Level-1 Trigger
Jet Energy Corrections

Joe Taylor
How to do it !!!

joseph.taylor@bristol.ac.uk

Outline of Procedure

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

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. LUTs and Closure Test. Remake the ntuples now using your derived energy corrs. Match the Jets again and validate that the corrections work.

5. Plots for presentations

EXTRA. A 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 *(somewhere that DICE can read from)*:
$ 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

(Some of these commands can take a little while!)

IMPORTANT: 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.
Fork the package git@github.com:joseph-taylor/L1JetEnergyCorrections.git on GitHub.
Then clone it into CMSSW.

$ cd ~/**CMSSW_8_0_24**/src/
$ git clone git@github.com:<username>/L1JetEnergyCorrections.git L1Trigger/L1JetEnergyCorrections

this also needs compiling
$ scram b -j 8
(you need to do this if your ever update the .cpp code)

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.
Note that his most recent version is not compatible with the L1JEC anymore.
I have a legacy version on my gitHub you should use instead.

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

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 within CMSSW_8_0_24/src…

1. There will be a param file containing the layer-1 corrections. Let's call it
   L1Trigger/L1TCalorimeter/python/caloStage2Params_2017_vXYZ_cfi.py

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

2. On ~Lines 69-73 is a section called calibration options.
   Comment out the line:
   caloStage2Params.jetCalibrationType = cms.string("LUT")
   (*This was for corrections being applied in the style of the firmware LookUpTable.*)
   And un-comment the following:
   #caloStage2Params.jetCalibrationType = cms.string("None")

3. Next ensure that L1Trigger/L1TCalorimeter/python/hackConditions_cff.py is pointing to the caloStage2Params_2017_vXYZ_cfi.py configuration (~Line 46)

4. Finally recompile with
   $ cd $CMSSW_BASE/src
   $ scram b -j8 (and you have turned of the JECs)

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

IMPORTANT: This config file needs updating to roll with the times as you change CMSSW version, l1t-integration-branch choice and MC sample. It is something to be aware of when calibrating again in the future! *It would be a matter of running a similar cmsDriver.py command within your new CMSSW version + l1t-integration-branch for a new MC input file*

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 genJets 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 to 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 I1t 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.
eg.
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

If you want to add a new dataset you need to add an entry to the dictionary in mc_samples.py

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 L1JetEnergyCorrections/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 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 log 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:
$ 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 run locally and fit the correction curves to these graphs.
Finally, these fits sometimes need massaging so I have developed an interactive method of doing that.

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!!

Next copy the output ROOT file locally to a sensible place 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 soolin
<local_output_rootFile> is the output of the process (If you don't set it, it will append the input file, 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 pt '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 an interactive way of massaging the fits I have developed.
Again I would make a copy of the new ROOT file with the fits that we just made. Work on this new file.

You run it interactively in a ROOT session:
$ .L L1JetEnergyCorrections/bin/local_L1JEC_scripts/tuneFits.h+

Load up a given fit function by doing the following:
$ tuneFits t**0** = tuneFits("fileNameToTune.root", **0**)
You should have the plot pop up on screen.
To redo the fit with a different start point and end point, relative to the initial fit:
$ t**0**.redoFit($\boldsymbol{\Delta}_{xmin}$, $\boldsymbol{\Delta}_{xmax}$) —>*can do this line multiple times*
to save the fit do the following
$ t**0**.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
You need to go through and save every fit function, even if you don't end up changing the fit at all.
NB: this is an easy tool to use but it probably best shown in person.
There is a little bit more info written in the script itself

Step 4: L.U.T's & Closure Test

Step4: LUTs:

to make the LUTs you need an environment with MATPLOTLIB

First we need a txt file with all the fits params in the ROOT file we have just worked with
$ root-q -b -l $CMSSW_BASE/src/L1Trigger/L1JetEnergyCorrections/bin/local_L1JEC_scripts/writeParamsForLUT.cxx
(change the input and output files in the script first)

Next, we feed these into to Human Readable LUT making script
$ python …/bin/correction_LUT_plot.py pathTo/paramsTextFile.txt pathToOutputDir/lut_HR.txt --stage2 --plots --text --ptCompressionFile lut_pt_compress.txt
(copy lut_pt_compress.txt from cms-l1t-offline/L1Trigger-L1TCalorimeter/lut_pt_compress.txt)

The output comes with a load of plots that should be checked to see if everything has gone okay.
We now have HR eta, pt and add_mult LUTs. All of which need converting to other formats.

To convert to firmware LUTS:
$ python …/bin/mif_maker.py lut_HR_pt.txt lut_pt.mif

To convert to XML LUTs:
$ python …/bin/programmable_lut_maker.py lut_HR_pt.txt lut_pt.xml

(also need to do for eta and add_mult LUTs)

Step4: Closure Test

THEN
Pass LUTs on for someone to make a new integration tag with

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: The composition of the |eta| bins is no longer uniform.