2D Alphabet Software Documentation

Lucas Corcodilos

March 22, 2018

Introduction to the method

As one might expect, the 2D Alphabet background estimation method is a two dimensional version of the Alphabet method. The 1D Alphabet method derives a background estimate in the jet mass signal region by interpolating a pass-to-fail ratio, $R_{P/F}$, from the calculated values in the jet mass sidebands and multiplying the failing distribution in the signal region by this $R_{P/F}$.

This method has been used successfully in the past. However, if the signal region is wide and the statistics in the high jet mass sideband are low, the background estimate is susceptible to shifts that do not accurately describe the behavior of the background being estimated. The 2D Alphabet method attempts to remedy this by constraining the shape of the $R_{P/F}$ with a second dimension.

Since the 1D Alphabet method requires a fit of the sideband values to interpolate the signal region, the 2D Alphabet method must do the same but in two dimensions. The statistical tool of choice to accomplish this is the Higgs Analysis Combine Tool. The documentation for the tool can be found here.

Prerequisites

The only environement this software has been tested in is CMSSW_7_4_7_patch2 on CMSLPC. The Higgs Analysis Combine Tool must be installed. To set this up on CMSLPC, follow the below recipe:

- `cmsrel CMSSW_7_4_7_patch2`
- `cd CMSSW_7_4_7_patch2/src`
- `cmsenv`
- `scram b -j 10`

- `git clone https://github.com/cms-analysis/HiggsAnalysis-CombinedLimit.git HiggsAnalysis`
  - Your target directory MUST be HiggsAnalysis/CombinedLimit or the package won't compile
- `cd HiggsAnalysis/CombinedLimit`
- `git checkout -b <new_branch> origin/74x-root6`
- `cat env_standalone.sh | sed 's/=/ /g' | sed 's/export/setenv/g' > env_standalone.csh`
- `make -j 8; make`
- `scram b -j 10`
- `cmsenv`

Installation

PLEASE NOTE THAT THE SOFTWARE IS NOT CURRENTLY READY TO BE INSTALLED AND USED. THIS SECTION IS BEING DEVELOPED ALONG WITH THE SOFTWARE.

Once a CMSSW environment with Combine is setup, one can install the 2D Alphabet software. There are two main steps. The first is to clone the master release from `https://github.com/lcorcodilos/2DAlphabet`. The second is to copy several files to your Combine directory and re-compile Combine so it can use a new class - RooParametricHist2D.

...

How To Run

The 2D Alphabet is designed to run by calling a single script (`2DAlphabet.py`) with a JSON configuration file as input via the command line option `-i`. There are several other options that will be discussed later but they are truly "optional" whereas the configuration file is not - so I will discuss that first.

JSON Configuration

While I've tried to keep the format of the JSON configuration file obvious, it's not fair to assume that everyone else will feel the same way. I've included "HELP" keys in dictionaries where I felt I could easily explain the formatting. This section is a necessary supplement to those. Please note that I may use phrases like, "The key 'HELP' is defined inside 'PROCESS'." I'm aware this is not correct since "PROCESS" is a key for a dictionary which has a key "HELP" and so "HELP" can't be "inside" "PROCESS" but it simplifies the explanation so I'm going to roll with it.

The goal of the JSON is to have an easily configurable input that allows the 2D Alphabet software to read and organize analysis files and histograms to its liking while also giving the user the ability to easily configure values like bin sizes and ranges. This means that while the user is encouraged to add what they need, there are some keys and values that must stay the same. **These static strings are always in capital letters to make them easy to distinguish.** I'll now describe what each of the five static sections is designed for.

PROCESS

In this section, the user can define as many processes as they need to account for. This includes data and MC. This section is NOT used to define the background to be estimated via the transfer function. That background is naturally defined by the difference between data and the other backgrounds defined here.

Each key in `PROCESS` is the name of each process of interest. Please name your data as "data_obs." This is a Combine convention that I'd like to maintain. Each process name is a key for a sub-dictionary that specifies

- `FILE:` the path to the file containing the nominal pass and fail histograms for the process (string);
- `HISTPASS:` the name of the histogram in the above file with the passing distribution (string);
- `HISTFAIL:` the name of the histogram in the above file with the failing distribution (string);
- `SYSTEMATICS:` a list of strings that correspond to the names of systematics in `SYSTEMATIC` (note `SYSTEMATIC` not `SYSTEMATICS`) that are applicable to this process (list of strings);
- `CODE:` a way to classify the treatment of the process: 0 (signal), 1 (data), 2 (unchanged MC), 3 (MC to be renormalized) (int).

SYSTEMATIC

Because it bears repeating, please note the difference between this section, `SYSTEMATIC`, and the list of `SYSTEMATICS` defined inside the `PROCESS` dictionary. The `SYSTEMATIC` dictionary is a place to define as many systematics as a user may need. Similar to the processes, each key in `SYSTEMATIC` is the name of the systematic in the analysis and each is classified by a code that determines how the systematic will be treated. However, the dictionary the user defines for a given systematic is different depending on what type it is. The self-explanatory types are:

- Symmetric, log-normal
  - `CODE: 0` (int)
  - `VAL:` uncertainty (float)
- Asymmetric, log-normal
  - `CODE: 1` (int)
  - `VALUP:` $+1\sigma$ uncertainty (float)
  - `VALDOWN:` -1σ uncertainty (float)

Less obvious are codes 2 and 3 which are for shape based uncertainties (and thus have corresponding histograms) and are either in the same file as the process's nominal histogram (code 2) or in a separate file (code 3). Additionally, they have a scale value which allows the user to change the normalization of the shape. For no change in the normalization, use 1.0. If you have a histogram with a 2 sigma shift, use 0.5 to divide the unit gaussian by 2 before doing the interpolation with Combine.

- Shape based uncertainty, in same file as nominal histogram
  - `CODE: 2` (int)
  - `HISTPASS_UP:` the name of the histogram (in the same file as the nominal histogram) for $+1\sigma$ uncertainty in the pass distribution (string)
  - `HISTPASS_DOWN:` the name of the histogram (in the same file as the nominal histogram) for -1σ uncertainty in the pass distribution (string)
  - `HISTFAIL_UP:` the name of the histogram (in the same file as the nominal histogram) for $+1\sigma$ uncertainty in the fail distribution (string)
  - `HISTFAIL_DOWN:` the name of the histogram (in the same file as the nominal histogram) for -1σ uncertainty in the fail distribution (string)
  - `SCALE:` a scale value which allows the user to change the normalization of the shape (float)
- Shape based uncertainty, in different file as nominal histogram. This is the more flexible but also more complicated option. The user can specify files two different ways. The first is by using `FILEUP:` and `FILEDOWN:` to pick a file that *every* process can pull the shape uncertainty histograms from. The second (and more likely to be used) way is to use keys of the form `FILEUP_myprocess:` where `myprocess` matches the name of a

process that is defined in the `PROCESS` dictionary and has this shape uncertainty associated with it. This allows each systematic and process to come from a separate file. The user can also specify histogram names in three different ways. The first is `HISTPASS` and `HISTFAIL` which allows the user to specify only two histogram names if they don't change between "up" and "down" shapes. The second is if the "up" and "down" shapes *do* have different histogram names and uses the form `HISTPASS_UP` and `HISTFAIL_UP`. Finally, the totally generic way allows the user to use the form `HISTPASS_UP_myprocess` where (again) `myprocess` matches the name of a process that is defined in the `PROCESS` dictionary and has this shape uncertainty associated with it. Below is the totally generic way (which hopefully gets no use).

  - `CODE: 3` (int)
  - `FILEUP_myprocess: /path/to/fileup_myprocess.root` which contains the $+1\sigma$ uncertainty histogram for myprocess (string)
  - `FILEDOWN_myprocess: /path/to/filedown_myprocess.root` which contains the -1σ uncertainty histogram for myprocess (string)
  - `HISTPASS_UP_myprocess:` the name of histogram for myprocess in `/path/to/fileup_myprocess.root` for $+1\sigma$ uncertainty in the pass distribution (string)
  - `HISTPASS_DOWN_myprocess:` the name of the histogram for myprocess in `/path/to/filedown_myprocess.root` for -1σ uncertainty in the pass distribution (string)
  - `HISTFAIL_UP_myprocess:` the name of histogram for myprocess in `/path/to/fileup_myprocess.root` for $+1\sigma$ uncertainty in the fail distribution (string)
  - `HISTFAIL_DOWN_myprocess:` the name of the histogram for myprocess in `/path/to/filedown_myprocess.root` for -1σ uncertainty in the fail distribution (string)
  - `SCALE:` a scale value which allows the user to change the normalization of the shape (float)

This scheme obviously relies on the user's ability to be somewhat organized. If for example, you have your $+1\sigma$ shape based uncertainty in file_up.root with histogram name "SystUp" and -1σ shape based uncertainty in file_down.root with histogram name "SystDown" then there's no applicable code for you to use.

I designed the scheme this way because 2 and 3 are the only applicable classifications to my analysis approach and so I ask the user to be at least as organized as I am so that this scheme works for them.

BINNING

One set of important user defined values is the 2D binning of the space being analyzed. This dictionary is the opportunity to define the axes binning that the user's input histograms already have. Note that while the `LOW` and `HIGH` bin edges defined here and in the user's input histograms *must* be consistent, the number of bins and signal bounds do not. The binning values are split into x and y axis definitions where the x-axis describes the variable whose signal region is blinded.

- `X`
  - `NAME:` name of your variable on the x-axis (string)
  - `LOW:` lower bound of x-axis (int)
  - `HIGH:` upper bound of x-axis (int)
  - `NBINS:` number of x bins from LOW to HIGH (int)
  - `SIGSTART:` lower bound of signal region of x-axis (int)
  - `SIGEND:` upper bound of signal region of x-axis (int)
- `Y`
  - `NAME:` name of your variable on the y-axis (string)
  - `LOW:` lower bound of y-axis (int)
  - `HIGH:` upper bound of y-axis (int)
  - `NBINS:` number of x bins from `LOW` to `HIGH` (int)

FIT

The other set of important user defined values is the fit parameters for the transfer function from the fail sideband to the passing (or pass-fail ratio). The 2D fit of the transfer function assumes a polynomial in both directions. The order of the polynomials are up to the user (technically they are capped at order 10) as long as each parameter is defined. For example, if the highest order parameter defined is 2 in x and 1 in y, the user needs to define six parameters here.

The naming for the parameters must follow the form `X#Y&` where `#` and `&` are the polynomial orders

- `X0Y0`
  - `NOMINAL:` nominal value (float)
  - `LOW:` lower bound (float)

    - `HIGH`: upper bound (float)
- `X1Y0`
    - `NOMINAL`: nominal value (float)
    - `LOW`: lower bound (float)
    - `HIGH`: upper bound (float)
- `X0Y1`
    - `NOMINAL`: nominal value (float)
    - `LOW`: lower bound (float)
    - `HIGH`: upper bound (float)
- `X1Y1`
    - `NOMINAL`: nominal value (float)
    - `LOW`: lower bound (float)
    - `HIGH`: upper bound (float)
- ...

GLOBAL

This dictionary is designed to help users with large configuration files by allowing them to create JSON-wide variables. For example, if all of your files are located in `/long/path/to/my/variables/`, you can store this string in the GLOBAL dictionary with a custom key (let's say `dir/`). Now instead of having to write the full directory path for every process and systematic, the user can just write `dir/`. This simplifies the JSON and also has the standard advantages of using variables over several instances of the same object.

This works by searching all strings in the JSON for instances of each key in `GLOBAL` and replacing the key with its corresponding dictionary value.

Thus, the user must be careful they don't accidentally replace strings in the JSON that are identical to keys in `GLOBAL` but that should be unchanged. This means keys in `GLOBAL` should be descriptive (single character keys would be a bad idea).

Command line options

Plotting

The 2D Alphabet software already makes some basic plots for the user including the shape of the transfer function and several 2D distributions. If you'd like to make more plots from the output of Combine, you'll have to access the RooWorkspaces manually. Most of the time, the workspace that is output by Combine, `MaxLikelihoodFitResult.root`, has the distribution of interest. However, Combine only saves what it deemed useful in its calculations meaning that some things (like distributions in your signal region which Combine is blind to) are not in `MaxLikelihoodFitResult.root`. You can either grab these yourself from the input files referrenced in your JSON or access `base.root` which contains the RooWorkspace input to Combine by the 2D Alphabet software.

To get a better idea of how to do all of this, please take a look at `plot_fit_results.py`.