Kernel Smoothing Package

Jeffrey Hetherly

August 21, 2015

Contents

Keywords statistics, smoothing, error analysis

Overview

This package is an implementation of a local polynomial kernel estimator (LPKE) along with utilities for smoothing a CDI file and performing an eigen-variation analysis. The package contains a generic implementation of an LPKE that can be customized to any polynomial order, domain dimensionality, numeric type, kernel function, and coordinate scaling. It also provides a convenience class for loading 1D, 2D, and 3D ROOT data from a `TH1` or `TGraphErrors`.
The utilities for smoothing a CDI file and performing the eigen-variation (EV) analysis are meant to aid in production of the smoothed CDI calibrations file and perform a complete EV analysis to help decide the appropriate number of EV to keep, respectively. They should be run from the command line as standalone executables and are not ROOT macros.
These utilities allow for some customizability, but in general will require modifications to their respective source codes for more realistic scenarios. The utility for smoothing a CDI file allows for command-line specification of the file to smooth, the number of p_T bins to store, and the (global) bandwidth. To specify which data sets to smooth, order of LPKE, etc. one needs to look at `smoothCalibrations.cxx`. The utility for the EV analysis can be customized at the command line to specify the required files, number of desired EV, and which data set to look analyze.

Quick Start

Here are the steps to quickly get up and running with smoothing a CDI file and run the eigen-variation analysis on LXPLUS (warning - some directories may need to change, but code works):

- Setup proper ROOT version (needs C++11)
  1. `localSetupROOT 6.02.10-x86_64-slc6-gcc48-opt`
- Checkout and setup RootCore
  1. `svn co svn+ssh://svn.cern.ch/reps/atlasoff/PhysicsAnalysis/D3PDTools/RootCore/tags/`svn ls svn+ssh://svn.cern.ch/reps/atlasoff/PhysicsAnalysis/D3PDTools/RootCore/tags | tail -n 1` RootCore`
  2. `source RootCore/scripts/setup.sh`
- Checkout relevant packages
  1. `rc checkout_pkg atlasoff/PhysicsAnalysis/JetTagging/JetTagPerformanceCalibration/CalibrationDataInterface/trunk`
  2. `rc checkout_pkg atlasoff/PhysicsAnalysis/JetTagging/JetTagPerformanceCalibration/CDIFiles/trunk`
  3. `rc checkout_pkg atlasperf/CombPerf/FlavorTag/JetTagPerformanceCalibration/NPandSmoothingTools/trunk`
- Compile packages
  1. `rc find_packages`
  2. `rc compile`
- Run smoothing procedure
  1. `ln -s NPandSmoothingTools/scripts/RUN_SMOOTHING.sh .`
  2. `source RUN_SMOOTHING.sh`
- Run EV analysis
  1. `ln -s NPandSmoothingTools/scripts/RUN_EIGEN_ANALYSIS.sh .`
  2. `source RUN_EIGEN_ANALYSIS.sh`

Mathematical Detail

Among the most common and crucial tasks in an analysis is fitting a curve with some functional form to a set of data. These curves are often used to interpolate or extrapolate into critical regions of phase space. Thus, the functional form of the fitted curve is of great importance. This form is usually chosen with a combination of domain specific knowledge (physical insight, etc.) and selecting from those forms with previous empirical success. However, sometimes the covariates and responses follow no obvious or previously encountered relationship. Additionally, one might not wish to impose a functional form on the data. In either case a "non-parametric" approach to constructing a smooth curve from data is attractive.
Broadly speaking, non-parametric curves have no innate functional form and can in principle adapt to any set of data. Splines, kernel estimation, and support vector regression are all forms of non-parametric regression. Each method will model the data differently and have its own unique "parameters" to determine the amount of "smoothness" of the curve. These are not parameters in the usual sense of fitting because one can normally find a value of the parameter(s) that can interpolate through all of the data. This is undesirable in our context as we are not seeking the "best" fit, but rather wish to follow the general trend of the data (otherwise, we could just use a histogram or empirical distribution function).

Local Polynomial Kernel Estimation

The non-parametric method used in the calibrations file and described here is known as local polynomial kernel estimation (LPKE). One can think of this method as fitting a polynomial by weighted least squares where the weights are functions of the covariates. These "weight" functions are referred to as kernels. Kernels are usually chosen to be even and integrate to unity, although for regression the requirement of integrability can be relaxed. A common kernel in physics and the one used here is a normalized guassian kernel. A general, multivariate local polynomial kernel estimator of polynomial order p, covariate dimension d, with kernel $K(\mathbf{x})$, $d \times d$ bandwidth matrix $\mathbf{H}$, and constructed from n data points with covariates $\mathbf{X}_i$ and responses Y_i has the form:

$$\hat{m}(\mathbf{x}; p, \mathbf{H}) = \hat{\mathbf{e}}_1^T \cdot \left(\mathbf{X}_\mathbf{x}^T \mathbf{W}_\mathbf{x} \mathbf{X}_\mathbf{x}\right)^{-1} \mathbf{X}_\mathbf{x}^T \mathbf{W}_\mathbf{x} \mathbf{Y}$$

where,

$$\mathbf{W}_\mathbf{x} = \begin{bmatrix} K_\mathbf{H}(\mathbf{X}_1 - \mathbf{x}) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & K_\mathbf{H}(\mathbf{X}_n - \mathbf{x}) \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}, \quad \mathbf{X}_\mathbf{x} = \begin{bmatrix} 1 & \left(\overrightarrow{\Delta}_1^1\right)^T & \dots & \left(\overrightarrow{\Delta}_1^p\right)^T \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\overrightarrow{\Delta}_n^1\right)^T & \dots & \left(\overrightarrow{\Delta}_n^p\right)^T \end{bmatrix}$$

$$K_\mathbf{H}(\mathbf{x}) = |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2}\mathbf{x})$$

$$\overrightarrow{\Delta}_i^m = \sum_{j=1}^{d} \left((\mathbf{X}_i - \mathbf{x}) \cdot \hat{\mathbf{e}}_j\right)^m \hat{\mathbf{e}}_j$$

The derivation of this for the univariate case is straightforward in the context of least squares minimization~[**?**]. In areas of phase space sufficiently far away from any data points, the weight matrix $\mathbf{W}_\mathbf{x}$ may approach machine zero due to small kernel values. The convention followed here is that in the case of a zero weight matrix the kernel evaluations are just replaced by 1. Replacing the kernels this way reduces this method to the least-squares fit of a multidimensional "polynomial." In this way, bandwidth values that are "very large" or "very small" yield similar results.
We also modified the weights matrix to allow for unequally weighted data by multiplying the kernel by $w_i = 1/\sigma_i^2$ since σ_i's are the variance (for uncorrelated errors) in the responses. This was motivated by wanting to approximate the best linear unbiased estimator (BLUE) where the weight matrix should be the

inverse of the variance-covariance matrix~[**?**]. Thus, the weight matrix looks like:

$$\mathbf{W_x} = \begin{bmatrix} w_1 K_{\mathbf{H}}(\mathbf{X}_1 - \mathbf{x}) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_n K_{\mathbf{H}}(\mathbf{X}_n - \mathbf{x}) \end{bmatrix}$$

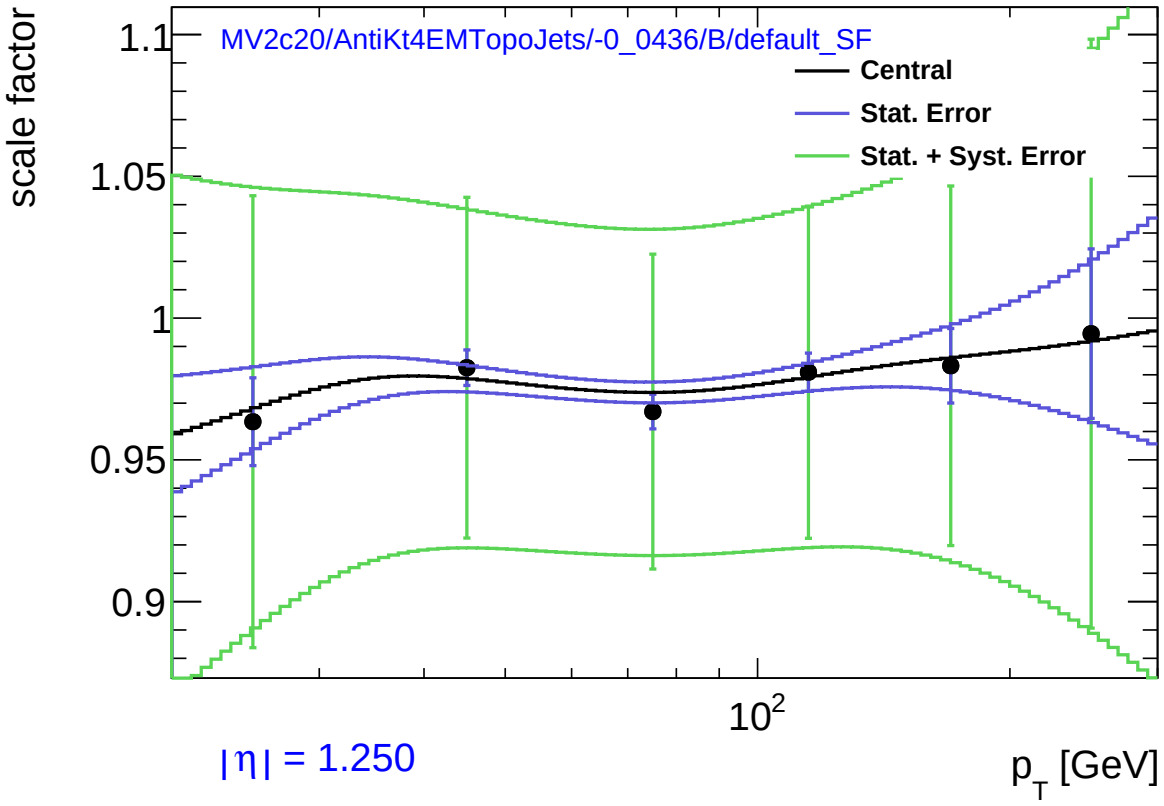
Lastly, for this implementation of the LPKE we use a simplified bandwidth matrix:

$$\mathbf{H} = \begin{bmatrix} h_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & h_d \end{bmatrix}$$

Thus, the kernel here is a function of the "scaled" covariates:

$$K(\mathbf{H}^{-1/2}\mathbf{x}) \rightarrow K\left(\frac{x_1}{h_1}, \dots, \frac{x_d}{h_d}\right)$$

The $p = 1$ case of the estimator is called a local linear kernel estimator and has less bias at the ends of the data than a local constant kernel estimator ($p = 0$). Note that all odd orders have less bias at the end points than even powers~[**?**]. The $p = 1$ local kernel estimator was chosen for this reason in addition to better accommodating the spacing in the SF distributions. The bandwidth plays the role of the smoothing parameter in an LPKE. There are a number of methods for constraining the smoothing parameters, some of which are~[**?**]: cross validation, minimization of squared residuals, or aesthetics. We implemented a cross-validation scheme that gave us a bandwidth of 0.4 (on a logarithmic scale in p_T). This is demonstrated in Fig.~**??**.



The procedure for smoothing the scale factor distributions is as follows:

- Scan through entire CDI file to find the light-, tau-, c-, and b-jet `CalibrationDataHistogramContainers` objects. Specifically, select the ones labeled `default_SF`.

- For each flavor, add in quadratue all of the bin-uncorrelated systematics along with the statistical bin uncertainties for the central SF distribution. This will serve as the weights given to the smoother when smoothing the central SF, denoted $\hat{m}_{\mathrm{W}}$. For comparison with other smoothed curve later in the procedure, we also smooth the central distribution where all bins are weighted equally, denoted $\hat{m}_{\mathrm{NW}}$.
- Bin-uncorrelated systematics and the statistical bin uncertainties are smoothed via a variational technique. In this approach, each bin i of the central distribution is varied by the content of the systematic or statistical uncertainty. This varied distribution is then smoothed with equally weighted bins and $\hat{m}_{\mathrm{NW}}$ is subtracted away, leaving only the smoothed variation, $\hat{m}_{\delta_i}$. This is repeated for each bin of the systematic until there are as many smoothed, varied systematics as there are bins in the original SF distribution (Fig.~**??**).
- Bin-correlated systematics are smoothed by varying the entire central SF distribution in all bins at once, smoothing, then subtracting away $\hat{m}_{\mathrm{NW}}$.
- For each η bin, smooth only along the p_T axis scaled logarithmically using a $p = 1$ LPKE with a bandwidth of 0.4.

Lastly, note that the form of the estimator has the following, seemingly insignificant, property. Let $\hat{m}_{\mathbf{Y}}$ be associated with $\mathbf{Y}$. Now shift the $\mathbf{Y}$ by $\overrightarrow{\delta}$ so that $\hat{m}_{\mathbf{Y}+\overrightarrow{\delta}}$ is the estimator associated with the "shifted" responses. Notice that $\hat{m}_{\mathbf{Y}+\overrightarrow{\delta}} - \hat{m}_{\mathbf{Y}} = \hat{m}_{\overrightarrow{\delta}}$. That is to say, the estimator of the "shifted" responses less the estimator for the original responses is simply the estimator associated with the shifts themselves, $\hat{m}_{\overrightarrow{\delta}}$. This is useful in nuisance variations where the nominal distribution is shifted and normally one would need to perform two LPKEs and a subtraction. This can introduce undesirable numerical effects. Fortunately, one can skip that procedure altogether and simply perform an LPKE on the $\overrightarrow{\delta}$'s themselves.

Eigen-Variation Reduction

Each scale factor distribution in a CDI data set has many (~40) systematic variations as well as statistical variations that need to be taken into account in order to properly access the uncertainty of the distribution. For a physics analysis this can lead to a large number of nuisance parameter when one needs to assess the effects of various systematics. Therefore, it is highly desirable to somehow reduce the number of systematics under consideration.
One method of reducing the number systematics that preserves the bin-to-bin correlations and total error is to perform an eigenvalue decomposition on the covariance matrix of systematic and statistical variations, also known as a principal component analysis. It is clear that the resulting number of variations will simply be the number of bins in the scale factor distribution. This is already a huge reduction of systematics to consider if your distribution has 10 bins. However, after smoothing the number of bins can far exceed the original number of systematics. Fortunately, most of the EV are very small in magnitude and can be thrown out without fear of losing correlations or total error. In principle one should recover the original number of "significant" EV after this pruning (i.e. number of "pre-smoothed" bins). The remaining EV can be further reduced by summing up the M least significant EV (ranked by eigenvalue). Whatever summing technique is used, the over goal is to try to satisfy the following constraint on the newly summed histogram, H':

$$H'(x_i)H'(x_j) = \sum_{k=M}^{N_{EV}} H_k(x_i)H_k(x_j)$$

This condition may only be satisfied if one of two criteria are met:

1. The $H'(x_i)$ are vectors with elements $H_k(x_i)$
2. The $H'(x_i)$ are scalars but the $H_k(x_i)$ conspire in such a way that makes this equality hold

Unfortunately, neither of these criteria are applicable for our case. Thus, our essential problem is that of how one can store the information of a vector in a single scalar. Using the (false) equality above one can set $i = j$

and take the positive square root to yield:

$$H'(x_i) = \sqrt{\sum_{k=M}^{N_{EV}} H_k^2(x_i)}$$

Using this equation (S_Q) preserves the total errors in the resulting covariance matrix. However, because all the information about the sign of the original systematics is lost this method of merging severely distorts the correlations. Another manipulation of the (false) equality is to first sum over j, sum over i, then finally back substitute to solve for $H'(x_i)$:

$$H'(x_i)S_{H'} = \sum_{j}^{N_{\text{bins}}} \sum_{k=M}^{N_{\text{EV}}} H_k(x_i)H_k(x_j)$$

$$S_{H'}^2 = \sum_{i}^{N_{\text{bins}}} \sum_{j}^{N_{\text{bins}}} \sum_{k=M}^{N_{\text{EV}}} H_k(x_i)H_k(x_j)$$

Thus,

$$H'(x_i) = \frac{\sum_{j}^{N_{\text{bins}}} \sum_{k=M}^{N_{\text{EV}}} H_k(x_i)H_k(x_j)}{\sqrt{\sum_{n}^{N_{\text{bins}}} \sum_{m}^{N_{\text{bins}}} \sum_{k=M}^{N_{\text{EV}}} H_k(x_n)H_k(x_m)}}$$

In principle this should be equal to the first method of merging, S_Q. But, because the original equation is false they give different results. This alternative method of summing (S_A) the M lowest eigen-variations is found empirically to better preserve the correlations over the S_Q method. However, S_A doesn't preserve the total error as S_Q does. Thus, a trade off must be made as to how much total error can be sacrificed for extra gains in correlations.

As a final note on EV reduction, another way of attempting to satisfy the (false) equality is to perform a multidimensional minimization against some metric treating each bin of the merged histogram (H') as the variables to minimize. An implementation of this using the Amoeba routine was explored, but yielded results similar to using S_A. Thus, it is not currently being used.

Implementation

LPKE

The LPKE implemented in this package (`LocalPolyKernelEstimator`) attempts to be as generic as possible. It is written in idiomatic C++11 and requires a compatible ROOT version (version 6 or version 5 with the c++11 flag enabled). The `LocalPolyKernelEstimator` class allows for any domain dimension, order of estimator, coordinate scaling, or kernel function (albeit the kernel must be a function of an array of normalized coordinates). It is a straightforward and naive implementation with respect to computational efficiency and numerical stability. No effort is made to reduce the number of kernel evaluations with a nearest-neighbors-based pruning or ensure that the minimum number of numerically unstable operations are performed. However, for our applications this doesn't appear to be an issue.

The coordinate scaling and kernel types are fed into the template parameters for `LocalPolyKernelEstimator`. The coordinate scaling function must take a single parameter (unnormalized coordinate) that returns the scaled coordinate. The kernel function should accept an `std::array` that represents the set of normalized coordinates and should return the value of the kernel at that coordinate.

The convenience class `LocalLinearKernelEstimator1D` is provided to easy the smoothing of a ROOT `TH1` or `TGraphErrors`. In addition to requiring a coordinate transform function an inverse coordinate transform is necessary to properly construct the smoothed histogram. It allows the easy retrieval of a smoothed `TH1D` given the number of bins to smooth to.

The utility for smoothing a given CDI file is `smoothCalibrations`. This is relatively simple routine that gathers the CDI datasets from a CDI file, smooths the relevant histograms, writes the "ReducedSets" numbers,

and stores them in a new CDI file. Which datasets it smooths are hard coded as well as the EV "ReducedSets" numbers. It checks each histogram to identify the systematics with uncorrelated bins. Those with uncorrelated bins are smoothed bin-by-bin to account for the bin-to-bin independence. Thus, one systematic with N uncorrelated bins will be stored as N smoothed systematics with correlated bins. The central value (denoted "result" in the dataset) is smoothed using the bin errors as weights. All other histograms are smoothed without using bin errors. Finally, the current implementation smooths only along p_T. It treats each η bin as a separate `TH1D` along p_T.

Eigen-Variation Analysis

The eigen-variation analysis utility is designed to aid in the determination of the appropriate number of EV to merge. Running this analysis executable yields two files:

1. A ROOT file with plots of the EV, central value, statistical nuisance analysis, percent difference of covariance matrices, etc.
2. A text file with information on the average and maximum relative differences for total error, correlations, etc.

This can then be used to determine an acceptable amount of EV merging. The number of EV before any merging should be the about the same as the number of EV before smoothing.

Usage

LPKE

The `LocalPolyKernelEstimator` class needs to be initialized with several template parameters:

1. `size_t Dim`: this is the desired domain dimension
2. `size_t Order`: this is the desired order of estimator
3. `typename CoordTransformFunc_t`: the type of the coordinate transformation function
4. `typename KernelFunc_t`: the type of the kernel function
5. `typename Numeric_t`: the numeric type to use in computation

Evaluation of this class is through the '()' operator that takes either a variable number of arguments or an array of coordinates (length `DIM`). Setting the coordinate data is accomplished through two functions; `SetCoordinate` and `AddCoordinate`. Both of these function accept either a variable number of arguments or an array of coordinates (length `DIM` + 2 or `DIM` + 1). Specifically, they need the domain coordinates along with the response coordinate and (optionally) the error associated with the response. `SetCoordinate` is best used in conjunction with `SetNumberOfCoordinates` method to reserve the exact number of data points needed or if you know the index of the point. `AddCoordinate` is convenient for quickly adding an additional data point. Modifying a coordinate can be accomplished through the `SetCoordinate` function with the appropriate index. The bandwidth should be set along each dimension with the function `SetBandwidth` and will be used to create the normalized coordinates.
The `LocalLinearKernelEstimator1D` class is a simplification of the `LocalPolyKernelEstimator` class that eases the creation of a smoothed `TH1D`. It has no template parameters of it's own and is effectively `LocalPolyKernelEstimator` with the following template parameters set:

1. `Dim = 1`
2. `Order = 1`

3. `CoordTransformFunc_t = std::function<double(double)>`
4. `KernelFunc_t = std::function<double(const std::array<double, 1>&)>`
5. `Numeric_t = double`

It adds a function to set the number of bins, set the inverse coordinate transform, load data from a `TH1` or `TGraphErrors`, and make a smoothed `TH1D`.
Example usage:

```
using KernelSmoother = Analysis::LocalLinearKernelEstimator1D;
KernelSmoother smoother;
smoother.SetNbins(100);
smoother.SetCoordTransform(0, TF1("log scale", "log(x)", 1, 10));
smoother.SetInvCoordTransform(0, TF1("exp scale", "exp(x)", 1, 10));
using Coord_t = typename KernelSmoother::Coord_t<KernelSmoother::DIM>;
TF1 k1D("kernel", "TMath::Gaus(x, 0, 1, 1)", -1, 1);
auto kernel = [k1D](const Coord_t &coord) {return k1D(coord[0]);};
smoother.SetKernel(kernel);
smoother.SetBandwidth(0, 0.4);
smoother.UseErrorWeights(false);
// ...
// import 1D histogram to_smooth
// ...
smoother.LoadData(to_smooth);
auto smoothed = smoother.MakeSmoothedTH1D();
// ...
```

The utility to smooth a CDI file is the `SmoothCDI` executable. It takes the following command line arguments:

1. the input CDI file to smooth
2. (optional: default = 100) the number of smooth bins
3. (optional: default = 0.4) the global bandwidth for the p_T axis for the kernel smoother

Look at `scripts/RUN_SMOOTHING.sh` for example usage.

Eigen-Variation Analysis

The utility to analyze eigen-variations is the `AnalyzeEigenVariation` executable. It takes the following command line arguments:

1. the input smoothed CDI file
2. an unsmoothed CDI file used for reference
3. the full path to the CDI dataset to analyze
4. (optional: default = -1) the number of EV to merge in the smoothed EV set, default is none
5. (optional: default = true) deprecated

Look at `scripts/RUN_EIGEN_ANALYSIS.sh` for example usage.