Table of Contents

Noise Bias Scan

A lot of this work was done previously by James Orcutt; he wrote the scripts that I use currently and tried fitting procedures and analysis. A link to his twiki can be found below.

Here is a list of useful information relating to the noise bias scan:

- Radiation damage talk

- Another Radiation Damage talk

- Run procedures for HV scans

- James' twiki on Noise Bias

- Dissertation on the strip tracker (page 59 for noise)

- List of HV runs so far

- Strip readout

- Here is a talk I did on the Noise Bias Scan

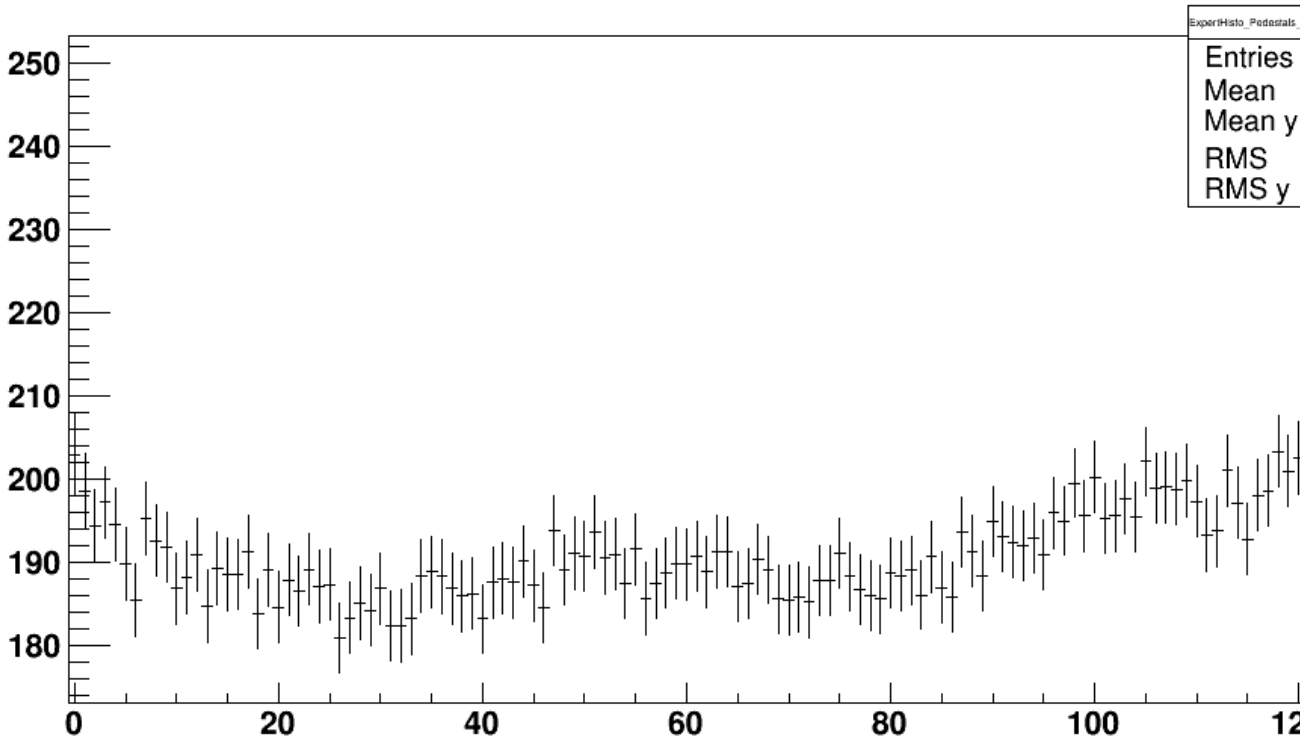
Introduction

The Noise Bias Scan is a way of keeping track of radiation damage in the detector and its effects on power consumption and performance. The way this is done is by keeping track of the full depletion voltage, or Vdep over time on modules of the tracker. Each module (modules hold the APVs, which hold 128 strips each) of the tracker has this full depletion voltage. A module is said to be "fully depleted" when the "noise" in the strip readout is at a minimum. To see what noise is, we have to look at a few other ideas.

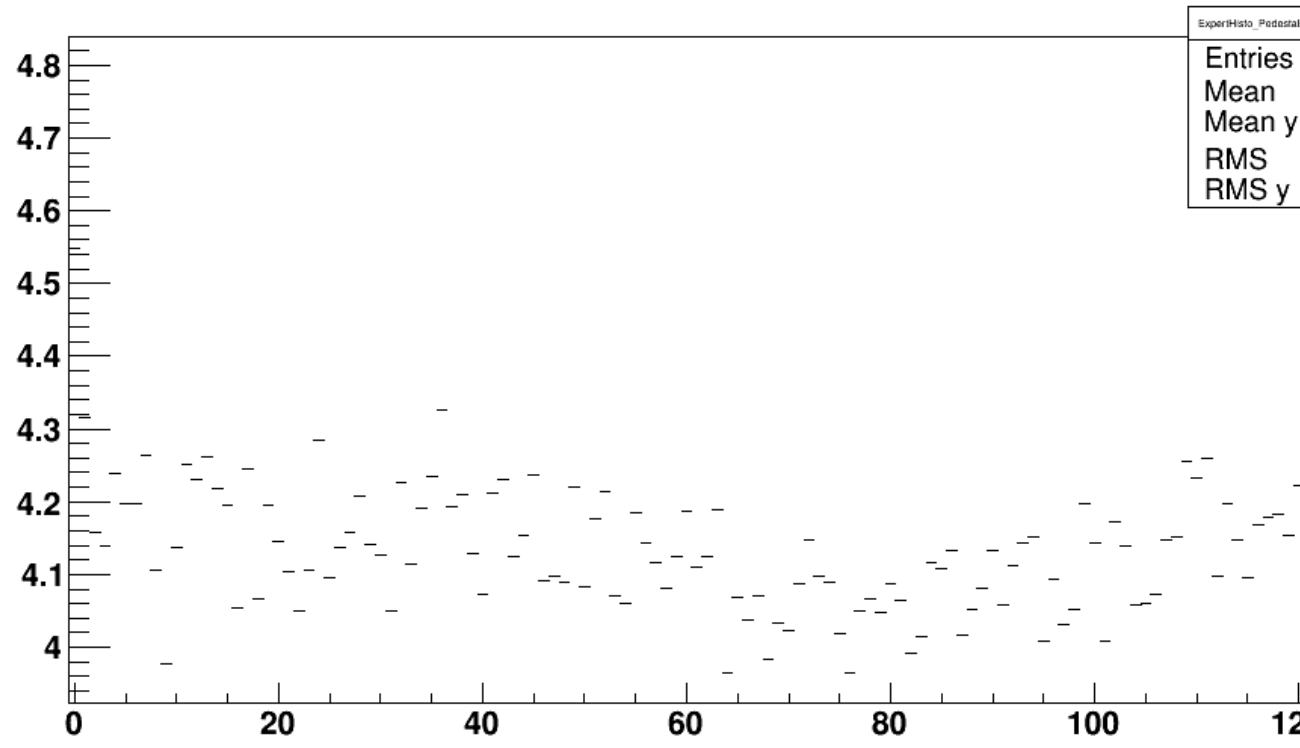
When the silicon strip tracker takes data, it reads out from all strips whenever a particle passes through the silicon and deposits charge on the strip. It looks at the charge on each strip and transmits only the strips which were determined to have a particle pass through them. To do this the module has to act like a capacitor, so there is high voltage put on one side, the other grounded. With the silicon having such a voltage however, there tends to be some electrons getting deposited on the strips anyway when the voltage is on. This is a property of the silicon and cannot be avoided, but we do look to minimize this effect. So when the tracker is turned on and a voltage is applied, it starts reading out data even though there are no colliding beams, no cosmics, etc. This opportunity can be used to see what the background level of readout is so that it can be subtracted from any data taken and more accurately know when a particle has actually passed through a strip. We keep track of this data in what are called pedestal runs.

Bias scan runs are pedestal runs done by starting the detector at a low voltage (say 30v) and taking a certain number of events readout per strip (say 2000) of a partition of the tracker at that voltage point. Then the voltage is ramped up by around 15V and the data is taken again. This continues to 350v. This is what makes the Noise Bias Scan a "bias scan". It scans different values of the bias voltage on the detector for noise data.

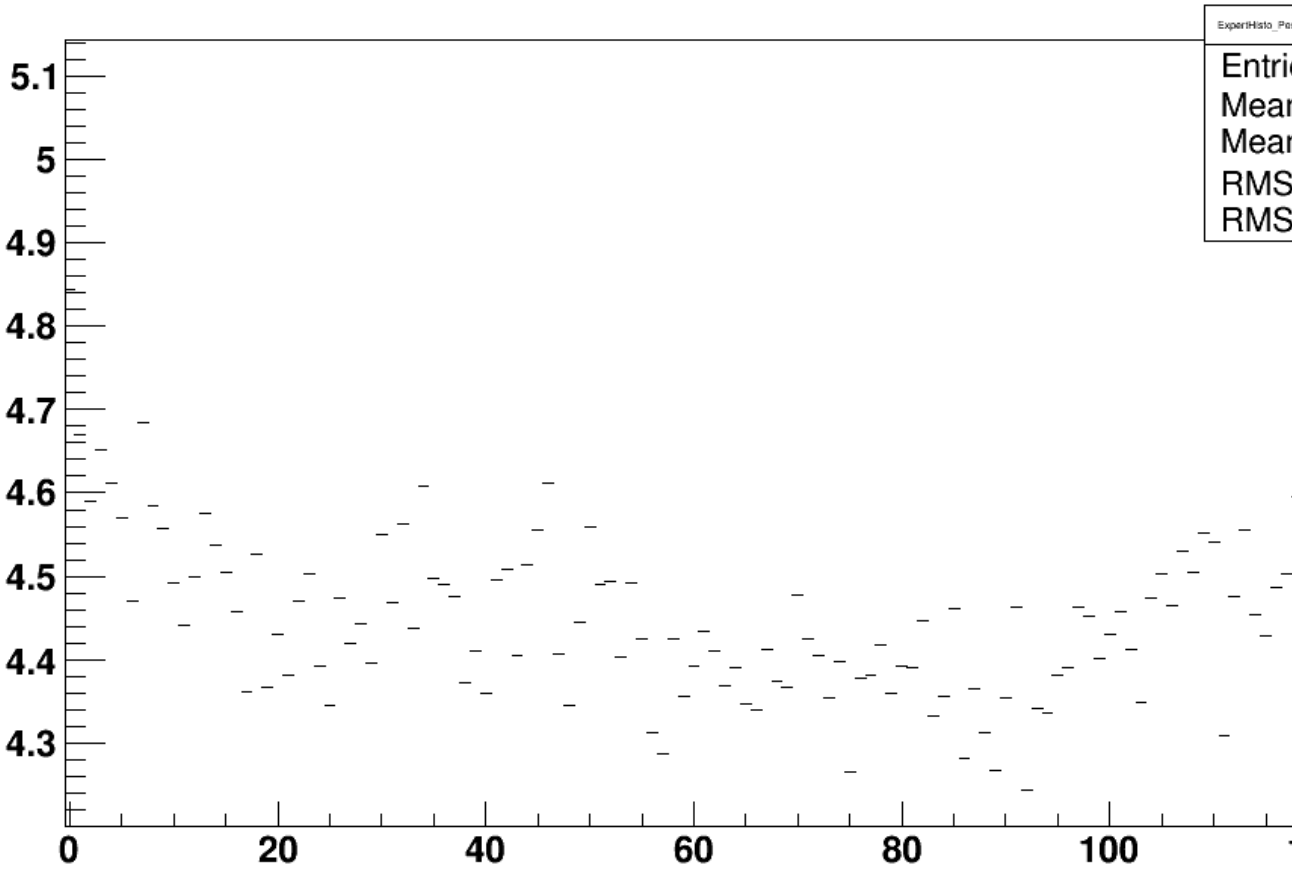
This is what a pedestal plot from one voltage point of the pedestal run looks like:



This is plotting ADC counts (analog to digital converter) vs. strip #, on one APV (128 strips). This plot is for the highest voltage point, 350V. As you can see, each of the strips are reading out a couple hundred ADC counts. The reason for the 512000 entries but only 128 strips is that I hid the other half of the plot. Normally it would be 256*2000 = ~512000. This pedestal plot can take us directly to the noise plot: the noise is the RMS of the pedestal.

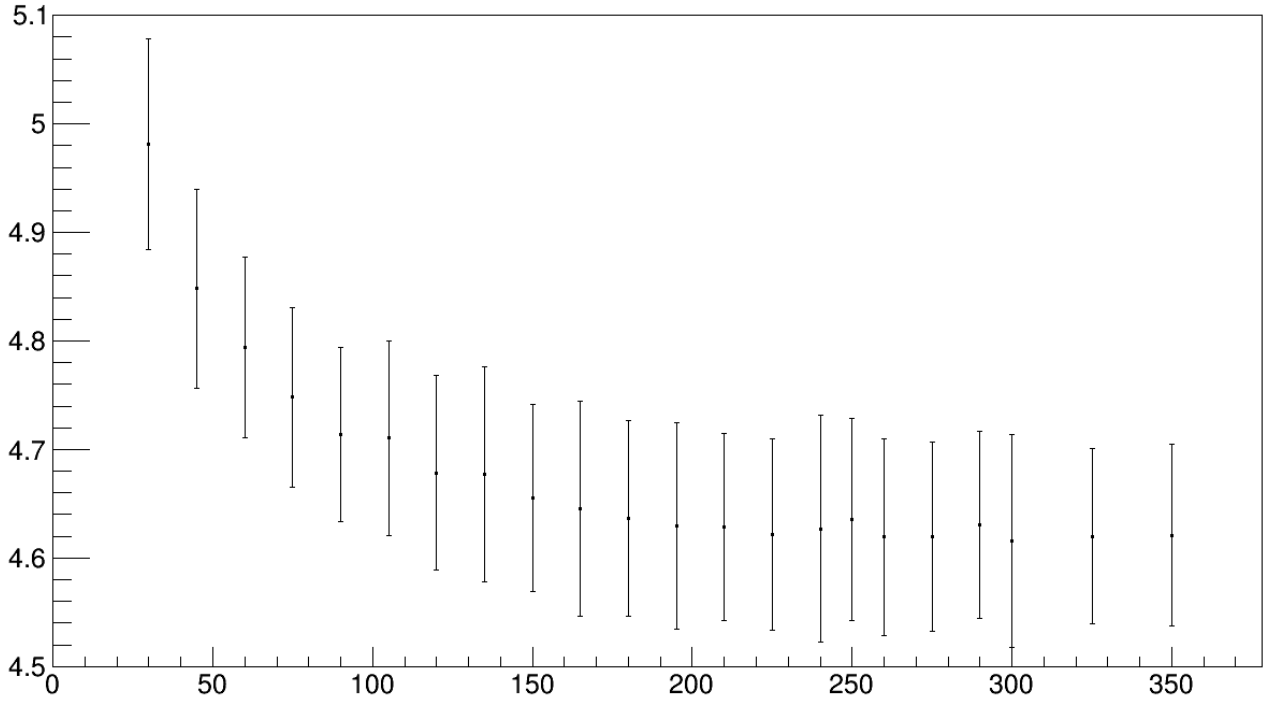


Here is the noise plot for that same APV. This is the plot that will be minimized when the voltage is highest, since the noise is minimized when the voltage is highest. Since the noise is the RMS of the mean, that means that the "noisy" behavior of the noise gets less at a higher voltage; the deviation of each strip's reading from the mean is less. Here is a plot of the same APV, but at 30V:



You can see that the mean is around 4.5 as opposed to 4.2. Higher average noise is what we would expect from a lower voltage value.

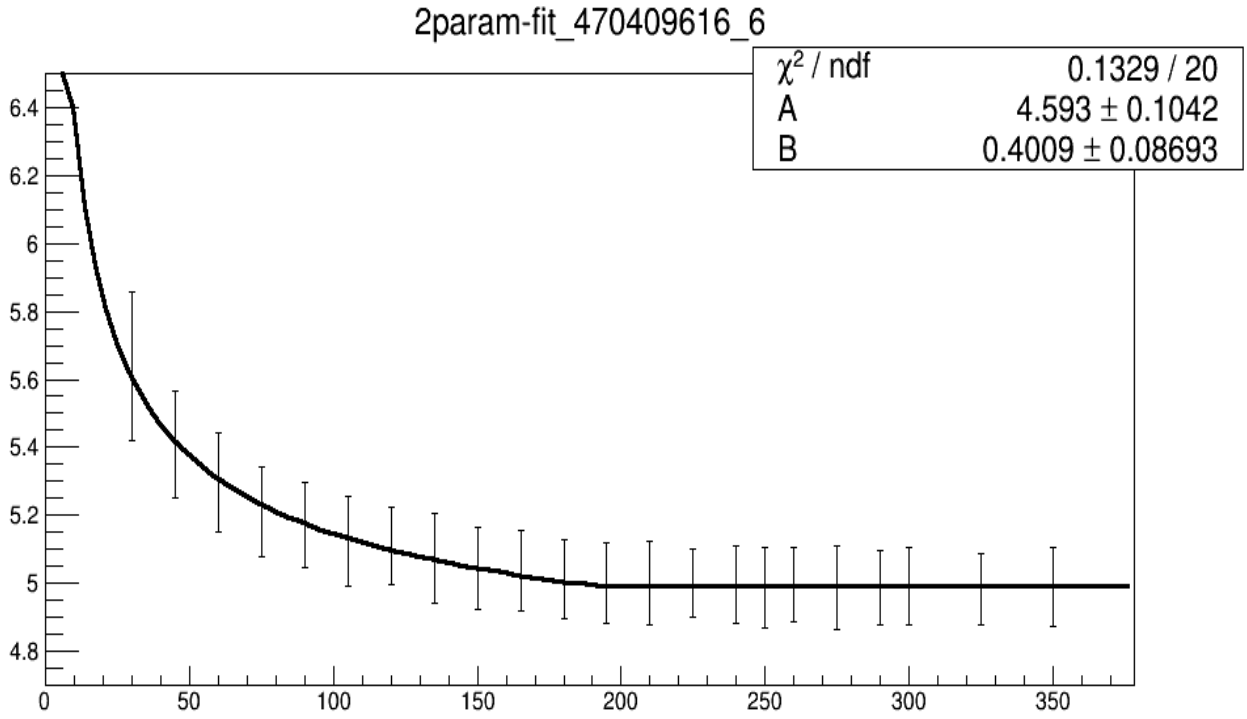
So we have these noise plots, the next thing to do is to look at the behavior of the APVs as the voltage increases. This will allow us to plot Noise vs. Voltage, and from that the full depletion voltage will be easily seen (by the eye at least) where the noise levels out to a nice line after the module has been fully depleted. Here is a plot of that same APV, telling us the mean noise of it at the different voltage points of the pedestal run:



This is plotting Noise vs. Voltage. You can see the trend of the noise evening out for higher voltages and there is a point, around 190V here, that can be said to begin a straight line. This voltage point is the full depletion voltage.

With this data we can try to fit a funtion to determine the exact depletion voltage, and with that information we can keep track of it as time progresses. Radiation damage will increase the depletion voltage over time, and knowing when we may not be able to provide enough current to a certain part of the tracker is valuable information. This is the purpose of the Noise Bias Scan.

Here is an example of a fitted plot:



Here the best fitted depletion voltage was 190V. You can see this is where the fit becomes a straight line. This fit was done with 2 parameters, using one function (a+b(sqrt(c)/x)) before the depletion voltage (c), and using another (a+b) after the depletion voltage. The leftward curve at the very top of the left of the fit is an error, and dissapears at different zoom levels.

Running the scripts, how they work

First off, all of the scripts can be found here:

/afs/cern.ch/user/d/dorzel/public/NoiseBiasPublic/CMSSW_6_0_1/src/NoiseBiasScan

And if you want to run them, copy the CMSSW_6_0_1 folder to your directory, cmsenv in CMSSW_6_0_1/src, and change any directories in the code that point to my area to yours. Most of these are found in the line "afs_path = '' and "afs_str =".

```
cp -r CMSSW_6_0_1 /some/local/dir
cd CMSSW_6_0_1/src
cmsenv
cd NoiseBiasScan
grep dorzel *.py  or  grep afs_path *.py  or grep afs_str *.py
change these lines to your path
```

The input files are currently located on EOS, here:
eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TECM/ for TECM. for others, substitute TECM for TECP,TIB,TOB. From James' twiki:

Noise bias scan run date	Partition	NBS data location	Run #s	APV 25 readout mode	Elogs

December 2010	TEC-	eos/cms/store/user/gbenelli/NoiseBiasScan/Dec2010/TECM		deco	
	TEC+	eos/cms/store/user/gbenelli/NoiseBiasScan/Dec2010/TECP			
	TIB	eos/cms/store/user/gbenelli/NoiseBiasScan/Dec2010/TIB			
	TOB	eos/cms/store/user/gbenelli/NoiseBiasScan/Dec2010/TOB			
July 2011	TEC-	eos/cms/store/user/gbenelli/NoiseBiasScan/Jul2011/TECM		peak	http://cmsonline.cern.
	TEC+	eos/cms/store/user/gbenelli/NoiseBiasScan/Jul2011/TECP			
	TIB	eos/cms/store/user/gbenelli/NoiseBiasScan/Jul2011/TIB			
	TOB	eos/cms/store/user/gbenelli/NoiseBiasScan/Jul2011/TOB			
September 2012	TEC-	eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TECM		deco	http://cmsonline.cern.
	TEC+	eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TECP			
	TIB	eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TIB			
	TOB	eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TOB			

Instructions for looking at these are at getting input files off of EOS.

data_harvester_profiles.py

This is the first script that is run. To start off, make sure you cmsenv'd in the CMSSW_6_0_1/src and then run the script using:

```
python data_harvester_profiles.py
```

This script runs on four different partitions of the tracker (TECM,TECP,TIB,TOB) for some run date and looks at the data in each of the voltage point files. So for instance, I can choose the run date to be Sep2012, and the partition to be just TECM, so it will go to the directory in which those files are stored (eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TECM/) and open the files in that directory (of the form TECM_run#_voltage).

Once it has analyzed all of the voltages in that directory, it will output a root file for that partition, of the form "meantr_Sep2012_TECM.root".

To specify which scan date you want, search for the line: "for scan date in [(some scan date)]:". There is a list in the comments of which you can choose. Just modify the list based on which scan date you want.

To specify the partition, go down a few lines and look for the declaration of "subdirs". Near the end, you can change "if subdir" to "if subdir == 'TECP'" for example. If you want to run on all, just leave it as "if subdir".

The analysis that the script does goes as follows:

- look at the mean noise of each APV
- remove dead or noisy strips from consideration
- collect mean noise values for each APV in each module at each voltage point

This takes about 30 minutes or longer to run. Once done, the meantr files hold the plots of Noise Vs. Bias voltage that can then be fitted.

run_batch_fitting_profiles.py

Now that data_harvester_profiles has outputted some root files, we can look at the plots of noise vs. voltage for each apv and see if we can get some fits. To run this script:

```
python run_batch_fitting_profiles.py 10000
```

This script will look at the output produced by data_harvester_profiles.py and try to fit a function to the plots of noise vs. voltage to get a full depletion voltage. It sends it off a batch job for each partition, which returns the output when it is finished in a directory of name "LSFJOB_#". If there are any errors, it will be in that dir, in a STDOUT file.

It sends off the batch job with the command:

- 'bsub -q 1nw -J p_script_'+date+'_'+partition+'_'+str(iteration)+'2param_'+date+'_'+partition +'_csh_'+str(iteration)+'_profiles.csh'

which comes out as, for example:

- 'bsub -q 1nw -J p_script_'+Sep2012+'_'+TECP+'_'+'0'+' '+'/afs/cern.ch/user/d/dorzel/work/HWdir/CMSSW_6_0_1/src/NoiseBiasScan/2param_'+'Sep2012'+'_'+'TECP'

The .csh files that it points to are created by the script beforehand. The only command they run is:

- python /afs/cern.ch/user/d/dorzel/work/HWdir/CMSSW_6_0_1/src/NoiseBiasScan/2param_Sep2012_TECM_script_0

which runs the corresponding python script. This is necessary because the csh files submitted tell the LSF system where the python file is to execute. **This is also the script that does the fitting and output.** It is modeled after fitting_template_profiles.py, and a few lines are changed to edit the date, partition, etc. So it is not fitting template.py that does the fitting, only each separate .py file for each partition. The fitting is done on the LSF system.

Currently the script may need a bit of fixing, I've been trying to get it to run and output for all partitions, but so far only two partitions have output files. You can look at one here. Originally I thought there were no fitted plots in this, but it turns out on certain channels there are!

The error message for the non-outputted root files is the same:

```
Traceback (most recent call last):
  File "/afs/cern.ch/user/d/dorzel/work/HWdir/CMSSW_6_0_1/src/NoiseBiasScan/2param_Sep2012_TIB_sc
    main()
  File "/afs/cern.ch/user/d/dorzel/work/HWdir/CMSSW_6_0_1/src/NoiseBiasScan/2param_Sep2012_TIB_sc
    detid = fedkey_detid_map[fedkey]
KeyError: '0x00013114'
```

This is for the TECM and TIB partitions. The TECP is outputted with sparse plots and TOB with no plots due to there being no "meantr_Sep2012_TOB.root" file

Looking at the parameters for the plots

-Each plot is stored as a TGraphErrors (http://root.cern.ch/root/html/TGraphErrors.html, for a full list of methods, check "show inherited") and originally shows only the fit line and data points. To show the parameter values and chi^2, you can:

Navigate to it in root:

- open root file in a TBrowser, and find the plot. Click on options->fit parameters and the box should be drawn with the plot
- to make the fit more visible, change the y axis scale by right clicking on the y axis and selecting SetRangeUser, and set it to what you want

Access them directly in root:

- open a terminal

```
 root -l meantr_Sep2012_TECP_fitted_profiles_0.root
 TGraphErrors* plot = _file0.FindObjectAny("2param-fit_470409616_6");
 plot.GetListOfFunctions().Print();
Collection name='TList', class='TList', size=2
TObjString = best_vd=190.00000
  2param-fit_470409616_6 : 2param-fit_470409616_6 Ndim= 1, Npar= 2, Noper= 0
 Par  0                  A = 4.59299
 Par  1                  B = 0.400945

Here you can see the value of the two parameters A and B for this plot, as well as the value of t
```

There is a TF1 that is named the same thing as the histogram here, so we can manually extract a, b, error in each, and the chi^2 of this fit as well:

```
TF1 *func = plot.GetFunction("2param-fit_470409616_6");
Double_t par1 = func.GetParameter(1);
Double_t par1error = func.GetParError(1);
Double_t par2 = func.GetParameter(2);
Double_t par2error = func.GetParError(2);
Double_t chi2 = func.GetChisquare();
```

Drawing the vdep text on the plot

-As we saw above, the vdep text is stored in a TObjString, which holds a TString. Here's a full working procedure script to plot this on the histo:

```
 root -l meantr_Sep2012_TECP_fitted_profiles_0.root
 TGraphErrors *plot = _file0.FindObjectAny("2param-fit_470409616_6");
 TObjString *vdeptext = plot.GetListOfFunctions().First();
 TText *vdeplabel = new TText();
 vdeplabel.SetNDC();
 vdeplabel.SetTextFont(1);
 vdeplabel.SetTextColor(1);
 vdeplabel.SetTextSize(0.03);
 vdeplabel.SetTextAlign(22);
 vdeplabel.SetTextAngle(0);
 gStyle.SetOptFit();  //displays the fit params
 plot.Draw();
 vdeplabel.DrawText(0.7, 0.75, vdeptext.String());  //the .7,.075 are percentages for each coordi
 Canvas_1.Update();
```

Figuring out why it isn't outputting correctly

A problem I immediately had with this script is that it didn't get all of the output back from when it submitted the LSF jobs. The root files that did come back had very few fitted profiles in them, and the rest were non-existent.

The first step is to see any error output from the batch jobs.

Getting input files off of EOS

Currently the input files for this (output files from pedestal runs) are located in EOS in a dir that looks something like: eos/cms/store/user/gbenelli/NoiseBiasScan/run_date/partition. From James' twiki:

To copy these off and have a look at them, do:

xrdcp
root://eocms.cern.ch//eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TECM/TECM_203247_350.root
whatever/dir/you/want in an lxplus terminal.

or

eos cp /eos/cms/store/user/gbenelli/NoiseBiasScan/Sep2012/TECM/TECM_203247_350.root
/whatever/dir/you/want

Fitting the plots

I'll start this off with the basic way of fitting some plot that you have to some function with the root fitter, using a plot from meantr_Sep2012_TECP_fitted_profiles_0.root as an example:

Load the plot:

- TGraphErrors *plot = _file0.FindObjectAny("2param-fit_470409616_6");

Define a function to fit with, [0],[1],[2] are parameters that we can name later:

- TF1 *fitfunc = new TF1("fitfunc","[0]+[1]*sqrt([2]/x)",0,350);

Define the parameters:

```
fitfunc->SetParName(0,"A");
fitfunc->SetParName(1,"B");
fitfunc->SetParName(2,"C");
fitfunc->SetParameter(0,4);
fitfunc->SetParameter(1,.4);
fitfunc->SetParameter(2,180);
```

Fit:

```
plot.Fit("fitfunc");
```

Fitting the plots with two lines:

To fit multiple lines to a plot, you have to define two line functions to fit with, which are just:

```
L1 = new TF1("Line1","[0]+[1]*x",xlow,xhigh);
L2 = new TF1("Line2","[0]+[1]*x",xlow,xhigh);
```

Where xlow and xhigh are the ranges of the x axis you want to use for the fits. Fit them like:

```
HistName->Fit(L1,"R");
HistName->Fit(L2,"R+"); //r+ to add them to the list of functions
```

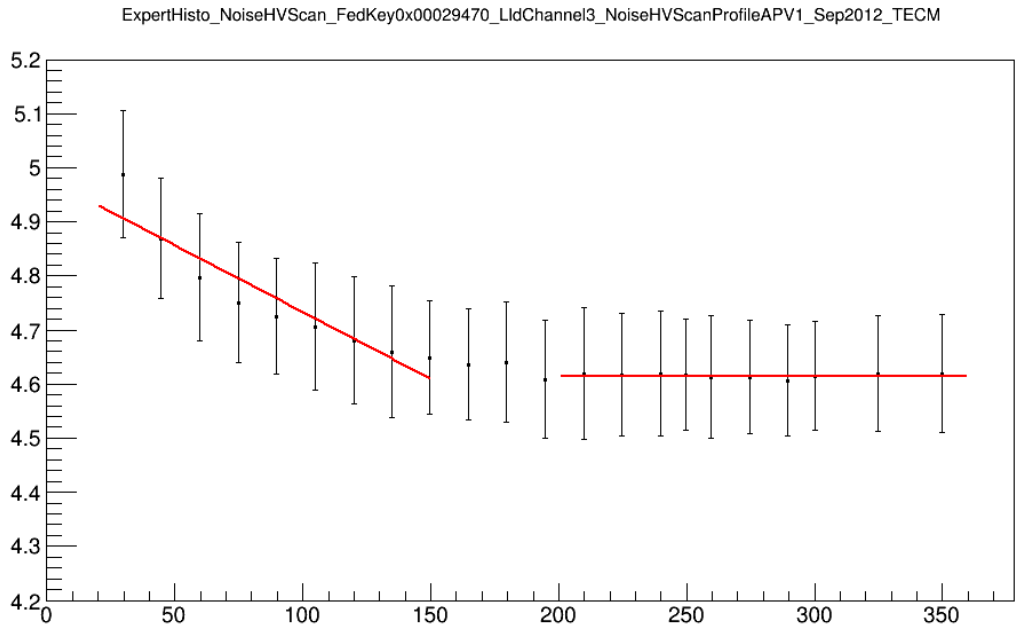
This will display the fit info, and if you want to see them:

```
L1->Draw("SAME");
L2->Draw("SAME");
Canvas_1.Update();
```

Or sometimes just this works:

```
HistName->Draw("AP");
```

My first attempts turned out like this, I just need to figure out how to draw the full lines and not segments:



I'm still trying to figure out how to display fit parameters for both lines on the plot, but it will probably involve creating a pavestats or something and using:

```
TF1 *fit = hist->GetFunction(function_name);
Double_t chi2 = fit->GetChisquare();
Double_t p1 = fit->GetParameter(0);
Double_t e1 = fit->GetParError(0);

or just:

Double_t chi2 = L1->GetChisquare();
Double_t p1 = L1->GetParameter(0);
Double_t e1 = L1->GetParError(0);

for each fit.
```

To look at where these lines converge to get the estimate for vdep, we equal the equations of the two lines and find the x value:

```
x=(b1-b2)/(m1-m2)
```

where b is the intercept and m is the slope. This translates into:

```
Double_t vdep = ((L1->GetParameter(0))-(L2->GetParameter(0)))/((L1->GetParameter(1))-(L2->GetPara
```

Take the absolute value of vdep, and that should be the result.

Different numbers of points used for fit

With the fits working properly, I explored how the fits changed with number of point used for each line for the fit, starting by including 5 points:

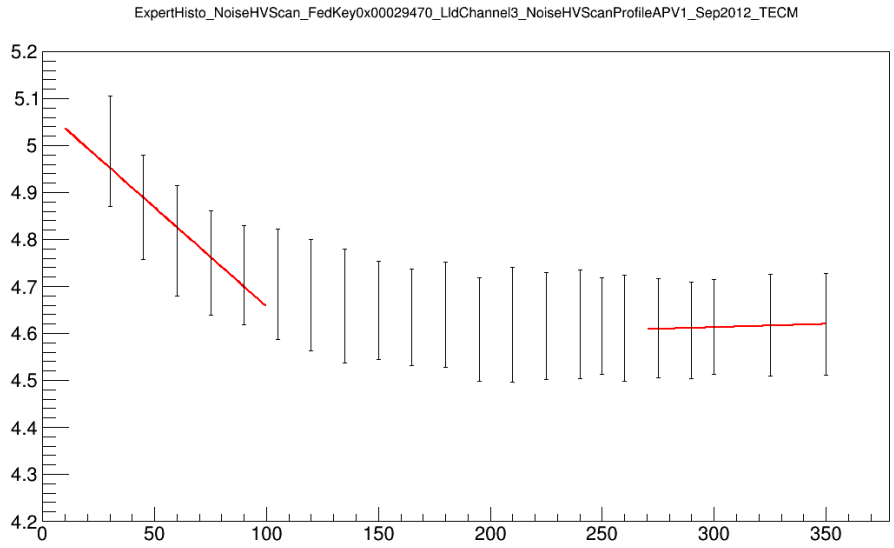
```
L1 = new TF1("Line1",10,100);
L2 = new TF1("Line2",270,350);
```

For future reference, you can remove functions and redraw fits with:

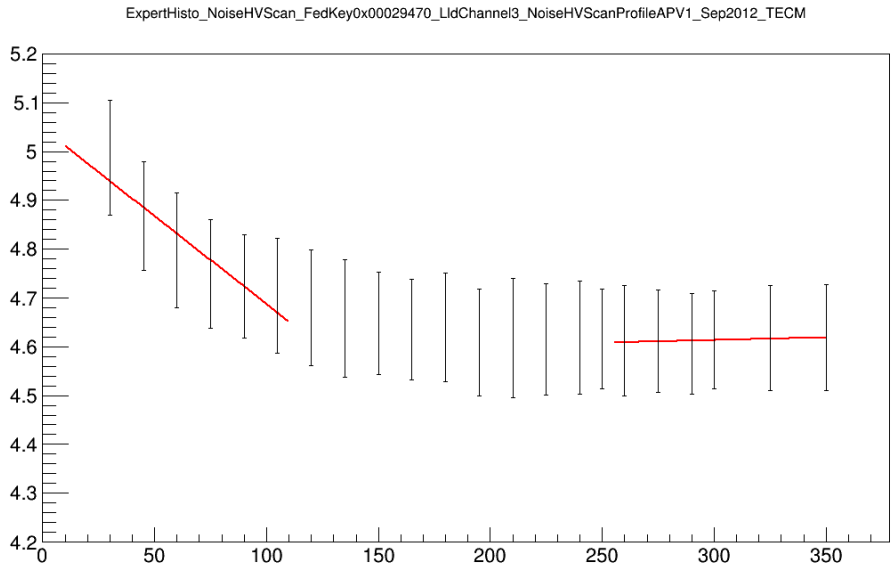
```
HistName.GetListOfFunctions().Delete();
```

This is on the same plot as above. Then I included 6,7,8,... by the same method. Here are the plots:

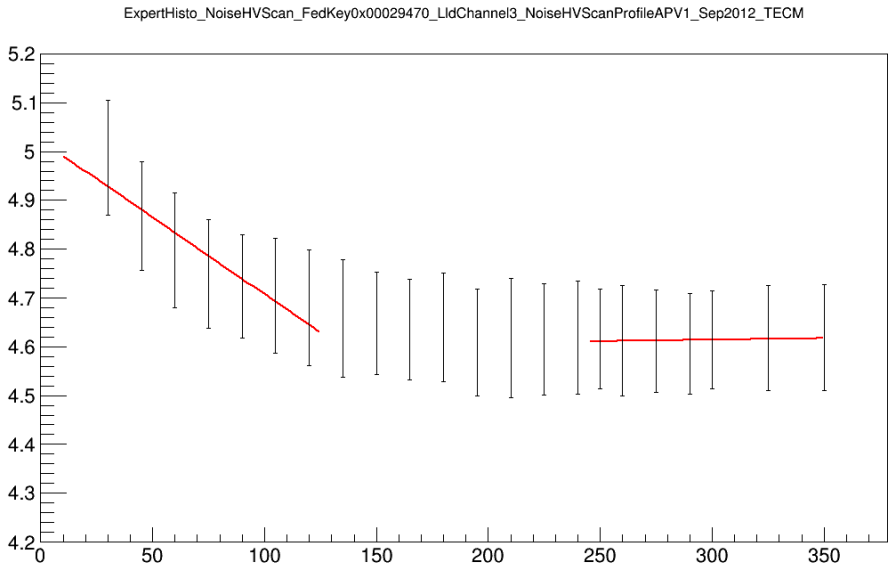
For 5 Points:



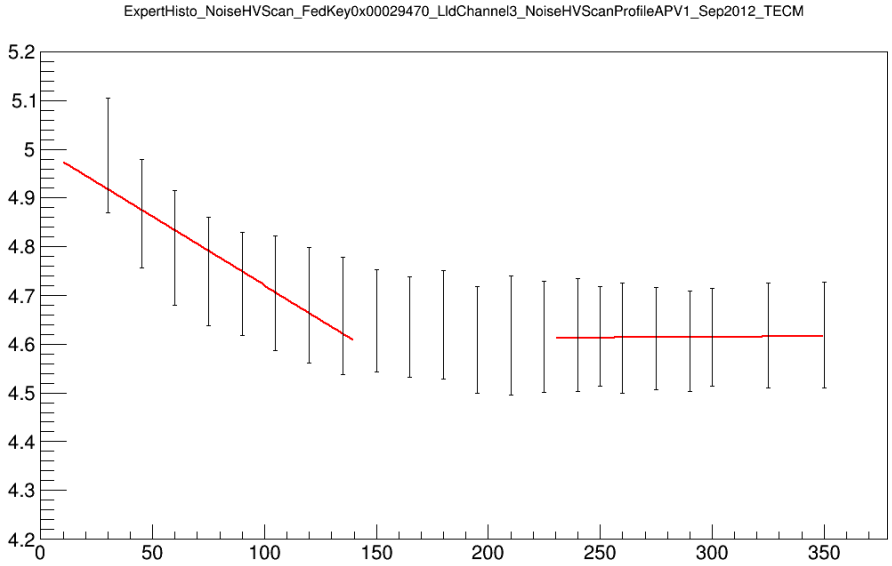
For 6 Points:



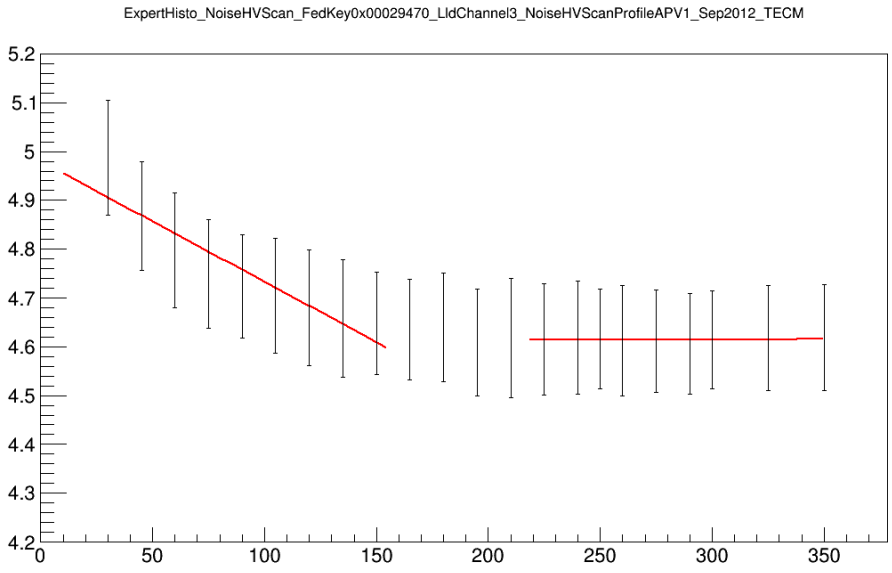
For 7 points:



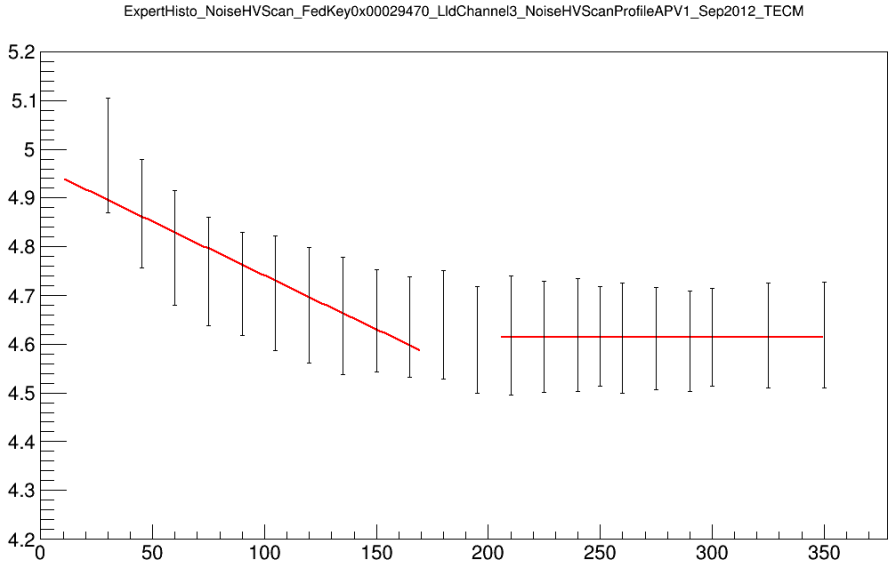
For 8 points:



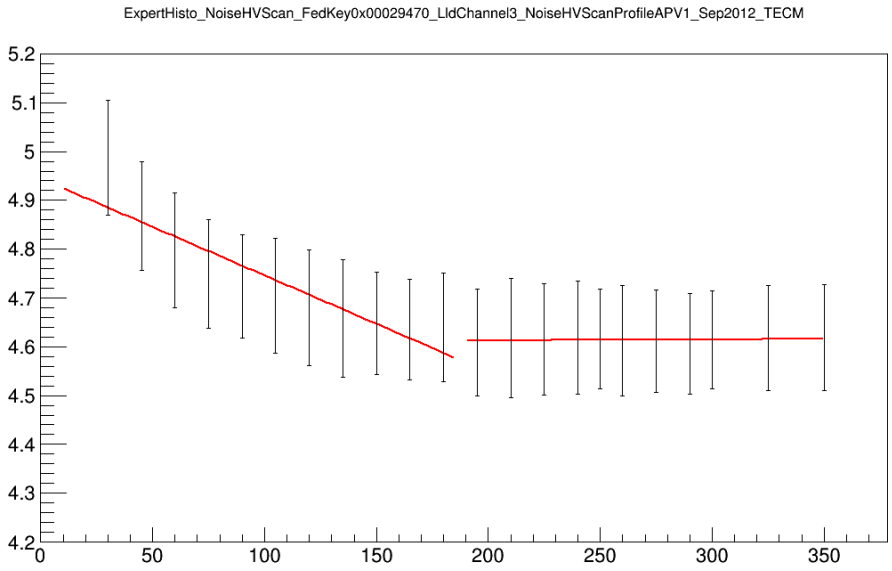
For 9 points:



For 10 points:



For 11 points:



The vdep appears to approach around 165, while the non-line fit predicts 190 for the vdep.

Plotting two lines with reduced errors:

To reduce the errors on each of the bins, this works:

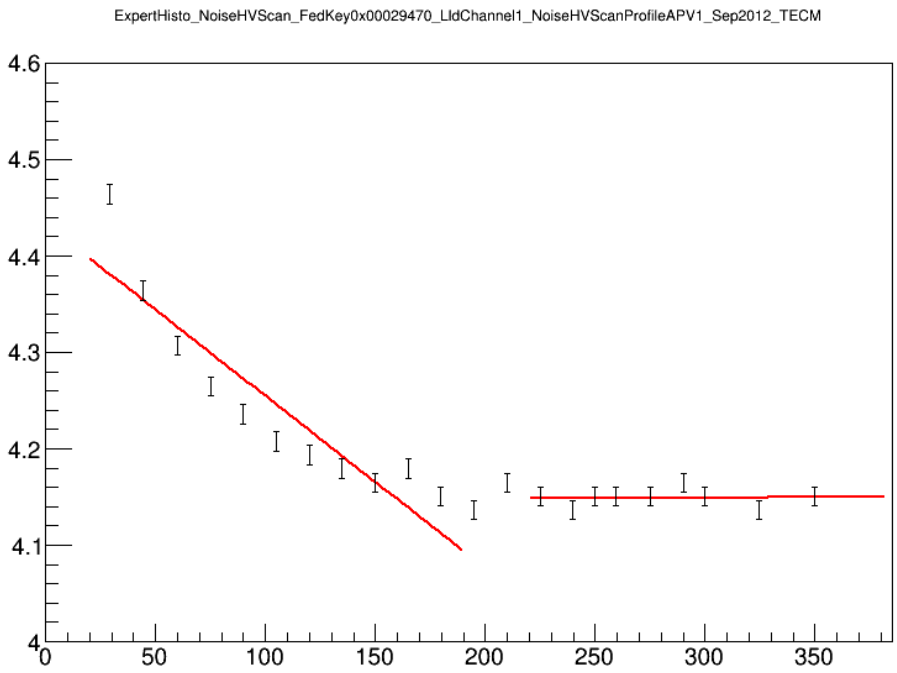
```
for(int i=0;i<HistName->GetXaxis()->GetNbins();i++){
     Double_t err = plot->GetErrorY(i);
     HistName->SetPointError(i,0,err/10);
}

HistName->Draw("AP");
Canvas_1.Update();
```

Fitting L1 and L2 as:

```
L1 = new TF1("Line1","[0]+[1]*x",20,190);
L2 = new TF1("Line2","[0]+[1]*x",220,350);
```

Gives the plot:



Common Mode Assertion - Noise/Pedestal Discrepancy

One thing we noticed when looking at the noise and pedestal plots of the output files of the noise scans is that the noise plots weren't exactly the error bars on the pedestal plots.

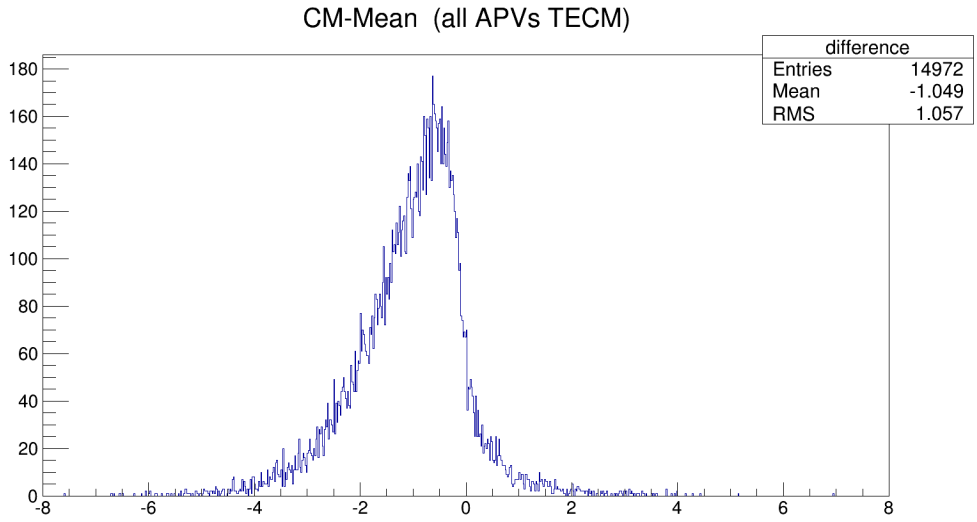
This didn't really make sense, as the noise is supposed to be the rms of the pedestal, so we wondered if the common mode had to do with this small discrepancy. So we had to know

how the common mode was calculated in the first place. The code that does that for the output files that we are looking at is here:

http://cvs.web.cern.ch/cvs/cgi-bin/viewcvs.cgi/CMSSW/DQM/SiStripCommissioningSources/src/NoiseHVScanTask.c

Which appears to take the first 32 strips of an apv and take the median. This doesn't seem to add up with the common mode plots in the files we are looking at, with many entries

and a mean of ~200 per plot. So, per APV, we looked at the difference between the common mode and the mean of the noise plot. This gave:



Which seems to suggest some correlation in the difference between the two. Data from comparing the pedestal and noise plots can be found in Comparisons_CM.zip

Reading from page 50 in Richard Bremer's disseration (dissertation_richard_bremer.pdf) it seems like the noise is indeed the error bar on the pedestal value. The common

mode seems to have nothing to do with the discrepancy we are seeing here. Perhaps the noise plot is not calculated from the pedestal plot, but taken from raw data?

Quarknet homeworks

I've completed some and am still working on some of James' homeworks for quarknet students related to Noise Bias Scan. Here is how I ran them and solved them.

HW1: Plotting two APVs

"Make a 1-d histogram of the noise for APV 1 and APV 2 (separately, so make two histograms) for the module located in a ControlView path using a PyROOT macro"

In this case I used a C++ macro and ran it in the ROOT interpreter, but pyROOT will be very similar. In the instructions, he gives a path: \DQMData\SiStrip\ControlView\FecCrate3\FecSlot15\FecRing1\CcuAddr123\CcuChan19 so I first opened the ROOT file and looked at the plots there. There are 3 plots of noise and three plots of pedestals. He wants us to plot APV's 1 and 2. Looking at how the apvs are numbered, it goes like:

channel 1 = apv1,apv2

channel 2 = apv3,apv4

channel 3 = apv5,apv6

So we want to grab the "lldchannel1" noise plot so we can plot both apvs. The code I used to plot them is here. I just plotted them on one canvas, but on two pads. To plot them, loop through the first 128 strips of the

hist and plot them on a new hist, then do the other 128 strips on another histogram.

Once this is done, save it as "filename.C" and run it with "root -l .x filename.C" It should display the histogram.

HW2: Cutting on bad strips

I had some trouble with this one, but the way I saw it, the assignment was to loop through all of the noise plots in the root file, and then loop through the plots to find strips that are behaving like what we would expect in the various failure modes, and then to show what the distributions look like before and after the cut. I didn't know how to loop through the root file, so I looked at Andre's code (couldn't find an attachment) and made a pyroot version of it: here.. This loops through all of the noise profiles, puts noisy, dead, and cosmic plots into lists, and chooses random ones to display and cut on. A current bug is that it can't find some strips when cutting. To run this, simply run it in a python version that is tied to pyroot.

Noise Bias Scan daily log

6/25/2014

-so far: have completed hw3 and set up the code correctly for hw4, quite a process without CVS.

6/27/2014

-going through and doing the homeworks again and making sure they are right, will upload for people who may need help on these in the future.

-when running run_batch_fitting_profiles.py 10000 for hw5, get an error about the .p files, since they have a .txt extension on them. Fixed these, to then get the error:

```
Traceback (most recent call last):
  File "batch_fitting_profiles.py", line 140, in <module>
    main()
  File "batch_fitting_profiles.py", line 93, in main
    out_python_script = open(py_file_to_be_submitted,'w')
IOError: [Errno 13] Permission denied: '/afs/cern.ch/work/e/eorcutt/public/noise/usercode/2param_
```

so I don't have access to that file.

7/08/2014

-I just realized that that was a 'w' open, so I'll just write to a /tmp directory.

7/15/2014

-Got run_batch_fitting_profiles to work by moving some files around, it sent a job off for four partitions, these came back with errors about file accesses to some files, to be fixed

7/31/2014

-managed to get the run_batch_fitting_profiles.py script running at least, and it outputs root files for TECP and TOB, but there are no plots in them.

8/6/2014

-working on documenting noise bias

-there are fitted plots in the root files... they are sparse but still... *facepalm

8/7/2014

-only the TECP file has plots in it, the TOB doesn't

-found out how to access fit parameters of the plots

-trying to figure out how to get the best vdep text in the plot

-you can do this with:

t1 = new TText(x,y,"text");

t1.Draw();

Canvas_1.Update();

-found out the other root files are not outputting due to "key errors" in the fedkeys.

8/18/2014

-looking into why only a few plots per root file are being outputted. Perhaps fits that don't converge aren't populated?

-can add the vdep by:

- TText *vdeplabel = new TText();
- vdeplabel -> SetNDC ();
- vdeplabel -> SetTextFont (1);
- vdeplabel -> SetTextColor (1);
- vdeplabel -> SetTextSize (0.03);
- vdeplabel -> SetTextAlign (22);
- vdeplabel -> SetTextAngle (0);
- vdeplabel -> DrawText (0.7, 0.75, "Best vdep"); //the .7,.075 are percentages for each coordinate. .75 means 75% of the way to the far right x coord
- Canvas_1.Update();

11/4/2014

-I have been forgetting to update this log for some time now, but here's what was done today

-Found a nice way to reduce the error bars on the noise plots

-Fitted a bunch of different point numbers with two lines for the noise plots

11/16/2014

-Possible reason why plots aren't outputting in line 79 of fitting_template_profiles

-set debug to true, no extra output from LSF job

-put a try/except on the error line in fitting_template_profiles, all root files were output, with sparse plots in all but TOB

-reduced the errors on each plot, breaks the fitting

-- DylanOrzel - 05 Aug 2014

This topic: Sandbox > DylanNoiseBiasScan
Topic revision: r26 - 2015-06-28 - unknown