DØNote 5818

Collie Hands-On Tutorial

Wade Fisher
Fermilab
(Dated: December 17, 2008)

This note includes hands-on tutorial materials for the Collie statistical software package.

This tutorial includes thirteen example problems designed to introduce users to the functional aspects of Collie and the calculations available within. These example problems should be considered a minimum prerequisite for Collie users and are necessary to understand the proper utilization of the Collie package. The example problems are designed to be performed in sequence.

- **Problem 0: Getting started**

  1. Log on to a clued0 node
  2. Choose a working directory (e.g., /home/username/collieWorkshop)
  3. Check out a copy of Collie from d0cvs:
     ```
     > setup d0cvs
     > cvs co -r V00-03-12 collie
     > cd collie/
     > source setup_Collie.tcsh
     ```
  4. Open the README files for the package and for the examples:
     ```
     > emacs README examples/README &
     ```
  5. Build the release:
     ```
     > make
     ```
  6. Read the README files while waiting for the build. A successful build will generate the following files:
     ```
     collie/lib/libCollieIO.so
     collie/lib/libCollieLimit.so
     collie/examples/collieIOmaker.exe
     collie/examples/collieLimitCalc.exe
     collie/examples/collieXsecCalc.exe
     ```

- **Problem 1: Make a CollieIO input file**

  1. An example for making a CollieIO file is given in collie/examples/collieIOexample.cc. Open this file to see what's going on.
  2. The API for making a CollieIO file is contained in io/include/CollieIOFile.hh. Open this file to see what's going on.
  3. The example CollieIO code creates a channel with 20 bins in the range 0.0 to 1.0, two backgrounds, one signal, and data for a single model parameter (100). It also assigns several signal and background systematic uncertainties. Run the executable to generate your example CollieIO file:
     ```
     > ./collieIOmaker.exe
     ==>Created mass point 100
     Mass:  100
     Data:  451
     Signal:  7.504
     Bkgd:  Bkgd1, 150.000
     Bkgd:  Bkgd2, 299.876
     Allbkgd:  449.88
     ==>Saving inspection histos to fv_exampleCollieIOfile.root
     ==>Saving channel data to exampleCollieIOfile.root....
     ```
  4. Two files are generated (see above).
     - The first (fv_exampleCollieIOfile.root) is a ROOT-browseable inspection file provided for users to view what input Collie will use for calculations. This is not your CollieIO file.
     - The second file (exampleCollieIOfile.root) is your CollieIO file and is not ROOT-browseable.
  5. Open your inspection file and see what Collie provides for user inspection:
     ```
     > root -l fv_exampleCollieIOfile.root
     root [1] new TBrowser
     ```
     Signal, background, and data histograms are normalized to the expected numbers of events. Systematic uncertainty histograms are given in the positive fractional deviation from nominal (ie, symmetric systematics will appear identical here).
  6. Plot the data, sum of backgrounds, and the signal on the same canvas:
     ```
     > root -l ../io/macro/plotInputs.C
     ```
     Try plotting signal plus background in one histogram.
  7. Plot a pair of positive and negative systematic uncertainty histograms:
     ```
     > root -l ../io/macro/plotSystematic.C
     ```
     Try plotting another systematic uncertainty distribution.

- **Problem 2: Generate a Log-Likelihood Ratio plot**

  1. The core of any Collie limit calculation are Log-Likelihood Ratio (LLR) distributions. Here we will generate LLR distributions and study their implications.
  2. Open the file collie/examples/exampleLimitCalculation.cc. This code will read your example CollieIO file from the last Problem and perform calculations. The CLfast calculator class is chosen by default (see tutorial for more details).
  3. For now, comment out the following lines:
     `//csLim.calculate(*sbd,clresults); L145`
     `//csLim.print(); L147`
  4. Recompile the package and run the example calculation:
     ```
     > cd ../; make; cd examples;
     > ./collieLimitCalc.exe CLtest.root 100
     Calculating for parameters:  100/-1/-1

     ****************************************:
     CLcompute Results:
     CLs_obs:  0.6443, CLs_med:  0.7261
     CLsb_obs:  0.2273, CLsb_med:  0.1369
     CLb_obs:  0.6389, CLb_med:  0.5107
     LLRobs:  0.4749, LLRb:  1.1659, LLRsb:  -1.2408, lumifactor:  3.249
     LLRb_m2s:  5.0993, LLRb_m1s:  3.2326
     LLRb_p1s:  -0.9139, LLRb_p2s:  -3.1569
     ****************************************:
     0.772721 sec run time
     ```
     Your output may be very slightly different, which we'll discuss later. This printout reports the results of a generic confidence level calculation for the nominal signal rate (ie, 1.0 × predicted).
  5. Now make histograms for the S+B LLR, B-Only LLR, observed LLR, and the $\pm 1\sigma, \pm 2\sigma$ B-Only LLR values. To do so, add the following lines into the code following line 136 (after clresults.print();):
     ```
     int bins = 500;
     double min = -15;
     double max = 15;
     TH1D* sigLLR = clcompute.getLLRdist_sb("LLR_SB",bins,min,max);
     TH1D* bkgLLR = clcompute.getLLRdist_b("LLR_B",bins,min,max);
     TH1D* LLRd = new TH1D("LLR_D","LLR_D",bins,min,max);
     TH1D* LLRsigma1 = new TH1D("LLR_B_1sigmas","LLR_B_1sigmas",bins,min,max);
     TH1D* LLRsigma2 = new TH1D("LLR_B_2sigmas","LLR_B_2sigmas",bins,min,max);

     LLRd->Fill(clresults.llrobs);
     LLRsigma2->Fill(clresults.llrb_m2s);
     LLRsigma1->Fill(clresults.llrb_m1s);
     LLRsigma1->Fill(clresults.llrb_p1s);
     LLRsigma2->Fill(clresults.llrb_p2s);
     ```
     These lines are also given at the end of the file collie/examples/exampleLimitCalculation.cc
  6. Plot the LLR distributions:
     `> root -l ../limit/macro/plotLLRdists.C`

- **Problem 3: Use the "brute force & awkwardness" method to find the observed 95% CL cross section limit**

  1. First, scale up your signal by some factor larger than 1.0 to change your CL values (after line 126):
     ```
     float sf = 1.1;
     printf(''Scaling signal by %.3f \n'',sf);
     sbd->scaleSignal(sf);
     ```
  2. Rerun your CL calculation:
     ```
     > cd ../; make; cd examples;
     > ./collieLimitCalc.exe CLtest_sf.root 100
     Calculating for parameters:  100/-1/-1
     Scaling signal by 1.10

     ****************************************:
     CLcompute Results:
     CLs_obs:  0.6933, CLs_med:  0.7697
     CLsb_obs:  0.1963, CLsb_med:  0.1151
     CLb_obs:  0.6400, CLb_med:  0.5101
     LLRobs:  0.6401, LLRb:  1.3945, LLRsb:  -1.4924, lumifactor:  2.709
     LLRb_m2s:  5.7106, LLRb_m1s:  3.6382
     LLRb_p1s:  -0.8695, LLRb_p2s:  -3.3043
     ****************************************:
     0.770733 sec run time
     ```
  3. Slowly increase your signal scaling factor until you achieve CLs_obs=0.95. Record this signal scaling factor value.

- **Problem 4: Adjust the precision of your calculation**

  1. Calculate the binomial uncertainty for your CLsb and CLb values. You should have 15000 pseudo-experiments in your histograms. *Hint: The binomial uncertainty will depend on the number of pseudo-experiments N and the outcome probability P.* This determines the dispersion you'd expect for CLsb or CLb if you repeat the calculation.
  2. Re-run your CL calculation a few times at your 95% CL limit signal scaling factor to see how CLsb_obs varies. Does this match what you calculated? Why or why not?
  3. Increase the number of pseudo-experiments you generate. You can find the nomenclature in limit/include/CLcompute.hh and its use in limit/src/CLfast.cc. For example, "CLcompute::LEVEL_STANDARD" gives you 50000 pseudo-experiments. Does the binomial uncertainty decrease as you'd expect?

  If you couldn't find the binomial uncertainty via Google or similar, you can take the author's word for it that the formula is:

$$\sigma^2 = N\ P\ (1-P) \tag{1}$$

$$\sigma = \sqrt{N\ P\ (1-P)} \tag{2}$$

- **Problem 5: Use Collie's algorithm to find the observed 95% CL cross section limit**

  1. Turn off any signal rate scaling as performed in Problem 3.
  2. Uncomment the two lines from step 3 of Problem 2. Recompile and rerun collieLimitCalc.exe. Collie should now be calculating the expected and observed 95% CL limits. Does the result agree with what you found in Problem 3?
  3. Turn off the expected limit calculation to speed things up. The job should speed up by roughly a factor of 2.
  4. Recalculate the observed limit ∼10 times. What is the dispersion of the outcomes in %?
  5. Increase the precision of the calculation (line 79) from 0 to 3. This increases the number of pseudo-experiments used in the limit calculation algorithm. Recalculate the observed limit ∼10 times. Does the dispersion decrease as expected? How is this related to the precision?
  6. What are the two parameters relevant to any algorithm that determines the 95% CL limit?

- **Problem 6: Turn on the systematic uncertainties**

  1. Up until now, you've been using CLfast, which ignores both statistical and systematic uncertainties. Switch to CLsyst, which will now include systematic uncertainties. For now, statistical uncertainties are ignored.
  2. Recalculate your observed 95% CL limit (use precision 0). How does it compare to what you found in Problem 5?
  3. Why does the limit change? Demonstrate using LLR distributions why this is the case.

- **Problem 7: Increase one of your systematic uncertainties**

  1. Try increasing one of your systematic uncertainties. For example, change the "Eff" uncertainty from 10% to 20% in collie/examples/collieIOexample.cc.
  2. Recompile and recalculate your observed 95% CL limit. Do you see a change?
  3. Return "Eff" from 20% to 10%.

- **Problem 8: Turn on the statistical uncertainties**

  1. Until now, your calculations have ignored the per-bin statistical uncertainties from the input histograms. Turn them on by adding the following lines after selecting your CLcompute class (*e.g.*, line 65):
     `clcompute.setNoviceFlag(false);`
     `clcompute.useHistoStats(true);`
  2. Recompile and recalculate your observed 95% CL limit. Do you see any change?
  3. Increase the statistical uncertainty by reducing the number of histogram entries generated in the CollieIO file. This can be done in collie/examples/collieIOexample.cc by changing the number of iterations (niter) from 1e6 to 1e3, for example. Reproduce your CollieIO file and recalculate your observed 95% CL limit. Do you see a difference?
  4. Return the number of iterations from 1e3 to 1e6 and remake your CollieIO file.

- **Problem 9: Find the $\pm 1\sigma$ values for your expected 95% CL limit**

  1. Your expected limit tells you what your limit would be if the data was identical to the background. An interesting comparison for your 95% CL limit are the values you would find if your data was similar to the pseudo-experiments that bound 68.27% of all pseudo-experiments ($\pm 1\sigma$) or 95.45% of all pseudo-experiments ($\pm 2\sigma$). Note: this is not an error or uncertainty on the limit. It's a gauge for understanding the signal and background uncertainties.
  2. Switch to calculating only the expected limit. Recompile and calculate the 95% CL expected limit.
  3. Adjust the toggle on line 86 to calculate the $\pm 1\sigma$ expected limits.
     `csLim.setNSigma(-1); (for` -1σ`)`
     `csLim.setNSigma(+1); (for` $+1\sigma$`)`
     Do these values make sense?

- **Problem 10: Submit a job to the ClueD0 batch**

  1. Given the combinatorial multitude of jobs that an analyzer may wish to run (expected, observed, several masses, $\pm 1, 2\sigma$, etc.), it makes sense to run in the batch. This example demonstrates how to run on the ClueD0 batch. Running on CAB is easiest from a /prj_root disk and is a simple modification to the script.
  2. Copy the file collie/limit/macro/CollieBatch.perl to the examples directory (ie, your working directory).
     `cp ../limit/macro/CollieBatch.perl .`
     Submit a limit calculation job to the batch (usage instructions are inside the script):
     `./CollieBatch.perl collieLimitCalc.exe 100 batchTest`
     Check that your job is running:
     `qstat -u <your username>`
  3. When your job has finished, you'll get a log file and the output root file in your working directory.

- **Problem 11: Perform a fit of the Monte Carlo templates to the data**

  1. Predict from your CLfast and CLsyst S+B LLR distributions whether your analysis is statistics limited or systematics limited. *Hint: the width of the CLsyst LLR distributions includes the statistical (ie, Poisson) and systematic uncertainties. CLfast only carries the Poisson width.*
  2. Inside collie/examples/exampleXsecMeasurement.cc, turn off the significance calculation. Comment out the following line (line 98):
     `// csCalc.testFitPE(*sbd, 0.0, 15000);`
  3. Recompile and run the example cross section measurement:
     `> ./collieXsecCalc.exe test.root 100`
  4. The output gives you the results of the best fit to the S+B hypothesis along with the covariance matrix from the fit. The fitted signal rate is given in units of the input cross section. Do the relative statistical and systematic uncertainties come out as you expected?
  5. The best fit value for each systematic uncertainty is given in units of its user-input 1-sigma value. MINUIT also returns values for what it determines to be one standard deviation in units of the user-input 1-sigma value. Do these seem reasonable? Try increasing/decreasing a systematic uncertainty to see the effect in the fit.

- **Problem 12: Test your fit model**

  1. One of the most important steps in generating a quality data/MC fit is to run the diagnostic tests for the user-defined fit. This fit test will perform a series of diagnostics and return the results for the user. Activate the fit test inside collie/examples/exampleXsecMeasurement.cc where indicated. Increase the number of pseudo-experiments to 3000 or more. Recompile and rerun the collieXsecCalc.exe executable.
     `> /collieXsecCalc.exe fitTest.root 100`
  2. Generate a postscript file with your fit results using the macro: limit/macro/fitResults.C
     ```
     > root -l
     root [0] .L ../limit/macro/fitResults.C
     root [1] makePlots("fitTest.root")
     ```
  3. Inspect the resulting output file (fitResults.ps) in order to identify any problems with your fit model. A detailed discussion of the diagnostic tests is included in the Collie user manual (Sec VIII, page 22).

- **Problem 13: Try out a few alternative systematic uncertainty formulations**

  1. To get an idea of how your system responds to differing systematics, try out a few alternatives. After each change, rerun your fit test. Try each one at a time and then consider mixing and matching.
     - Increase the "Eff" systematic to 75% (ie, from 0.10 to 0.75).
     - Use a log-normal PDF for the "Eff" systematic to get a more realistic description of a 75% uncertainty. Add the following line to collie/examples/collieIOexample.cc before writing out your CollieIO file (*e.g.*, line 133):
       `cfile->setLogNormalFlag("Eff",true,100);`
     - Try floating a systematic parameter (*i.e.*, remove any constraint on its value in the fits. A free parameter.). Add the following lines to collie/examples/collieIOexample.cc before writing out your CollieIO file (*e.g.*, line 133):
       ```
       cfile->setBkgdFloatFlag(0,"Eff",true,100);
       cfile->setBkgdFloatFlag(1,"Eff",true,100);
       cfile->setSigFloatFlag("Eff",true,100);
       ```

     You can experiment with including a single event source instead of all event sources.