Table of Contents

CMS Data Analysis School at CERN 2020: $\mathrm{t}\bar{\mathrm{t}}$ cross section measurement at $\sqrt{s} = 5.02$ TeV -- Exercise

Facilitators

- Juan González (University of Nebraska Lincoln)
- Andrea Giammanco (Université catholique de Louvain)
- Pietro Vischia (Université catholique de Louvain)
- Pieter David (Université catholique de Louvain)

Introduction

This exercise will guide you through a measurement of the top-quark cross-section in final state with one or two leptons using the 2017 5TeV dataset, and physics analysis in CMS in general, using the standard NanoAOD format.

The following colour scheme will be used:

- Commands that you should run in the shell will be shown in a gray box, e.g.:

```
ls -ltr
```

- Code snippets to be looked at (e.g, as examples) or edited will be shown in a pink box, e.g.:

```
std::cout << "This is a test" << std::endl;
```

- Output such as screen printouts will be shown in a green box, e.g.:

```
This is a test
```

orange notes are to-do items for the facilitators while preparing the exercise - so these should be gone by the time we start 🙂

Getting set up

As for the pre-exercises and short exercises, we will use lxplus:

```
ssh -Y USERNAME@lxplus.cern.ch
```

bamboo

For most of the analysis we will use the bamboo framework, which only needs a recent version of ROOT (it's based on RDataFrame) and python3. We created a virtual environment with everything you will need (following this recipe). You can pick it up with

```
source /cvmfs/sft.cern.ch/lcg/views/LCG_98python3/x86_64-centos7-gcc9-opt/setup.sh
source ~davidp/public/CMSDAS2020CERN/bamboovenv/bin/activate
```

create "final" virtualenv (latest platform and bamboo) in the cmsdas area (Pieter)

If this does not work If this does not work...
... the most likely reason is that you did not start from a clean shell (the LCG software distribution brings its own gcc, python etc., similarly to CMSSW, but mixing several of these is generally a bad idea).

Please check your `~/.bashrc` and try again in a new shell. If that doesn't help, please ask the facilitators.

In case you are using (t)csh instead of bash or zsh, you should use the csh version of the scripts above instead (`setup.csh` and `activate.csh`).

Other installation options Other installation options
It is recommended to use lxplus, at least for this part not so much for bamboo, but for the input files that you also need to copy to your laptop or institute storage, and for accessing lxbatch, which is useful for the single lepton channel.

To install bamboo there are three options: using the LCG software distribution from CVMFS and a virtual environment (that's what's done above), using conda and pip, or from a docker image. All three are described in the installation guide.

git repository for analysis code

Next, clone the skeleton repository. To be able to exchange code within your team, and with other teams, it's a good idea to create a personal fork on Github (with the "fork" button on the top right), and add it as a remote.

```
git clone -o skeleton https://github.com/pieterdavid/tt5TeVCMSDAS.git
cd ...
git remote add origin git@github.com:GITHUBUSERNAME/tt5TeVCMSDAS.git
```

You can test that git and the repository are correctly set up with `git fetch origin`. It will not fetch anything at this point, but it will check that you can push to your fork from lxplus (you should have gone through the necessary steps in the pre-exercises; if something does not work (anymore) you can find all the necessary information there). If you want you can already add your colleagues' forks

```
git remote add THEIRNAME https://github.com/THEIRGITHUBUSERNAME/tt5TeVCMSDAS.git
```

We will have a closer look at the framework in a minute, but you can already produce a few plots with the following command (this will print some information about the files and samples being processed, and take a

few minutes):

```
bambooRun -m tt5TeV.py:HelloWorld dilepton.yml -o ../test_out_1
```

The output directory `../test_out_1` is chosen to avoid unintentionally committing the outputs; you can also make an `output` directory and add it to the `.gitignore` file, as you prefer.

update file lists, copy/move to the CMSDAS EOS directory, as needed

combine

One team (D) will work on the statistical analysis, with the combine tool. The setup for this is a bit different, but should be familiar, since based on CMSSW; that means that you should do this in a fresh shell (screen or tmux can be really convenient in such cases, just keep in mind that you'll need to type `kinit` and give your password once a day to keep it working).

You can use either the instructions below or look into the installation page in the documentation, where you also will find alternative ways of installation that you will need in case you will contribute to the combine development.

```
export SCRAM_ARCH=slc6_amd64_gcc530
cmsrel CMSSW_8_1_0
cd CMSSW_8_1_0/src
cmsenv
git clone https://github.com/cms-analysis/HiggsAnalysis-CombinedLimit.git HiggsAnalysis/CombinedL
cd HiggsAnalysis/CombinedLimit
cd $CMSSW_BASE/src/HiggsAnalysis/CombinedLimit
git fetch origin
git checkout v7.0.1 # this version might not be the latest one; always check the documentation to
scramv1 b clean; scramv1 b # always make a clean build
```

Measurement of the $\mathrm{t}\bar{\mathrm{t}}$ production cross section

You can take a look to the cross section measurement by CMS with the 2015 dataset, CMS-PAS-TOP-16-023.

The dataset that we will use correspond to the Run2017G era, with a total of 296.08 pb^{-1}. This dataset was collected at low pileup. We will use three different datasets `SingleMuon`, `DoubleMuon` and `HighEGJet` (containing events passing electron triggers). Take a look to the top trigger recommendations and remember to perform an overlap removal. Note that the `DoubleMuon` dataset does not have an invariant mass threshold!

List of tasks:

- Control plots for leptons, jets, and event variables
- Optimizing the selection: electron, muon and jet p$_{T}$, MET cut
- Measurement of efficiency, acceptance and inclusive cross section
- Data-driven estimate for the DY background
- Data-driven estimate for the Nonprompt leptons background
- Trigger paths and trigger scale factors
- B-tagging discriminant (CSVv2 vs DeepCSV), working points and scale factors
- Modeling uncertainties: PDF, Matrix element scale (μ_{R}, μ_{F}), h$_{damp}$, Underlying Event. Fiducial cross section.
- Combination of measurements in different channels

A: Dilepton channels -- Background estimate

Basic tasks:

- Optimize the lepton and event selection for the dilepton analysis in the three channels (ee, $\mu\mu$, eμ)
- Calculate the DY data-driven estimate and uncertainty
- Calculate the nonprompt leptons background estimate and uncertainties
- Produce control plots for electrons, muons, jets and other event observables

Advanced tasks:

- Calculate the estimations for several levels of selection

You need to take inputs from:

- Nobody

You need to provide inputs to:

- Groups B and D

The dibosons and tW backgrounds are estimated from MC. The uncertainties on the normalization for these backgrounds is 30% (see here for instructions about adding this to the plots). The Drell-Yan and nonprompt leptons background are estimated using data-driven techniques.

Drell-Yan background

The Drell-Yan background can be estimated using the R$_{out/in}$ method, documented in AN-2015/305. Events inside the Z peak are counted in ee/$\mu\mu$ channels. The non DY contribution inside the peak is estimated as half the contribution from the eμ channel. The DY background estimate outside the peak is taken by multiplying the DY measurement in the peak by the R$_{out/in}$ factor, that is calculated as the ratio between DY number of events outside and inside the Z peak, from MC.

The number of predicted events in the $\mu\mu$ is:

$N^{\mu\mu}_{out} = R^{\mu\mu}_{out/in}(N^{\mu\mu}_{in} - 0.5 N_{in}^{e\mu} k_{\mu\mu})$

where $k_{\mu\mu}$ is a factor that corrects for the acceptance of electrons, and is calculated using events in the Z peak as:

$k_{\mu\mu} = \sqrt{\frac{N^{\mu\mu}_{in}}{N^{ee}_{in}}}$

The R$_{out/in}$ factor is calculated using Monte Carlo as:

$R_{out/in} = \frac{N_{out}^{\mu\mu, DY}}{N_{in}^{\mu\mu, DY}}$

For the DY estimate in the $e\mu$ channel, a scale factor is applied to the DY MC estimate, defined as:

$SF_{DY}^{e\mu} = \sqrt{SF^{\mu\mu}\cdot SF^{ee}}$

and $SF^{\mu\mu}$ is the ratio between the expected number of DY events with the data-driven and MC estimations.

For the normalization uncertainty in MC you can assume a 10% uncertainty. You must propagate the systematic an statistic uncertainties on the estimate to the final result. Also, as an additional uncertainty you can perform the estimate for several jet multiplicities and see how stable the SF is.

Nonprompt leptons background

To calculate the nonprompt leptons background, first define your selection in the same way as for the main analysis but selecting only same-sign dilepton pairs. This will be your control region. This method is funded on the hypothesis that nonprompt leptons has equal probability of being positively or negatively charged, so should be equally distributed in same-sign (SS) and opposite-sign (OS) selections. In the standard model, events with final states containing two isolated same-sign leptons are very rare, so the expected number of events in this control region should be small. A large fraction of the events in this control region will come from charge misassignment of electrons. The contribution of prompt same-sign backgrounds in this region is calculated from MC and substracted to the observed data. Then the nonprompt lepton estimate is:

$N^{OS, fakes}_{data} = (N^{SS}_{data} - N^{SS, prompt}_{MC})\cdot R$

Where R is a transfer factor from the same-sign to the opposite sign region can be calculated using events from MC, with the assumption that most of the events with nonprompt leptons come from `W+Jets` and `semileptonic $t\bar{t}$` events (which is true). Then we can compute:

$R = \frac{N^{OS, nonprompt}_{MC}}{N^{SS, nonpromt}_{MC}}$

R must be close to one if our initial assumption is true.

Note: if the number of events in the same-sign region is too small to have a proper estimate of the background, you can try relaxing the isolation cut in you lepton selection.

Hint Hint
You may find some inspiration for the implementation in this recipe (but it may be overkill in this case; think about what group D needs and how you can obtain that).

B: Dilepton channels -- Systematic uncertainties and cross section calculation

Basic tasks:

- Calculate the inclusive and fiducial cross section, with all the uncertainties, for the three dilepton channels
- Calculate and propagate uncertainties from: electron and muon efficiencies, jet energy scale.
- Calculate and propagate uncertainties from: h_{damp}, underlying event tune, PDF, ME scales.

Advanced tasks:

- Calculate as much other uncertainties as you can
- Calculate the modeling uncertainties at fiducial level
- Propagate the uncertainties to all the plots of the analysis using error bands

You need to take inputs from:

- Group A: the event selection and background estimate

You need to provide inputs to:

- Group D

There are several sources of uncertainties on the computation of the efficiency and the acceptance. These uncertainties must be propagated to the final results.

add the scalefactors and PU reweighting to an example

Experimental uncertainties

Uncertainties to the jet energy scale and jet energy resolution should be propagated to the yields and cross section measurements. It can be done by using the collections of jets stored in the trees with the varied p_{T}. Uncertainties on lepton ID and isolation efficiencies can be estimated by varying the lepton scale factors by their uncertainties. Check if there is any specific recommendation from the POGs. Uncertainties to the trigger efficiencies may be estimated from the scale factors (if you use some trigger scale factors, which are not provided in this exercise). If you tag b jets, you should assign uncertainties to the b-tagging and mistag rate. Normaly, the uncertainties are calculated by varying the p_{T}- and eta-dependent b-tagging scale factors. If you apply PU reweighting, you should calculate an uncertainty associated to this reweighting.

Modeling uncertainties

The uncertainty on the h$_{damp}$ modeling and the tune of the underlying event are calculated using dedicated samples. The uncertainty is propagated to the acceptance by comparing the varied samples with the

nominal samples.

The uncertainty on the matrix element scales are calculated by varying μ_{R} and μ_{F} by 2 and 0.5 independently. The events are reweighted according to 8 different variations and the uncertainty is propagated to the acceptance taking the maximum variation with respect to the nominal value. The variations where $\mu_{R} = 0.5$ and $\mu_{F} = 2$ and $\mu_{R} = 2$ and $\mu_{F} = 0.5$ are considered unphysical.

The uncertainties on the proton PDF and the value of α_{S} are propagated using 36 set of weights with variations of the PDF values and 2 extra set of weights with variations of $alpha_{S}$. Calculate the uncertainty following the recipe in 1510.03865.

Calculate these modeling uncertainties also at fiducial level: use events in the fiducial region (at generator level) and normalize to the number of fiducial events the nominal value and the variations.

Hint Hint
There's no explicit support for adding the uncertainties from alternative samples in plotIt and, therefore, bamboo. You can try to follow the procedure discussed here.

prepare an implementation for this

C: Lepton + jets channels

Basic tasks:

- Define you event selection and signal regions for e+jets and μ+jets channels
- Estimate the QCD events from a control region
- Use basic uncertainties (e.g.: normalization uncertainties, lepton efficiencies...)
- Extract the cross section with a fit where (at least) QCD of W+Jets normalization is constrained

Advanced tasks:

- Calculate as much other uncertainties as you can
- Try fitting other distributions
- Study and optimitze the b-tagging efficiency for both algorithms CSVv2 and Deep CSV

You need to take inputs from:

- Nobody

You need to provide inputs to:

- Group D

Follow the strategy in AN-2016/320. In this exercise you don't need to precisely estimate all the systematics but try to define a good strategy for the signal extraction and have a good control of the QCD background and W+Jets estimations.

Objects and event selection

Select events with exactly one electron or muon and at least 4 jets. You have to veto events with more than one lepton. Try to also veto events containing leptons passing relaxed ID/ISO cuts.

Create event categories according to the b-tag multiplicity of the events. Try different b-tagging working points and try CSVv2 and DeepCSV algorithms.

Compute several variables with the non b-tagged jets related to the W mass and the angular distances. Follow the ideas in AN-2016/320.

QCD control region

Use non-isolated events for estimating the QCD background. Use the low MET region to extrapolate the estimation to the signal region. Follow the explanation in AN-2016/320.

D: Combination of measurements

Basic tasks:

- Create a framework for combining the measurements in all channels (ee, $\mu\mu$, eμ, e+jets, μ+jets).

Advanced tasks:

- Compare the combinations using `BLUE` and `combine`

You need to take inputs from:

- Everybody

You need to provide inputs to:

- Nobody

Your task is to compare measurements from the different channel. This is eminently a statistical problem, and you won't work with the details of the event selection. You will regard the various final states as "regions". You will have a signal region per each of the channels, and possibly also control regions from them.

You will eminently use the Higgs combination tool ("combine"); this tool is an advanced interface to RooStats that contains full implementations for most of the common tasks you might be interested in; in such a way you can focus on the measurement at hand rather than having to reinvent the wheel. As an advanced exercise, you will look into the Best Linear Unbiased Estimator (BLUE), which is a simplified way of combining measurements, and you will compare the results from the two frameworks.

In this exercise you will teach yourself how to use the tool by starting from the requested task and working your way to the command lines for it. To achieve that, you will make heavy use of the documentation of the combine tool. You will first need to install the tool (see the instructions above).

Combine commands act on "datacards" that contain all the information about the yields per each process in your signal or control regions, and the uncertainties. The input yields will all depend on the cross section of each process and on the luminosity. The parametrization of the problem is such that the result of a likelihood fit is given in terms of "signal strength", which is denoted μ in notes and papers, and r within the tool and defined in the case of the cross section as;

$\mu=\frac{\sigma_{\text{observed}}}{\sigma_{\text{expected}}}$

In other words, a result of $\mu=1.2$ means that your best estimate for the cross section is a value which is 20% higher than the expected cross section (the one you normalized your yields to). Note that here you are

measuring the fiducial cross section, and you will therefore need to later extrapolate to the inclusive cross section.

It is important to never look at the data until you are satisfied that the method is correctly in place; for this reason, for each measurement you will first perform all the needed checks with the Asimov dataset.

Your first objective is to perform a profile likelihood fit to extract the ttbar cross section in one signal region, coding in a datacard the yields you obtain for each process and their uncertainties. For this you can either use a basic datacard, or a datacard supporting "shapes" (in your case, the shape will be made of one bin).

- Remember to first use the Asimov dataset;
- Always check the "impact plots" to learn if there are pathological constraints of the nuisance parameters of your statistical model; the Asimov dataset should realize the ideal "no pull, no constraint" situation;
- You should break down the contribution from each systematic uncertainty; play with different groupings, and isolate the uncertainties you think are the most interesting;
- Look at the post-fit yields for each proces, and compare them to the corresponding pre-fit yields. What can you learn?

Your inputs will arrive from each of the other groups, but since they will need some time to produce it, you can set up a mock exercise by reproducing in your card the yields from this CMS note. In Section 4.2 of the note, you will find information on the systematic uncertainties; you will implement at this stage only the ones that are "flat" (i.e. the ones that you can code with a single number rather than having a distribution; such distributions will be provided by the other groups. In the perfect CMS style, you can even ping them every now and then to know if they have already some output available; just be gentle!

You will notice that there are three regions in the table; implement each region in a separate datacard, compute the cross section separately in each one, and then combine the three datacards to perform a simultaneous fit to the three regions.

When you have the inputs from other groups, you will then add the control regions they developed, properly modelling the type of nuisance parameter in the datacard, and check how the result changes, in particular with respect to the uncertainties and of post-fit yields.

Advanced; BLUE vs combine

Combining two (or more) measurements can also be done post-facto, by taking the results (central value plus uncertainties) of the individual measurements, and "combining" them. This is somewhat a less reliable procedure with respect to the simultaneous fit to all the regions, because it is performed under some simplifying assumptions that are not always valid. It is interesting to compare the combination of measurements done by performing simultaneous fits, and the one done by combining the individual fit results.

You can use an easy python implementation of BLUE to perform such comparison; split the uncertainty in each individual "combine" fit in a reasonable number of systematic uncertainties; encode each of them in BLUE, and compute the combined measurement. Finally, compare the result with the result you obtain from a likelihood fit of all categories simultaneously.

All teams: presentation on Friday

You can add material to [[][this google doc presentation]]

make a new presentation on google docs

Useful links

- TOP PAG twiki
- JetMET POG twiki
- EGamma POG twiki
- Muon POG twiki

copy, update, and add a few more (NanoAOD doc, for instance)

A brief introduction to the bamboo framework

All frameworks by definition impose some constraints on what you can do, and how you can do that. That is done for good reasons, usually, but it makes it a little bit unfair to impose a specific one for a project of just a few days, since you have very little time to get acquainted to it. On the other hand, we do have enough time to build up a fairly complete analysis, and run into the limitations of ad-hoc code.

This introduction is an attempt to quickly get you up to speed with the mental model and assumptions in bamboo, but it will certainly miss some points that may be important, so feel free to ask and point those out. After the exercise you can open an issue (also for missing or unclear documentation) in the bamboo repository on gitlab (or discuss in the mattermost channel). There is also lengthier HTML documentation; for the exercises you will probably use the helper function list and API reference most.

Principles

bamboo is based on `ROOT::RDataFrame`. If you have not done the ROOT short exercise please have a look, especially at the second part, it is a very good introduction.

To summarise, the key point of RDataFrame is that you do not directly write the code that will loop over the events, read branches etc., but declare a computation graph based on the structure of the tree (the branch names and their types), and everything is computed efficiently (in a single loop over the inputs, if possible, and in (JIT-)compiled C++) afterwards, when retrieving the results. An important consequence of this is that the code you write needs to work for all possible events, e.g. if you want to make a histogram with the leading jet p_T, the `Histo1D` node that does that should be under a `Filter` node that asks for at least one jet, otherwise the first event without jets will cause a segmentation fault.

The goal of bamboo is to make it easy to build a full analysis with RDataFrame. For the most common case of producing stack plots comparing the sum of MC samples to data, that means:

1. since you will want to build (almost) the same RDataFrame graph for the different samples, you should define a list of samples (minimally a unique name and list of files, but it's useful to add more information, indicating e.g. if it is an MC sample or not, if it should be grouped with others in the plots) in a YAML configuration file, and define a module with a method that produces the RDataFrame graph for a sample.
   This is a strong constraint, but it also brings an important benefit: the code needed for splitting the work over batch jobs (one or several per sample) and combining the results is fully separated from the analysis-specific part, so to use lxbatch (or a different batch system) you only need to pass one command-line switch.
   Now it's also clear why the typical way to run something with bamboo is

   ```
   bambooRun -m tt5TeV.py:HelloWorld dilepton.yml -o ../test_out_1
   ```

   `dilepton.yml` is the YAML configuration file with sample definitions, and `tt5TeV.py` a python module with a class `HelloWorld` that has a `definePlots` method to build the RDataFrame graph. Feel free to launch this, it should not take more than a few minutes and give you a few plots in the `../test_out_1` directory.
2. to make plots, you need an RDataFrame with `Filter` nodes (to apply cuts), `Histo1D` nodes (to fill histograms), and `Define` nodes (to make things efficient, by reusing intermediate results instead of calculating them again). In bamboo, you instead define `Selection` and `Plot` objects, which correspond almost one-to-one to `Filter` and `Histo1D` nodes, respectively, but add some additional information. `Define` nodes are in most cases inserted automatically behind the scenes.
   A `Selection` corresponds to a set of cuts (`Filter` node), and also holds an event weight. It is constructed by adding cuts and/or weight factors to another `Selection` (starting from the trivial one,

with all events in the input and unit event weight), by calling the `refine` method of the parent selection. This is a rather natural way to think about an analysis, once one gets used to it: cuts define selection stages, or selection regions, and the corrections that are multiplied with the event weight depend on the cuts applied so far.
A `Plot` is then the combination of a `Selection` with an x-axis variable, a binning, and layout options (two-dimensional plots are also supported, the only difference is that they have two variables and two binnings).
So essentially, the `definePlots` method takes the root (trivial) `Selection`, and returns a list of `Plot` objects defining other `Selection` objects as needed.

3. in RDataFrame you can pass C++ callables (from C++), code strings (from C++ and python), and numba-compiled python methods. In bamboo, there is a python object representation of the tree (you can think of it as a generic event), which puts the structure back by combining branches with the same prefix, adding some helpers, etc. You can get attributes and call functions on this tree view to build expressions, e.g. you can use `leadMu = tree.Muon[0]` and fill `leadMu.pt` instead of `Muon_pt[0]`, or `(tree.Muon[0].p4+tree.Muon[1].p4).M()` instead of something long enough to need a helper function. These expressions will automatically be converted into code strings (inserting `Define` nodes if needed) when needed for RDataFrame, if used in a `Selection` or `Plot`. Since the expressions are python objects, you can save intermediate results and reuse them to make your code simpler.
This may seem a bit strange at first, but it's just a more pythonic view of the NanoAOD branches. The easiest way to get an idea of how this can be used is by trying. The following command will give you an IPython shell, the event view is called `tree` here:

```
bambooRun -i -m tt5TeV.py:HelloWorld dilepton.yml
# try tree.<TAB>https://cmsdas.github.io/root-short-exercise/
# or j = tree.Jet[0], and then j.<TAB>
```

An example in detail

Let's start from the example command that you may already have launched above:

```
bambooRun -m tt5TeV.py:HelloWorld dilepton.yml -o ../test_out_1
```

this shows the main inputs: a YAML configuration file, with a list of samples to process, options for the plots, and some other pieces of informations (e.g. the integrated luminosity in the data files), and a python class (which inherits from a base class, making it a 'bamboo analysis module' that `bambooRun` can call certain methods on). It will proceed in two steps:

- first all samples (defined in the YAML configuration file) are "processed": in the case of plots this means that the histograms needed by the `Plot` objects defined in the module are filled (it's also possible to do other things, e.g. output reduced trees for training a multivariate classifier, but we don't need that here)
- then a "postprocessing" step is run, combining the different samples into plots. The plotIt tool (code, documentation) is used for this, a standalone C++ executable that does just this. It needs the ROOT files and another YAML file, but bamboo can generate the latter (it is written to `plots.yml` in the output directory, but you probably won't need it)

Since the first step usually is by far the most time-consuming, bamboo makes it possible to split the two: `bambooRun --onlypost` (with otherwise the same arguments) will run just the second step (very convenient if all you need to do is change some colour or axis title). The first step can also be run in an distributed way on HTCondor or slurm, since it's always only looking at one sample at a time (samples can also be split, and the results merged). This is done by adding the `--distributed=driver` option.

To use lxbatch To use lxbatch...
... bamboo needs to know a few things about it (the code is generic, such that it can be used also on other HTCondor and slurm clusters). The easiest way is by saving this file as `~/.config/bamboorc`. You can add

more HTCondor options (job flavour, maximum CPU time and memory etc.) under the corresponding section.

The histograms for each sample will be written to `/results/.root`. It is important to know that for MC no scaling with the cross-section and integrated luminosity is done: these, together with the number of generated events, are passed to plotIt to do the normalisation. This has some advantages (there is one place where the scaling is done, and you can change the cross-section in the plots without rerunning), but may be unexpected, or need a bit of extra care when making the datacards.

Please have a look at `dilepton.yml`, the YAML configuration file. Most of it should be clear, or easy to guess, now.

Now we can have a closer look at the python code of the module we just ran. We can for now ignore the first part of the file, before the `HelloWorld` class definition: it defines a base class to deal with the input samples (they are very similar to NanoAODv4 with some processing, but there are a few differences that need to be taken into account).

Let's start with the "hello world" example (a dimuon invariant mass plot):

```
class HelloWorld(Nano5TeVHistoModule):
    def definePlots(self, t, noSel, sample=None, sampleCfg=None):
        from bamboo.plots import Plot, CutFlowReport, SummedPlot
        from bamboo.plots import EquidistantBinning as EqB
        from bamboo import treefunctions as op

        isMC = self.isMC(sample)
        if isMC:
            noSel = noSel.refine("mcWeight", weight=t.genWeight)
        noSel = noSel.refine("trig", cut=op.OR(t.HLT.HIL3DoubleMu0, t.HLT.HIEle20_Ele12_Calo

        plots = []

        muons = op.select(t.Muon, lambda mu : mu.pt > 20.)
        twoMuSel = noSel.refine("twoMuons", cut=[ op.rng_len(muons) > 1 ])
        plots.append(Plot.make1D("dimu_M",
            op.invariant_mass(muons[0].p4, muons[1].p4), twoMuSel, EqB(100, 20., 120.),
            title="Dimuon invariant mass", plotopts={"show-overflow":False,
            "legend-position": [0.2, 0.6, 0.5, 0.9]}))

        return plots
```

It's quite short, so let's go through line by line: first, we declare a class, our module, and inherit from the base class (if you go up a few lines, you see that it combines everything for histograms with the adjustments for our samples, but you do not need to worry about the details of this). Since it's a module to fill histograms, there's the `definePlots` method which returns a list of `Plot` objects, as explained above; this will be called once for each sample (or once per job, if the sample is split over multiple batch jobs); the event loop will run in (mostly JIT-compiled) C++, that's what makes this fast. The arguments to `definePlots` are `self`, a reference to the module itself, `t`, a view of an event, `noSel`, the base selection, `sample`, the name of the sample, and `sampleCfg`, a dictionary with the fields defined in the YAML analysis config for this sample.

Most of the other imports speak for themselves: a `CutFlowReport` will print the numbers of events passing different selections, and a `SummedPlot` simply adds up the histograms from different plots (e.g. to make a combined jet pt plot from those for the dimuon and dielectron categories). `EquidistantBinning` holds an axis binning (number of bins, minimum, and maximum); `VariableBinning` also exists. `bamboo.treefunctions` defines a collection of helper functions, that can be used on objects retrieved from the event view, e.g. `op.invariant_mass(tree.Muon[0], tree.Muon[1])` the full list is available here (this is necessary because we're working with placeholder objects instead of the actual values; these functions record the operations instead of immediately performing them).

Now things get more interesting: by default every event would have weight 1, but for MC samples we want to use the generator weight (this is especially important for NLO samples), so we add that to the `Selection`. We also only want to look at events that pass a trigger, so that's added as well (here the new selection is called `noSel` again, but it's just a python variable, so the name can be anything). At this stage it's quite normal to apply some cuts that are specific to the sample (

Next, an empty list is defined that we can add plots to, and return, eventually; this example only defines one plot, but a realistic analysis quickly grows to hundreds of plots that are produced in one go.

```
muons = op.select(t.Muon, lambda mu : mu.pt > 20.)
```

makes a list of muons that pass a selection. Adding more cuts is easy, by replacing `mu.pt > 20` with `op.AND(mu.pt > 20, ...)`, and `muons` can be used in the same way as `t.Muon`. This syntax with python lambda expressions passed to "higher order functions" may seem a bit strange at first, but it's very powerful (and easy to implement). As an exercise, think about how you could select jets that are not within a certain angular distance from any electron or muon the answer is also in the bamboo documentation.

The next line goes on to select events with two muons (so all selections and plots attached to this will only see those events)

```
twoMuSel = noSel.refine("twoMuons", cut=[ op.rng_len(muons) > 1 ])
```

and the line after that defines a `Plot` with the dimuon invariant mass (which can safely be calculated now):

```
plots.append(Plot.make1D("dimu_M",
            op.invariant_mass(muons[0].p4, muons[1].p4), twoMuSel, EqB(100, 20., 120.),
            title="Dimuon invariant mass", plotopts={"show-overflow":False}))
```

A few more ingredients you may need

Triggers and primary datasets

For data you will combine data with different triggers to cover all channels (dielectron, dimuon, and mixed for dilepton, or single electron and single muon for the single lepton channel). These are not all on the same primary dataset (a primary dataset contains all events selected by one of a group of triggers, such that analyses can run on smaller input files), so it may happen that an event is selected by triggers on different primary datasets, and would be considered twice in the analysis. How could you prevent this? An additional complication is that not all triggers were there for the whole data-taking period.

Hint Hint
In most cases the `bamboo.analysisutils.makeMultiPrimaryDatasetTriggerSelection` helper method can be used for this; it may not directly apply in this case, but it could serve as inspiration nonetheless.

finish writing this (Pieter); tasks A, B and C (or the section that defines them) should link here

---

This topic: Main > Draft_tt5TeV_CMSDASCERN2020_piedavid
Topic revision: r11 - 2020-09-07 - PieterDavid