Table of Contents

Alignment Position Error (APE) Estimator

Complete: 🟩

Goal of this page

Description of the configuration and operation of the tool for estimating the Alignment Position Error (APE).

Introduction

The tool is called ApeEstimator and resides in Alignment/APEEstimation. The code was implemented into the official release in CMSSW_7_6_0. Working versions can be found for CMSSW_7_4_X, CMSSW_7_5_X. It also uses subtools located in Alignment/TrackerAlignment and contains several scripts to run the procedure and to produce validation plots.

Very concise usage instructions

- Set up the tool following instructions from Setting up the Tool below.
- Configure the data set for a skim in `Alignment/APEEstimation/python/samples`. Configure your skim in `Alignment/APEEstimation/test/SkimProducer/skimProducer_cfg.py`. There you include the sample file, used track collection, output file name and folder, and the GlobalTag for your skim.
- Start skimming using `Alignment/APEEstimation/test/batch/startSkim.py`. Provide the option `-s` which is passed to `skimProducer_cfg.py`.
- Create a configuration file for the APE measurement in `Alignment/APEEstimation/test/autoSubmitter` following the instructions in `config.ini`. If you do not have a Design baseline measurement using ideal conditions MC, you need to create one yourself before starting the actual measurement. This is done in the same way as doing a measurement, but adding the option `isDesign=True` to the alignment section of your configuration.
- Start your APE measurement using `Alignment/APEEstimation/test/autoSubmitter/autoSubmitter.py`. The config file you created is given using the option =-c .
- After the autoSubmitter has finished without errors, your results are found in `Alignment/APEEstimation/hists/measurementName/iter14` and `Alignment/APEEstimation/hists/measurementName/apeObjects/apeIter14.db for the APE objects`. Check the `allData.root` file to see if the normalized residuals behave as expected (roughly gaussian and with width 1). You want to do these checks for early iterations while the tool is running so that you can detect mistakes you might have made in the configuration early.
- Result plots are created using `Alignment/APEEstimation/test/plottingTools/drawResults.py`.

Recipe for your favorite CMSSW version

```
cmsrel CMSSW_X_Y_Z
cd CMSSW_X_Y_Z/src
cmsenv
git cms-addpkg Alignment/APEEstimation
git cms-addpkg Alignment/TrackerAlignment
scram b
cmsenv
```

If you want to run quickly the tool with the current configuration without knowing anything about details, you can read only Setting up the Tool, Autosubmitter for multiple parallel measurements and Scripts producing Validation Plots, and read the other parts only if you need them. A very short summary is given here.

Setting up the Tool

After setting up the CMSSW area, do the following:

- `bash Alignment/APEEstimation/scripts/initialise.bash`

The command creates all relevant folders needed for the outputs, thus automated procedures in the scripts.

Now, it is necessary to create a .root-file containing the TTree with all relevant information about the silicon modules of the tracker. This is done by a simple standalone tool named TrackerTreeGenerator, placed in `Alignment/TrackerAlignment`, which uses the ideal geometry. The ideal geometry is chosen, since it guarantees that selections of modules via their position space coordinates chose always the same modules. E.g. TOB modules on the same rod have the same design position in phi, but misalignment could cause a selection choosing only some modules if the cut is by accident selected around the nominal position. It is crucial that the config file for creating the tracker tree is edited to use a global tag with the correct geometry, i.e., phase 0/1/2, depending on the data set one wants to run on. The tracker tree is created with:

- `cmsRun Alignment/TrackerAlignment/test/trackerTreeGenerator_cfg.py`

The .root-file containing the TTree can be found and browsed in `Alignment/TrackerAlignment/hists/TrackerTree.root`, and there it is read from in the APE calculation.

Now, the tool is set up and the procedure for calculating APEs can be configured and started. However, in order to allow fast iterations and parallelisation, a private skim of the files which should be used is created, as explained in the following step.

Creation of Private Skim

Default Creation

In order to allow parallelisation and fast iterations, a private skim of files is created from the AlCaReco files. The event content is minimised to the needs for the ApeEstimator, a tighter preselection is also applied, and the files are split in size for optimising between number of files and number of events inside. Skimming is done using the configuration in `Alignment/APEEstimation/test/SkimProducer/skimProducer_cfg.py`. There, the used track selection is defined in `Alignment/APEEstimation/python/AlignmentTrackSelector_cff.py`, and the trigger selection in `Alignment/APEEstimation/python/TriggerSelection_cff.py` (the trigger selection is disabled by default). The event content is defined in `Alignment/APEEstimation/python/PrivateSkim_EventContent_cff.py`.

Which dataset to process is steered via a configurable parameter using the `VarParsing`, which also allows a local test run. After having run those two scripts for each dataset that one wants to process, the skim is ready for the automated workflow of the APE estimation.

It is recommended use `/test/SkimProducer/startSkim.py` to start skims. This script automatically runs skims, renames the output file to match the _N.root naming scheme, and, if configured in `/test/SkimProducer/skimProducer_cfg.py`, copies the files to a desired location. The script takes `-s` as an argument, where an arbitrary number of sample names is allowed. It is also possible to define sample names with ranges, for example `-s Sample[1-3,ONE,TWO,THREE]` will input the sample names Sample1, Sample2, Sample3, SampleONE, SampleTWO, SampleTHREE which will be passed to `/test/SkimProducer/skimProducer_cfg.py`. Note that for this definition to be parsed correctly, the may be no whitespaces in a sample name definition. For multi-IOV campaigns with many different data sets, a sample name option `iov` is predefined in `skimProducer_cfg.py`, which can then be further parsed to get the correct input and output file names etc, but it is transparent to add further custom namings. The skimming of more than one sample will be done with parallel processes. While it is possible to have an arbitrary number of parallel skims in one session, it is recommended to not skim more than 20 samples per ssh-session in order to quickly finish the skims. Using the option `-n ncores`, one can limit the maximum number of skims to run in parallel (by default, all samples are skimmed at the same time. Generally, it is likely that too many parallel skims will take too much computing time and will be killed by lxplus admins.

Especially for large numbers of parallel skims, it is recommended to use the option `-c` (or `--condor`), which enables the submission of skims to condor. For this purpose, `-C` is a possible switch to submit condor jobs to the CAF queue for prioritzed execution.

Alternatively, the bash script `/test/batch/skimProducer.bash` can be used for skims.

The APE Estimation Tool

In order to allow parallel processing, the tool is based on two different modules. The first one (ApeEstimator) reads the events and gathers all relevant information in several .root-files. The second one (ApeEstimatorSummary) then calculates the APE values afterwards, requiring to merge the files from the first step. Since the method is a local method, iterations are necessary, so the chain needs to be repeated. These steps are all automatically performed by the autoSubmitter. In the following, the configuration of the two modules is explained, the scripts to run are explained in a later section.

In general, all configurations can be done in your final `apeEstimator_cfg.py`, `apeEstimatorSummary_cfg.py` and `apeEstimatorLocalSetting_cfg.py` in `Alignment/APEEstimation/test/cfgTemplate` if one needs to do changes that cannot be configured in `Alignment/APEEstimation/test/autoSubmitter/config.ini`. The `_cfi.py` files in `Alignment/APEEstimation/python` define the configurable parameters, mainly without doing any selection, and should never be changed, they are only templates. The `_cff.py` files in the same folder give the default settings, and should only be changed in exceptional cases.

Usually, all necessary configurations for APE measurements can be made in the `.ini` file that is used to configure the `autoSubmitter`. Therefore, it is recommended to use the config provided to `autoSubmitter.py`, except for in cases where exotic configurations are required. The different configurables and their default values are documented in `test/autoSubmitter/config.ini`.

Configuration of ApeEstimator

The ApeEstimator module is coded in `Alignment/APEEstimation/plugins/ApeEstimator.cc`, having the configuration template in `Alignment/APEEstimation/python/ApeEstimator_cfi.py` with documentation of the configurable parameters, and the default settings in `Alignment/APEEstimation/python/ApeEstimator_cff.py`.

For testing purposes there is one configuration file `Alignment/APEEstimation/test/testApeestimator_cfg.py`, but the general configuration used in the automated workflow can be found elsewhere as explained later.

Event, Track and Hit Selections

The module contains the possibility of a dedicated hit and cluster selection. However, the cluster selection is common for all pixel modules, respectively common for all strip modules. Some selections are applied to both pixel and strip hits. These selections are based on intervals, you need to specify always pairs of numbers to select specific intervals, e.g for one interval (0.3,0.4) or for three intervals (0.3,0.4, 1.8,1.9,-1.7,-1.5). In case of integers, a single number can be selected by e.g. (3,3). If no number is given, no selection is applied.

The track selection is hardcoded and can only be switched on and off. This has historical reasons, and should probably be excluded, and instead the official AlignmentTrackSelector should be used before the applied refit. This would also guarantee, that the track selection is identical during the iterations, since the change of the APE values might lead to small migrations of the track parameters inside/outside the selection window due to the refit.

Some additional selections and configurations can be applied.

Choose between APE Calculation and Control Plots

Furthermore, there are two important switches, since the module can be used for the calculation of APE values, but also as analyzer only, producing zillions of control plots, including plots for track parameters. Calculating APEs is defined in the `cff.py` as the module `ApeEstimator`, setting the switch `calculateApe = True`. Using the analyzer is defined in the `cff.py` as the module `ApeAnalyzer`, setting the switch `analyzerMode = True`. In principle, one could use both things simultaneously in one module, but this often makes no sense due to the sector definitions explained later: the APEs should be calculated for the whole tracker, while the huge amount of detailed validation plots should be chosen for some exemplary regions. The APE calculation also contains some validation plots which are in principle not necessary for the calculation, but since these are the most important basic validation plots, they are implemented there in order to understand the general quality of the estimated APEs.

Granularity of APEs: Sector Definition

A group of modules which should be analysed combined, and for which the same APE value is calculated and assigned, is called "sector". The sectors can be defined based on all module information stored in the TTree produced by the standalone tool mentioned above. The sector definitions need to be given to the ApeEstimator using the parameter `Sectors`, which is a VPSet. An empty template, not selecting anything but defining all selection parameters, is in `Alignment/APEEstimation/python/SectorBuilder_cfi.py`, explaining the possible arguments. Each sector is defined as a PSet, already defined sectors for the subdetectors are given in `SectorBuilder_Bpix_cff.py`, `SectorBuilder_Fpix_cff.py`, `SectorBuilder_Tib_cff.py`, `SectorBuilder_Tid_cff.py`, `SectorBuilder_Tob_cff.py`, `SectorBuilder_Tec_cff.py` in `Alignment/APEEstimation/python`. Further subdefinitions should be built in the same way as shown there. It is important to assign to each sector a name reflecting clearly the exact definition, because this can be found in all .root-files and in printouts and also histogram names, in order to see which sector the results are for. All sector definitions are then gathered in the only file to include, `Alignment/APEEstimation/python/SectorBuilder_cff.py`. There, the two important sector definitions (VPSets) which are used at present can be found, it is `ValidationSectors` for the tool in analyzer mode having the full set of validation plots, and should contain only those sectors where one wants to have a closer look at, and `RecentSectors`, which defines the granularity for the APE calculations, and should span the whole tracker.

Configuration of the Cluster Parameter Estimator (CPE)

The configuration of the CPE which should be used in the refit is given in `Alignment/APEEstimation/python/TrackRefitter_38T_cff.py`. There it is chosen which PixelCPE and which StripCPE should be used. The recent one in use is called `TTRHBuilderAngleAndTemplate` , but of course the parameters can be changed also in the specific `cfg.py`, your configuration. But you need to ensure that it is also included in the refit definition, see below.

Configuration of the Refit

The refit itself is also defined in `Alignment/APEEstimation/python/TrackRefitter_38T_cff.py`. There the CPE has to be specified by its `ComponentName`, which is for the one mentioned above `WithAngleAndTemplate`. Very important parameters which might have an influence on the results are the ones steering the hit rejection (outliers and bad pixel template fits). Again, this can be overwritten in your specific configuration.

For the refitter, a sequence is defined which needs to be included in the `cfg.py`, since the refit also needs the `offlineBeamSpot`. It also contains the selection of tracks flagged as of `highPurity`, since in many alignment tasks only those are selected, and so it is done here. There, one could also apply the track selection instead of within the ApeEstimator, but in the present configuration this is not done, it selects only for `highPurity`.

Configuration of the Geometry and the GlobalTag

The global tag and the geometry need to be specified in the `cfg.py`. But never change the APE, this always has to be the design one with zero APE everywhere. During the iterations of the automated workflow, the correct APE object as created in the previous iteration is taken automatically.

Output

The final output of the ApeEstimator is one file containing the relevant distributions for the second step, the ApeEstimatorSummary, and all validation plots. The output is structured in numerated folders for the defined sectors. Within each folder there is a histogram `z_name`, which contains only the name given to the sector and allows its identification.

Configuration of ApeEstimatorSummary

The ApeEstimator module is coded in `Alignment/APEEstimation/plugins/ApeEstimatorSummary.cc`, having the configuration template in `Alignment/APEEstimation/python/ApeEstimatorSummary_cfi.py` with documentation of the configurable parameters, and the default settings in `Alignment/APEEstimation/python/ApeEstimatorSummary_cff.py`. The module needs as input a file (parameter `InputFile`) produced with the ApeEstimator.

For testing purposes there is one configuration file `Alignment/APEEstimation/test/testApeestimatorSummary_cfg.py`, but the general configuration used in the automated workflow can be found elsewhere as explained later.

Choose between Calculation of APEs or Setting the Baseline from Design MC

When setting the flag `setBaseline = True`, the nominal residual width for each defined sector is estimated. I.e. that not APE values are calculated, but the nominal residual width which returns exactly APE=0 is estimated. This should be done on design MC only, to get this as a reference instead of a fixed assumption of the nominal residual width. A .root-file containing a TTree is created, specified by parameter `BaselineFile`.

If the flag is not set, APEs are calculated. If a baseline .root-file was produced and is specified by `BaselineFile`, then the nominal residual width is read from this file for each sector. If no such file is found, the assumption of residual width equal to 1 is used for each sector. The calculated APE values are also stored in a TTree of a .root-file, specified by `IterationFile`. However, this file is not created newly when you run the tool again, since it is used for iterations of the APE. If it is found, the last stored entry is assumed to be the squared APE as estimated in the previous iteration, and the estimated squared correction is added to it and gives the new APE, which is stored as new entry in the TTree. The APE values for each module contained in a sector are written to an ASCII-file specified by `ApeOutputFile` in the format needed to use the module `Alignment/CommonAlignmentAlgorithm/python/ApeSettingAlgorithm_cfi.py` to create a DB object. The configuration of the tool creating the DB object as it is used during the automated workflow can be found in `Alignment/APEEstimation/test/cfgTemplate/apeLocalSetting_cfg.py`.

Parameters steering the APE Calculation

How the weight of the individual intervals in the residual resolution for the APE calculation within one sector should be estimated is specified by `apeWeight`, where the variant with `"entriesOverSigmaX2"` works best.

The minimum number of hits for using an interval in the calculation is defined by `minHitsPerInterval`.

The parameter `sigmaFactorFit` was used earlier to use a two-step Gaussian fit procedure, but it caused some instabilities due to obvious non-Gaussian behaviour in several intervals. Thus it is not used at all in the present implementation of the code, except for additional plots. In the recent implementation, a Gaussian is fit to each residual distribution in each interval spanning the full range. This second fit should then fit only the core, with +- the specified factor times the width of the first fit around the mean of the first fit.

The parameter `correctionScaling` is used as the damping factor to avoid overestimations and thus convergence problems due to the correlations of the APEs of all modules penetrated by a track. The estimated squared correction is scaled with this factor.

The two parameters `smoothIteration` and `smoothFraction` should not be used and removed from the code. This was another implementation of the damping factor for the iterations, which is mathematically equivalent.

Output

The output of the tool are the following files. There is a .root-file containing a set of validation plots important for the APE estimate. In the mode setting the baseline, the baseline .root-file mentioned above is produced. In the mode for calculating the APE, the .root-file for the iterations of the APE value and the ASCII-file with the APE values used for producing a DB object as mentioned above is produced. Additionally, a root file similar to the one for the iterations is included which contains the default APEs from the global tag before aplying the tool.

Validation plots

In the first iteration (called iteration 0), also validation plots of the analyzer mode of ApeEstimator are created automatically. During the iterations, this is not done, only the relevant things for the iterations are produced. If you specified your last iteration as stated above, again a set of validation plots is created. This allows the comparison and the automated production (see later) of validation plots, comparing the distributions before iterations (with APE=0) to distributions after iterations (final APE as estimated by the tool). The tool is normally used with 15 iterations, so running iteration 0,...14 with the first set of commands, and then running iteration 15 with the second set of commands. This very last iteration (called iteration 15) is not to do another iteration, but to get the validation on the APE after the 15 iterations 0,...14. If you would like to use another number of iterations, it is not a problem, but some of the automated scripts producing validation plots as explained later need to be adjusted.

Be aware that the APE DB object of the iteration called iteration 14 is your final result, and not the APE DB object of the one called iteration 15!

All output except of the DB object is stored in `Alignment/APEEstimation/hists/measurementName/iter*/`, where the * corresponds to the number of the given iteration. The important ones are `iter0` (containing the validations with zero APE), `iter14` (containing the results concerning the estimated final APE values), and `iter15` (containing the validations with the final APE).

The DB objects are stored in `Alignment/APEEstimation/hists/measurementName/apeObjects/`. Always use the one named `apeIter14.db` as final result, the one named `apeIter15.db` is only produced due to the automation.

Autosubmitter to automatically manage measurements

The APE tool includes the autosubmitter in `test/autoSubmitter/`, which handles all steps of an APE measurement and is configured with a `.ini` file. It is possible to stop and resume the autosubmitter in order to add new measurements when others are already running. The autosubmitter is also used to do the required baseline measurements that are necessary for APE determination. For this, one has to use a data set with the flag `isMC=True` and an alignment object with the flag `isDesign=True`. The currently running measurements are stored and updated in a shelve file. Measurements that fail or finished are logged in `/test/autoSubmitter/history.log`. Currently, the following options can be used to run `autoSubmitter.py`:

- `-c <config.ini>`: The config from which the measurements are to be loaded
- `-d <dump.shelve>`: Defines the .pkl file in which the current measurements are to be stored. Can be customized in case one wants to have multiple instances of autoSubmitter.py running in different sessions.
- `-r <dump.shelve>`: Resume measurements from a .pkl file.
- `-k/-p <measurement.Name>`: Kill or purge (kill and remove folders) a measurement with a certain name. For this, you need to also resume the measurement from a .pkl file.
- `-C`: Use this option to enable submission to the CAF queue for faster job execution. For this to work, one has to be in the "cms-caf-alca-TRACKERALIGN" e-group.

Data set configurations start with a section declaration `[dataset:datasetName]`, while alignment object names start with `[alignment:alignmentName]`. Measurements are defined as `[ape:measurementName]`.

Data set configuration:

Option	Description	Default value
baseDirectory	Directory to read files from, usually some /eos directory	has to be defined, can be empty if path is included in fileNames
fileNames	Names of input .root files, separated by whitespaces. on can use ranges [A-B] with integer numbers inside to add multiple files. Multiple ranges per filename are allowed.	is mandatory
maxEvents	Number of events to be analyzed per input file	-1 (which means run over entire input file)
isMC	Determines whether file contains simulated events or data	False
condition	Implemented as "condition record=source tag" or "condition tag:source" in latter case the record name is guessed. Replaces a record with the object found for a given tag in the source. Multiple conditions are possible.	

Alignment object configuration:

Option	Description	Default value
globalTag	Global tag to load	None, will load an auto tag depending on whether data or simulation is used
alignmentName	Name of alignment record to load in apeEstimator_cfg.py	only needs to be be specified if no `condition TrackerAlignmentRcd` is used
baselineDir	Name of directory from which to load baseline for APE measurement	Design

isDesign	Determines whether the alignment object contains an ideal alignment	False
condition	Implemented as "condition record=source tag" or "condition tag:source" in latter case, the record name is guessed. Replaces a record with the object found for a given tag in the source. Multiple conditions are possible. An alignment object needs either an alignmentName or a condition for a TrackerAlignmentRcd. If both are defined, the condition is used and the alignmentName is ignored.	

In conditions, one can use shortcuts for the source. Possible shortcuts are "mpXXXX", "mpXXXX_jobmY", "hpXXXX_iterY", "smXXXX_iterY", and "prod". Further shortcuts can be added as regular expressions in the header of `autoSubmitter.py`.

Alignment measurement configuration:

Option	Description	Default value
dataset	Name of dataset to be used	None, is mandatory
alignment	Name of alignment object to be used	None, is mandatory
firstIteration	First iteration to be run, can be used to rerun cancelled measurements	0
maxIterations	Max number of iterations. For design measurements, this is always 0	15
maxEvents	Number of events to be analyzed per input file. Overwrites whatever is defined in dataset configuration.	from dataset

The different options for the configuration of the .ini file can also be found as comments in `/test/autoSubmitter/config.ini`, which includes an example configuration.

New python scripts for result plots

Result plots can be created using scripts in `test/plottingTools/`. For now, these include `drawResults.py` for the graphical representation of APE measurements and their comparison with other measurements and with default values, and `drawIterations.py` for the graphical representation of the development of APE values over different iterations, and `drawTrends.py` for APE dependency plots against run number or integrated luminosity. The granularity of these plots can be configured in `granularity.py`, where phase 0 and phase 1 geometries are predefined. When plotting trends combining multiple years, one should take care that they have compatible geometries or rewrite the relevant parts of the script to handle the differences. Basic functionality for the representation of uncertainties is provided with `systematicErrors.py`, which can be used to display symmetric and asymmetric uncertainties in result plots, although no systematic studies are available, yet.

The ApeEstimator module runs in Analyzer mode in the iterations 0 and 15, which means that additional validation histograms are created. No plotting tool is available for these in the python-based framework, yet.

Scripts producing Validation Plots

The following scripts are no longer updated, but should still work, given that the results are located in `Alignment/APEEstimation/hists/workingArea`

Scripts for producing validation plots based on root-macros can be found in `Alignment/APEEstimation/macros/`. There are two different scripts to run. First do:

- `cd Alignment/APEEstimation/macros/`

The first script produces all validation plots of iteration 0, and prints them in .pdf-files. This is done for the design MC (the baseline), and for the geometry under study (data or misaligned MC). They are mainly used for optimising the track and hit selection. These validation plots can only be obtained for the sectors which are defined in the analyzer mode module of the ApeEstimator, but there are also files with specific validation plots which are produced for each sector defined in the APE calculation of the ApeEstimator. To produce them run:

- `bash ./apeOverview.sh`

The output is stored in `Alignment/APEEstimation/hists/plots/`, the subfolder `ideal/` contains those of the design MC, the subfolder `data/` those of the geometry under study.

The other script produces single .eps-files containing one histogram each, and overlays them for design geometry, and geometry under study. These are the most important plots, but others could in principle be added in the root-macros. To produce them run:

- `bash ./drawPlotAndIteration.sh`
- `bash ./sortPlots.sh`

The output is stored in `Alignment/APEEstimation/hists/workingArea/iter*/plots/`, where the * stands for 0, 14 or 15, corresponding to the iteration where they are obtained from. They are sorted in subfolders. Important are the following plots in the following subfolders.

- The calculated APE values can be found in `iter14/plots/result/`.
- The important validation plots are in `iter15/plots/Sector/` and `iter15/plots/Track/`.
- For the modelling of data, look at `iter0/plots/Sector/` and `iter0/plots/Track/`.

The last bullet (iter0) is not that important, since the distributions are also overaid in the the one above (iter15), together with the distributions of the final APE. But there the distributions are not scaled to integral, in order to see the absolute number of events/tracks/hits, thus the statistics of the modelling.

If one wants to overlay the resulting APE values for different geometries, this can be done with the corresponding root-macro, no explicit script exists. The macro is `Alignment/APEEstimation/macros/commandsDrawComparison.C`.

Addtional Tools

The following tools are not needed for measuring Ape or creating validation plots but provide extra functions that are useful.

Writing and reading Ape payloads

Using `Alignment/APEEstimation/test/createTrackerAlignmentErrorExtendedRcd_cfg.py`, one can read/write Ape payloads from/to db files. This is useful if one wishes to define a custom Ape based on measurements. The Ape payloads themselves are configured in `Alignment/APEEstimation/macros/writeAPEsInASCII.C`, where one can set the individual Ape values for an arbitrary granularity. These values are then written to an ASCII file that is used by the `cfg.py`.

Writing Ape payload to Tree

In the case that one wishes to plot existing Ape payloads using the Ape comparison tools, these payloads have to be converted to `.root` format first. One possibility is to load one payload as reference in `Alignment/APEEstimation/test/cfgTemplate/apeEstimatorSummary_cfg.py` using ESPrefer. However this requires you to run at least one iteration of a measurement (including sending and retrieving batch jobs) and only allows for one Ape payload to be converted. For a more convenient workflow, `Alignment/APEEstimation/test/apeTreeCreateDefault_cfg.py` can be used. In this file, the used global tag, IOV and Ape tag can be specified along with the desired granularity. Note that all trees that are compared to each other need to have the same granularity with the same sector definitions. The sectors are defined in `Alignment/APEEstimation/python/SectorBuilder_cff.py`.

References

The method used for the APE estimation is described in the following two documents:

- AN 2012/235 (link needs iCMS login)
- PhD Thesis of Johannes Hauk "Measurement of Associated Z-Boson and b-Jet Production in Proton-Proton Collisions with the CMS Experiment" (Chapter 4)

-- JohannesHauk - 24-Jul-2012