README instructions for analysis

P. Dudero

April 2011

1 2010 analysis procedures

1.1 Storage

- Due to its superior performance all samples were migrated to hadoop (/hdfs/cms/...) as much as possible. Phedexed samples were migrated to /hdfs/cms/phedex by Jeremy, whereas mu skims of data and large background samples were put under /hdfs/cms/skim/mu, and signal samples were put under /hdfs/cms/user/heavynu/HeavyNuRecoFromHLT/38X.
- Analysis output was stored in various directories under `/local/cms/user/dudero`. Typically the directory was named after the configuration file that was run, with "_cfg.py" stripped from the name. Lots of other analysis flotsam and jetsam, some worthwhile and some ancient history, can probably also be found under these directories...

1.2 MC Scaling to Lumi

Subsequent sections of this README presume that MC samples are scaled to some luminosity prior to execution of subsequent steps. In fact, historically the MC samples were always scaled immediately after completion of execution. This strategy may be revisited and altered as necessary; in any case the raw MC samples were always kept around as well, as backup.

The multiscale.C script is checked in to CVS and provides capability to scale all the histograms in many files at once. It takes a text file appropriately formatted, as well as other formal parameters as shown below.

The downside of this implementation is the number of times one has to rescale, since every study indicated below leads to a new set of files, the number of "files2scale.txt" files themselves proliferated.

1.2.1 Preparation

1. edit a "files2scale.txt" file with the appropriate information (see the one checked in to CVS already).

1.2.2 Synopsis

```
root [0] .L multiscale.C+
root [1] multiscale("files2scale.txt",36.1,"NLO")
```

1.3 Signature

```
void multiscale(const char* filewithpaths,
                float integluminvpb,
                const char* csordersuffix="",
                bool writeErrors=false)
```

1.4 Formal Parameter Inputs

filewithpaths	space or tab-separated columns with filename, cross-section, number of events, and optional k-factors
integluminvpb	pretty self explanatory, what lumi to scale to
csordersuffix	suffix to affix to the output files, e.g. "NLO", "Meas", etc.
writeErrors	optional argument to turn on errors (if they haven't been already) and an attempt at handling them correctly during scaling

2 Z+Jets normalization

2.1 Preparation

1. Make data and Z+Jets MC histograms available
2. Create the "other backgrounds" root file containing the sum of backgrounds to be considered (both ttbar and VV are minimally required). This is currently done with a superPlot config file:

   ```
   root [0] .L superPlot.C+
   root [1] superPlot("addback2zpeak.cfg")
   ```

3. CAREFULLY adjust the parameters defined below in myChi2Fit.C

2.2 Synopsis

```
root [0] .L myChi2Fit.C+
root [1] myChi2Fit()
```

2.3 Signature

```
void myChi2Fit(double ctr_xsecpb=1.12,
               double xsecincpb=0.01,
               const string& refpath="data.root:hNu/cut6_Mu1HighPt/mMuMuZoom",
               const string& path2scale="fall10zjets.root:hNu/cut6_Mu1HighPt/mMuMuZoom",
               const string& otherbckgrnd="zpeakbackgrnd.root:sumback_mMuMuZoom_cut6",
               double minMLL=72.0,
               double maxMLL=112.0)
```

2.4 Formal Parameter Inputs

ctr_xsecpb	where the multiplier value is expected; the scan is centered here (the scan was historically done in xsec space)
xsecincpb	how much to increment the scan by for each iteration
refpath	the histogram path for the "reference" (data) histogram
path2scale	path to the histogram to be scaled (MC)
otherbckgrnd	path to the histogram containing "other background"
minMLL,maxMLL	defines the Z-peak window over which to do the normalization

2.5 Global Parameter Inputs

h1sf,h2presf,h3sf	prescale factors for reference (perhaps another MC), MC and other
rebinx	rebinning on the x-axis to boost per-bin statistics, 0=none
luminvpb,nevents	required for doing the scan in xsec space; can be set trivially to accomplish a unitless normalization

3 Trigger Efficiency studies

3.1 On Data

3.1.1 Preparation

1. All data samples available
2. Configure trigger matching on data samples with proper parameters (`HeavyNu/AnalysisModules/python/hnutrigmatch_cfi.py`)

3.1.2 Synopsis

1. `cmsRun heavyNuAnalysis_cfg.py` on data, trigger efficiency is collected in folders MuTightInZwin, Mu1TrigMatchesInZwin, Mu2TrigMatchesInZwin, Mu1Mu2TrigMatchesInZwin

2. use superPlot to generate plots, print trigger eff. bin info in xterm window (contents of `~/.rootrc` contains path to superPlot source:

   ```
   ACLiC.IncludePaths:        -Imypath -I$(ROOTSYS)/include
   ```

   ```
   root [0] .L superPlot.C+
   root [1] superPlot("hnutrigEffAsymOfficial.cfg")
   ```

3.2 On MC background

3.2.1 Preparation

1. All MC background samples available and scaled to the appropriate lumi

2. Adjust code in HeavyNuTrigger::simulateForMC with new bin info from run on data

3.2.2 Synopsis

1. `cmsRun hnuTrigEffStudy_cfg.py` on all MC samples

2. Use superPlot to print final statistics (also scales Z histos according to Z norm)

   ```
   root [0] .L superPlot.C+
   root [1] superPlot("finalstatsTrigEffback.cfg",false,false) # suppresses diagnostics
   ```

3. Calculate manually from the numbers printed what the up/down fluctuation is.

3.3 On MC signal

Same as for background except as noted. Given the number of signal files, some more automation is needed. `spDriver.C` is a script that invokes `superPlot` a number of times using a text file as input to configure variables such as filename. It takes the input configuration script (which presumably contains unresolved aliii - plural of alias in my book) and for each line in the ".drv" file it writes a new fully resolved config file to the `/tmp` directory and executes `superPlot` on that temporary config file.

1. Run spDriver:

   ```
   root [0] .L spDriver.C+
   root [1] spDriver("finalstatsTrigEffSig.cfg","signalfiles.drv",false,false)
   ```

2. To save the output to a file, ROOT syntax allows `root[0] myscript(); >myoutputfile.txt`

3. Throw the text file into a spreadsheet - since the text here is formatted for human readability and not spreadsheet parseability, one can overcome the problem by including the following characters besides space, tab and comma, as "text csv" separators: (%) and set "merge delimiters" to true (oocalc). An example ".ods" spreadsheet (`finalstats_trigeffunc_signal.ods`) has been checked in to CVS (with subsequent formatting) for perusal.

4 Muon ID/Isolation Efficiency studies

In contrast to the trigger studies, these are comparison studies between MC and data, so the same steps apply to both initially:

4.1 Preparation

1. Data and Z+Jets MC samples available

4.2 Synopsis

1. `cmsRun hnuMuSelectEffStudy_cfg.py` on data and Z+Jets MC samples, muon loose and tight selection and isolation efficiencies are collected in folders Mu1tagInZwin, Mu2tagInZwin, Mu1tagMu2passesLooseInZw Mu2tagMu1passesLooseInZwin, Mu1passesTightInZwin, Mu2passesTightInZwin, Mu1Mu2passesTightInZwin.

2. use superPlot to generate plots

   ```
   root [0] .L superPlot.C+
   root [1] superPlot("hnuMuSelectEffAsymOfficial.cfg")
   ```

3. Edit hnuMuSelectEffAsymOfficial.cfg to switch between loose and tight plots (comment/uncomment aliii at the top), repeat previous step.

4. Another ".cfg" file is in CVS that shows more detail, "hnuMuSelectEffAsymDetailed.cfg"

Since the last study performed on 2010 data showed no substantial difference between MC and data, no further computation was required, although the machinery for it is commented out in the python config file, and is probably buried in CVS history for the source.

5 Cut Optimization

5.1 Preparation

1. run "savehistos.C", which extracts histograms from all of the background and signal MC samples and saves them to a local file.

   ```
   root [0] .L savehistos.C+
   root [1] savehistos("files2optim.list","optimMWR.root")
   ```

   File "files2optim.list" contains the list of histograms to pull and a second column that indicates shortened unique names to give them in the new flat root file (otherwise there would be over a hundred versions of the same histogram "mwr"); "optimMWR.root" is the name of the output file to be generated.

5.2 Synopsis

To scan for optimal MWR cut values,

```
root [0] .L opticut.C+
root [1] opticut(0) # mode 0 => saves bkgrnd fit parameters to a ".h" file
root [2] opticut(1) # mode 1 => scans MWR cut values, saves results to "mwrscantree.root"
root [3] opticut(2) # mode 2 => reads tree, prints results table & tons of plots
```

To scan for optimal MLL cut values, modes 3,4, and 5 do the same thing for MLL (must have optimMLL.root prepared).

6 Final Statistics, Background

6.1 Cut Flow Table

6.1.1 Synopsis

To get a cut flow table for background, run spDriver as follows:

```
root [0] .L spDriver.C+
root [1] spDriver("cuttable.cfg","cuttable.drv",false,false)
```

This now produces latex-formatted output that can be captured as described above and then inserted directly into a tabular environment within a latex document. Some further editing is still required to add total background numbers, errors, etc.

6.2 Systematics and total errors, Background

Currently it is necessary to determine total systematics for background process by using the “spreadsheet of power”. This handles correlated and uncorrelated errors correctly, and totals uncorrelated errors in quadrature. A number of rows at the top of the spreadsheet are reserved for cutflow numbers and final numbers after optimization. Any of these rows can be copied into the top row, which is used to calculate the total error for the final number on the far right. The error shows up in red at the bottom right.

6.3 Final numbers with optimized cuts, Background

6.3.1 Preparation

1. Run Cut Optimization, which generates information necessary to decide what the optimal cuts (MLL, MWR) should be for the analysis. This will also produce files in preparation for use in this step.

7 Final Statistics, Signal

One ROOT script puts out all final yields, systematic errors, and total errors, once each for every mass point hypothesis.:

7.1 Preparation

- File “optimMWR.root” must be ready to go, as described above.
- Make sure the optimized cut values and total “other” uncertainty numbers at the top of the script are as you want them to be.

7.2 Synopsis

```
root [0] .L sigtable.C+
root [1] sigtable()
```

This file spits out a table that currently looks like this:

```
MWR   MNu  Cut     Ssig   Serr  JES(%) MES(%) MC(%) Tot(%)
============================================================
 700  100  560 |  16.58   1.86   9.2%   1.3%   3.3%  11.2%
 700  200  560 |  44.16   3.70   6.1%   0.7%   1.9%   8.4%
 700  300  560 |  48.66   3.36   3.8%   1.0%   1.6%   6.9%
 700  400  560 |  36.50   2.47   3.6%   0.9%   1.6%   6.8%
 700  500  560 |  19.79   1.31   3.3%   1.1%   1.6%   6.6%
 700  600  560 |   4.66   0.32   2.9%   2.2%   1.9%   6.8%
============================================================
 800  100  640 |   7.79   0.81   8.0%   2.1%   3.5%  10.5%
 800  200  640 |  26.62   1.94   4.5%   0.6%   1.8%   7.3%
 800  300  640 |  30.65   2.04   3.4%   0.7%   1.6%   6.6%
...
...
...
```

where Ssig = total signal yield, Serr = total signal error; the subsequent columns break down the percentage uncertainties.