

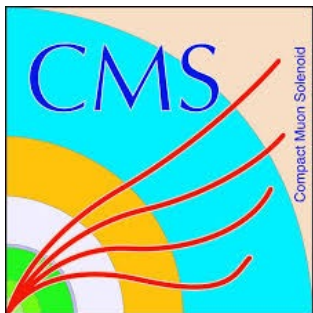
Level-1 Trigger Jet Energy Corrections

Joe Taylor

Method Explainer

01/02/2017

`joseph.taylor@bristol.ac.uk`



Outline of Procedure

INTRO - Information on the code we use and other necessary tools

1. Create Level-1 Trigger specific ROOT ntuples **without** any Level-1 Jet Energy Corrections applied.
2. Match Level-1 Jets with offline reference jets.
3. Derive the Calibrations.
4. Closure Test. Remake the ntuples now using your derived energy corrections. Match the Jets again and validate that the corrections work.
5. Plots for presentations

EXTRA. I will note the key things that are different to Robins initial instructions that I sent you.

INTRO: Code and Tools

Intro: Code and Tools

Tools needed/things we use in the workflow:


- CMSSW workspace
- Github for pulling code / managing code
- CRAB3 for creating the ntuples, here is a good tutorial:
<https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCRAB3Tutorial>
- /hdfs and DICE via soolin

Intro: Code and Tools: Setting up CMSSW


Work on soolin so that we can make use of /hdfs and DICE facilities.

The most recent version of CMSSW that has the relevant information is 8_0_24

Setup the CMSSW workspace on soolin:

```
$ cmsrel CMSSW_8_0_24   
$ cd CMSSW_8_0_24/scr  
$ cmsenv
```

Now add the L1Trigger specific package within this workspace:

```
$ git cms-init  
$ git remote add cms-l1t-offline git@github.com:cms-l1t-offline/cmssw.git  
$ git fetch cms-l1t-offline   
$ git cms-merge-topic --unsafe cms-l1t-offline:l1t-integration-v89.20  
$ scram b -j 8
```

**THIS WILL DEFINITELY BE DIFFERENT.
BUT THE VERSION WE NEED
DOES NOT EXIST YET**



(The final line compiles the package, it can take a little while!)

CMSSW_8_0_24 and v82.0 in bold as they will change with time and before doing any work must ensure they are valid for what we are doing!

Intro: Code and Tools: Setting up L1JEC code

We place the JEC specific code that we will use within this L1Trigger package in CMSSW

```
$ cd ~/CMSSW_8_0_24/src/
```

```
$ git clone git@github.com:joseph-taylor/L1JetEnergyCorrections.git L1Trigger/L1JetEnergyCorrections
```

this also needs compiling

```
$ scram b -j 8
```

(need to do this if your ever update the .cpp code)

Create your own offline remote for the repository on github, and push changes to there. If you make a significant change to the code you can send me a Pull Request so that I can merge the updates to my repository.

NB: there is a lot of code in this repository. Have a look around. Luckily we only need to use a small fraction of it :)

Intro: Code and Tools: Setting up htcondenser

This is a library written by ex-student and L1JEC maestro Robin.
It is just a framework of his for submitting HTCondor jobs @ Bristol
Keep it separate from your CMSSW project area.

```
$ git clone git@github.com:raggleton/htcondenser.git ~/htcondenser  
$ cd ~/htcondenser  
$ source setup.sh
```

(IMPORTANT: Need to source it anytime we are going to put Jobs on DICE in this workflow

STEPS 2,3,4,5 need it!!!!!!)

Step 1: Creating the Ntuples

Step 1: Creating the Ntuples (Intro)

First a good taster of how grid jobs work using crab3:

<https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCRAB3Tutorial>

In short,

You have a CMSSW configuration file written in python, e.g. `cmsswConfigFile.py`, this will do some task like produce the ntuples from a CMS dataset.

Running `$ cmsRun cmsswConfigFile.py` will do this task locally.

What Crab3 does is scale that up and run it on the grid.
Using Crab3 requires its own config file to tell it what to do.

NB: We shouldn't need to tinker with this stuff too much, but it will be a handy thing to know about for any future CMS work.

Step 1: Creating the Ntuples (Turn off JEC)

We want to derive our L1 Jet Energy Corrections on Level-1 Jets that **do not** have any corrections applied. Thus before creating the ntuples we must turn off the L1JECs. This is done by hacking slightly with the CMSSW code...

**THIS FILE DOES NOT EXIST YET.
it will be in a later integration tag**

Within CMSSW_8_0_24/src, Look at the file:

L1Trigger/L1TCalorimeter/python/calStage2Params_2017_vXYZ_cfi.py

(*vXYZ is some currently undetermined unique identifier for the most recent version*)

As a sanity check ensure that L1Trigger/L1TCalorimeter/python/hackConditions_cff.py is pointing to the calStage2Params_2017_vXYZ_cfi.py configuration rather than an old one

Lines 69-73 is a section called calibration options.

Comment out the line:

```
calStage2Params.jetCalibrationType = cms.string("LUT")
```

(*This was for corrections being applied in the style of the firmware LookUpTable.*)

And un-comment:

```
#calStage2Params.jetCalibrationType = cms.string("None")
```

recompile with (not 100% if this is necessary, but definitely not wrong)

```
$ scram b -j 8
```

And you have turned off the current L1JECs.

Step 1: Creating the Ntuples (CMSSW config 1)

There are a few different CMSSW config files for creating ntuples in
L1JetEnergyCorrections/python/

The most recent python CMSSW configuration file for making ntuples derived from
Monte Carlo is:

L1JetEnergyCorrections/python/l1NtupleMcMaker2017_RAW2DIGI.py

Have a look in these files, there is a lot of random stuff going on. But look on line 5, this
tells you how I created the file, just by running the following cmsDriver command.

```
$ cmsDriver.py -s L1REPACK:FullMC,RAW2DIGI --  
python_filename=l1NtupleMcMaker2017_RAW2DIGI.py -n 100 --no_output --no_exec  
--era=Run2_2016 --mc --conditions=80X_mcRun2_asymptotic_2016_miniAODv2_v1 --  
customise=L1Trigger/L1TNtuples/customiseL1Ntuple.L1NtupleRAWEMUGEN_MC --  
filein=/store/mc/RunIISpring16DR80/  
QCD_Pt-15to3000_TuneCUETP8M1_Flat_13TeV_pythia8/GEN-SIM-RAW/  
FlatPU20to70HcalNZSRAW_withHLT_80X_mcRun2_asymptotic_v14-  
v1/40000/0002348A-1664-E611-81DA-0CC47A4C8F0C.root
```

Step 1: Creating the Ntuples (CMSSW config 2)

We will initially use MC to derive our corrections, so let's focus on the CMSSW config:
L1JetEnergyCorrections/python/l1NtupleMcMaker2017_RAW2DIGI.py

NB: This config file needs updating to rolls with the times as you change CMSSW version and l1t-integration-branch choice. We won't worry about this now as it is all up to date. But it is something to be aware of when calibrating again in the future. *It would just be a matter of running a new cmsDriver.py command within your new CMSSW version + l1t-integration-branch*

Before running on CRAB let's check that this code works locally
\$ cmsRun l1NtupleMcMaker2017_RAW2DIGI.py

Does it run without failing? Does it produce a ROOT file? Have a look inside the ROOT file, is it filled? (Also take a moment to familiarise yourself with the objects inside it. Information about the *emulated* L1jets is in l1UpgradeEmuTree and info about the jets is in l1GeneratorTree)

(BTW you might need to load your grid certificate to get the above to work
\$ voms-proxy-init -voms cms --valid 168:00
as the CMSSW config uses remote data)

Step 1: Creating the Ntuples (CRAB3:one)

Before using crab3 you need to load your grid certificate, running:

```
$ voms-proxy-init -voms cms --valid 168:00
```

Will give you a valid one for a week.

Then you need to source crab itself

```
$ source /cvmfs/cms.cern.ch/crab3/crab.sh
```

Okay, so Robin has put a bit of a wrapper around the crab3 config file, which we will use to submit our jobs to the grid. I think it perhaps just adds additional confusion, but anyway... how to use it...

The python script in question is:

```
L1JetEnergyCorrections/crab/crab3_stage2.py
```

you run it with (**but don't run it just yet**)

```
$ python crab3_stage2.py
```

This is different to the normal way of submitting crab jobs.

(But then you do have to use ordinary crab commands to check on the jobs and resubmit failed jobs. So do have a look at the crab tutorial!!!)

But before you submit...There are a few things we need to change first

Look at the file: L1JetEnergyCorrections/crab/crab3_stage2.py ...**PTO**

Step 1: Creating the Ntuples (CRAB3:two)

Line 27: update the job_append variable


job_append = "qcdSpring16_genEmu_1Feb2017_8024v89p20_noJEC_086b699"

This tag describes what we were doing in our crab3 job

This should include the dataset we run over = qcdSpring16

Date = 1Feb2017

CMSSW and l1t integration tag = 8024v89p20

beginning most recent git commit hash = 086b699 (so you can trace back the state of the repository) 

nb: crab job names can only have 100 character's, putting a limit on how long job_append can be

Line 18: ensure that the PY_CONFIG variable points to the correct CMSSW config.

PY_CONFIG = '../python/l1NtupleMcMaker2017_RAW2DIGI.py'

line 33ish: set the datasets variable. This variable is a *key* to the full dataset name we want to use. The corresponding full datasets are set in dictionaries in:

L1JetEnergyCorrections/python/mc_samples.py

(this bit is currently set up correctly so don't worry about it for now, in the future you may wish to use a different dataset to run over, in wish case we can discuss this in more detail)

Step 1: Creating the Ntuples (CRAB3:three)

Okay, so let's run it with
\$ python crab3_stage2.py



This will now take ~1 day to run and come back. Check on it / resubmit failed jobs using traditional crab3 commands, these are all in the tutorial I pointed you to.

The output will end up on
/hdfs/dpm/phy.bris.ac.uk/home/cms/store/user/<cernUserName>/...

You then want to use hadoop file system commands

<https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html>

To copy the output ROOT files into a L1JEC specific area in:

/hdfs/L1JEC/CMSSW_8_0_24/

NB Look in /hdfs/L1JEC/CMSSW_8_0_9/ as an example of what it looked like last year.

NB you may not have permission to write to the /hdfs/L1JEC area. But Winnie/Luke could fix that:)

Step 2: Matching

Step 2: Matching (1)

We only care about level-1 jets and the corresponding offline jet (eg the MC genJet). They are matched using dR.

We don't care about which event the Jets come from, so in this stage we create ROOT files where each entry contains info about a single L1Jet and the corresponding offline jet only.

The output file name will be pairs.root

The source code for this stage (if using MC datasets) is:

L1JetEnergyCorrections/bin/RunMatcherStage2L1Gen.cpp


But we don't need to touch this code.

What we do play with is the code:

L1JetEnergyCorrections/bin/HTCondor/submit_matcher_dag.py

This script allows us to play with the settings and it submits the necessary jobs to DICE for us (*automatically saving the output in the ~same /hdfs location as the input ntuples*)

Look inside the code...

Line 39: NTUPLE_DIRS variable. Set this to your ntuple input you just created with crab in stage 1. 

Other variables include:

DELTA_R, PT_REF_MIN, in future you may want to play with these, but they are good as they are.

Step 2: Matching (2)

To submit the jobs enter the HTCondor directory (you have to be in the directory of the code for this to work) and run:


```
$ ./submit_matcher_dag.py
```

JobLogs/JobInfo will be found in the following:

```
/storage/<bristolUserName>/L1JEC/<cmsswVersion>/L1JetEnergyCorrections/jobs/pairs/<date>
```

Here one thing you should find is a .status file. If you run:

```
$ DAGstatus.py pathTo/XYZ.status
```

you will get some nice information about how the jobs are going. 

All going well you should get pairs ROOT files in ~1hr.

You will also find a .dag file in the same directory.

NB: these are DAG type jobs, i.e. some jobs only begin once another set of jobs are completed.

If a job(s) in the chain fails and you wish to resubmit, do the following:

in the HTCONDOR directory again

```
$ condor_submit_dag pathTo/XYZ.dag
```

(You may have to kill the controlling DAG jobs before resubmission, find it via \$ condor_q -dag)

TIP: to save time and effort looking for the .status they are printed to the terminal when you submit the job. So make a note of them then!

Step 3: Derive Calibrations

Step 3: Derive Calibrations (1)

We first run on DICE to get the correction graphs (as a function of pt) for each $|\eta|$ bin. After that we fit the correction curves locally to these graphs (they sometimes need a bit of massaging before they are right, so we don't do it on the initial step)

The main code is in:

L1JetEnergyCorrections/bin/runCalibration.py

But again we don't touch this and instead use:

L1JetEnergyCorrections/bin/HTCondor/submit_runCalibration_dag.py

With this we can alter input and submit the jobs to DICE

You want to set the PAIRS_FILES variable to match your output ROOT files from the matching in step2.

You can also play about with the PU_BINS which is actually set at the bottom of:

L1JetEnergyCorrections/bin/binning.py

We only want to correct the L1 Jets for a PU range which will correspond to what is expected for the years run. e.g. PU 40-50

We will need to find out what PU we expect for 2017.

Step 3: Derive Calibrations (2)

Run on DICE by doing the following in the HTCondor directory again:

```
$ ./submit_runCalibration_dag.py
```

It will take ~1hr again.

Logs and Info quite like before are located here:

```
/storage/<bristolUserName>/L1JEC/<cmsswVersion>/L1JetEnergyCorrections/jobs/calib/<date>
```

Output will be in ~same /hdfs location as the ntuples and pairs files.

After the jobs are done:

Open the output ROOT file for the PU bin we wish to correct for (eg PU 30-40 version). Then look to check that the correction graphs and gaussian fits look good. There will be a lot of gaussian fits!!

NOW copy the output ROOT file locally to a sensible place /users on soolin. It is okay to do so as this ROOT file is only small. You want to use a copy also because we are going to meddle with it and don't want to risk screwing the original if it goes badly.

Step 3: Derive Calibrations (3)

****So these next few slides probs represent the most confusing steps of the process****

We run the following

```
$ python pathTo/runCalibrations.py <local_input_rootFile> <local_output_rootFile> --redo-corrections-fit --inherit-params --stage2
```

<local_input_rootFile> is what you have just copied locally onto /users

<local_output_rootFile> is the output of the process (If you don't set it, it will append itself onto the input, hence how you can screw with the original file and why we make a copy)

The output ROOT file should now have fitted a function to the correction graph.
Open it up and check it out.

Few things to look out for:

For higher $|\eta|$ bins the graphs/fits start to degrade and are over a smaller range
Most graphs have a low p_t 'flick' which the function can never really capture

Great, we have the correction curves!!! But, could we massage them to be a bit better...

Step 3: Derive Calibrations (4)

This is a little thing I wrote to massage the fits. It's a bit weird but works;p It re-fits starting from the parameters it had in the last fit but for slightly different ranges that you can set.

You run it interactively in a ROOT session:

```
$ .L L1JetEnergyCorrections/bin/local_L1JEC_scripts/tuneFits.h+
```

Load up a given fit function do the following:

```
$ t0.setup("inputRootFileWithTheCorrectionFits.root", etaNames[0])
```



You should have the plot pop up on screen

to redo the fit with a different start point and end point, relative to where they are now:

```
$ t0.redoFit( $\Delta_{xmin}$ ,  $\Delta_{xmax}$ )
```

can do this line multiple times

to save and replace the original fit do the the following

```
$ t0.save()
```

instead exit root session if you have screwed it up and don't want to save!!!

Loop through all the |eta| bins by changing the **0** to 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Step 4: Closure Test

Step4: Closure Test: Updating Corrections in CMSSW (1)

We now need to re-make the ntuples with our new correction functions put in the CMSSW emulator.
How do we do that?

An example of how I have done this in the past can be seen in:
`L1Trigger/L1TCalorimeter/python/calorStage2Params_2016_v3_0_inconsistent_cfi.py`

We will need to insert the same things into the new version which we hacked earlier:
`L1Trigger/L1TCalorimeter/python/calorStage2Params_2017_vXYZ_cfi.py`

KEY THINGS:

line73:
`calorStage2Params.jetCalibrationType = cms.string("functionErf11PtParams16EtaBins")`
This set the corrections according to what we have going on on lines 112-140

We will need to do the same in the new version of the config file

Remember that earlier, when we wanted no calibrations, that we had it set as
`calorStage2Params.jetCalibrationType = cms.string("None")`

lines112-140:
Explains and sets the the correction functions for this style of function. See PTO

Step4: Closure Test: Updating Corrections in CMSSW (2)

lines 112-140: **We will need to insert this into the new config file but with a new set of numbers**

```
# vector with 11 parameters for each eta bin
# each eta bin represented by a new line, starting at first eta bin
# the first seven parameters of a line are the 7 parameters for the correction function
# the next four parameters are:
# 1. the function value at the minimum pt of fit
# 2. the minimum pt of the fit
# 3. the function value at the maximum pt of the fit
# 4. the maximum pt of the fit
jetCalibParamsVector16 = cms.vdouble(
jetCalibParamsVector16.extend([
0.661201, -1.08715e+06, -2.59519e-08, -10.8044, -1.04431e-06, -1.358, 0.830229, 1.9517, 21.7232, 1.11497, 308.136,
0.595461, -242991, -5.38e-08, -31.5184, -8.65299e-06, -0.963764, 0.382539, 2.0419, 22.4169, 1.13073, 304.49,
2.55569, -1.76867e+06, 6.783e-08, -8.05711, -3.97612e-07, -2.83151, 1.1268, 2.00221, 20.7898, 1.14923, 248.034,
2.96018, -4.20761e+06, 7.63023e-08, -2.55568, -9.93322e-08, -6.08005, 1.29864, 2.04699, 17.3628, 1.20751, 193.13,
3.08393, -4.49833e+06, 8.14721e-08, -2.23442, -8.46574e-08, -7.22304, 1.32952, 2.03851, 21.509, 1.19884, 211.281,
2.1004, -2.69476e+06, 4.91209e-08, -4.5254, -2.40174e-07, -3.2305, 1.21127, 1.97181, 16.7042, 1.06621, 273.416,
2.14597, -2.8508e+06, 5.17083e-08, -4.21701, -2.28694e-07, -3.40691, 1.18177, 1.98222, 15.4658, 1.05329, 242.235,
2.35967, -3.55304e+06, 6.32171e-08, -2.95984, -1.3141e-07, -4.07551, 1.21547, 1.83235, 15.7161, 1.03703, 190.26,
0.856007, -364884, -1.29766e-08, -22.9827, -2.11044e-06, -2.01542, 1.02893, 1.76636, 18.165, 1.00783, 273.301,
3.21455, -286314, 3.16865e-07, -19.3786, -1.14632e-06, -4.6408, 1.33054, 1.46451, 21.5884, 0.987992, 246.126,
-2.88427, -21731.7, -1.44395e-06, -106.655, -1.78369e-05, -3.42763, 1.28446, 1.37317, 21.5443, 0.983075, 288.211,
-0.787852, -12400.7, -8.08595e-07, -152.002, -2.55708e-05, -3.91554, 1.28934, 1.27682, 27.3396, 0.962841, 225.501,
-1.90211, -17583.4, -1.16672e-06, -119.776, -8.63159e-06, -7.56003, 1.45489, 1.06435, 36.2542, 0.929621, 112.707,
-74.6452, -258031, -1.76366e-05, -11.9507, -3.53722e-05, -0.493146, 0.593305, 1.11432, 46.5187, 1.03823, 1024,
0.568061, -7606.13, -5.39389e-07, -150.07, 4.85322e-05, -2.42215, 2.07552, 1.11686, 27.012, 0.967464, 1024,
-0.819561, -12223.5, -7.93953e-07, -149.116, -3.22883e-05, -5.31212, 1.26463, 1.23498, 24.8899, 0.840316, 186
])
# this vector corresponds to "functionErf11PtParams16EtaBins"
caloStage2Params.jetCalibrationParams = jetCalibParamsVector16
```

Step4: Closure Test: Updating Corrections in CMSSW (3)

So how do we get the numbers from the fit functions to insert into the CMSSW config file?

Look at:

L1JetEnergyCorrections/bin/local_L1JEC_scripts/writeParams.cxx

Set the input ROOT file containing the correction functions

Set the output file which will be a .txt file with the info we need to put in the CMSSW code
(It will be at the bottom of the file)

And run with

```
$ root -q -b -l pathTo/writeParams.cxx
```

THEN

Put these in the emulator

Recompile with `$ scram b -j8`

repeat STEP 1: Making the ntuples (will now use our corrections)

repeat STEP 2: Matching the L1Jets with MC genJets (using ntuples which now have the corrections)



All we need now is to make some nice plots to convince TSG what a great job we've done

Step 5: Pretty plots for presentations

Step5: Pretty plots for presentations (1)

Plots made by running the script

L1JetEnergyCorrecitons/bin/showOffPlots.py with various different extensions.

You can see the various extensions available on lines 840-867 of the code

First off we are going to want plots of our correction curves.

```
$ python showOffPlots.py --calib <localRootFileWithCorrectionFits> --title <titleForThePlots> --oDir <outputDirectoryPath>
```

Finally we are also going to want to look at the results before and after the calibrations.

To do this we need to run one more bit of code on the matched jets pairs file.

We will run it both on the pairs root file for no calibrations and also for the pairs root file we have just made that include our new calibrations.

Then we make some simple plots from the output of these jobs, and jobs a good'un

(we may have to additionally make turnOn curves, which will involve using a package by Shane from IC, but we will save this for another time)

Step5: Pretty plots for presentations (2)

The final bit of code is:



```
L1JetEnergyCorrections/bin/checkCalibration.py
```

Again we don't touch this and instead run it on DICE using:

```
L1JetEnergyCorrections/bin/HTCondor/submit_checkCalib_dag.py
```

It is just like the other htcondor things we have already been running:

Make sure the pairs file in this code is correct

and then run from the HTCondor directory as

```
./submit_matcher_dag.py
```

outputs on /hdfs

to get the pdf's out of it...

```
$ python showOffPlots.py --checkcal <rootFileWeJustCreated> --title <titleForThePlots> --oDir  
<outputDirectoryPath>
```

Do it for all the PU bins as we want to see how our corrections work on different PU bins also (ie not just the one we corrected the jets for).

bosh. we are done.

EXTRA: things different to Robin's Slides

EXTRA:

Robins slides say we match Level-1 Jets and MC genJets with $dR < 0.4$
This has been tightened to 0.25 and that is what's currently in the setup.

Robins slides say there are 11 $|\eta|$ bins (each 4 trigger towers wide)
We now use 16 of them (the max allowed in the firmware).
NB: They are no longer composed of the same longer of trigger towers