_binning.fill_from_csv_file("real_data.txt")
reco_binning.plot_values("real_data.png", variables=(None, None))
```



Truth and reco binning are combined into a response matrix object, which is filled with simulated data:

```
with open("reco-binning.yml", 'rt') as f:
    reco_binning = binning.yaml.load(f)
with open("truth-binning.yml", 'rt') as f:
    truth_binning = binning.yaml.load(f)

resp = migration.ResponseMatrix(reco_binning, truth_binning)
resp.fill_from_csv_file("model_data.txt")
```

"LikelihoodMachine" class implements likelihood calculations for arbitrary models, i.e. truth expectations:

```
lm = likelihood.LikelihoodMachine(data, response_matrix)
lm.log_likelihood(model)
```

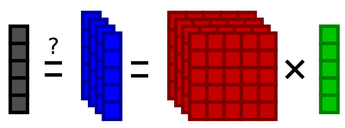
$$L(\mu) = \prod_i \frac{\nu_i^{n_i} \exp(-\nu_i)}{n_i!} = \prod_i \frac{(R_{ij}\mu_j)^{n_i} \exp(-R_{ij}\mu_j)}{n_i!} \qquad R_{ij}\mu_j = \sum_j R_{ij}\mu_j$$

Uncertain detector properties are included as prior $P(\phi)$:

$$L(\mu) = \int_\phi P(\phi) \prod_i \frac{(R(\phi)_{ij}\mu_j)^{n_i} \exp(-R(\phi)_{ij}\mu_j)}{n_i!}$$

ReMU replaces integral with sum over sample from $P(\phi)$:

$$L(\mu) = \frac{1}{N_\text{toys}} \sum_t^{N_\text{toys}} \prod_i \frac{(R^t_{ij}\mu_j)^{n_i} \exp(-R^t_{ij}\mu_j)}{n_i!}$$



Convenience functions likelihood maximisation:

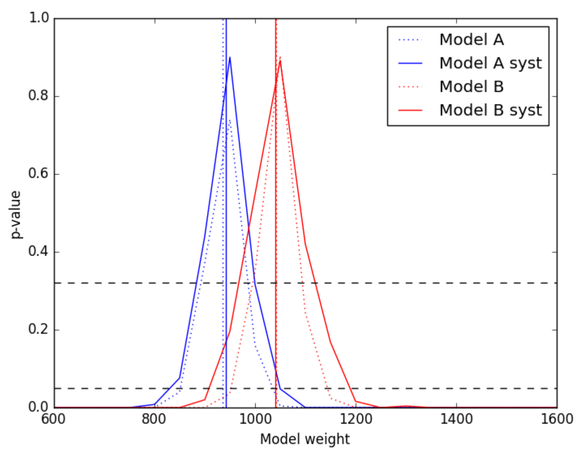
```
model_shape = likelihood.TemplateHypothesis([model])
lm.max_log_likelihood(model_shape)
```

and for numerically calculating different p-values:

```
lm.likelihood_p_value(model)
lm.max_likelihood_p_value(model_shape)
lm.max_likelihood_ratio_p_value(compositeA, compositeB)
```

Allows very simple construction of confidence intervals:

```
for v in values:
    fixed_model = model_shape.fix_parameters((v,))
    p = lm.max_likelihood_ratio_p_value(fixed_model, model_shape)
    p_values.append(p)
```



Support for PyMC

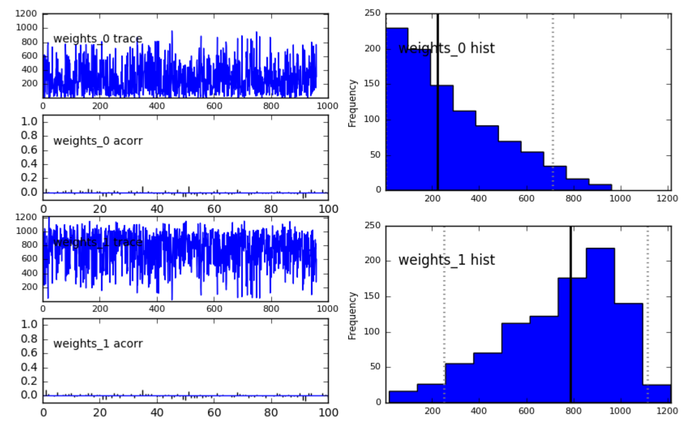
PyMC is a Markov Chain Monte Carlo (MCMC) sampling toolkit:

http://pymc-devs.github.io/pymc/

ReMU can create PyMC MCMC objects, allowing very easy sampling from the posterior distribution:

```
mixed_model = likelihood.TemplateHypothesis([modelA, modelB])

mcmc = lm.mcmc(mixed_model)
mcmc.sample(iter=250000, burn=10000, tune_throughout=True, thin=250)
pymc.Matplot.plot(mcmc, suffix='_mixed')
```

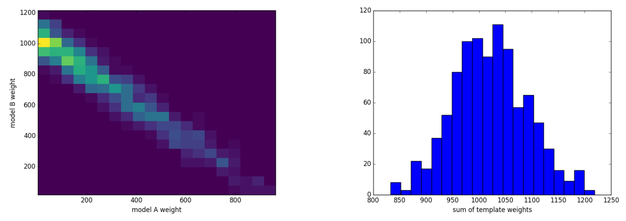


This is useful especially for models with many free (nuisance) parameters, as the MCMC sampling scales much better than the numerical Frequentist methods.

With the fill trace available, it is also easily possible to investigate correlations of parameters or the distribution of functions of parameters:

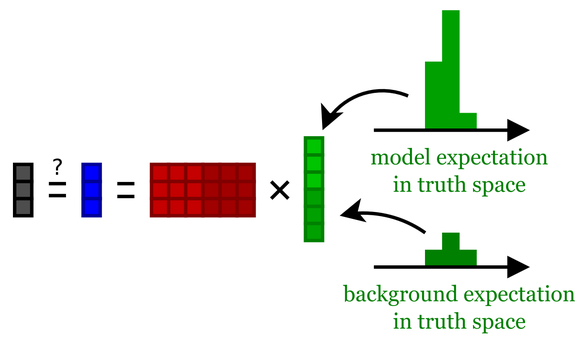
```
weights = mcmc.trace('weights')[:]
pyplot.hist2d(weights[:,0],weights[:,1], bins=20)
```

```
ax.hist(weights.sum(axis=1), bins=20)
```



A Note on Backgrounds

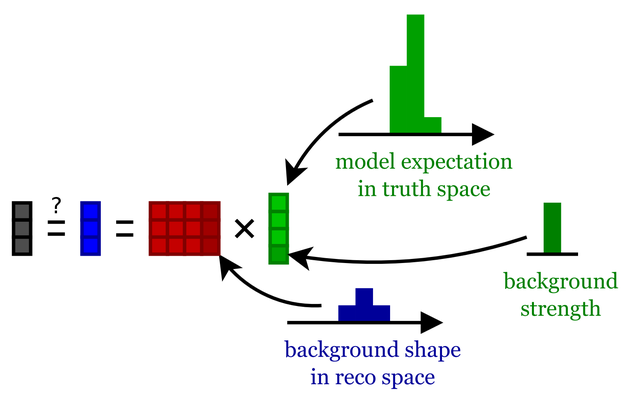
Subtracting background events from the data vector would break the Poisson statistics, so backgrounds must be added to the expectation values:



If the data is not able to constrain the backgrounds, e.g. via control regions, detector experts can provide templates to be used as is or in simultaneous fits:

```
templates = likelihood.TemplateHypothesis([model, background_template])
lm.max_log_likelihood(templates)
```

If the background is very detector specific, its shape can also be provided as a column in the response matrix:



*All code snippets are abridged and will require additional code to run successfully. See examples in documentation.

Science & Technology Facilities Council