RooStats updates: Bernstein Correction

Kyle Cranmer (NYU)
**RooStats: Project info**

Joint ATLAS/CMS project

- core developers
  - K. Cranmer (ATLAS)
  - Gregory Schott (CMS)
  - Wouter Verkerke (RooFit)
  - Lorenzo Moneta (ROOT)

- open project, you are welcome to join
  - Max Baak, Mario Pelliccioni, Alfio Lazzaro contributing now

Included since ROOT v5.22

- Example macros in $ROOTSYS/tutorials/roostats

Documentation

- code doc. via ROOT
- users manual is in development

**Release notes:**
http://root.cern.ch/root/v524/Version524.news.html#roofit

**Code documentation:**
http://root.cern.ch/root/html/ROOFIT_ROOSTATS_Index.html

https://twiki.cern.ch/twiki/bin/view/RooStats/WebHome
Correcting Models of Distributions

At our last Statistics Forum meeting, Glen presented a nice way to correct models (histograms or parametric functions) based on data.

- motivated by Stephan and Sascha’s work (also presented today)
- brief reminder in next few slides

Likelihood ratio test to determine best number of parameters

ATLAS Statistics Forum
CERN, 18 February, 2009

Glen Cowan
Physics Department
Royal Holloway, University of London
g.cowan@rhul.ac.uk
www.pp.rhul.ac.uk/~cowan

“Present study motivated by discussions with Eilam, Stephan Horner, Sascha Caron, et al., regarding Stephan's presentation on SUSYFit at 3 December 2008 Statistics Forum.”
A simple example

The naive model (a) could have been e.g. from MC (here statistical errors suppressed; point is to illustrate how to incorporate systematics.)

Note: “true model” was also a polynomial in Glen’s talk

G. Cowan
RHUL Physics

LR test to determine number of parameters
Enlarging the model

Here try to enlarge the model by multiplying the 0th order distribution by a function $s$:

$$\nu_i \rightarrow \nu_i s(x_i; \theta)$$

where $s(x)$ is a linear superposition of Bernstein basis polynomials of order $m$:

$$s(x) = \sum_{k=0}^{m} \beta_k b_{k,m}(x)$$

$$b_{k,m}(x) = \frac{m!}{k!(m-k)!} x^k (1 - x)^{m-k}$$

Note: $i$ here is an index over bins of a histogram, not needed in general.
Bernstein basis polynomials

Note: Bernstein basis polynomials are positive definite, an essential property for pdfs.

Figure 2: Bernstein basis polynomials of different orders $n$. 
Enlarging the parameter space

Using increasingly high order for the basis polynomials gives an increasingly flexible function.

At each stage compare the \( p \)-value to some threshold, e.g., 0.1 or 0.2, to decide whether to include the additional parameter.

Now iterate this procedure, and stop when the data do not require addition of further parameters based on the likelihood ratio test.

Once the enlarged model has been found, simply include it in any further statistical procedures, and the statistical errors from the additional parameters will account for the systematic uncertainty in the original model.
Fits using increasing numbers of parameters

Stop here

Note: “true model” was also a polynomial in Glen’s talk
Goodness-of-fit for the extended models

$q_\nu$ gives overall goodness-of-fit

$q$ compares model with $n_{\text{par}}$ parameters to that with $n_{\text{par}}+1$

<table>
<thead>
<tr>
<th>$n_{\text{par}}$</th>
<th>$q_\nu$</th>
<th>$p_\nu$</th>
<th>$q$</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>258.8</td>
<td>$6.1 \times 10^{-30}$</td>
<td>98.9</td>
<td>$2.6 \times 10^{-23}$</td>
</tr>
<tr>
<td>1</td>
<td>159.9</td>
<td>$1.1 \times 10^{-13}$</td>
<td>15.4</td>
<td>$8.9 \times 10^{-05}$</td>
</tr>
<tr>
<td>2</td>
<td>144.5</td>
<td>$1.3 \times 10^{-11}$</td>
<td>112.0</td>
<td>$3.5 \times 10^{-26}$</td>
</tr>
<tr>
<td>3</td>
<td>32.5</td>
<td>0.95</td>
<td>0.0013</td>
<td>0.97</td>
</tr>
<tr>
<td>4</td>
<td>32.5</td>
<td>0.93</td>
<td>0.26</td>
<td>0.61</td>
</tr>
<tr>
<td>5</td>
<td>32.2</td>
<td>0.92</td>
<td>0.37</td>
<td>0.54</td>
</tr>
</tbody>
</table>

$p$-values

Note: p-value can be seen as Type I error, eg. probability to falsely reject model with $n_{\text{par}}$ parameters when it’s true. (More relevantly, interpret as probability to add additional term when it’s unnecessary.)
I thought this was a very nice technique and should be incorporated into RooStats.

- Bernstein basis polynomials were not yet in RooFit
  - one point: want to provide analytic integral for the functions
  - should be easy, they are just polynomials
  - naive integral in Bernstein basis is not so easy
  - re-write in normal “power basis” of polynomials, trivial to integrate

**Converting from the Bernstein Basis to the Power Basis**

Since the power basis \{1, t, t^2, ..., t^n\} forms a basis for the space of polynomials of degree less than or equal to \(n\), any Bernstein polynomial of degree \(n\) can be written in terms of the power basis. This can be directly calculated using the definition of the Bernstein polynomials and the binomial theorem, as follows:

\[
B_{k,n}(t) = \binom{n}{k} t^k (1 - t)^{n-k}
\]

\[
= \binom{n}{k} t^k \sum_{i=0}^{n-k} (-1)^i \binom{n-k}{i} t^i
\]

\[
= \sum_{i=0}^{n-k} (-1)^i \binom{n}{k} \binom{n-k}{i} t^i + k
\]

\[
= \sum_{i=k}^{n} (-1)^{i-k} \binom{n}{i-k} \binom{n-k}{i-k} t^i
\]

where we have used the binomial theorem to expand \((1 - t)^{n-k}\).
RooStats Implementation

Now in RooFit, released last week in ROOT 5.23

- RooBernstein is a nice Polynomial PDF
  - guaranteed to be positive definite

Bernstein basis polynomials are positive-definite in the range [0,1]. In this implementation, we extend [0,1] to be the range of the parameter. There are n+1 Bernstein basis polynomials of degree n. Thus, by providing N coefficients that are positive-definite, there is a natural way to have well behaved polynomial PDFs. For any n, the n+1 basis polynomials form a partition of unity', e.g. they sum to one for all values of x. See [http://www.idav.ucdavis.edu/education/CAG2Notes/Bernstein-Polynomials.pdf](http://www.idav.ucdavis.edu/education/CAG2Notes/Bernstein-Polynomials.pdf)
RooStats Implementation

Requirements: provide a utility that implements the algorithm described by Glen last time.

- Able to start with an arbitrary “naive model”
  - it could be a histogram via RooHistPdf
  - it could be a non-parametric PDF via RooKeysPdf
  - it could be a parametric PDF based on physics insight
  - regardless, the class inherits form RooAbsPdf

- Base the correction on dataset (binned or unbinned)

- Specify the tolerance in some meaningful terms
  - choose probability that an additional term is added without really being needed
    - not exact, relies on Wilks’ theorem and assumption that current correction includes ‘true’ model
    - when true model is not naive * polynomial, this is an approximation
  - Therefore, add a utility to explicitly calculate the distribution of the Likelihood ratio (N_{par} vs. N_{par}+1) to double check
**Example (Step 1)**

An example similar to Glen’s but true model is not naive polynomial.

Here nature is a double gaussian

Generate toy data and forget we know what “nature” is.

```cpp
// set range of observable
Double_t lowRange = -1, highRange = 5;

// make a RooRealVar for the observable
RooRealVar x("x", "x", lowRange, highRange);

// true model
RooGaussian narrow("narrow", "", x, RooConst(0.), RooConst(.8));
RooGaussian wide("wide", "", x, RooConst(0.), RooConst(2.));
RooAddPdf reality("reality", "", RooArgList(narrow, wide), RooConst(0.8));

RooDataSet* data = reality.generate(x, 1000);
```
Example (Step 2)

Here our nominal (naive) model is not a histogram, but a parametrized function

- eg. a single Gaussian

Best fit model is not very good.

```cpp
// set range of observable
Double_t lowRange = -1, highRange = 5;

// make a RooRealVar for the observable
RooRealVar x("x", "x", lowRange, highRange);

// nominal model
RooRealVar sigma("sigma","",1,0,10);
RooGaussian nominal("nominal","",x,RooConst(0.), sigma);
```
Example (Step 3)

Import data & model into a workspace

Create a BernsteinCorrection object (specify tolerance)

Ask Bernstein correction object to import the corrected model into workspace

```cpp
// nominal model
RooRealVar sigma("sigma","",1.,0,10);
RooGaussian nominal("nominal","",x,RooConst(0);

RooWorkspace* wks = new RooWorkspace("myWorksspace");

wks->import(*data, Rename("data");
wks->import(nominal);

// The tolerance sets the probability to add an unnecessary term.
// lower tolerance will add fewer terms, while higher tolerance
// will add more terms and provide a more flexible function.
Double_t tolerance = 0.05;
BernsteinCorrection bernsteinCorrection(tolerance);
Int_t degree = bernsteinCorrection.ImportCorrectedPdf(wks,"nominal","x","data");
```
Example (Step 4)

Code below is to show how one can access the corrected model and the correction term from the workspace.

```cpp
RooPlot* frame = x.frame();
frame->SetTitle("An Example of RooStats BernsteinCorrection");
data->plotOn(frame);

// plot the best fit nominal model in blue
nominal.fitTo(*data, PrintLevel(-1));
nominal.plotOn(frame);

// plot the best fit corrected model in red
RooAbsPdf* corrected = wks->pdf("corrected");
corrected->fitTo(*data, PrintLevel(-1));
corrected->plotOn(frame, LineColor(kRed));

// plot the correction term (* norm constant) in dashed green
// should make norm constant just be 1, not depend on binning of data
RooAbsPdf* poly = wks->pdf("poly");
poly->plotOn(frame, LineColor(kGreen), LineStyle(kDashed));
```

Note: ratio of Double Gaussian / Gaussian can be very large in tails. BernsteinCorrection class allows one to limit the size of the correction.
When you ask the BernsteinCorrection to ImportCorrectedPdf it will print a log of the comparisons of $N_{\text{par}}$ vs $N_{\text{par}}+1$

```
// nominal model
RooRealVar sigma("sigma","",1.,0,10);
RooGaussian nominal("nominal","",x,RooConst(0.), sigma);

RooWorkspace* wks = new RooWorkspace("myWorksspace");

wks->import(*data, Rename("data"));
wks->import(nominal);

// The tolerance sets the probability to add an unnecessary term.
// lower tolerance will add fewer terms, while higher tolerance
// will add more terms and provide a more flexible function.
Double_t tolerance = 0.05;
BernsteinCorrection bernsteinCorrection(tolerance);
Int_t degree = bernsteinCorrection.ImportCorrectedPdf(wks,"nominal","x","data");
```

------ Begin Bernstein Correction Log -------

degree = 1  -log L(0) = 1216.78  -log L(1) = 1208.89  q = 15.7692  P(chi^2_1 > q) = 7.15583e-05
degree = 2  -log L(1) = 1208.89  -log L(2) = 1203.21  q = 11.3692  P(chi^2_1 > q) = 0.000746714
degree = 3  -log L(2) = 1203.21  -log L(3) = 1198.85  q = 8.72266  P(chi^2_1 > q) = 0.00314279
degree = 4  -log L(3) = 1198.85  -log L(4) = 1190.19  q = 17.3157  P(chi^2_1 > q) = 3.16557e-05
degree = 5  -log L(4) = 1190.19  -log L(5) = 1183.56  q = 13.2591  P(chi^2_1 > q) = 0.000271266
degree = 6  -log L(5) = 1183.56  -log L(6) = 1182.68  q = 1.7518  P(chi^2_1 > q) = 0.185651
Checking Sampling Distributions

choose probability that an additional term is added without really being needed

- not exact, relies on Wilks’ theorem and assumption that current correction includes ‘true’ model
- when true model is not naive * polynomial, this is an approximation

Therefore, add a utility to explicitly calculate the distribution of the Likelihood ratio ($N_{\text{par}}$ vs. $N_{\text{par}}+1$) to double check

```cpp
TH1* samplingDist = new TH1("samplingDist","",20,0,10);
TH1* samplingDistExtra = new TH1("samplingDistExtra","",20,0,10);
int numToyMC = 1000;
bernsteinCorrection.CreateQSamplingDist(wks,"nominal","x","data",
    samplingDist, samplingDistExtra, degree,numToyMC);
c1->cd(2);
samplingDistExtra->SetLineColor(kRed);
samplingDistExtra->Draw();
samplingDist->Draw("same");
```
Tutorial in last ROOT release

Go to root.cern.ch; click Documentation -> Tutorials
Tutorials

Choose the roostats tutorials

- see 7. rs_bernsteinCorrection.C
- used to make plots in this talk
Conclusions & To-do

RooFit now has a PDF class that implements Bernstein basis polynomials
  › nice because they are positive definite
RooStats now has BernsteinCorrection utility which implements the corrections as described by Glen
  › empirical correction of a nominal model based on observed data
  › can correct an arbitrary model (histogram, non-parametric, or parametric PDF)
  › corrections are based on polynomials and there is a well defined algorithm with a meaningful ‘stopping rule’

Considered: extend this method by abstracting the basis of the correction term and the algorithmic ‘stopping rule’ for adding more corrections
  › not too hard, but no obvious interfaces. Would turn into some templated code. Not clear that it’s worth it because it was pretty easy to write from scratch and it would make usage less obvious to users

In progress: extend method to include differences between distribution in control region and in signal region
  › basic idea is to have two copies of polynomial and add a term which relates parameters of polynomial in control region and signal region