8.882 LHC Physics

Experimental Methods and Measurements

Data Analysis Strategies and Essentials

[Lecture 6, February 23, 2009]

Organizational Issues

Project 1 – Charge Multiplicity

- due date: March 12 (2.5 weeks)

Project 2 – Bottomonia Cross Sections

- due date: April 6 (3.5 weeks)

Project 3 – *B* lifetime

- due date: May 2 (3.5 weeks)

Lecture Outline

Data Analysis Strategies and Essential

- motherhood and apple pie
- proper work style
  - preparation of setup
  - manuals and tutorials
  - prototyping
- design of an analysis
  - data processing
  - histograms and ntuples
- rule of thumb for coding

Most of the following seems obvious but so many people do it incorrectly

Motherhood and Apple Pie

Performing analysis = repeating the same thing over and over with small variations.

Everybody has to develop a work style for him/herself, and they are not all going to be the same.

There are some general patterns though which make sense to consider.

Most of this applies in one way or another beyond High Energy or Heavy Ion Physics.

Proper Work Style

Determine the objectives before you start

- measure physics quantity with statistical and systematic uncertainties
- document it in form of an internal note (including numbers and figures)

Determine your tools

- analysis tool is root: numbers and figures
- documentation tool is latex
- make sure to have a decent setup for your tools
  - you'll need it anyway
  - steal and copy as best as you can (no glory in this one)
  - you do not show your smarts here, though it has to look good

Proper Work Style

Manuals and tutorials

- reading a program manual is very ineffective, don't do it!
- learn to use a program by example
  - best: use examples from people you know are good
  - second best: use tutorials which are relevant for what you do
- in case of technical question
  - first try yourself, but not for too long (<15min)
  - then ask someone who you think might know the answer
  - or try to find an example (google is a great resource)
  - or check the manual (online if possible, search works best)

Proper Work Style

Prototyping

- in most cases with an analysis in the beginning you do not know what is important and what not
- prototyping the analysis will get you a quick feeling for what it is all about
- prototyping means do it quick and dirty
- after the prototype is done: cleanup and define the essential blocks (spent enough time here)
- then prototype the blocks and refine as needed

Design of an Analysis

Data processing

- analysis usually starts from bulky format
  - how much time does it takes to run over the entire sample?
  - how much time does a short peak job take?
- decide whether you need a dedicated small format: ntuple (today: TTree) usually the answer is yes, sometimes it is enough to work with histograms

Ntuple design

- ntuple always has run and event number
- design it to be fast, but as inclusive as you can
- make sure no information is duplicated
- split event information and run information

Design of an Analysis

Larger ntuple projects (not for this class)

- always organize ntuple on event basis
- write clean atomic objects to be streamed
- understand how to generically link objects
- make sure to include versions to all objects
- define split between input format and output format cleanly
- consider large subdivision of code (avoid circular dependencies)
  - lowest level: objects
  - next higher: algorithms (uses objects)
  - highest: modules (uses objects and algorithms)

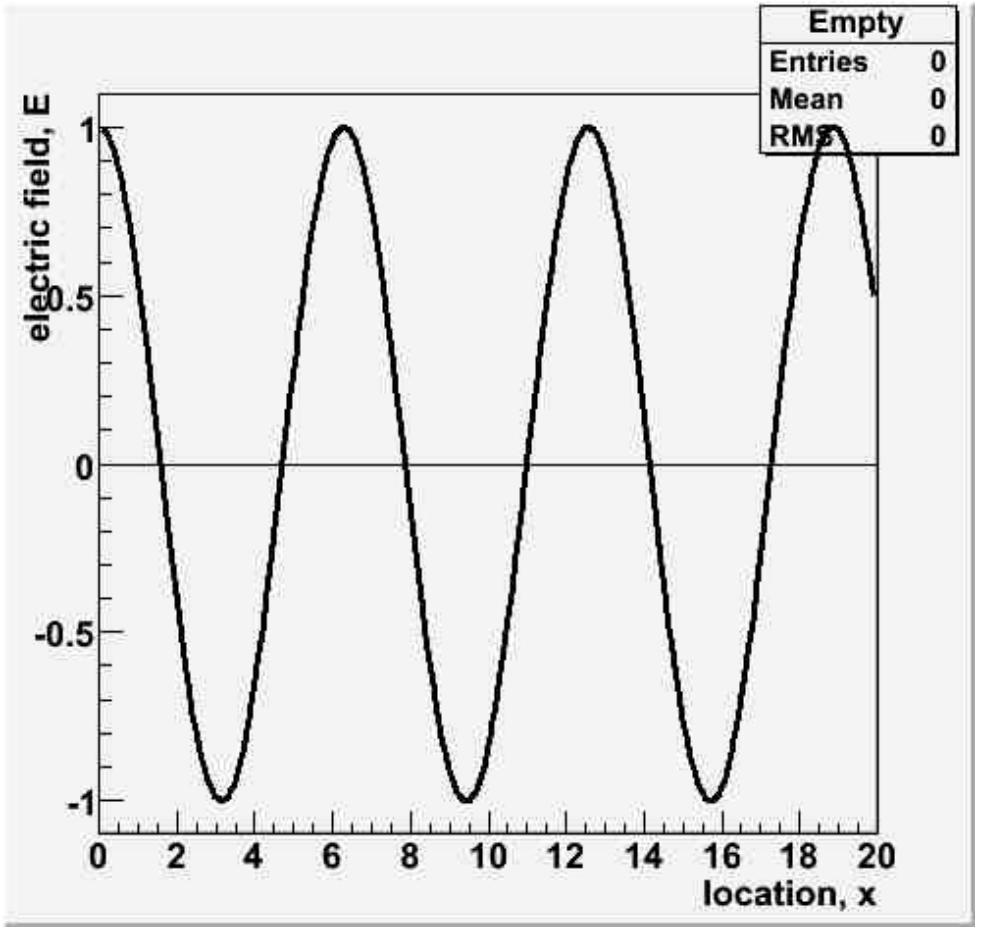
Last Step in Analysis

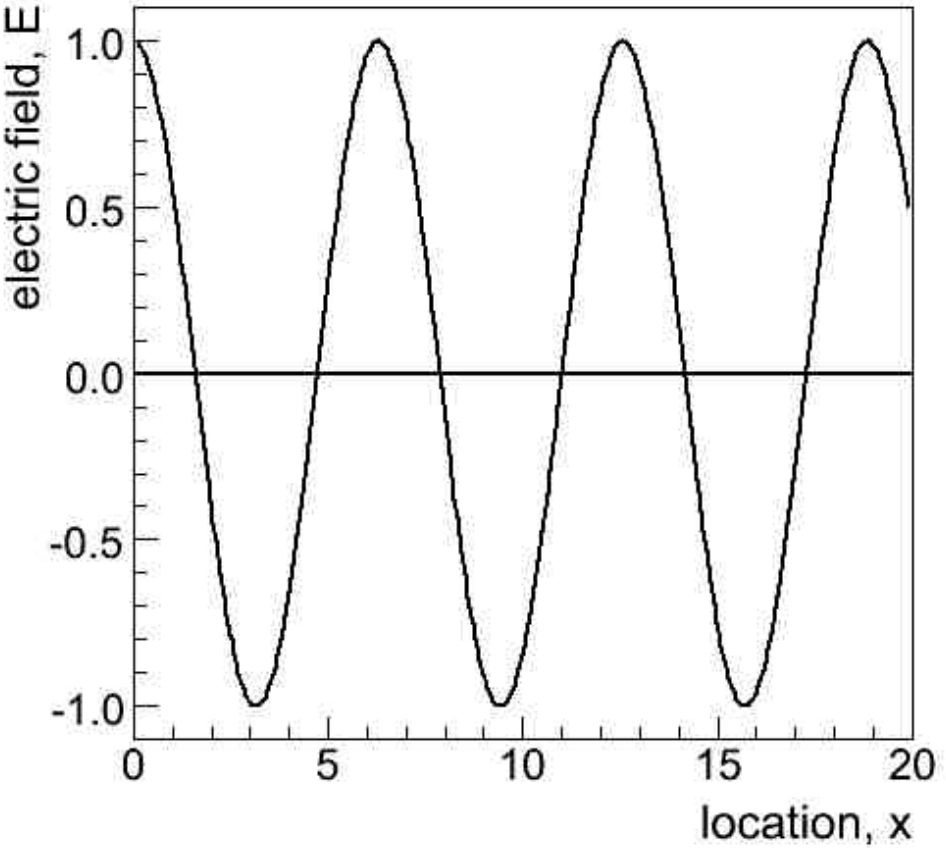
Histograms and fit

- in all analysis you need to present figures or plots
  - plots are, apart from the numbers, the result of your analysis
  - good plots explain: *"a picture says more then 1000 words"*
  - they have to be presentable, invest time in them
  - time invested in good figures always pays

- most analysis end up with a fit
  - fitting is an essential component of doing analysis
  - fitting is not difficult but needs quite some experience
  - always worth to invest some time into it
  - start with histogram fits in root and develop from there
  - TMinuit should be sufficient for almost any fit you'll do

Example: Making a Plot – Style





gray background, bold font, adjusted axis label position, larger font, adjusted axis division, adjusted line width, no useless information, no boxes

Example: Making a Plot

How to make a good plot, efficiently?

- important figures are never made interactively
  - write a little macro (in CINT) for each plot
  - re-making the figure has to be very quick
  - be guaranteed: you will remake a plot many, many times

- use standards
  - invest some work into your personal root style, copy someone style if it is good and you like it: MitStyle.C (SetStyle)
  - setups are often repeated, write some generic tools to help you make it go quicker

Example: Making a Plot – Macro

Load your preferred style

- generally: choose no serif fonts for plots
  - this would be: Helvetica, Arial, *etc.*
  - choose all fonts to be completely consistent
  - choose default plot layout, maybe put together two three layouts
  - choose the setup ones and do not touch them afterwards
  - remember small tunings have to be applied to plots, one size never really fits all
- use your default helpers to make the plot
- tune the plots so they look nice

Example: Making a Plot – Macro

cosine.C:

```
// Make funtion global so it is available after macro execution
TF1  *fitFunc = 0;
TH1F *empty   = 0;

void plot(TString xTitle = "title x", TString yTitle = "title y",
          TString opt    = "nonFlat");

// Main
void cosine()
{
  plot("location, x","electric field, E");
}
```

```
void plot(TString xTitle, TString yTitle, TString opt)
{
  // define the fitting function
  fitFunc = new TF1("cosine","[0]*cos(x)",0.0,20.0);
  fitFunc->SetParName  (0,"Amplitude");
  fitFunc->SetParameter(0,1.0);

  // prepare the ground for a plot by defining an empty histogram
  empty = (TH1F*) gROOT->FindObject("Empty");
  if (! empty)
    empty = new TH1F("Empty", "",10,0.0,20.0);
  InitHist((TH1*) empty,xTitle,yTitle);
  empty->SetMinimum(-1.1);
  empty->SetMaximum( 1.1);
  empty->SetTitleOffset(1.0,"X");
  empty->SetTitleOffset(1.4,"Y");

  // finally draw everything
  MakeCanvas("ShowCanvas","Just for Show");
  empty  ->Draw("");
  fitFunc->Draw("same,AC");
}
```

Example: Ntuple for N Track Analysis

Considerations

- analyzing all data (241k, MC: 203k) takes ≈20 secs
- reasonable test sample maybe ≈2 secs
- ntuple not really needed, work with histograms directly

But just to exercise, design ntuple for yourself

- content
  - run/event number
  - nTracks for various sets of cuts
  - add each track with η, z_0, p_T, ... to recalculate nTracks with ntuple (create event based structure)

Make tools for raw data and ntuple the same

Coding Rules of Thumb

Our choice of language is driven by CMS/root

- don't fight it, just go with it
- compile any bigger task (no CINT please)
- C++ is a fine language but
  - never try to show your smarts by using the last possible feature
  - technique is secondary: large scale organization is important
- root has its shortcomings but there is no alternative

General suggestions

- most analysis problems are straight forward
  - develop your standard solutions for standard problems (90%)
  - decompose the big task to a reasonable level of subtasks
  - avoid duplication of code
- your code should be a pleasure to read

Coding Rules of Thumb

Some general suggestions, *continued*

- use coherent syntax: helps to read your own code
- clarity is more important than compactness
- use constants to describe numbers, explicit numbers will appear in various places and will go out of sync
- if you use global variables or singletons, make sure it is really a good idea, avoid them if possible
- class design is essential
  - clarify who is the owner of the data you are using
  - draw yourself a little diagram to make sure you understand the interactions
  - always initialize variables and think about creation and destruction of allocated memory

Coding Rules of Thumb

Class design

- make sure not to overload a class
- the header file should fit into a large editor window
- choose good names and always document the non-obvious features of the class
- keep it to maybe 7 features
- hide details of the class as much as possible
  - important to avoid far reaching dependencies
  - more efficient for re-design
- try to use base classes, but only if it is useful
- do not be lazy, re-write if you see it is necessary

Syntax Suggestions

Write down how you like to write your code: makes your code coherent and easy to read

Line length:

- at some point a line gets too long, break it at some point and stick to it (like 80,100, .. characters)

Variable names:

- variable names are important, but keep them short
- variables start with small letter
- avoid underlines, use capitalization: myVar (my_var)
- identify class variables: use leading letter “f”: fMyField
- abbreviations: CMS becomes Cms

Syntax Suggestions

Statements:

- one per line should be fine, avoid several

```
if (myVar >25)
  cout << “myVar is larger 25\n”;
else
  cout << “myVar is smaller or equal 25\n”;
```

Methods

- use well defined syntax to make them recognizable

```
void MyClass::ThisMethod(int iVar)
{
   double here = ...

   return;
}
```

Comments: organize them well

Conclusions

Analysis hands on

- prepare a good setup for your analysis
- do not read manuals but work with examples
- prototype your analysis
- evaluate what analysis style is best for the problem
- keep coding simple and well organized in a global way

Next Lecture

Detectors: Tracking

- gas tracking detectors
- the Central Outer Tracker (COT) at CDF
- silicon detectors
- the silicon tracking system at CDF
- the tracker at CMS