Statistical Methods in the NPStat Package

I. Volobouev, *i.volobouev@ttu.edu*

Version: 5.5.0 Date: April 15, 2022

Contents

1 Preliminaries

The NPStat package provides a C++ implementation of several nonparametric statistical modeling tools together with various supporting code. Nonparametric modeling becomes very useful when there is little prior information available to justify an assumption that the data belongs to a certain parametric family of distributions. Even though nonparametric techniques are usually more computationally intensive than parametric ones, judicious use of fast algorithms in combination with increasing power of modern computers often results in very practical solutions — at least until the "curse of dimensionality" starts imposing its heavy toll.

It will be assumed that the reader of this note is reasonably proficient in C++ programming, and that the meaning of such words as "class", "function", "template", "method", "container", "namespace", *etc.* is obvious from their context.

The latest version of the package code can be downloaded from http://npstat.hepforge.org/ Most classes and functions implemented in NPStat are placed in the "npstat" namespace. A few functions and classes that need Minuit2 [1] installation in order to be meaningfully used belong to the "npsi" namespace. Such functions and classes can be found in the "interfaces" directory of the package.

When a C++ function, class, or template is mentioned in the text for the first time, the header file which contains its declaration is often mentioned as well. If the header file is not mentioned, it is most likely "npstat/stat/NNNN.hh", where NNNN stands for the actual name of the class or function.

An expression "$I(Q)$" is frequently employed in the text, where Q is some logical proposition. $I(Q)$ is the proposition indicator function defined as follows: $I(Q) = 1$ if Q is true and $I(Q) = 0$ if Q is false. For example, a product of $I(0 \leq x \leq 1)$ with some other mathematical function is often used to denote some probability density supported on the $[0, 1]$ interval. Note that $I(\neg Q) = 1 - I(Q)$, where $\neg Q$ is the logical negation of Q.

2 Data Representation

For data storage, the NPStat software relies on the object serialization and I/O capabilities provided by the C++ Geners package [2]. Geners supports persistence of all standard library containers (vectors, lists, maps, *etc.*) including those introduced in the C++11 Standard [3] (arrays, tuples, unordered sets and maps). In addition to data structures supported by Geners, NPStat offers several other persistent containers useful in statistical data analysis: multidimensional arrays, homogeneous n-tuples, and histograms.

Persistent multidimensional arrays are implemented with the ArrayND template (header file "npstat/nm/ArrayND.hh"). This is a reasonably full-featured array class, with support for subscripting (with and without bounds checking), vector space operations (addition, subtraction, multiplication by a scalar), certain tensor operations (index contraction, outer product), slicing, iteration over subranges, linear and cubic interpolation of array values to fractional indices, *etc.* Arrays whose maximum number of elements is known at the compile

time can be efficiently created on the stack.

Homogeneous n-tuples are two-dimensional tables in which the number of columns is fixed while the number of rows can grow dynamically. The stored object type is chosen when the n-tuple is created (it is a template parameter). For homogeneous n-tuples, storage type is the same for every column which allows for more efficient optimization of memory allocation in comparison with heterogeneous n-tuples.[1] Two persistent template classes are provided for homogeneous n-tuples: InMemoryNtuple for n-tuples which can completely fit inside the computer memory (header file "npstat/stat/InMemoryNtuple.hh") and ArchivedNtuple for n-tuples which can grow as large as the available disk space (header file "npstat/stat/ArchivedNtuple.hh"). Both of these classes inherit from the abstract class AbsNtuple which defines all interfaces relevant for statistical analysis of the data these n-tuples contain. Thus InMemoryNtuple and ArchivedNtuple are completely interchangeable for use in various data analysis algorithms.

Arbitrary-dimensional[2] persistent histograms are implemented with the HistoND template (header file "npstat/stat/HistoND.hh"). Both uniform and non-uniform placing of bin edges can be created using axis classes HistoAxis and NUHistoAxis, respectively, as HistoND template parameters. The DualHistoAxis class can be employed for histograms in which only a fraction of axes must have non-uniform bins. Two binning schemes are supported for data sample processing. In the normal scheme, the bin count is incremented if the point coordinates fall inside the given bin (the fill method of the class). In the centroid-preserving scheme, all bins in the neighborhood of the sample point are incremented in such a way that the center of mass of bin increments coincides exactly with the point location (the fillC method)[3]. The objects used as histogram bin contents and the objects used as weights added to the bins when the histogram is filled are not required to have anything in common; the only requirement is that a meaningful binary function (typically, operator+=) can be defined for them. This level of abstraction allows, for example, for using vectors and tensors as histogram bin contents or for implementing profile histograms with the same HistoND template. As the bin contents are implemented with the ArrayND class, all features of ArrayND (vector space operations, slicing, interpolation, *etc*) are available for them as well. Overflows are stored in separate arrays with three cells in each dimension: underflow, overflow, and the central part covered by the regular histogram bins.

[1] The Geners serialization package supports several types of persistent heterogeneous n-tuples including std::vector<std::tuple<...>>, RowPacker variadic template optimized for accessing large data samples row-by-row, and ColumnPacker variadic template optimized for data access column-by-column.

[2] To be precise, the number of dimensions for both histograms and multidimensional arrays can not exceed 31 on 32-bit systems and 63 on 64-bit systems. However, your computer will probably run out of memory long before this limitation becomes relevant for your data analysis.

[3] For one-dimensional histograms, this is called "linear binning" [4]. NPStat supports this technique for uniformly binned histograms of arbitrary dimensionality.

3 Descriptive Statistics

A number of facilities is included in the NPStat package for calculating standard descriptive sample statistics such as mean, covariance matrix, *etc.* Relevant functions, classes, and templates are described below.

3.1 Functions and Classes for Use with Point Clouds

arrayStats (header file "npstat/stat/arrayStats.hh") — This function estimates the population mean (μ), standard deviation (σ), skewness (s), and kurtosis (k) for a one-dimensional set of points. A numerically sound two-pass algorithm is used. The following formulae are implemented (N is the number of points in the sample):

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i \tag{1}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu)^2} \tag{2}$$

$$s = \frac{N}{(N-1)(N-2)\sigma^3}\sum_{i=0}^{N-1}(x_i-\mu)^3 \tag{3}$$

$$g_2 = N\frac{\sum_{i=0}^{N-1}(x_i-\mu)^4}{\left(\sum_{i=0}^{N-1}(x_i-\mu)^2\right)^2} - 3 \tag{4}$$

$$k = \frac{N-1}{(N-2)(N-3)}((N+1)g_2+6)+3 \tag{5}$$

Note that the population kurtosis estimate is defined in such a way that its expectation value for a sample drawn from the Gaussian distribution equals 3 rather than 0. For more details on the origin of these formulae consult Ref. [5].

empiricalCdf (header file "npstat/stat/StatUtils.hh") — This function evaluates the empirical cumulative distribution function (ECDF) for a sample of points. The points must be sorted in the increasing order by the user before the function call is made. The ECDF is defined as follows. Suppose, the N sorted point values are $\{x_0, x_1, ..., x_{N-1}\}$ and all x_i are distinct (*i.e.,* there are no ties). Then

$$\mathrm{ECDF}(x) = \begin{cases} 0 & \text{if } x \le x_0, \\ \frac{1}{N}\left(\frac{x-x_i}{x_{i+1}-x_i} + i + 1/2\right) & \text{if } x_i < x \le x_{i+1} \text{ and } x < x_{N-1}, \\ 1 & \text{if } x \ge x_{N-1}. \end{cases} \tag{6}$$

When all sample points are distinct, ECDF(x) is discontinuous at $x = x_0$ and $x = x_{N-1}$ and continuous for all other x. If the sample has more than one point with the same x, let say x_k

and x_{k+1} where $k > 0$ and $k < N-2$, x_k also becomes a point of discontinuity. ECDF(x_k) will be set to $\frac{1}{N}(k+1/2)$, the value coincident with the function left limit.

empiricalQuantile (header file "npstat/stat/StatUtils.hh") — This is the empirical quantile function, EQF(y), which provides an inverse to ECDF(x). For any x from the interval $[x_0, x_{N-1}]$, EQF(ECDF(x)) = x (up to numerical round-off errors). If $x < x_0$ then EQF(ECDF(x)) = x_0 and if $x > x_{N-1}$ then EQF(ECDF(x)) = x_{N-1}.

MultivariateSumAccumulator and MultivariateSumsqAccumulator — These classes are intended for calculating means and covariance matrices for multivariate data samples in a numerically stable manner. The classes can be used inside user-implemented cycles over sets of points or with cycleOverRows and other similar methods of the AbsNtuple class. Two passes over the data are necessary for calculating the covariance matrix, first with MultivariateSumAccumulator and second with MultivariateSumsqAccumulator. The multivariate mean is initially found from

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{i=0}^{N-1}\mathbf{x}_i \tag{7}$$

and then an estimate of the population covariance matrix is calculated as

$$V = \frac{1}{N-1}\sum_{i=0}^{N-1}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T \tag{8}$$

MultivariateWeightedSumAccumulator and MultivariateWeightedSumsqAccumulator — These classes are intended for calculating means and covariance matrices for multivariate data samples in which points enter with non-negative weights:

$$\overline{\mathbf{x}} = \frac{\sum_{i=0}^{N-1} w_i\mathbf{x}_i}{\sum_{i=0}^{N-1} w_i}, \tag{9}$$

$$V = \frac{N_{eff}}{N_{eff}-1}\,\frac{\sum_{i=0}^{N-1} w_i(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T}{\sum_{i=0}^{N-1} w_i}, \tag{10}$$

where N_{eff} is the effective number of entries in a weighted sample[4]:

$$N_{eff} = \frac{\left(\sum_{i=0}^{N-1} w_i\right)^2}{\sum_{i=0}^{N-1} w_i^2}. \tag{11}$$

Two passes over the data are required, first with MultivariateWeightedSumAccumulator and then with MultivariateWeightedSumsqAccumulator.

StatAccumulator — Persistent class which determines the minimum, the maximum, the mean, and the standard deviation of a sample of values using a single-pass algorithm. It can

[4] N_{eff} is also known as the Kish's effective sample size. Note that this estimate of V makes sense only if the weights and point locations are statistically independent from each other. If the weights are functions of the observed $\mathbf{x}_i$ then there appears to be no general estimate that is independent of the weighting function.

be used when the two-pass calculation is either impossible or impractical for some reason. This class can be conveniently employed as a bin content type for HistoND (this turns HistoND into a profile histogram). StatAccumulator maintains a running average which is updated after accumulating 2^K points, $K = 0, 1, 2, ...$ The numerical precision of the standard deviation determined in this manner is not as good as in the two-pass method but it is usually much better than that expected from a naive implementation which simply accumulates sample moments about 0 and then evaluates σ^2 according to $\frac{N}{N-1}(\overline{\mathbf{x}^2} - \overline{\mathbf{x}}^2)$ (this formula can suffer from a severe subtractive cancellation problem).

WeightedStatAccumulator — Similar class for calculating mean and standard deviation of a sample of weighted values using a single-pass algorithm.

StatAccumulatorArr — This class provides functionality similar to an array of StatAccumulator objects but with a more convenient interface for accumulating sample values (cycling over the input values is performed by the class itself).

StatAccumulatorPair — Two StatAccumulator objects augmented by the sum of cross terms which allows for subsequent calculation of covariance. The covariance is estimated using a formula which can potentially suffer from subtractive cancellation[5]. Therefore, this class should not be used to process large samples of points in which, for one of the variables or for both of them, the absolute value of the mean can be much larger than the standard deviation. This class is persistent.

CrossCovarianceAccumulator — To accumulate the sums for covariance matrix calculation according to formula 8 in d-dimensional space, one has to keep track of $d(d+1)/2$ sums of squares and cross terms. Sometimes only a small part of the complete covariance matrix is of interest. In this case d can often be split into subspaces of dimensionalities d_1 and d_2, $d = d_1 + d_2$, so that the interesting part of covariance matrix is dimensioned $d_1 \times d_2$ (plus d diagonal terms). For large values of d, accumulating $d + d_1 \times d_2$ sums instead of $d(d+1)/2$ can mean big difference in speed and memory consumption, especially in case $d_1 \ll d_2$ (or $d_1 \gg d_2$). The CrossCovarianceAccumulator allows its users to do just that: it accumulates cross terms between two arrays with d_1 and d_2 elements but not the cross terms between elements of the same array. The covariances are estimated by a single-pass method using a formula which can suffer from subtractive cancellation[5]. Use with care.

SampleAccumulator — This class stores all values it gets in an internal buffer. Whenever mean, standard deviation, or some quantile function values are needed for the accumulated sample, they are calculated from the stored values using numerically sound techniques. Mean and standard deviation are calculated according to Eqs. 1 and 2. Covariance and correlations can be calculated for a pair of SampleAccumulator objects with the same number of stored elements by corresponding methods of the class. Empirical CDF and empirical quantile calculations are handled by calling empiricalCdf and empiricalQuantile functions internally, preceded by data sorting. This class is persistent and can be used as the bin content template parameter for HistoND.

WeightedSampleAccumulator — Similar to SampleAccumulator, but intended for calculating various statistics for samples of weighted points.

[5]This problem is partially alleviated by using long double type for internal calculations.

Kendall's τ and Spearman's ρ rank correlation coefficients can be estimated by functions sampleKendallsTau and sampleSpearmansRho, respectively. These functions are declared in the header files "npstat/stat/kendallsTau.hh" and "npstat/stat/spearmansRho.hh", respectively.

3.2 Functions for Use with Binned Data

histoMean (header file "npstat/stat/histoStats.hh") — This function calculates the histogram center of mass according to Eq. 9 with bin contents used as weights for bin center coordinates. Note that the code of this function does not enforce non-negativity of all bin values, and the result of 9 will not necessarily make a lot of sense in case negative weights are present. In order to check that all bins are non-negative and that there is at least one positive bin, you can call the isDensity method of the ArrayND class on the histogram bin contents. The histoMean function is standalone and not a member of the HistoND class because Eq. 9 is not going to make sense for all possible bin types.

histoCovariance (header file "npstat/stat/histoStats.hh") — This function estimates the population covariance matrix for histogrammed data according to Eq. 10, with bin contents used as weights for bin center coordinates. As for histoMean, results produced by histoCovariance will not make much sense in case negative bin values are present.

arrayCoordMean (header file "npstat/stat/arrayStats.hh") — Similar to histoMean but the bin values are provided in an ArrayND object and bin locations are specified by additional arguments. Only uniform bin spacing is supported by this function (unlike histoMean which simply gets its bin centers from the argument histogram).

arrayCoordCovariance (header file "npstat/stat/arrayStats.hh") — Similar to histoCovariance but the bin values are provided in an ArrayND object and bin locations are specified by additional arguments. Uniform binning only.

arrayShape1D (header file "npstat/stat/arrayStats.hh") — Estimates population mean, standard deviation, skewness, and kurtosis for one-dimensional histogrammed data with equidistant bins. Bin values b_i used as weights are provided in an ArrayND object and bin locations x_i are specified by additional arguments. The mean and the standard deviations squared are calculated according to Eqs. 9 and 10 reduced to one dimension. The skewness is estimated as

$$s = \frac{N_{eff}^2}{(N_{eff}-1)(N_{eff}-2)\sigma^3}\,\frac{\sum_{i=0}^{N_b-1} b_i(x_i-\mu)^3}{\sum_{i=0}^{N_b-1} b_i}, \tag{12}$$

with N_{eff} defined by Eq. 11. The kurtosis is found from Eq. 5 in which N is replaced by N_{eff} and, instead of Eq. 4, g_2 is determined from

$$g_2 = \sum_{i=0}^{N_b-1} b_i\,\frac{\sum_{i=0}^{N_b-1} b_i(x_i-\mu)^4}{\left(\sum_{i=0}^{N_b-1} b_i(x_i-\mu)^2\right)^2} - 3. \tag{13}$$

arrayQuantiles1D (header file "npstat/stat/arrayStats.hh") — This function treats one-dimensional arrays as histograms and determines the values of the quantile function for

a given set of cumulative density values (after normalizing the histogram area to 1). Each bin is considered to have uniform probability density between its edges. If you need to perform this type of calculation more than once with the same array, it will be more efficient to construct a BinnedDensity1D object and then to use its quantile method instead of calling the arrayQuantiles1D function multiple times.

arrayEntropy (header file "npstat/stat/arrayStats.hh") — This function calculates

$$H = -\frac{1}{N_b}\sum_{i=0}^{N_b-1} b_i \ln b_i \tag{14}$$

This formula is appropriate for arrays that tabulate probability density values on uniform grids. To convert the result into the actual entropy of the density, multiply it by the volume of the distribution support.

3.3 ArrayND Projections

The following set of classes works with ArrayND class and assists in making array "projections". Projections reduce array dimensionality by calculating certain array properties over projected indices. For example, one might want to create a one-dimensional array, a, from a three-dimensional array, b, by summing all array elements with the given second index: $a_j = \sum_{i,k} b_{ijk}$ (this means that the first and the third indices of the array are "projected"). This and other similar operations can be performed by the project method of the ArrayND class. What is done with the array elements when the projection is performed can be specified by the "projector" argument of the project method. In particular, this argument can be an object which performs some statistical analysis of the projected elements.

ArrayMaxProjector (header file "npstat/stat/ArrayProjectors.hh") — For each value of the array indices which are not projected, finds the maximum array element for all possible values of projected indices.

ArrayMinProjector (header file "npstat/stat/ArrayProjectors.hh") — Finds the minimum array element for all possible values of projected indices.

ArraySumProjector (header file "npstat/stat/ArrayProjectors.hh") — Sums all array values in the iteration over projected indices.

ArrayMeanProjector (header file "npstat/stat/ArrayProjectors.hh") — Calculates the mean array value over projected indices.

ArrayMedianProjector (header file "npstat/stat/ArrayProjectors.hh") — Calculates the median array value over projected indices.

ArrayStdevProjector (header file "npstat/stat/ArrayProjectors.hh") — Calculates the standard deviation of array values encountered during the iteration over projected indices.

ArrayRangeProjector (header file "npstat/stat/ArrayProjectors.hh") — Calculates the "range" of array values over projected indices. "Range" is defined here as the difference between 75^{th} and 25^{th} percentiles of the sample values divided by the distance between 75^{th} and 25^{th} percentiles of the Gaussian distribution with $\sigma = 1$. Therefore, "range" can be used as a robust estimate of the standard deviation.

All "projectors" are derived either from AbsArrayProjector class in case the calculated quantity depends on the array indices (header file "npstat/nm/AbsArrayProjector.hh") or from AbsVisitor class in case it is sufficient to know just the element values (header file "npstat/nm/AbsVisitor.hh"). Naturally, the user can supply his/her own projector implementations derived from these base classes for use with the the project method of the ArrayND class.

4 Statistical Distributions

A number of univariate and multivariate statistical distributions is supported by the NPStat package. All implementations of univariate continuous distributions share a common interface and can be used interchangeably in a variety of statistical algorithms. A similar approach has been adopted for univariate discrete and multivariate continuous distributions.

4.1 Univariate Continuous Distributions

All classes which represent univariate continuous distributions inherit from the AbsDistribution1D abstract base class. These classes must implement methods density (probability density function), cdf (cumulative distribution function), exceedance (exceedance probability is just $1 - \text{cdf}$, but direct application of this formula is often unacceptable in numerical calculations due to subtractive cancellation), and quantile (the quantile function, inverse of cdf). For a large number of statistical distributions, the probability density function looks like $\frac{1}{\sigma}\, p\left(\frac{x-\mu}{\sigma}\right)$, where μ and σ are the distribution location and scale parameters, respectively. Distributions of this kind should inherit from the base class AbsScalableDistribution1D which handles proper scaling and shifting of the argument (this base class itself inherits from AbsDistribution1D, and it is declared in the same header file "npstat/stat/AbsDistribution1D.hh").

Some of the standard univariate continuous distributions implemented in NPStat are listed in Table 1. A few special distributions less frequently encountered in the statistical literature are described in more detail in the following subsections. Of course, user-developed classes inheriting from AbsDistribution1D or AbsScalableDistribution1D can also be employed with all NPStat functions and classes that take an instance of AbsDistribution1D as one of parameters.

Table 1: Continuous univariate distributions included in NPStat. $P_n(x)$ are the Legendre polynomials. Parameters μ and σ are not shown for scalable distributions. When not given explicitly, the normalization constant $\mathcal{N}$ ensures that $\int_{-\infty}^{\infty} p(x)dx = 1$. Most of the classes listed in this table are declared in the header file "npstat/stat/Distributions1D.hh". If the distribution is not declared in that header then it has a dedicated header with the same name, *e.g.*, "npstat/stat/TruncatedDistribution1D.hh" for TruncatedDistribution1D.

Class Name	$p(x)$	Scalable?
Uniform1D	$I(0 \le x \le 1)$	yes
IsoscelesTriangle1D	$(1 - \lvert x\rvert) I(-1 \le x \le 1)$	yes
Exponential1D	$e^{-x} I(x \ge 0)$	yes
Quadratic1D	$(1 + aP_1(2x-1) + bP_2(2x-1))\, I(0 \le x \le 1)$	yes
LogQuadratic1D	$\mathcal{N} \exp(aP_1(2x-1) + bP_2(2x-1))\, I(0 \le x \le 1)$	yes
Gauss1D	$\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$	yes
GaussianMixture1D	$\frac{1}{\sqrt{2\pi}} \sum_i \frac{w_i}{\sigma_i} e^{-(x-\mu_i)^2/(2\sigma_i^2)}$, where $\sum_i w_i = 1$	yes
TruncatedGauss1D	$\frac{\mathcal{N}}{\sqrt{2\pi}} e^{-x^2/2}\, I(-n_\sigma \le x \le n_\sigma)$	yes
MirroredGauss1D	$I(0 \le x \le 1)\, \frac{1}{\sqrt{2\pi} s} \sum_{i=-\infty}^{\infty} \left[e^{-(x-m+2\,i)^2/(2s^2)} + e^{-(x+m+2\,i)^2/(2s^2)} \right]$, where $0 \le m \le 1$ and $s > 0$	yes
BifurcatedGauss1D	$\mathcal{N} \left[e^{-x^2/(2\alpha)} I(-n_{\sigma,L} \le x < 0) + e^{-x^2/(2(1-\alpha))} I(0 \le x < n_{\sigma,R}) \right]$	yes
UGaussConvolution1D	$\frac{1}{\sqrt{2\pi}(b-a)}\, e^{-x^2/2} * I(a \le x \le b)$	yes
SymmetricBeta1D	$\mathcal{N}\, (1-x^2)^p\, I(-1 < x < 1)$	yes
Beta1D	$\mathcal{N}\, x^{\alpha-1}(1-x)^{\beta-1}\, I(0 < x < 1)$	yes
Gamma1D	$\mathcal{N}\, x^{\alpha-1} e^{-x}\, I(x > 0)$	yes
Pareto1D	$\alpha x^{-\alpha-1} I(x \ge 1)$	yes
Huber1D	$\mathcal{N}\, [e^{-x^2/2} I(\lvert x\rvert \le a) + e^{a(a/2-\lvert x\rvert)}(1 - I(\lvert x\rvert \le a))]$	yes
Cauchy1D	$\pi^{-1}(1+x^2)^{-1}$	yes
StudentsT1D	$\mathcal{N}\, (1 + x^2/N_{dof})^{-(N_{dof}+1)/2}$	yes
Moyal1D	$\frac{1}{\sqrt{2\pi}} e^{-(x+e^{-x})/2}$	yes
Logistic1D	$e^{-x}\, (1+e^{-x})^{-2}$	yes
Continued on the next page		

Table 1 — continued from the previous page

Class Name	$p(x)$	Scalable?
Tabulated1D	Defined by a table of equidistant values on the [0, 1] interval, interpolated by a polynomial (up to cubic). The first table point is at $x = 0$ and the last is at $x = 1$.	yes
BinnedDensity1D	Defined by a table of N equidistant values on the [0, 1] interval, with optional linear interpolation. The first table point is at $x = 1/(2N)$ and the last is at $x = 1 - 1/(2N)$. Useful for converting 1-d histograms into distributions.	yes
QuantileTable1D	Defined by a table of N equidistant quantile function values on the [0, 1] interval with linear interpolation between these values (so that density looks like a histogram with equal area bins). The first table point is at $x = 1/(2N)$ and the last is at $x = 1 - 1/(2N)$. Useful for converting data samples into distributions by sampling empirical quantiles.	yes
DistributionMix1D	$\sum_i w_i p_i(x)$, where $\sum_i w_i = 1$	no
DeltaMixture1D	$\sum_i w_i \delta(x - x_i)$, where $\sum_i w_i = 1$	yes
LocationScaleFamily1D	$\frac{1}{s} p_{\text{other}}\left(\frac{x-m}{s}\right)$, useful for converting non-scalable 1-d distributions into scalable.	yes
RatioOfNormals	Ratio of two correlated normal random variables, as described in [6].	no
LeftCensoredDistribution	$f\, p_{\text{other}}(x) + (1 - f)\, \delta(x - x_{-\infty})$	no
RightCensoredDistribution	$f\, p_{\text{other}}(x) + (1 - f)\, \delta(x - x_{+\infty})$	no
TruncatedDistribution1D	$\mathcal{N}\, p_{\text{other}}(x)\, I(x_{\min} \le x \le x_{\max})$	no
TransformedDistribution1D	$p_{\text{other}}(y) \left\lvert \frac{dy}{dx} \right\rvert$ for monotonous $y(x)$	no

4.2 Johnson Curves

The density functions of the Johnson distributions [7, 8, 9] are defined as follows:

$$S_U \text{ (unbounded)}: \quad p(x) = \frac{\delta}{\lambda\sqrt{2\pi\left(1 + \left(\frac{x-\xi}{\lambda}\right)^2\right)}}\, e^{-\frac{1}{2}\left(\gamma + \delta \sinh^{-1}\left(\frac{x-\xi}{\lambda}\right)\right)^2} \tag{15}$$

$$S_B \text{ (bounded)}: \quad p(x) = \frac{\delta}{\lambda\sqrt{2\pi}\left(\frac{x-\xi}{\lambda}\right)\left(1 - \frac{x-\xi}{\lambda}\right)}\, e^{-\frac{1}{2}\left(\gamma + \delta \log\left(\frac{x-\xi}{\xi+\lambda-x}\right)\right)^2} I(\xi < x < \xi + \lambda) \tag{16}$$

They are related to the normal distribution, $N(\mu, \sigma)$, by simple variable transformations. Variable z distributed according to $N(0, 1)$ can be obtained from Johnson's variates x by

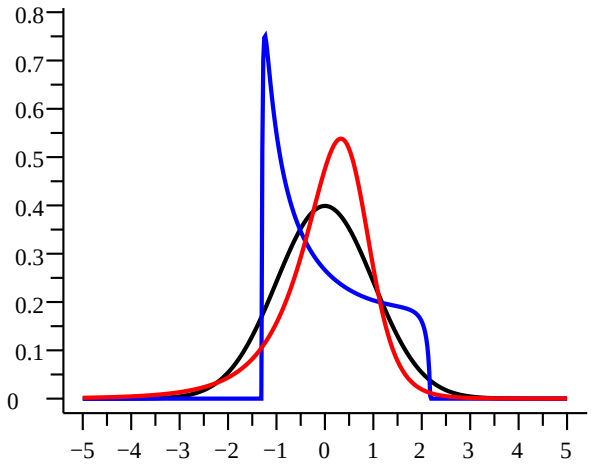

Figure 1: Black: $s = 0,\ k = 3$, Gaussian. Red: $s = -1.5,\ k = 10,\ S_U$. Blue: $s = 0.5,\ k = 2,\ S_B$. For all three curves, the mean is 0 and the standard deviation is 1.

transformation $z = \gamma + \delta f\left(\frac{x-\xi}{\lambda}\right)$:

$$\begin{aligned} f(y) &= \sinh^{-1}(y) && S_U \text{ curves} \\ f(y) &= \log\left(\frac{y}{1-y}\right) \quad (0 < y < 1) && S_B \text{ curves} \end{aligned}$$

Both densities become arbitrarily close to the lognormal density (for which variable $z = \gamma + \delta \log(x - \xi)$ is normally distributed) in the limit $\gamma \to \infty$ and to the normal distribution in the limit $\delta \to \infty$ (ξ and λ have to be adjusted accordingly). Johnson's parameterization of the lognormal density is

$$p(x) = \frac{\delta}{\sqrt{2\pi}(x-\xi)} e^{-\frac{1}{2}(\gamma+\delta\log(x-\xi))^2} I(x > \xi) \tag{17}$$

Together with their limiting cases, Johnson's S_U and S_B distributions attain *all possible values of skewness and kurtosis*. Unfortunately, parameters ξ, λ, γ, and δ of S_U and S_B in Eqs. 15 and 16 have no direct relation to each other. Crossing the lognormal boundary requires a discontinuous change in the parameter values which is rather inconvenient for practical data fitting purposes. This problem can be alleviated by reparameterizing the functions in terms of mean μ, standard deviation σ, skewness s, and kurtosis k, so that the corresponding curve type and the original parameters $\xi, \lambda, \gamma, \delta$ are determined numerically. Examples of Johnson's density functions are shown in Fig 1.

Johnson's S_U and S_B curves are implemented in NPStat with classes JohnsonSu and JohnsonSb, respectively (header file "npstat/stat/JohnsonCurves.hh"). Both of these distributions are parameterized by μ, σ, s, and k, with an automatic internal conversion into

$\xi, \lambda, \gamma, \delta$. An original algorithm was developed to perform this conversion, based in part on ideas from Refs. [10, 11]. The lognormal distribution parameterized by μ, σ, and s is implemented by the LogNormal class (header file "npstat/stat/Distributions1D.hh"). The class JohnsonSystem (header file "npstat/stat/JohnsonCurves.hh") can be used when automatic switching between S_U, S_B, lognormal, and Gaussian distributions is desired.

4.3 Composite Distributions

Composite distributions are built out of two or more component distributions. One of these component distributions is arbitrary while all others must have a density supported on the interval [0, 1]. Suppose, $G_k(x)$, $k = 1, 2, 3, ...$ are cumulative distribution functions with corresponding densities $g_i(x)$ proportional to $I(0 \leq x \leq 1)$. Then, if $H(x)$ is a cumulative distribution function with density $h(x)$, $F_1(x) = G_1(H(x))$ is also a cumulative distribution function with density $f_1(x) = h(x)\, g_1(H(x))$. Similarly, $F_2(x) = G_2(F_1(x))$ is a cumulative distribution with density $f_2(x) = f_1(x) g_2(F_1(x)) = h(x)\, g_1(H(x)) g_2(G_1(H(x)))$. We can now construct $F_3(x) = G_3(F_2(x))$ and so on. This sequence can be terminated after an arbitrary number of steps.

Note that $f_1(x) = h(x)$ in case $g_1(x)$ is a uniform probability density. Small deviations from uniformity in $g_1(x)$ will lead to corresponding small changes in $f_1(x)$. The resulting density model can be quite flexible, even if the component distributions $H(x)$ and $G_1(x)$ are simple. Therefore, data samples with complicated sets of features can be modeled as follows: construct an approximate model $h(x)$ first, even if it does not fit the sample quite right. Then transform the data points x_i to the [0, 1] interval according to $y_i = H(x_i)$. The density of points y_i[6] can now be fitted to another parametric distribution (including composite one) or it can be modeled by nonparametric techniques [14]. Due to the elimination of the boundary bias, the LOrPE density estimation method described in Section 6.2 becomes especially useful in the latter approach. Once the appropriate $G(y)$ is constructed (parametrically or not), the resulting composite density will provide a good fit to the original set of points x_i[7].

The composite distributions are implemented in NPStat with the CompositeDistribution1D class. One composite distribution can be used to construct another, thus allowing for composition chains of arbitrary length, as described at the beginning of this subsection. Several pre-built distributions of this type are included in the NPStat package (they are declared in the header file "npstat/stat/CompositeDistros1D.hh"). Flexible models potentially capable of fitting wide varieties of univariate data samples are implemented by the JohnsonLadder and

[6]This density is called "relative density" [12] or "comparison density" [13] in the statistical literature. Note that $h(x)$ should be selected in such a way that the ratio between the unknown population density of the sample under study and $h(x)$ should be bounded for all x. If it is not, the relative density will usually be unbounded, and its subsequent representation by polynomials or log-polynomials will not lead to a consistent estimate. Johnson curves often work reasonably well as $h(x)$.

[7]There is, of course, a close connection between this density modeling approach and a number of goodness-of-fit techniques based on the comparison of the empirical cdf with the cdf of the fitted density. For example, using Legendre polynomials to model $\log(g_1(x))$, as in the LogQuadratic1D density, directly improves the test criterion used in the Neyman's smooth test for goodness-of-fit [13, 15].

BinnedCompositeJohnson classes. In both of these, Johnson curves are used as $H(x)$. JohnsonLadder takes an arbitrarily long sequence of parametric LogQuadratic1D distributions for $G_k(x)$ while BinnedCompositeJohnson is using a single nonparametric BinnedDensity1D as $G_1(x)$.

4.4 Univariate Discrete Distributions

Classes which represent univariate discrete distributions inherit from the AbsDiscreteDistribution1D abstract base class. The interface defined by this base class differs in a number of ways from the AbsDistribution1D interface. Instead of the method density used with arguments of type "double", discrete distributions have the method probability defined for the arguments of type "long int". The methods cdf and exceedance have the same signatures as corresponding methods of continuous distributions, but the quantile function is returning long integers. The method random generates long integers as well.

Discrete univariate distributions which can be trivially shifted should inherit from the ShiftableDiscreteDistribution1D base class which handles the shift operation. There is, however, no operation analogous to scaling of continuous distributions.

Univariate discrete distributions implemented in NPStat are listed in Table 2. Mixtures

Table 2: Discrete univariate distributions included in NPStat. The location parameter is not shown explicitly for shiftable distributions. Classes listed in this table are declared in the header file "npstat/stat/DiscreteDistributions1D.hh".

Class Name	$p(n)$	Shiftable?
DiscreteTabulated1D	Defined by a table of probability values. Normalization is computed automatically.	yes
Poisson1D	$\frac{\lambda^n}{n!}e^{-\lambda}$	no

of discrete univariate distributions with finite support can be implemented using the function pooledDiscreteTabulated1D (header file "npstat/stat/DiscreteDistributions1D.hh").

4.5 Multivariate Continuous Distributions

All classes which represent multivariate continuous distributions inherit from the AbsDistributionND abstract base class. These classes must implement methods density (probability density function) and unitMap (mapping from the unit d-dimensional cube, U_d, into the density support region). Densities that can be shifted and scaled in each coordinate separately should be derived from the AbsScalableDistributionND base class (which itself inherits from AbsDistributionND). Classes representing densities that look like $p(\mathbf{x}) = \prod q(x_i)$, where $q(x)$ is some one-dimensional density, should be derived from the HomogeneousProductDistroND base class (the three base classes just mentioned are declared in the "np-

stat/stat/AbsDistributionND.hh" header file). Simple classes inheriting from AbsDistributionND are listed in Table 3.

Table 3: Continuous multivariate distributions included in NPStat. These distributions are predominantly intended for use as multivariate density estimation kernels. Shifts and scale factors are not shown for scalable distributions. Here, "scalability" means the ability to adjust the shift and scale parameters in each dimension, not the complete bandwidth matrix. When not given explicitly, the normalization constant $\mathcal{N}$ ensures that $\int p(\mathbf{x})d\mathbf{x} = 1$. All of these distributions are declared in the header file "npstat/stat/DistributionsND.hh" with exception of ScalableGaussND which has its own header.

Class Name	$p(\mathbf{x})$	Scalable?
ProductDistributionND	$\prod_{i=1}^{d} p_i(x_i)$	depends on components
UniformND	$\prod_{i=1}^{d} I(0 \leq x_i \leq 1)$	yes
ScalableGaussND	$(2\pi)^{-d/2} e^{-\lvert\mathbf{x}\rvert^2/2}$	yes
ProductSymmetricBetaND	$\mathcal{N} \prod_{i=1}^{d} (1 - x_i^2)^p I(-1 < x_i < 1)$	yes
ScalableSymmetricBetaND	$\mathcal{N}(1 - \lvert\mathbf{x}\rvert^2)^p I(\lvert\mathbf{x}\rvert < 1)$	yes
ScalableHuberND	$\mathcal{N}\left[e^{-\lvert\mathbf{x}\rvert^2/2} I(\lvert\mathbf{x}\rvert \leq a) + e^{\,a(a/2-\lvert\mathbf{x}\rvert)}(1 - I(\lvert\mathbf{x}\rvert \leq a))\right]$	yes
RadialProfileND	Arbitrary centrally-symmetric density. Defined by its radial profile: a table of equidistant values on the [0, 1] interval, interpolated by a polynomial (up to cubic). The first table point is at $\lvert\mathbf{x}\rvert = 0$ and the last is at $\lvert\mathbf{x}\rvert = 1$. For $\lvert\mathbf{x}\rvert > 1$ the density is 0. Normalization is computed automatically.	yes
BinnedDensityND	Defined by a table of values on U_d, equidistant in each dimension, with optional multilinear interpolation. In each dimension, the first table point is at $x_i = 1/(2N_i)$ and the last is at $x_i = 1 - 1/(2N_i)$. Useful for converting multivariate histograms into distributions.	yes

4.6 Copulas

For any continuous multivariate density $p(\mathbf{x}|\mathbf{a})$ in a d-dimensional space X of random variables $\mathbf{x} \in X$ depending on a vector of parameters $\mathbf{a}$, we define d marginal densities $p_i(x_i|\mathbf{a})$,

$i = 1, \ldots, d$ by

$$p_i(x_i|\mathbf{a}) \equiv \int p(x_1, \ldots, x_d|\mathbf{a}) \prod_{\substack{j=1 \\ j\neq i}}^{d} dx_j$$

with their corresponding cumulative distribution functions

$$F_i(x_i|\mathbf{a}) \equiv \int_{-\infty}^{x_i} p_i(\tau|\mathbf{a}) d\tau.$$

For each point $\mathbf{x} \in X$, there is a corresponding point $\mathbf{y}$ in a unit d-dimensional cube U_d such that $y_i(x_i) \equiv F_i(x_i|\mathbf{a})$, $i = 1, \ldots, d$. The *copula density* is defined on U_d by

$$c(\mathbf{y}(\mathbf{x})|\mathbf{a}) \equiv \frac{p(\mathbf{x}|\mathbf{a})}{\prod_{i=1}^{d} p_i(x_i|\mathbf{a})}. \tag{18}$$

Copula density (as well as its corresponding multivariate distribution function $C(\mathbf{y}|\mathbf{a})$, or just *copula*) contains all information about mutual dependence of individual variables x_i. It can be shown that all copula marginals are uniform and, conversely, that any distribution on U_d whose marginals are uniform is a copula [16].

Naturally, when the copula and the marginals of some multivariate density are known, the density itself is expressed by

$$p(\mathbf{x}|\mathbf{a}) = c(\mathbf{y}(\mathbf{x})|\mathbf{a}) \prod_{i=1}^{d} p_i(x_i|\mathbf{a}). \tag{19}$$

Note that $c(\mathbf{y}|\mathbf{a}) = 1$ for all $\mathbf{y}$ if and only if all x_i are independent and the density $p(\mathbf{x}|\mathbf{a})$ is fully factorizable.

The NPStat package allows its users to model multivariate continuous distributions using copula and marginals with the aid of CompositeDistributionND class. An object of this class is normally constructed out of user-provided copula and marginals. CompositeDistributionND object can also be constructed from histogram bins. Several standard copulas are implemented: Gaussian, Student's-t, and Farlie-Gumbel-Morgenstern [16]. The corresponding class names are GaussianCopula, TCopula, and FGMCopula (these classes are declared in the header file "npstat/stat/Copulas.hh").

Empirical multivariate copulas densities can be constructed as follows. Let's assume that there are no coincident point coordinates in each dataset variable. We can sort all N elements of the data set in the increasing order in each coordinate separately and assign to each data point i a multi-index $\{m_{i0}, ..., m_{id}\}$. For each dimension k, m_{ik} represents the number of the point i in the sequence ordered by the dimension k coordinate, so that $0 \leq m_{ik} < N$. The empirical copula density, ECD($\mathbf{x}$), is then defined by

$$\mathrm{ECD}(\mathbf{x}) = \sum_{i=0}^{N-1} \prod_{k=0}^{d-1} \delta\left(x_k - \frac{m_{ik} + 1/2}{N}\right), \tag{20}$$

where $\delta(x)$ is the Dirac delta function. To get a better idea about locations of these delta functions, think of an $N \times N \times ... \times N$ uniform grid in d dimensions on which N points are placed in such a way that no two points share the same coordinate in any of the dimensions. In two dimensions, this is like the placement of chess pieces in the eight queens puzzle [17] which is simplified to use rooks instead of queens [18].

In the NPStat package, empirical copula densities can be approximated by histograms defined on U_d. Construction of empirical copula histograms can be performed by functions empiricalCopulaHisto (header file "npstat/stat/empiricalCopulaHisto.hh") and empiricalCopulaDensity (header file "npstat/stat/empiricalCopula.hh"). The integrated empirical copulas can be constructed on uniform grids by the calculateEmpiricalCopula function (header "npstat/stat/empiricalCopula.hh").

The Spearman's rank correlation coefficient, ρ, can be estimated from two-dimensional empirical copulas using functions spearmansRhoFromCopula and spearmansRhoFromCopulaDensity declared in the header file "npstat/stat/spearmansRho.hh". These functions evaluate the following integrals numerically:

$$\rho = 12 \int_0^1 \int_0^1 C(\mathbf{y}|\mathbf{a}) dy_0 dy_1 - 3 \qquad \text{used by spearmansRhoFromCopula} \tag{21}$$

$$\rho = 12 \int_0^1 \int_0^1 c(\mathbf{y}|\mathbf{a}) y_0 y_1 dy_0 dy_1 - 3 \qquad \text{used by spearmansRhoFromCopulaDensity} \tag{22}$$

While these two formulas are identical mathematically [16], the approximations used in numerical integration and differentiation will usually lead to slightly different results returned by these functions for some copula and its corresponding density.

The Kendall's rank correlation coefficient, τ, can be estimated from the empirical copulas using the function kendallsTauFromCopula declared in the header "npstat/stat/kendallsTau.hh". This function evaluates the following formula numerically:

$$\tau = 4 \int_0^1 \int_0^1 C(\mathbf{y}|\mathbf{a}) c(\mathbf{y}|\mathbf{a}) dy_0 dy_1 - 1. \tag{23}$$

The mutual information between variables can be estimated from copula densities represented on uniform grids according to Eq. 14[8].

Decomposition of multivariate densities into the marginals and the copula is useful not only for a subsequent analysis of mutual dependencies of the variates but also for implementing nonparametric density interpolation, as described in the next section.

5 Nonparametric Interpolation of Densities

There are numerous problems in High Energy Physics in which construction of some probability density requires extensive simulations and, due to the CPU time limitations, can only

[8] Mutual information is simply the negative of the copula entropy [19].

be performed for a limited number of parameter settings. This is typical, for example, for "mass templates" which depend on such parameters as the pole mass of the particle under study, sample background fraction, detector jet energy scale, *etc.* It is often desirable to have the capability to evaluate such a density for arbitrary parameter values. It is sometimes possible to address this problem by postulating an explicit parametric model and fitting that model to the sets of simulated distributions. However, complex dependence of the distribution shapes on the parameter values often leads to infeasibility of this approach. Then it becomes necessary to interpolate the densities without postulating a concrete model.

5.1 Interpolation of Univariate Densities using Quantile Functions

As it was shown in [20], a general interpolation of one-dimensional distributions which leads to very naturally looking results can be achieved by interpolating the *quantile function* (defined as the inverse of the cumulative distribution function). While study [20] considers linear interpolation in one-dimensional parameter space, it is obvious that similar weighted average interpolation can be easily constructed in multivariate parameter settings and with higher order interpolation schemes. In general, the interpolated quantile function for an arbitrary parameter value $\mathbf{a}$ is expressed by

$$q(y|\mathbf{a}) = \sum_{j=1}^{m} w_j q(y|\mathbf{a}_j), \tag{24}$$

where the weighted quantile functions are summed at the m "nearby" parameter settings $\mathbf{a}_j$ for which the distribution was explicitly constructed. The weights w_j are normalized by $\sum_{j=1}^{m} w_j = 1$. Their precise values depend on the location of $\mathbf{a}$ w.r.t. nearby $\mathbf{a}_j$ and on the interpolation scheme used. Simple high-order interpolation schemes can be constructed in one-dimensional parameter space by using Lagrange interpolating polynomials to determine the weights[9]. In r-dimensional parameter space, rectangular grids can be utilized with multilinear or multicubic interpolation. For example, in the case of multilinear interpolation, the weights can be calculated as follows:

— Find the hyperrectangular parameter grid cell inside which the value of $\mathbf{a}$ falls.

— Shift and scale this cell so that it becomes a hypercube with diagonal vertices at $(0, 0, ..., 0)$ and $(1, 1, ..., 1)$.

— Let's designate the shifted and scaled value of $\mathbf{a}$ by $\mathbf{z}$, with components $(z_1, z_2, ..., z_r)$. If we were to interpolate towards $\mathbf{z}$ a scalar function f defined at the vertices with coordinates $(c_1, c_2, \ldots, c_r)$, where all c_k values are now either 0 or 1, then we would use

[9]To determine interpolated value of a function using Lagrange interpolating polynomials, weights are assigned to function values at a set of nearby points according to a rule published by Lagrange in 1795. The weights depend only on the abscissae of the interpolated points and on the coordinate at which the function value is evaluated but not on the interpolated function values.

the formula

$$f(z_1, z_2, ..., z_r) = \sum_{\substack{c_1 \in \{0,1\} \\ c_2 \in \{0,1\} \\ \cdots\cdots\cdots \\ c_r \in \{0,1\}}} f(c_1, c_2, ..., c_r) \prod_{k=1}^{r} z_k^{c_k} (1 - z_k)^{1-c_k} \tag{25}$$

which is obviously linear in every z_k and has correct function values when evaluated at the vertices. In complete analogy, we define the weights for the quantile functions constructed at the vertices with coordinates $(c_1, c_2, \ldots, c_r)$ to be $w(c_1, c_2, \ldots, c_r) = \prod_{k=1}^{r} z_k^{c_k} (1 - z_k)^{1-c_k}$. Naturally, there are $m = 2^r$ weights total.

The quantile interpolation of univariate distributions is implemented in the NPStat package with the InterpolatedDistribution1D class. A collection of quantile functions is assembled incrementally together with their weights (formula for calculating the weights has to be supplied by the user). For calculating the interpolated density at point x, the equation $x = q(y|\mathbf{a})$, with $q(y|\mathbf{a})$ from 24, is solved numerically for y. The density is then evaluated by numerically differentiating the interpolated quantile function: $p(x|\mathbf{a}) = \left(\frac{\partial q(y|\mathbf{a})}{\partial y}\right)^{-1}$.

Direct interpolation of univariate density functions, also known as "vertical interpolation", is implemented by the VerticallyInterpolatedDistribution1D class.

A class with a simpler interface implementing automatic linear weight assignments between nearby points in a 1-d parameter space is called InterpolatedDistro1D1P. This class can be used to interpolate either quantile functions or densities, depending on a switch. A similar class which interpolates univariate densities on a rectangular grid in a miltidimensional parameter space is called InterpolatedDistro1DNP.

5.2 Interpolation of Multivariate Densities

The procedure described in the previous section can be generalized to interpolate multivariate distributions. Let $p(\mathbf{x}|\mathbf{a})$ be a multivariate probability density in a d-dimensional space X of random variables $\mathbf{x} \in X$. $\mathbf{a}$ is a vector of parameters, and $\int_X p(\mathbf{x}|\mathbf{a}) d\mathbf{x} = 1$ for every $\mathbf{a}$[10]. For a "well-behaved" density (Riemann-integrable, *etc*), it is always possible to construct a one-to-one mapping from the space X into the unit d-dimensional cube, U_d, using a sequence of one-dimensional conditional cumulative distribution functions. These functions are defined as follows:

$$\begin{array}{c} F_1(x_1|x_2, x_3, \ldots, x_d, \mathbf{a}) \equiv \int_{-\infty}^{x_1} p(z_1, x_2, x_3, \ldots, x_d|\mathbf{a}) dz_1 / \int_{-\infty}^{\infty} p(z_1, x_2, x_3, \ldots, x_d|\mathbf{a}) dz_1 \\ F_2(x_2|x_3, \ldots, x_d, \mathbf{a}) \equiv \int_{-\infty}^{x_2} F_1(\infty|z_2, x_3, \ldots, x_d, \mathbf{a}) dz_2 / \int_{-\infty}^{\infty} F_1(\infty|z_2, x_3, \ldots, x_d, \mathbf{a}) dz_2 \\ F_3(x_3| \ldots, x_d, \mathbf{a}) \equiv \int_{-\infty}^{x_3} F_2(\infty|z_3, \ldots, x_d, \mathbf{a}) dz_3 / \int_{-\infty}^{\infty} F_2(\infty|z_3, \ldots, x_d, \mathbf{a}) dz_3 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ F_d(x_d|\mathbf{a}) \equiv \int_{-\infty}^{x_d} F_{d-1}(\infty|z_d, \mathbf{a}) dz_d / \int_{-\infty}^{\infty} F_{d-1}(\infty|z_d, \mathbf{a}) dz_d \end{array}$$

[10]Compared to interpolation of arbitrary functions, preserving this normalization is one of the major complications in interpolating multivariate densities.

Naturally, $F_k(\infty|x_{k+1},\ldots,x_d,\mathbf{a})$ is just renormalized $p(\mathbf{x}|\mathbf{a})$ in which the first k components of $\mathbf{x}$ are integrated out (*i.e.,* marginalized). The mapping from $\mathbf{x}\in X$ into $\mathbf{y}\in U_d$ is defined by $y_i = F_i(x_i|\ldots),\ i=1,\ldots,d$. In terms of conditional cumulative distribution functions,

$$p(\mathbf{x}|\mathbf{a}) = \prod_{i=1}^{d} \frac{\partial F_i(x_i|\ldots)}{\partial x_i}.$$

This method is not unique and other mappings from X to U_d are possible. What makes this particular construction useful is that the inverse mapping, from U_d into X, can be easily constructed as well: we simply solve the equations

$$y_i = F_i(x_i|\ldots) \tag{26}$$

in the reverse order, starting from dimension d and going back to dimension 1. Each equation in this sequence *has only one unknown* and therefore it can be efficiently solved numerically (and sometimes algebraically) by a variety of standard root finding techniques. The solutions of these equations, $x_i = q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}) \equiv F_i^{-1}(y_i|\ldots)$, are the *conditional quantile functions* (CQFs). Note that

$$p(\mathbf{x}|\mathbf{a}) = \left(\prod_{i=1}^{d} \frac{\partial q_i(y_i|\ldots)}{\partial y_i}\right)^{-1}. \tag{27}$$

Now, if the CQFs are known for some parameter values $\mathbf{a}_1$ and $\mathbf{a}_2$, interpolation towards $\mathbf{a} = (1-\lambda)\mathbf{a}_1 + \lambda\mathbf{a}_2$ is made in the same manner as described in Section 5.1: $x_i = (1-\lambda)q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}_1) + \lambda q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}_2)$. If the CQFs are known on a grid in the parameter space, we have to use an appropriate interpolation technique (multilinear, multicubic, *etc*) in that space in order to assign the weights to the CQFs at the nearby grid points. In general, the interpolated CQFs are defined by a weighted average

$$q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}) = \sum_{j=1}^{m} w_j q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}_j), \tag{28}$$

where the sum is performed over m nearby parameter points, weights w_j are normalized by $\sum_{j=1}^{m} w_j = 1$, and their exact values depend on the parameter grid chosen and the interpolation method used.

Basically, it is the whole mapping from $\mathbf{y}$ into $\mathbf{x}$ which gets interpolated in this method. The CQF interpolation results look very natural, but the process is rather CPU-intensive: for each $\mathbf{x}$ we need to solve d one-dimensional nonlinear equations

$$x_i = q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}), \qquad i=d,\ldots,1 \tag{29}$$

in order to determine $\mathbf{y}$ of the interpolated mapping. In this process, each call to evaluate $q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a})$ triggers m calls to evaluate $q_i(y_i|x_{i+1},\ldots,x_d,\mathbf{a}_j)$. Depending on implementation details, each of these m calls may in turn trigger root finding in an equation

like 26. Once $\mathbf{y}$ is found for the interpolated CQFs, density is determined by numerical evaluation of 27 in which $q_i(y_i|\ldots)$ are given by 28.

In practice, finite steps of the parameter grids will cause dependence of the interpolation results on the order in which conditional quantiles are evaluated. If choosing some particular order is considered undesirable and increased CPU loads are acceptable, all $d!$ possible permutations should be averaged.

In the NPStat package, multivariate densities whose mapping from the multidimensional unit cube into the density support region is implemented via CQFs return "true" when their virtual function mappedByQuantiles is called. User-developed implementations of multivariate densities should follow this convention as well. In particular, mapping of the densities represented by the BinnedDensityND class (lookup tables on hyperrectangular grids with multilinear interpolation) is performed by CQFs, as well as mapping of all fully factorizable densities.

When the fidelity of the model is less critical or when the correlation structure of the distribution is more stable w.r.t. parameter changes than its location and scales, much faster multivariate density interpolation can be performed by decomposing the density into the copula and the marginals. The support of the copula density $c(\mathbf{y}(\mathbf{x})|\mathbf{a})$ defined in Eq. 18 is always U_d and it does not depend on the parameter $\mathbf{a}$. This suggests that the marginals and the copula can be interpolated separately, using quantile function interpolation for the marginals and the standard weighted average interpolation for the copula.

The NPStat package uses the same data structure to perform both CQF-based and copula-based interpolation of multivariate densities. Multilinear interpolation is supported on a rectangular parameter grid (not necessarily equidistant). The distributions at the grid points are collected together using the GridInterpolatedDistribution class. Internally, the interpolation is performed by either UnitMapInterpolationND or CopulaInterpolationND class, depending on a switch. These classes perform CQF-based and copula-based interpolation, respectively.

6 Nonparametric Density Estimation

The problem of estimating population probability density function from a finite sample drawn from this population is ubiquitous in data analysis practice. It is often the case that there are no substantial reasons for choosing a particular parametric density model but certain assumptions, such as continuity of the density together with some number of derivatives or absence of narrow spatial features, can still be justified. There is a number of approaches by which assumptions of this kind can be introduced into the statistical model. These approaches are collectively known as "nonparametric density estimation" methods [21, 22, 23]. The NPStat package provides an efficient implementation of one of such methods, kernel density estimation (KDE), together with its extension to polynomial density models and densities with bounded support. This extension is called "local orthogonal polynomial expansion" (LOrPE).

6.1 Kernel Density Estimation (KDE)

Suppose, we have an i.i.d. sample of measurements x_i, $i = 0, 1, ..., N-1$ from a univariate probability density $p(x)$. The empirical probability density function (EPDF) for this sample is defined by

$$\mathrm{EPDF}(x) = \frac{1}{N}\sum_{i=0}^{N-1}\delta(x - x_i), \tag{30}$$

where $\delta(x)$ is the Dirac delta function. $\mathrm{EPDF}(x)$ can itself be considered an estimate of $p(x)$. However, this estimate can be substantially improved if some additional information about $p(x)$ is available. For example, we can often assume that $p(x)$ is continuous together with its first few derivatives or that it can have at most a few modes. In such cases a convolution of $\mathrm{EPDF}(x)$ with a kernel function, $K(x)$, often provides a much better estimate of the population density. $K(x)$ itself is usually chosen to be a symmetric continuous density with a location and scale parameter, so that the resulting estimate looks like

$$\hat{p}_{\mathrm{KDE}}(x|h) = \frac{1}{Nh}\sum_{i=0}^{N-1} K\left(\frac{x - x_i}{h}\right). \tag{31}$$

In the context of density estimation, parameter h is usually referred to as "bandwidth". Use of the Gaussian distribution or one of the distributions from the symmetric beta family as $K(x)$ is very common. In fact, it is so common that beta family kernels have their own names in the density estimation literature. These names are listed in Table 4.

Table 4: One-dimensional kernels from the symmetric beta family. All these kernels look like $\mathcal{N}(1 - x^2)^p\, I(-1 < x < 1)$, where $\mathcal{N}$ is the appropriate normalization factor. In the NPStat package, their formulae are implemented by the SymmetricBeta1D function.

Kernel name	Power p
Uniform (also called "boxcar")	0
Epanechnikov	1
Biweight (also called "quartic")	2
Triweight	3
Quadweight	4

In the limit $N \to \infty$ and with proper choice of h so that $h \to 0$ and $Nh \to \infty$, $\hat{p}_{\mathrm{KDE}}(x)$ becomes a consistent estimate of $p(x)$ in terms of integrated squared error (ISE):

$$\mathrm{ISE}(h) = \int_{-\infty}^{\infty}(\hat{p}_{\mathrm{KDE}}(x|h) - p(x))^2 dx, \qquad \lim_{N\to\infty}\mathrm{ISE}(h) = 0. \tag{32}$$

For the method analysis purposes, it is useful to understand what happens when we reconstruct known densities $p(x)$. Here, another measure of distance between the true density and its estimator becomes indispensable, called "mean integrated squared error" (MISE):

$$\mathrm{MISE}(h) = E(\mathrm{ISE}(h)) = E\left(\int_{-\infty}^{\infty} (\hat{p}_{\mathrm{KDE}}(x|h) - p(x))^2 dx\right), \tag{33}$$

where $E(...)$ stands for the expectation value over samples of N points drawn from $p(x)$. While other distance measures can be defined (and may be more relevant for your problem), $\mathrm{MISE}(h)$ is usually the easiest to analyze mathematically[11]. A typical goal of such an analysis consists in finding the value of h which minimizes $\mathrm{MISE}(h)$ for a sample of given size, and a significant amount of effort has been devoted to such bandwidth optimization studies (see, *e.g.,* Refs. [24, 25] for a review). A large fraction of these studies employs a simple MISE approximation valid for large values of N known as "AMISE" (asymptotic MISE). This approximation includes just the two leading terms known as "bias" which increases with increasing bandwidth and "variance" which decreases with increasing bandwidth, so that the bandwidth optimization procedure is reduced to finding the best bias-variance trade-off.

For subsequent discussion, it will be useful to introduce the concept of kernel order. Let's define the functional

$$\mu_j(f) = \int_{-\infty}^{\infty} x^j f(x) dx \tag{34}$$

which is the j-th moment of $f(x)$ about 0. Then it is said that the kernel $K(x)$ is of order m if

$$\mu_0(K) = 1,\ \mu_j(K) = 0 \text{ for } j = 1, ..., m-1, \text{ and } \mu_m(K) \neq 0. \tag{35}$$

It can be shown that, for one-dimensional KDE, the rate of AMISE convergence to 0 is proportional to $N^{-\frac{2m}{2m+1}}$ (see, for example, section 2.8 of [26]). Therefore, kernels with high values of m should be preferred for large samples. At the same time, only in case $m = 2$ the kernels can be everywhere non-negative (*i.e.,* bona fide densities). When $m > 2$ (so called "high-order" kernels), in order to have $\mu_2(K) = 0$, $K(x)$ must become negative somewhere. Therefore, negative values of $\hat{p}_{\mathrm{KDE}}(x|h)$ also become possible, and a mechanism for dealing with this problem must be specified. In NPStat, this problem is normally taken care of by setting negative values of $\hat{p}_{\mathrm{KDE}}(x|h)$ to 0 with subsequent renormalization of the density estimate so that its integral is 1.

It is instructive to consider $p(x)$, $\mathrm{EPDF}(x)$, and $\hat{p}_{\mathrm{KDE}}(x|h)$ in the frequency domain[12]. There, the Fourier transform of $K(x)$ acts as a low-pass filter which suppresses the sampling noise present in $\mathrm{EPDF}(x)$ so that the result, $\hat{p}_{\mathrm{KDE}}(x|h)$, has a better match to the spectrum of $p(x)$. By Parseval's identity, this leads to the reduction of the ISE. High-order kernels allow for a sharper frequency cutoff in the filter. Fortunately, the precise shape of $p(x)$ frequency spectrum is relatively unimportant, it is only important that its high frequency

[11] Note that both ISE and MISE are not dimensionless and, therefore, not invariant under scaling transformations. It is OK to compare different $\hat{p}_{\mathrm{KDE}}(x|h)$ with each other if the underlying $p(x)$ is the same, but ISE or MISE comparison for different $p(x)$ requires certain care if meaningful results are to be obtained.

[12] The Fourier transform of a probability density is called "characteristic function" of the distribution.

components decay "fast enough" so that their suppression together with the noise does not cause a significant distortion of $\hat{p}_{\mathrm{KDE}}(x|h)$ in comparison with $p(x)$. Powerful automatic bandwidth selection rules can be derived from this type of analysis if certain assumptions about the $p(x)$ spectrum decay rate at high frequencies are satisfied [27, 28].

With a few modifications, the ideas described above can be translated to multivariate settings. The kernel function becomes a multivariate density, and the bandwidth parameter becomes, in general, the bandwidth matrix (for example, the covariance matrix in the case of multivariate Gaussian kernel).

The NPStat package calculates $\hat{p}_{\mathrm{KDE}}(x|h)$ on an equidistant grid in one or more dimensions. Initially, the data sample is histogrammed using a finely binned histogram and then convoluted with a kernel using Discrete Fast Fourier Transform (DFFT). The number of histogram bins, N_b, should be selected taking into account the following considerations:

— The bins should be sufficiently small so that no "interesting" detail will be missed due to discretization of the density.

— The bins should be sufficiently small so that the expected optimal MISE is significantly larger than the ISE due to density discretization. Detailed exposition of this requirement can be found in Ref. [29].

— DFFT should be efficient for this number of bins. It is best to use $N_b = 2^k$ bins in each dimension, where k is a positive integer.

The computational complexity of this method is $\mathcal{O}(N) + \mathcal{O}(N_b \ln N_b)$ which is usually much better than the $\mathcal{O}(N \times N_b)$ complexity of a "naive" KDE implementation. However, for large sample dimensionalities N_b can become very large which limits the applicability of this technique. There are other computational methods (not yet in NPStat) which can work efficiently for high-dimensional samples [30, 31].

The following NPStat functions and classes can be used to perform KDE and to assist in bandwidth selection:

amiseOptimalBwGauss (header file "npstat/stat/amiseOptimalBandwidth.hh") — calculates AMISE-optimal bandwidth for fixed-bandwidth KDE with Gaussian kernel as well as high-order kernels derived from Gaussian. The following formula is implemented:

$$h_{\,\mathrm{AMISE}} = \left(\frac{R(K)(m!)^2}{2m\mu_m^2(K)R(p^{(m)})N} \right)^{\frac{1}{2m+1}}. \tag{36}$$

In this formula, $R(f)$ denotes the functional $R(f) = \int_{-\infty}^{\infty} f^2(x)dx$ defined for any square-integrable function f, $\mu_m(f)$ is the functional defined in Eq. 34, K is the kernel, m is the kernel order, $p^{(m)}$ is the m-th derivative of the reconstructed density[13], and N is the number of points in the sample. The expected AMISE corresponding to this bandwidth is calculated from

$$\mathrm{AMISE}(h_{\,\mathrm{AMISE}}) = \frac{(2m+1)R(K)}{2m\,h_{\,\mathrm{AMISE}}\,N}. \tag{37}$$

[13]Of course, in practice the reconstructed density is usually unknown. It is expected that a look-alike density will be substituted.

For the origin and further discussion of these formulae consult, for example, section 2.8 in Ref. [26]. It is assumed that high-order kernels are generated according to Eq. 46.

amiseOptimalBwSymbeta (header file "npstat/stat/amiseOptimalBandwidth.hh") — calculates AMISE-optimal bandwidth and corresponding expected AMISE according to Eqs. 36 and 37 for fixed-bandwidth KDE with kernels from symmetric beta family as well as with high-order kernels derived from symmetric beta distributions.

amisePluginBwGauss (header file "npstat/stat/amiseOptimalBandwidth.hh") — Gaussian $p(x)$ is substituted in Eqs. 36 and 37 and the corresponding quantities are found for the Gaussian kernel as well as high-order kernels derived from Gaussian.

amisePluginBwSymbeta (header file "npstat/stat/amiseOptimalBandwidth.hh") — Gaussian $p(x)$ is substituted in Eqs. 36 and 37 and the corresponding quantities are found for the kernels from the symmetric beta family as well as high-order kernels derived from symmetric beta distributions.

miseOptimalBw and gaussianMISE methods of the GaussianMixture1D class — these methods calculate MISE-optimal bandwidth and the corresponding exact MISE for Gaussian mixture densities estimated with the Gaussian kernel (or high-order kernels derived from the Gaussian) according to the formulae from Ref. [32].

ConstantBandwidthSmoother1D — This class implements fixed-bandwidth KDE for one-dimensional samples of points using Gaussian kernels or kernels from the symmetric beta family (including high-order kernels which are generated internally). Boundary effects can be alleviated by data mirroring. Kernel convolutions are performed by DFFT after sample discretization.

ConstantBandwidthSmootherND — This class implements fixed-bandwidth KDE for multivariate histograms. Arbitrary density implementations which inherit from AbsDistributionND can be used as weight functions for generating high-order kernels. Boundary effects can be alleviated by data mirroring.

JohnsonKDESmoother — This class constructs adaptive bandwidth KDE estimates for one-dimensional samples. It operates in several steps:

— The sample is discretized using centroid-preserving binning.

— Population mean, μ, standard deviation, σ, skewness, s, and kurtosis, k, are estimated by the arrayShape1D function.

— A distribution from the Johnson system with these values of μ, σ, s, and k is used as a template for the amiseOptimalBwSymbeta (or amiseOptimalBwGauss) function which calculates optimal constant bandwidth according to Eq. 36.

— A pilot density estimate is built by KDE (using LocalPolyFilter1D class) with the bandwidth determined in the previous step.

— The sample is smoothed by the variableBandwidthSmooth1D function (described below) using this pilot estimate.

This method achieves better MISE convergence rate without using high-order kernels (so that density truncation below zero is unnecessary). Give this method a serious consideration if you are working with one-dimensional samples, expect to reconstruct a unimodal distribution,

and do not have to worry about boundary effects.

KDECopulaSmoother — multivariate KDE in which extra care is applied in order to ensure that the estimation result is a bona fide copula (*i.e.,* that all of its marginals are uniform). This class should normally be used to smooth empirical copula densities constructed by empiricalCopulaHisto or empiricalCopulaDensity. The bandwidth can be supplied by the user or it can be chosen by cross-validation. In general, LOrPECopulaSmoother will produce better results in this context, so KDECopulaSmoother should be used only in case the slower speed of LOrPECopulaSmoother is deemed unacceptable.

KDEFilterND — A collection of KDEFilterND objects which utilize common workspace and DFFT engine can be employed to perform cross-validation calculations and bandwidth scans. Such a collection is used, for example, by the KDECopulaSmoother class described above. If you already know the bandwidth, the ConstantBandwidthSmootherND class will likely be more convenient to use than this one.

variableBandwidthSmooth1D — implements one-dimensional variable kernel adaptive KDE (which should not be confused with *local kernel*, or *balloon*, method — see section 2.10 in Ref. [26] for further discussion). In general, this approach consists in assigning different bandwidth values to each sample point. In the variableBandwidthSmooth1D function, this assignment is performed according to the formula

$$h_i = \frac{c}{\hat{\rho}^{\alpha}(x_i)}, \tag{38}$$

where $\hat{\rho}(x)$ is a pilot density estimate constructed, for example, by fixed-bandwidth KDE. Boundary kernel adjustments are performed automatically. The normalization constant c is determined so that the geometric mean of h_i equals to a user-provided bandwidth. There are certain reasons to believe that the choice of power parameter $\alpha = 1/2$ is a particularly good one [33].

simpleVariableBandwidthSmooth1D — a high-level driver function for variable kernel adaptive KDE which uses kernels from the symmetric beta family and automatically generates a pilot density estimate employing AMISE plugin bandwidth.

6.2 Local Orthogonal Polynomial Expansion (LOrPE)

Local Orthogonal Polynomial Expansion (LOrPE) can be viewed as a convenient method for creating kernels with desired properties (including high-order kernels) and for eliminating the KDE boundary bias[14]. LOrPE amounts to constructing a truncated expansion of the EPDF defined by Eq. 30 into orthogonal polynomial series near each point $x_{\,\mathrm{fit}}$ where we want to build the initial density estimate:

$$\hat{f}_{\mathrm{LOrPE}}(x|h) = \sum_{k=0}^{M} c_k(x_{\,\mathrm{fit}}, h) P_k\left(\frac{x - x_{\,\mathrm{fit}}}{h}\right). \tag{39}$$

[14]If you are familiar with orthogonal series density estimators (OSDE), you can also view LOrPE as a localized version of OSDE.

The polynomials $P_k(x)$ are built to satisfy the normalization condition

$$\frac{1}{h}\int_a^b P_j\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right) P_k\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right) K\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right) dx = \delta_{jk}, \tag{40}$$

which is equivalent to

$$\int_{(a-x_{\,\mathrm{fit}})/h}^{(b-x_{\,\mathrm{fit}})/h} P_j(y)P_k(y)K(y)dy = \delta_{jk}, \tag{41}$$

where δ_{jk} is the Kronecker delta, $K(x)$ is a suitably chosen kernel function, and $[a,b]$ is the support interval of the estimated density. For commonly used kernels from the beta family (Epanechnikov, biweight, triweight, *etc.*), condition (41) generates the normalized Gegenbauer polynomials (up to a common multiplicative constant) at points $x_{\,\mathrm{fit}}$ sufficiently deep inside the support interval, provided h is small enough to guarantee that $(a-x_{\,\mathrm{fit}})/h \leq -1$ and $(b-x_{\,\mathrm{fit}})/h \geq 1$. If $x_{\,\mathrm{fit}}$ is sufficiently close to the boundaries of the density support $[a,b]$ then the polynomial system will vary depending on $x_{\,\mathrm{fit}}$, and the notation $P_k(\cdot, x_{\,\mathrm{fit}})$ would be more appropriate. However, in the subsequent text this dependence on $x_{\,\mathrm{fit}}$ will be suppressed in order to simplify the notation.

The expansion coefficients $c_k(x_{\,\mathrm{fit}}, h)$ are determined by the usual scalar product of the expanded function with P_k:

$$c_k(x_{\,\mathrm{fit}}, h) = \frac{1}{h}\int \mathrm{EPDF}(x) P_k\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right) K\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right) dx, \tag{42}$$

which, after substituting $\mathrm{EPDF}(x)$ from 30, leads to

$$c_k(x_{\,\mathrm{fit}}, h) = \frac{1}{Nh}\sum_{i=0}^{N-1} P_k\left(\frac{x_i-x_{\,\mathrm{fit}}}{h}\right) K\left(\frac{x_i-x_{\,\mathrm{fit}}}{h}\right). \tag{43}$$

Note that the coefficients $c_k(x_{\,\mathrm{fit}}, h)$ calculated according to Eq. 42 can be naturally interpreted as localized expectation values of orthogonal polynomials $P_k\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right)$ w.r.t. probability density $\mathrm{EPDF}(x)$ in which localization weights are given by $\frac{1}{h}K\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right)$.

The density estimate at $x_{\,\mathrm{fit}}$ is defined by

$$\hat{p}_{\mathrm{LOrPE}}(x_{\,\mathrm{fit}}|h) = \max\{0, \hat{f}_{\mathrm{LOrPE}}(x_{\,\mathrm{fit}}|h)\}. \tag{44}$$

In general, LOrPE does not produce a bona fide density (in this respect it is similar to the orthogonal series estimator), and after calculating the density for all $x_{\,\mathrm{fit}}$ values one has to perform the overall renormalization.

Equation 39 can be usefully generalized as follows:

$$\hat{f}_{\mathrm{LOrPE}}(x|h) = \sum_{k=0}^{\infty} g(k) c_k(x_{\,\mathrm{fit}}, h) P_k\left(\frac{x-x_{\,\mathrm{fit}}}{h}\right). \tag{45}$$

Here, $g(k)$ is a "taper function". Normally, $g(0) = 1$ and there is an integer M such that $g(k) = 0$ for any $k > M$. The taper function suppresses high order terms gradually instead of using a sharp cutoff at M.

When evaluated at points x_{fit} which are sufficiently far away from the density support boundaries and if $K(x)$ is an even function, Eq. 45 is equivalent to a kernel density estimator with the effective kernel

$$K_{\text{eff}}(x) = K(x) \sum_{j=0}^{\infty} g(2j) P_{2j}(0) P_{2j}(x). \tag{46}$$

Moreover, if $g(k)$ is a step function, *i.e.,* $g(k) = 1$ for all $k \leq M$ and $g(k) = 0$ for all $k > M$, it can be shown that the effective kernel is of order $M + 1$ if M is odd and $M + 2$ if M is even [34][15].

A slightly different modification of LOrPE is based on the following identity:

$$\frac{1}{h} \sum_{j=0}^{\infty} P_j \left(\frac{x - x_i}{h} \right) P_j(0) K \left(\frac{x - x_i}{h} \right) = \delta(x - x_i). \tag{47}$$

Substituting this into Eq. 30, we obtain

$$\text{EPDF}(x) = \frac{1}{Nh} \sum_{i=0}^{N-1} \sum_{j=0}^{\infty} P_j \left(\frac{x - x_i}{h} \right) P_j(0) K \left(\frac{x - x_i}{h} \right). \tag{48}$$

The modified density estimate is obtained from this expansion by introducing a taper:

$$\hat{\hat{f}}_{\text{LOrPE}}(x_{\text{fit}}|h) = \frac{1}{Nh} \sum_{i=0}^{N-1} \sum_{j=0}^{\infty} g(j) P_j \left(\frac{x_{\text{fit}} - x_i}{h} \right) P_j(0) K \left(\frac{x_{\text{fit}} - x_i}{h} \right) \tag{49}$$

and then truncation is handled just like in Eq. 44. For even kernels and points x_{fit} sufficiently far away from the density support boundaries, Eq. 49 is equivalent to Eq. 45 evaluated at $x = x_{\text{fit}}$. However, this is no longer true near the boundaries. Perhaps, the easiest way to think about it is that, in Eq. 49, an effective kernel is placed at the location of each data point (and, in general, shapes of these effective kernels depend on the data point location x_i). All these kernels are then summed to obtain the density estimates at all x_{fit}. On the other hand, in Eq. 45 the effective kernel is placed at the location of each point at which the density estimate is made (so that the effective kernel shape depends on x_{fit}). This kernel generates the weights for each data point which are summed to obtain the estimate. The difference between these two approaches is usually ignored for KDE (simple KDE is unable to deal with boundary bias anyway), but for LOrPE substantially different results can be obtained near the boundaries.

[15]For such kernels the sum in Eq. 46 can be reduced to a simple algebraic form via the Christoffel-Darboux identity. However, the resulting formula is not easy to evaluate in a numerically stable manner near $x = 0$.

It is not obvious apriori which density estimate is better: $\hat{\hat{f}}_{\mathrm{LOrPE}}$ from Eq. 49 or $\hat{f}_{\mathrm{LOrPE}}$ from Eq. 45, although some preliminary experimentation with simple distributions does indicate that $\hat{f}_{\mathrm{LOrPE}}$ typically results in smaller MISE. The integral of the $\hat{\hat{f}}_{\mathrm{LOrPE}}$ estimate on the complete density support interval is automatically 1 which is not true for $\hat{f}_{\mathrm{LOrPE}}$. On the other hand, $\hat{f}_{\mathrm{LOrPE}}$ admits an appealing interpretation in terms of the local density expansion 45 in which the localized expectation values of the orthogonal polynomials P_k are matched to their observed values in the data sample (this also leads to automatic matching of localized distribution moments about x_{fit}).

One-dimensional KDE with fixed kernel $K(x)$ has only one important parameter which regulates the amount of smoothing: bandwidth h. LOrPE has two such parameters: bandwidth h and the highest polynomial order M (or, in general, the shape of the taper function). It is intuitively obvious that polynomial modeling should result in a smaller bias than KDE for densities with several (at least M) continuous derivatives, and that a proper balance of h and M should result in a better estimator overall.

LOrPE calculations remain essentially unchanged in multivariate settings: the only difference is switching to multivariate orthogonal polynomial systems.

Even though LOrPE is equivalent to KDE far away from the density support boundaries, LOrPE does not suffer from the boundary bias because Eq. 41 automatically adjusts the shape of orthogonal polynomials near the boundary. This makes LOrPE applicable in a wider set of problems than KDE. In addition to just making better estimates of densities with a sharp cutoff at the boundary, LOrPE fixes the main problem with some existing KDE-based methods which are rarely used in practice due to their severe degradation from the boundary bias. Examples of such methods include transformation kernel density estimation (see, for example, section 2.10.3 of [26]) in which the transformation target is the uniform distribution, as well as separate estimation of the marginals and the copula for multivariate densities.

Unfortunately, LOrPE improvements over KDE do not come without a price in terms of the algorithmic complexity of the method. The density estimate can no longer be represented as a simple convolution of the sample EPDF and a kernel. Because of this, DFFT-based calculations are no longer sufficient. In the LOrPE implementation within NPStat, simple algorithms are used instead which perform pointwise convolutions. Their computational complexity scales as $\mathcal{O}(N) + \mathcal{O}(N_b N_s)$, where N_b is the number of bins in the sample discretization histogram and N_s is the number of bins of the same length (area, volume, *etc*) inside the kernel support. For large bandwidth values (or for kernels with infinite support) this essentially becomes $\mathcal{O}(N) + \mathcal{O}(N_b^2)$ which can be significantly slower than the KDE implementation based on DFFT.

The following NPStat classes and functions can be used to perform LOrPE and to assist in choosing the bandwidth:

LocalPolyFilter1D — LOrPE for one-dimensional samples. A numerical Gram-Schmidt procedure is used to build polynomials defined by Eq. 40 on an equidistant grid. A linear filter which combines formulae 42 and 45 is then constructed for each grid point x_{fit} from the density support region (the same "central" filter is used for all x_{fit} points far away

from the support boundaries). The filter method of the class can then be used to build density estimates defined by Eq. 45 with $x = x_{\rm fit}$ from sample histograms. Alternatively, the convolve method can be used to to make estimates according to Eq. 49. If necessary, subsequent truncation of the reconstructed densities below 0 together with renormalization should be performed by the user.

WeightTableFilter1DBuilder (header file "npstat/stat/WeightTableFilter1DBuilder.hh") — Helper class designed to work with LocalPolyFilter1D. This helper constructs linear filters out of arbitrary scanned weights utilizing orthogonal polynomials. If it is necessary to introduce exclusion regions into the data (for example, for the purpose of interpolating background density from sidebands), constructor of this class is the place where this can be done.

NonmodifyingFilter1DBuilder (header file "npstat/stat/WeightTableFilter1DBuilder.hh") — Helper class designed to work with LocalPolyFilter1D. This filter does not change the data (*i.e.*, this filter is represented by the unit matrix).

getBoundaryFilter1DBuilder (header file "npstat/stat/AbsFilter1DBuilder.hh") — This function can be used to construct various filters that inherit from AbsBoundaryFilter1DBuilder class. These filters differ by the kernel adjustments they perform near the density support boundaries. Many filters of this kind are declared in the "npstat/stat/Filter1DBuilders.hh" header file.

PolyFilterCollection1D — A collection of LocalPolyFilter1D objects which can be used, for example, in bandwidth scans or in cross-validation calculations.

LocalPolyFilterND — similar to LocalPolyFilter1D but intended for estimating multivariate densities.

SequentialPolyFilterND — similar to LocalPolyFilterND but each dimension is processed sequentially using 1-d filtering. Employs a collection of LocalPolyFilter1D objects, one for each dataset dimension.

LOrPECopulaSmoother — multivariate LOrPE in which extra care is applied in order to ensure that the estimation result is a bona fide copula (*i.e.,* that all of its marginals are uniform). This class should normally be used to smooth empirical copula densities constructed by empiricalCopulaHisto or empiricalCopulaDensity. The bandwidth can be supplied by the user or it can be chosen by cross-validation. Less reliable but faster calculations of this type can be performed with the KDECopulaSmoother class described in Section 6.1.

SequentialCopulaSmoother — similar to LOrPECopulaSmoother but each dimension is processed sequentially using 1-d filtering.

NonparametricCompositeBuilder — a high-level API for estimating multivariate densities by applying KDE or LOrPE separately to each marginal and to the copula. This class builds CompositeDistributionND objects from collections of sample points.

symbetaLOrPEFilter1D (header file "npstat/stat/LocalPolyFilter1D.hh") — a convenience utility for building one-dimensional LOrPE filters using kernels from the symmetric beta family (including the Gaussian).

lorpeMise1D (header file "npstat/stat/lorpeMise1D.hh") — calculates LOrPE MISE for an arbitrary known density according to Eq. 33. The support of the density is split into M subintervals. It is assumed that the sample points are distributed in these subintervals

according to the multinomial distribution. The covariance matrix of this distribution is then propagated to the density estimation result. The trace of the propagated covariance, multiplied by the width of the subinterval, is added to the integrated squared bias in order to obtain the MISE. Of course, this method reproduces Eq. 33 exactly only in the limit $M \to \infty$, while in practice the $O(M^3)$ computational complexity of the error propagation formulae (based on conventional matrix multiplication) will necessitate a reasonable choice of finite M. I suggest choosing M in such a manner that the ISE introduced by the discretization of the density is significantly smaller than the estimated MISE.

Further remarks on theoretical development of LOrPE, as well as a comparison of its performance with other density estimation techniques, can be found in [35].

6.3 Density Estimation with Bernstein Polynomials

Density representation by Bernstein polynomial series is an alternative approach which can be used to alleviate the boundary bias problem of KDE. Bernstein polynomials are defined as follows:

$$b_{m,n}(x) = C_n^m x^m (1-x)^{n-m}, \tag{50}$$

where $m = 0, 1, ..., n$ and C_n^m are the binomial coefficients:

$$C_n^m = \frac{n!}{m!(n-m)!}. \tag{51}$$

In the density estimation context, Bernstein polynomials are often generalized to non-integer values of n and m in which case C_n^m is replaced by $\frac{\Gamma(n+1)}{\Gamma(m+1)\Gamma(n-m+1)}$. Up to normalization, this representation is equivalent to the beta distribution with parameters $\alpha = m + 1$ and $\beta = n - m + 1$. For notational simplicity, I will use the term "Bernstein polynomials" even if n and m are not integer.

There are two substantially different variations of this density estimation method. In the first scheme [36], Bernstein polynomials are used as variable-shape kernels in a KDE-like formula:

$$\hat{f}_{\mathrm{B}}(x|n) = \frac{n+1}{N} \sum_{i=0}^{N-1} b_{m(x),n}(x_i), \tag{52}$$

where it is assumed that the reconstructed density is supported on the [0, 1] interval (naturally, this interval can be shifted and scaled as needed). The requirement of asymptotic estimator consistency does not fix $m(x)$ uniquely, and this mapping can be chosen in a variety of ways. In NPStat, the following relationship is implemented:

$$m(x) = \begin{cases} c & \text{if } x(n-2s) + s \leq c \\ x(n-2s) + s & \text{if } c < x(n-2s) + s < n - c \\ n - c & \text{if } x(n-2s) + s \geq n - c \end{cases} \tag{53}$$

This formula reproduces the simple mapping $m(x) = xn$ considered in Ref. [36] in case $s = 0$ and $c = 0$. The offset parameter s plays the role of effective Bernstein polynomial degree

used at $x = 0$ and regulates the amount of boundary bias. Meaningful values of s lie between -1 and 0. As $m \to -1$, the mean of the generalized Bernstein polynomial kernels tends to x so that the estimator becomes uniformly unbiased for linear density functions. At the same time, the width of the kernel tends to 0 at the boundary which leads to an increase in the estimator variance. As it makes little sense to use kernels whose width is smaller than the discretization bin size, the cutoff parameter c was introduced. This cutoff effectively limits kernel width from below in a manner which preserves asymptotic consistency of the estimator (the useful range of c values is also [-1, 0]). In addition to appropriate selection of the main bandwidth parameter n, proper choice of parameters s and c can significantly improve estimator convergence at the boundary.

In the second variation of this density estimation technique [37], the polynomials are chosen based on the location of the observed points:

$$\hat{\hat{f}}_{\mathrm{B}}(x|n) = \frac{n+1}{N}\sum_{i=0}^{N-1} b_{m(x_i),n}(x). \tag{54}$$

For any m and n, $\int_0^1 b_{m,n}(x)dx = \frac{1}{n+1}$, so this particular estimate is a bona fide density. Due to the reasons that will be mentioned later in this subsection, it can be advantageous to keep integer m and n in this approach. As in the case of x-dependent kernel shape, there is some amount of freedom in the $m(x_i)$ assignment. For asymptotic consistency we must require that $m(x_i)/n \to x_i$ as $n \to \infty$. However, such assignments are not unique. One can choose, for example, $m(x_i) = \lfloor x_i/(n+1)\rfloor$ as in Ref. [37] (the symbol $\lfloor\cdot\rfloor$ stands for the "floor" function), but it can also be useful to assign more than one polynomial to x_i. The following scheme is implemented in NPStat in addition to the $m(x_i)$ assignment just mentioned. First, an integer k is found such that $k + 0.5 \leq x_i(n+1) < k + 1.5$. Then, if $0 \leq k < n$,

$$m(x_i) = \begin{cases} k & \text{with weight } k + 1.5 - x_i(n+1) \\ k+1 & \text{with weight } x_i(n+1) - k - 0.5 \end{cases} \tag{55}$$

If $k < 0$ then $m(x_i) = 0$ with weight 1, and if $k >= n$ then $m(x_i) = n$ with weight 1. The polynomial weighting schemes actually implemented in the code are slightly more complicated as they take into account data binning.

With integer values of m and n, density estimates constructed according to Eq. 54 (or its weighted version just described) have an important property of being positive doubly stochastic. What it means is that not only $\int_0^1 \hat{\hat{f}}_{\mathrm{B}}(x|n)dx = 1$ but also a sum of an arbitrary number of separate $\hat{\hat{f}}_{\mathrm{B}}(x|n)$ estimators will be flat in x as long as the sum of x_i values used to build all these estimators is itself flat. If the x_i values are flat between 0 and 1 then assigned m values will be flat between 0 and n (inclusive). Then double stochasticity follows directly from the partition of unity property of Bernstein polynomials: $\sum_{m=0}^{n} b_{m,n}(x) = \sum_{m=0}^{n} C_n^m x^m (1-x)^{n-m} = [x + (1-x)]^n = 1$.

A collection of positive double stochastic estimators can be used for copula filtering by sequentially applying these estimators in each dimension (with the help of SequentialPolyFilterND class). If the initial data set processed by this sequence represents a copula density (for

example, in case it is an empirical copula density histogram), the result is also guaranteed to be a copula density.

In NPStat, any density estimator implemented via the LocalPolyFilter1D class can be turned into closest (in some sense) doubly stochastic estimator by calling the doublyStochasticFilter method of that class. Non-negative estimators will be converted into non-negative doubly stochastic estimators using an iterative procedure similar to the one described in [38] while filters with negative entries will be converted into generalized doubly stochastic filters according to the method described in [39].

The following facilities are provided by NPStat for estimating densities with Bernstein polynomials and beta distribution kernels:

BetaFilter1DBuilder — Constructs linear filters for LocalPolyFilter1D class according to Eq. 52.

BernsteinFilter1DBuilder — Constructs linear filters for LocalPolyFilter1D according to Eq. 54. These filters are intended for use with the convolve method of LocalPolyFilter1D class rather than the filter method.

betaKernelsBandwidth — This function estimates optimal bandwidth (order n of the generalized Bernstein polynomial) for Eq. 52 according to the AMISE calculation presented in [36]. This bandwidth estimate should not be taken very seriously for realistic sample sizes, as finite sample performance of Bernstein polynomial methods is not well understood.

It is worth mentioning that there are other ways to create doubly stochastic filters. For example, the filter which represents the discretized Green's function for the homogeneous 1-d heat equation with the Neumann boundary conditions is doubly stochastic. This particular filter can be constructed with the help of the symbetaLOrPEFilter1D function, using Gaussian kernel and specifying 0 for the degree of the LOrPE polynomial as well as BM_FOLD for the boundary handling option. Subsequently, the convolve method of this filter should be utilized to smooth the data.

6.4 Using Cross-Validation for Choosing the Bandwidth

Cross-validation is a technique for adaptive bandwidth selection applicable to both KDE and LOrPE. Two types of cross-validation are supported by NPStat: least squares and pseudo-likelihood.

The least squares cross-validation is based on the following idea. The MISE from Eq. 33 can be written as

$$\begin{aligned}\mathrm{MISE}(h) &= E\left(\int_{-\infty}^{\infty}(\hat{p}(x|h)-p(x))^2dx\right)\\ &= E\left(\int_{-\infty}^{\infty}\hat{p}^2(x|h)dx\right) - 2E\left(\int_{-\infty}^{\infty}\hat{p}(x|h)p(x)dx\right) + \int_{-\infty}^{\infty}p^2(x)dx.\end{aligned}$$

The last term, $\int_{-\infty}^{\infty}p^2(x)dx$, does not depend on h, so minimization of MISE is equivalent to minimization of $B(h) \equiv E\left(\int_{-\infty}^{\infty}\hat{p}^2(x|h)dx - 2\int_{-\infty}^{\infty}\hat{p}(x|h)p(x)dx\right)$. Of course, $p(x)$ itself is

unknown. However, it can be shown (as in section 3.4.3 of [21]) that an unbiased estimator of $B(h)$ can be constructed as

$$\mathrm{LSCV}(h) = \int_{-\infty}^{\infty} \hat{p}^2(x|h)dx - \frac{2}{N}\sum_{i=0}^{N-1} \hat{p}_{-1,i}(x_i, h), \tag{56}$$

where $\hat{p}_{-1,i}(x, h)$ is a "leaving-one-out" density estimator to which the point at x_i does not contribute. For example, in the case of KDE this estimator is defined by

$$\hat{p}_{-1,i}(x|h) = \frac{1}{(N-1)\,h}\sum_{\substack{j=0 \\ j\neq i}}^{N-1} K\left(\frac{x - x_j}{h}\right). \tag{57}$$

Minimization of $\mathrm{LSCV}(h)$ can lead to a reasonable bandwidth estimate, h_{LSCV}. However, as h_{LSCV} is itself a random quantity, its convergence towards the bandwidth that optimizes MISE, h_{MISE}, is known to be rather slow. Moreover, $\mathrm{LSCV}(h)$ can have multiple minima, so its minimization is best carried out by simply scanning h within a certain range in the proximity of some value h^* suggested by plug-in methods. For more information on the issues related to the least squares cross-validation see Refs. [21, 26].

The pseudo-likelihood cross-validation (also sometimes called likelihood cross-validation) is based on maximizing the "leaving-one-out" likelihood:

$$\mathrm{PLCV}(h) = \prod_{i=0}^{N-1} \hat{p}_{-1,i}(x_i, h) \tag{58}$$

The criterion of maximum $\mathrm{PLCV}(h)$ can be obtained by minimizing an approximate Kullback-Leibler distance between the density and its estimate [21]. Maximizing $\mathrm{PLCV}(h)$ is only appropriate in certain situations. In particular, whenever $\hat{p}_{-1,i}(x_i, h)$ becomes 0 even for a single point, this criterion fails to produce meaningful results. Its use is also problematic for densities with infinite support due to the strong influence fluctuations in the distribution tails exert on $\mathrm{PLCV}(h)$. Because of these problems, NPStat implements a "regularized" version of $\mathrm{PLCV}(h)$ defined by

$$\mathrm{RPLCV}(h) = \prod_{i=0}^{N-1} \max\left(\hat{p}_{-1,i}(x_i, h),\, \frac{\hat{p}_{\mathrm{self},i}(x_i, h)}{N^{\alpha}}\right), \tag{59}$$

where α is the regularization parameter chosen by the user ($\alpha = 1/2$ usually works reasonably well) and $\hat{p}_{\mathrm{self},i}(x, h)$ is the contribution of the data point at x_i into the original density estimator. For KDE, this contribution is

$$\hat{p}_{\mathrm{self},i}(x|h) = \frac{1}{Nh} K\left(\frac{x - x_i}{h}\right). \tag{60}$$

If the bandwidth is fixed, $\hat{p}_{\mathrm{self},i}(x_i|h) = K(0)/(Nh)$ for every point x_i. $\hat{p}_{\mathrm{self},i}(x_i, h)$ changes from one point to another for LOrPE and for variable-bandwidth KDE.

Cross-validation in the NPStat package is implemented for discretized KDE and LOrPE density estimators. It is assumed that the optimal bandwidth corresponds to the maximum of some quantity, as in the case of RPLCV(h). All classes which perform cross-validation for univariate densities inherit from the abstract base class AbsBandwidthCV1D. For multivariate densities, the corresponding base class is AbsBandwidthCVND (both of these base classes are declared in the header file "npstat/stat/AbsBandwidthCV.hh"). The following concrete classes can be used:

BandwidthCVLeastSquares1D — implements Eq. 56 for KDE and LOrPE in 1-d.

BandwidthCVLeastSquaresND — implements Eq. 56 for multivariate KDE and LOrPE.

BandwidthCVPseudoLogli1D — implements Eq. 59 for KDE and LOrPE in 1-d.

BandwidthCVPseudoLogliND — implements Eq. 59 for multivariate KDE and LOrPE.

The cross-validation classes are used internally by such high-level classes as KDECopulaSmoother, LOrPECopulaSmoother, and SequentialCopulaSmoother.

6.5 The Nearest Neighbor Method

Using NPStat tools, a simple density estimation algorithm similar to the k-nearest neighbor method [21] can be implemented for one-dimensional samples as follows:

— Discretize the data using a finely binned histogram.

— Convert this histogram into a distribution by constructing a BinnedDensity1D object.

— For any point x at which a density estimate is desired, calculate the corresponding cumulative distribution value $y = F(x)$.

— For some interval Δ, $0 < \Delta < 1$, estimate the density at x by

$$\hat{p}_{\mathrm{NN}}(x) = \frac{\Delta}{q(y + \Delta/2) - q(y - \Delta/2)}, \tag{61}$$

where $q(y)$ is the quantile function: $q(F(x)) = x$. This formula assumes $y - \Delta/2 \geq 0$ and $y + \Delta/2 \leq 1$. If $y + \Delta/2 > 1$ then the $\hat{p}_{\mathrm{NN}}(x)$ denominator should be replaced by $q(1) - q(1 - \Delta)$, and if $y - \Delta/2 < 0$ then the denominator should become $q(\Delta) - q(0)$.

In this approach, the parameter Δ plays the same role as the k/N ratio in the standard k-nearest neighbor method. For best results, Δ should scale with the number of sample points as $N^{-1/5}$, and the optimal constant of proportionality depends on the estimated density itself [21]. The k-nearest neighbor method (and its modification just described) is not recommended for estimating densities with infinite support as it leads to a diverging density integral.

For multivariate samples, a similar estimate can be constructed with the help of HistoNDCdf class. Its method coveringBox can be used to find the smallest d-dimensional box with the given center and fixed proportions which encloses the desired sample fraction.

7 Nonparametric Regression

"Regression" refers to a form of data analysis in which the behavior of the dependent variable, called "response", is deduced as a function of the independent variable, called "predictor", from a sample of observations. The response values are considered random (*e.g.*, contaminated by noise) while the predictor values are usually assumed to be deterministic. The analysis purpose is thus to determine the location parameter of the response distribution (mean, median, mode, *etc*) as a function of the predictor. In the NPStat algorithms, the response is always assumed to be a univariate quantity while the predictor can be either univariate or multivariate. In the discussion below, predictor will be denoted by $\mathbf{x}$, response by y, $\mu(\mathbf{x})$ will be used to describe the response location function, and $\hat{\mu}(\mathbf{x})$ will denote an estimator of $\mu(\mathbf{x})$.

"Nonparametric regression" refers to a form of regression analysis in which no global parametric model is postulated for $\mu(\mathbf{x})$. Instead, for every $\mathbf{x}_{\rm fit}$, $\mu(\mathbf{x})$ is described in the vicinity of $\mathbf{x}_{\rm fit}$ by a relatively simple model which is fitted using sample points located nearby. Further discussion of $\mu(\mathbf{x})$ estimation depends critically on the assumptions which can be made about the distribution of response values.

7.1 Local Least Squares

With the additional assumption of Gaussian error distribution (*i.e.*, $y_i = \mu(\mathbf{x}_i) + \epsilon_i$, where ϵ_i are normally distributed with mean 0 and standard deviation σ_i), the model fitting can be efficiently performed by the method of local least squares. In this method, $\hat{\mu}(\mathbf{x})$ is found by minimizing the quantity:

$$\chi^2(\mathbf{x}_{\rm fit}, h) = \sum_{i=0}^{N-1} \left(\frac{y_i - \hat{\mu}(\mathbf{x}_i|\mathbf{x}_{\rm fit}, h)}{\sigma_i} \right)^2 K_h(\mathbf{x}_i - \mathbf{x}_{\rm fit}). \tag{62}$$

Here, h refers to one or more parameters which determine the extent of the kernel $K_h(\mathbf{x})$. In NPStat, $\hat{\mu}(\mathbf{x}|\mathbf{x}_{\rm fit}, h)$ is usually decomposed into orthogonal polynomials. In the case of univariate predictor,

$$\hat{\mu}(x|x_{\rm fit}, h) = \sum_{k=0}^{M} \hat{a}_k(x_{\rm fit}, h) P_k\left(\frac{x - x_{\rm fit}}{h} \right), \tag{63}$$

where polynomials $P_k(x)$ are subject to normalization condition 40. The expansion coefficients $\hat{a}_k(x_{\rm fit}, h)$ are determined from the equations $\partial\chi^2(x_{\rm fit}, h)/\partial\hat{a}_k = 0$ which leads to $M+1$ simultaneous equations for $k = 0, 1, ..., M$:

$$\sum_{i=0}^{N-1} \frac{1}{\sigma_i^2} \left(y_i - \sum_{j=0}^{M} \hat{a}_j(x_{\rm fit}, h) P_j\left(\frac{x_i - x_{\rm fit}}{h} \right) \right) K\left(\frac{x_i - x_{\rm fit}}{h} \right) P_k\left(\frac{x_i - x_{\rm fit}}{h} \right) = 0. \tag{64}$$

If the predictor values x_i are specified on a regular grid of points then the discretized version of Eq 40 is just

$$\frac{h_g}{h}\sum_{i=0}^{N-1} P_j\left(\frac{x_i - x_{\text{fit}}}{h}\right) K\left(\frac{x_i - x_{\text{fit}}}{h}\right) P_k\left(\frac{x_i - x_{\text{fit}}}{h}\right) = \delta_{jk}, \tag{65}$$

where h_g is the distance between any two adjacent values of x_i. This leads to a particularly simple solution for $\hat{a}_k(x_{\text{fit}}, h)$ if the model can be assumed at least locally homoscedastic (*i.e.,* if all σ_i are the same in the vicinity of x_{fit}):

$$\hat{a}_k(x_{\text{fit}}, h) = \frac{h_g}{h}\sum_{i=0}^{N-1} y_i K\left(\frac{x_i - x_{\text{fit}}}{h}\right) P_k\left(\frac{x_i - x_{\text{fit}}}{h}\right). \tag{66}$$

Substituting this into Eq. 63, one gets

$$\hat{\mu}(x_{\text{fit}}|h) = \frac{h_g}{h}\sum_{i=0}^{N-1}\sum_{k=0}^{M} y_i K\left(\frac{x_i - x_{\text{fit}}}{h}\right) P_k\left(\frac{x_i - x_{\text{fit}}}{h}\right) P_k(0). \tag{67}$$

If $K(x)$ is an even function and x is far away from the boundaries of the interval on which the regression is performed, Eq. 67 is equivalent to the well-known Nadaraya-Watson estimator,

$$\hat{\mu}_{NW}(x_{\text{fit}}|h) = \frac{\sum_{i=0}^{N-1} y_i K_{\text{eff}}\left(\frac{x_i - x_{\text{fit}}}{h}\right)}{\sum_{i=0}^{N-1} K_{\text{eff}}\left(\frac{x_i - x_{\text{fit}}}{h}\right)}, \tag{68}$$

with the effective kernel given by

$$K_{\text{eff}}(x) = \sum_{k=0}^{M} P_k(0) P_k(x) K(x). \tag{69}$$

Just as in the case of LOrPE, a taper function can be introduced for this kernel which leads to Eq 46.

If the predictor values x_i are arbitrary or if the model can not be considered homoscedastic even locally, there is no simple formula which solves the linear system 64. In this case the equations must be solved numerically. To perform this calculation, NPStat calls appropriate routines from LAPACK [40].

Generalization of local least squares methods to multivariate predictors is straightforward: one simply switches to multivariate kernels and polynomial systems.

The following NPStat classes can be used to perform local least squares regression of locally homoscedastic polynomial models on regular grids:

LocalPolyFilter1D, LocalPolyFilterND, and SequentialPolyFilterND — these classes have already been mentioned in Section 6.2. It turns out that LOrPE of discretized data essentially amounts to local least squares regression on histogram bin contents. To see that, compare

Eqs. 66 and 42. Up to an overall normalization constant, Eq. 66 is just a discretized version of Eq. 42. In fact, all three "PolyFilter" classes actually perform local least squares regression which, in the signal analysis terminology, is a linear filtering procedure. If the result is to be treated as a density, it has to be truncated below zero and renormalized by the user.

QuadraticOrthoPolyND — this class supports a finer interface to the local least squares regression functionality on a grid than LocalPolyFilterND, but only for polynomials up to second degree. In addition to the response itself, this class can be used to calculate the gradient and the hessian of the local response surface defined by Eq. 63[16]. The predictor/response data can be provided by a method of some class (callback) which is sometimes more convenient than using just a grid of points.

LocalQuadraticLeastSquaresND — this class fits local least squares regression models for arbitrary multivariate predictor values (no longer required to be on a grid). The models can be heteroscedastic, as in the most general case of Eq. 64. The polynomials can be at most quadratic. Calculation of the gradient and the hessian of the local response surface is supported.

7.2 Local Logistic Regression

For regressing binary response variables, NPStat implements a method known as "local quadratic logistic regression" (LQLR). This method is a trivial extension of the local linear logistic regression originally proposed in [41]. In this type of analysis, "response" is the probability of success in a Bernoulli trial, estimated as a function of one or more predictors. Due to the manner in which it is often used, this probability will be called "efficiency" for the remainder of this section.

In the LQLR model, the efficiency dependence on $\mathbf{x}$ is represented by

$$\epsilon(\mathbf{x}) = \frac{1}{1 + e^{-P(\mathbf{x})}} \tag{70}$$

where $P(\mathbf{x})$ is a multivariate quadratic polynomial whose coefficients are determined at each predictor value $\mathbf{x}_{\rm fit}$ by maximizing the local log-likelihood:

$$L(\mathbf{x}_{\rm fit}, h) = \sum_{i=0}^{N-1} K_h(\mathbf{x}_i - \mathbf{x}_{\rm fit}) \left[y_i \ln(\hat{\epsilon}(\mathbf{x}_i)) + (1 - y_i) \ln(1 - \hat{\epsilon}(\mathbf{x}_i)) \right]. \tag{71}$$

Here, $K_h(\mathbf{x}_i - \mathbf{x}_{\rm fit})$ is a suitable localizing kernel which decays to 0 when $\mathbf{x}_i$ is far from $\mathbf{x}_{\rm fit}$, and y_i are the observed values of the Bernoulli trial: 1 if the point "passes" and 0 if it "fails". The local log-likelihood 71 is very similar to the one implemented in the Locfit package [42], the only difference is that orthogonal polynomials are used in NPStat to construct $P(\mathbf{x})$ series instead of monomials.

Setting partial derivatives of $L(\mathbf{x}_{\rm fit}, h)$ with respect to polynomial coefficients to 0 results in a system of nonlinear equations for these coefficients. Solving such a system of equations

[16] This can be useful, for example, for summarizing properties of log-likelihoods defined on grids in the space of estimated parameters for some parametric statistical models.

does not appear to be any easier than dealing with the original log-likelihood optimization problem by applying, let say, the Levenberg-Marquardt algorithm [43].

NPStat includes facilities for efficient calculation of the LQLR log-likelihood together with its gradient with respect to $P(\mathbf{x})$ expansion coefficients. This code does not rely on any specific optimization solver, and can be easily interfaced to a number of different external optimization tools. The relevant classes are:

LogisticRegressionOnKDTree (header file "npstat/stat/LocalLogisticRegression.hh") — this class calculates the log-likelihood from Eq. 71 in the assumption that the kernel $K_h(\mathbf{x})$ has finite support. In this case iterating over all N points in the sample becomes rather inefficient: there is no reason to cycle over values of $\mathbf{x}_i$ far away from $\mathbf{x}_{\rm fit}$ because $K_h(\mathbf{x}_i - \mathbf{x}_{\rm fit})$ is identically zero for all such points. To automatically restrict the iterated range, the predictor values are arranged into a space-partitioning data structure known as k-d tree [44] which is implemented in NPStat with the KDTree class (header file "npstat/nm/KDTree.hh").

LogisticRegressionOnGrid (header file "npstat/stat/LocalLogisticRegression.hh") — this class calculates the log-likelihood from Eq. 71 in the assumption that the predictor values are histogrammed. Two identically binned histograms must be available: the one with all values of y_i ("the denominator") and the one which collects only those $\mathbf{x}_i$ for which $y_i = 1$ ("the numerator"). Naturally, only the bins sufficiently close to $\mathbf{x}_{\rm fit}$ are processed when the LQLR log-likelihood is evaluated for these histograms.

The "interfaces" directory of the NPStat package includes two high-level driver functions for fitting LQLR response surfaces (header file "npstat/interfaces/minuitLocalRegression.hh"). These functions employ a general-purpose optimization package Minuit [1] for maximizing the log-likelihood. The names of the functions are minuitUnbinnedLogisticRegression (intended for use with LogisticRegressionOnKDTree) and minuitLogisticRegressionOnGrid (for use with LogisticRegressionOnGrid). To use these functions, the Minuit package has to be compiled and linked together with the user code which provides the data and calls the functions.

7.3 Local Quantile Regression

The method of least squares allows us to solve the problem of response mean determination in the regression context. By recasting calculation of the sample mean as a minimization problem, we have gained the ability to condition the mean on the value of the predictor. In a similar manner, the method of least absolute deviations can be used to determine conditional median. In the method of least squares, the expression $S(f) = \sum_i f(y_i - \hat{\mu}(\mathbf{x_i}))$ is minimized, with $f(t) = t^2$. The method of least absolute deviations differs only by setting $f(t) = |t|$. Moreover, just like the problem of determination of response mean can be localized by introducing a kernel in the predictor space (resulting in local least squares, as in Eq. 62), the problem of response median determination can be subjected to the same localization treatment. As a method of determination of response location, local median regression is extremely robust (insensitive to outliers).

Not only the median but an arbitrary distribution quantile can also be determined in

this manner. The corresponding function to use is

$$f_q(t) = q\,t\,I(t \geq 0) - (1-q)\,t\,I(t < 0), \tag{72}$$

where q is the cumulative distribution value of interest, $0 < q < 1$. You can easily convince yourself of the validity of this statement as follows. For a sample of points $y_0, ..., y_{N-1}$, define $t = y - y_q$, where y_q is a parameter on which $S(f_q)$ depends. The condition for the minimum of $S(f_q)$, $\frac{dS(f_q)}{dy_q} = 0$, is then equivalent to $\frac{dS(f_q)}{dt} = 0$. As $\frac{df_q(t)}{dt} = q\,I(t > 0) - (1-q)\,I(t < 0)$, the minimum of $S(f_q)$ is reached when

$$q\sum_{i=0}^{N-1} I(y_i > y_q) - (1-q)\sum_{i=0}^{N-1} I(y_i < y_q) = 0. \tag{73}$$

This equation is solved when the number of sample points for which $y_i > y_q$ is $(1-q)N$ and the number of sample points for which $y_i < y_q$ is qN. But this is precisely the definition of the sample quantile which, in the limit $N \to \infty$, becomes the population quantile of interest.

Unfortunately, solving Eq. 73 numerically in the regression context is usually significantly more challenging than solving the corresponding χ^2 minimization problem. For realistic finite samples, $\frac{dS(f_q)}{dy_q}$ is not a continuous function, and Eq. 73 can have multiple solutions. This basically rules out the use of standard gradient-based methods for $S(f_q)$ minimization.

The following NPStat classes facilitate the solution of the local quantile regression problem:

QuantileRegression1D — calculates the expression to be minimized (also called the "loss function") for a single univariate predictor value $x_{\,\mathrm{fit}}$. This expression looks as follows:

$$S_{\mathrm{LQR}}(x_{\,\mathrm{fit}}) = \sum_{i=0}^{N-1} f_q(y_i - \hat{y}_q(x_i|x_{\,\mathrm{fit}}))w_i, \tag{74}$$

where weights w_i are provided by the user (for example, these weights can be calculated as $w_i = K((x_i - x_{\,\mathrm{fit}})/h)$, but more sophisticated weighting schemes can be applied as well). The quantile dependence on the predictor is modeled by

$$\hat{y}_q(x|x_{\,\mathrm{fit}}) = \sum_{k=0}^{M} \hat{a}_k(x_{\,\mathrm{fit}})P_k\left(\frac{x - x_{\,\mathrm{fit}}}{h}\right), \tag{75}$$

where $P_k(x)$ are either Legendre or Gegenbauer polynomials. The use of Legendre polynomials is appropriate for a global fit of the regression curve, while Gegenbauer polynomials are intended for use in combination with symmetric beta kernels that generate localizing weights w_i. The expansion coefficients $\hat{a}_k(x_{\,\mathrm{fit}})$ are to be determined by minimizing $S_{\mathrm{LQR}}(x_{\,\mathrm{fit}})$.

QuantileRegressionOnKDTree (header file "npstat/stat/LocalQuantileRegression.hh") — calculates the quantile regression loss function for a multivariate predictor (the class method

which returns it is called linearLoss):

$$S_{\mathrm{LQR}}(\mathbf{x}_{\mathrm{fit}}, h) = \sum_{i=0}^{N-1} f_q(y_i - \hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\mathrm{fit}}))K_h(\mathbf{x}_i - \mathbf{x}_{\mathrm{fit}}). \tag{76}$$

The quantile dependence on the predictor is modeled by an expansion similar to Eq. 75 in which multivariate orthogonal polynomials (up to second order) are generated by $K_h(\mathbf{x})$ used as the weight function. The expansion coefficients are to be determined for each $\mathbf{x}_{\mathrm{fit}}$ separately by minimizing $S_{\mathrm{LQR}}(\mathbf{x}_{\mathrm{fit}}, h)$. The predictor values are arranged into a k-d tree structure for reasons similar to those mentioned when the LogisticRegressionOnKDTree class was described.

QuantileRegressionOnHisto (header file "npstat/stat/LocalQuantileRegression.hh") — calculates loss function 76 in the assumption that the response is histogrammed on a regular grid of predictor values (so that the input histogram dimensionality is larger by one than the dimensionality of the predictor variable).

CensoredQuantileRegressionOnKDTree (header "npstat/stat/ensoredQuantileRegression.hh") — this class constructs an appropriate quantile regression loss function in case some of the response values are unavailable due to censoring (*i.e.,* there is a cutoff from above or from below on the response values). For example, this situation can occur in modeling of jet response of a particle detector when jets with visible energy below certain cutoff are not reconstructed due to limitations in the clustering procedure. It is assumed that the censoring efficiency (*i.e.,* the fraction of points not affected by the cutoff) can be determined as a function of the predictor by other techniques, like the local logistic regression described in section 7.2. It is also assumed that the cutoff value is known as a function of the predictor, and that the presence of this cutoff is the only reason for inefficiency. The appropriate loss function in this case is

$$S_{\mathrm{CLQR}}(\mathbf{x}_{\mathrm{fit}}, h) = \sum_{i=0}^{N_{\mathrm{p}}-1} \left[f_q(y_i - \hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\mathrm{fit}})) + \frac{1-\epsilon_i}{\epsilon_i} g_q(\epsilon_i, y_{\mathrm{cut},i} - \hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\mathrm{fit}})) \right] K_h(\mathbf{x}_i - \mathbf{x}_{\mathrm{fit}}), \tag{77}$$

where the summation is performed only over the N_{p} points in the sample surviving the cutoff. ϵ_i is the censoring efficiency for the given $\mathbf{x}_i$. The function $g_q(\epsilon, t)$ is defined differently for right-censored (R-C) samples in which surviving points are below the cutoff and left-censored (L-C) samples in which surviving points are above the cutoff:

$$g_{q,\text{R-C}}(\epsilon, t) = I(q < \epsilon) f_q(t) + I(q \geq \epsilon) \left(\frac{q-\epsilon}{1-\epsilon} f_q(t) + \frac{1-q}{1-\epsilon} f_q(+\infty) \right) \tag{78}$$

$$g_{q,\text{L-C}}(\epsilon, t) = I(q > 1-\epsilon) f_q(t) + I(q \leq 1-\epsilon) \left(\frac{1-\epsilon-q}{1-\epsilon} f_q(t) + \frac{q}{1-\epsilon} f_q(-\infty) \right) \tag{79}$$

Formulae 77, 78, and 79 were inspired by Ref. [45]. They can be understood as follows. Consider, for example, a right-censored sample. If this sample was not censored, the value of the response cumulative distribution function at $y_{\mathrm{cut},i}$ would be equal ϵ_i. For each point with

response y_i below the cutoff, there are $(1-\epsilon_i)/\epsilon_i$ unobserved points above the cutoff (and $1/\epsilon_i$ points total). Before localization, points below the cutoff contribute the usual amount $f_q(y_i - \hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit}))$ into $S_{\rm CLQR}(\mathbf{x}_{\rm fit}, h)$ (compare with Eq. 76). To the points above cutoff, we assign the value of response which equals either $y_{{\rm cut},i}$ or $+\infty$, in such a manner that the estimate $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit})$ is "pushed" in the right direction and by the right amount when it crosses $y_{{\rm cut},i}$. If the estimated quantile is less than the efficiency, all unobserved points are assigned the response value $y_{{\rm cut},i}$ (this corresponds to the term $I(q < \epsilon)f_q(t)$ in the Eq. 78). This is because we know that the correct value of $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit})$ should be below $y_{{\rm cut},i}$, so the penalty for placing $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit})$ above the cutoff is generated by all points, including the unobserved ones. If the chosen quantile is larger than the efficiency, the only thing we know is that the correct value of $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit})$ should be inside the interval $(y_{{\rm cut},i}, +\infty)$. There is no reason to prefer any particular value from this interval, so the overall contribution of sample point i for which $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit}) \in (y_{{\rm cut},i}, +\infty)$ into $S_{\rm CLQR}(\mathbf{x}_{\rm fit}, h)$ must not depend on $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit})$[17] We do, however, want to prevent $\hat{y}_q(\mathbf{x}_i|\mathbf{x}_{\rm fit})$ from leaving this interval. This is achieved by placing so many points at $y_{{\rm cut},i}$ that the fraction of sample points (including both observed and unobserved ones) at or below $y_{{\rm cut},i}$ is exactly q, and this is precisely what the term proportional to $I(q \geq \epsilon)$ does in Eq. 78. Similar reasoning applied to a left-censored sample leads to Eq. 79.

Naturally, in the computer program, $-\infty$ and $+\infty$ in Eqs. 78 and 79 should be replaced by suitable user-provided numbers which are known to be below and above, respectively, all possible response values. In order to avoid deterioration in $S_{\rm CLQR}(\mathbf{x}_{\rm fit}, h)$ numerical precision, these numbers should not be very different from the minimum and maximum observed response.

CensoredQuantileRegressionOnHisto (header "npstat/stat/ensoredQuantileRegression.hh") — this class calculates the loss function 77 in the assumption that all information about response, efficiency, and cutoffs is provided on a regular grid in the space of predictor values.

The "interfaces" directory of the NPStat package includes several high-level driver functions for performing local quantile regression. These driver functions use the simplex minimization method of the Minuit package to perform local fitting of the quantile curve expansion coefficients. minuitLocalQuantileRegression1D function uses QuantileRegression1D class internally to perform local quantile regression with one-dimensional predictors. This driver function can be used, for example, to construct Neyman belts from numerical simulations of some statistical estimator. A similar function, weightedLocalQuantileRegression1D, can be used to perform local quantile regression with one-dimensional predictors when the points are weighted. The minuitQuantileRegression driver function can use one of the QuantileRegressionOnKDTree, QuantileRegressionOnHisto, CensoredQuantileRegressionOnKDTree, or CensoredQuantileRegressionOnHisto classes (all of which inherit from a common base) to perform local quantile regression with multivariate predictors. The function minuitQuantileRegressionIncrBW has similar functionality but it can also automatically increase the localization

[17] If most points $\mathbf{x}_i$ are like that for some $\mathbf{x}_{\rm fit}$ value, the fit becomes unreliable. Consider increasing the bandwidth of the localization kernel or avoid such situations altogether. You should not expect to obtain good results for high (low) q values and all possible $\mathbf{x}_{\rm fit}$ in a right (left)-censored sample.

kernel bandwidth so that not less than a certain predefined fraction of the whole sample participates in the the local quantile determination for each $\mathbf{x}_{\text{fit}}$.

7.4 Iterative Local Least Trimmed Squares

The NPStat package includes an implementation of an iterative local least trimmed squares (ILLTS) algorithm applicable when predictor/response values are supplied on a regular grid of points in a multivariate predictor space (think image denoising). The algorithm operation consists of the following steps:

1. Systems of orthogonal polynomials up to user-defined degree M are constructed using weight functions $K_{h,-j}(\mathbf{x})$. These weight functions are defined using symmetric finite support kernels in which the point in the kernel center together with $j-1$ other points are set to 0. Imagine, for example, a 5×5 grid in two dimensions. Start with the uniform $K_h(\mathbf{x})$ kernel which is 1 at every grid point. The $K_{h,-1}(\mathbf{x})$ weight function is produced from $K_h(\mathbf{x})$ by setting the central grid cell to 0. 24 weight functions of type $K_{h,-2}(\mathbf{x})$ are produced from $K_{h,-1}(\mathbf{x})$ by setting one other grid cell to 0, in addition to the central one. 276 weight functions of type $K_{h,-3}(\mathbf{x})$ are produced from $K_{h,-1}(\mathbf{x})$ by choosing 2 out of 24 remaining cells which are set to 0, and so on.

2. Local least squares regression is performed using all polynomial systems generated by $K_{h,-j}(\mathbf{x})$ weights for all possible positions of the kernel inside the predictor grid (sliding window)[18]. For each kernel position, we find the polynomial system which produces the best χ^2 calculated over grid points for which the weight function is not 0. For this polynomial system, we determine the value $\Delta = |y_c - y_{c,\text{fit}}|$ for the kernel center, where y_c is the response value in the data sample and $y_{c,\text{fit}}$ is the response value produced by the local least squares fit.

3. The position of the kernel is found for which Δ is the largest in the whole data sample.

4. The response for the position found in the previous step is adjusted by setting it to the value fitted at the kernel center.

5. Δ is recalculated for all kernel positions affected by the adjustment performed in the previous step.

6. The previous three steps are repeated until some stopping criterion is satisfied. For example, the requirement that the largest Δ in the grid becomes small (below certain cutoff) can serve as such a stopping criterion.

The ILLTS algorithm works best if the fraction of outliers in the sample is relatively small and the response errors are homoscedastic. ILLTS is expected to be more efficient (in the statistical sense) than the local quantile regression. If the spectrum of response errors is

[18] Edge effects are taking into account by constructing special polynomial systems which use boundary kernels similar to $K_{h,-j}(\mathbf{x})$ but with different placement of zeros.

not known in advance, it becomes very instructive to plot the history of Δ values for which response adjustments were performed. This plot often exhibits two characteristic "knees" which correspond to the suppression of outliers and suppression of "normal" response noise. The procedure can be stopped somewhere between these knees and followed up by normal local least squares on the adjusted sample, perhaps utilizing different bandwidth.

Unfortunately, computational complexity of the ILLTS algorithm is increasing exponentially with increasing j, so only very small values of j are practical. The inability to use high values of j can be partially compensated for by choosing smaller bandwidth values and performing more iteration cycles. More cycles result in larger *effective* bandwidth — think, for example, what happens when you perform local least squares multiple times. For certain kernels like Gaussian or Cauchy (*i.e.,* stable distributions) and polynomial degree $M = 0$ (local constant fit), this is exactly equivalent to choosing larger bandwidth. For other types of kernels not only the effective bandwidth increases with the number of passes but also the effective kernel shape gets modified[19].

The top-level API function for running the ILLTS algorithm is called griddedRobustRegression. The $K_{h,-1}(\mathbf{x})$ weights can be used by supplying an object of WeightedLTSLoss type as its loss calculator argument, and $K_{h,-2}(\mathbf{x})$ weights are used by choosing TwoPointsLTSLoss instead. A simple stopping criterion based on the Δ value, local least trimmed squares χ^2, or the number of adjustment cycles can be specified with an object of GriddedRobustRegressionStop type. The griddedRobustRegression implementation is rather general, and can accept user-developed loss calculators and stopping criteria.

7.5 Organizing Regression Results

It is usually desirable to calculate the regression surface on a reasonably fine predictor grid and save the result of this calculation for subsequent fast lookup. A general inerface for such a lookup is provided by the AbsMultivariateFunctor class (header file "npstat/nm/AbsMultivariateFunctor.hh"). This class can be used to represent results of both parametric and nonparametric fits.

Persistent classes StorableInterpolationFunctor and StorableHistoNDFunctor derived from AbsMultivariateFunctor are designed to represent nonparametric regression results. Both of these classes assume that the regression was performed on a (hyper)rectangular grid of predictor points. StorableInterpolationFunctor supports multilinear interpolation and extrapolation of results, with flexible extrapolation (constant or linear) for each dimension beyond the grid boundaries. This class essentially combines the AbsMultivariateFunctor interface with the functionality of the LinInterpolatedTableND class discussed in more detail in Section 11.

The class StorableHistoNDFunctor allows for constant, multilinear, and multicubic interpolation[20] inside the grid boundaries, and for constant extrapolation outside the boundaries

[19]ILLTS adds the dimension of time (adjustment cycle number) to the solution of robust regression problem. It would be interesting to explore this, for example, by introducing time-dependent bandwidth or by studying the connection between the ILLTS updating scheme and the numerous updating schemes employed in solving partial differential equations.

[20]Multicubic interpolation is supported for uniform grids only.

(the extrapolated response value is set to its value at the closest boundary point). Both StorableInterpolationFunctor and StorableHistoNDFunctor support arbitrary transformations of the response variable via a user-provided functor. If a transformation was initially applied to the response values in order to simplify subsequent modeling, this is a good place to perform the inverse.

8 Unfolding with Smoothing

In particle physics, the term *unfolding* is used to describe methods of nonparametric reconstruction of probability densities using observations affected by particle detector resolution and inefficiency (see [46] for a contemporary review). In other natural sciences, the term *inverse problem* is commonly used [47], while in the statistical literature a more specific name, *deconvolution density estimate*, is becoming the norm [48].

8.1 Unfolding Problem

For the purpose of stating the unfolding problem, it will be assumed that the detector can be described by an operator K. This operator (also called *kernel*, *transfer function*, *observation function*, or *response function*, depending on the author and context) converts probability densities $p(\mathbf{x})$ in the physical process space $\mathbf{x}$ into the densities $q(\mathbf{y})$ in the observation space $\mathbf{y}$: $q = Kp \equiv \int K(\mathbf{y}, \mathbf{x}) p(\mathbf{x}) \mathbf{dx}$. The response function does not have to be fully efficient: q does not have to integrate to 1 when p is normalized. In the subsequent discussion, operator K will be assumed linear and exactly known but not necessarily invertible.

In many situations of interest, observations are described by the empirical density function (*i.e.*, there is no error term associated with each individual observation):

$$\rho_e(\mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{y} - \mathbf{y}_i). \tag{80}$$

In this case, the probability to observe a point at $\mathbf{y}_i$ is given by the normalized version of q called r: $r = q/\epsilon$. In case p is normalized,

$$\epsilon = \int q(\mathbf{y}) \, \mathbf{dy} = \int Kp \, \mathbf{dy} \tag{81}$$

is the overall detector acceptance for the physical process under study.

The purpose of unfolding is to learn as much as possible about $p(\mathbf{x})$ given $\rho_e(\mathbf{y})$ when a parametric model for $p(\mathbf{x})$ is lacking. The difficulty of this problem can be easily appreciated from the following argument. K typically acts as low pass filter. For measurements of a single scalar quantity, it can often be assumed that the detector has resolution σ and that $K(y, x) = \mathcal{N}(y - x, \sigma^2)$, so that the detector simply convolves $p(x)$ with the normal distribution. If q was exactly known, the Fourier transform of p could simply be obtained from $p(\omega) = q(\omega)/K(\omega)$. As $q(\omega)$ is not known, the closest available approximation is the

characteristic function of $\rho_e(y)$: $\rho_e(\omega) = \int \rho_e(y) e^{i\omega y} dy = \frac{1}{N} \sum_{i=1}^{N} e^{i\omega y_i}$. As the characteristic function of the normal distribution is just $K(\omega) = e^{-\sigma^2\omega^2/2}$, the ratio $\rho_e(\omega)/K(\omega)$ becomes arbitrarily large as $\omega \to \infty$. The "naive" method of estimating $p(\omega)$ as $\rho_e(\omega)/K(\omega)$ thus fails miserably: the high frequency components of the noise contained in the $\rho_e(\omega)$ are multiplied by an arbitrarily large factor so that $\rho_e(\omega)/K(\omega)$ isn't even square-integrable.

A number of effective approaches to solving the pure deconvolution problem just described are discussed in [48]. These approaches invariably involve introduction of additional smoothness assumptions about either $p(x)$ or $q(y)$ or both. Such assumptions essentially declare that the high frequency components of $p(x)$ are of little interest and, therefore, can be suppressed in the $\rho_e(\omega)/K(\omega)$ ratio (so that the inverse Fourier transform can exist). Introduction of new information by applying additional assumptions which make an originally ill-posed problem treatable is called *regularization*.

In the problems of interest for particle physics, the action of $K(\mathbf{y}, \mathbf{x})$ on $p(\mathbf{x})$ is usually more complicated than simple convolution. At the time of this writing, reconstruction of $p(\mathbf{x})$ in particle physics applications is most often performed by either the SVD unfolding [49] or the expectation-maximization (a.k.a. D'Agostini, or Bayesian) unfolding [50][21]. Both of these methods start with the assumption that $\mathbf{x}$ and $\mathbf{y}$ spaces are binned, and that partitioning of these spaces is beyond the control of the method. In the SVD unfolding, one-dimensional $\mathbf{x}$ is assumed, and the regularization is performed by penalizing discretized second derivative of the $p(x)$ density[22]. In the expectation-maximization unfolding, regularization usually consists in imposing a subjective limit on the number of iteration cycles. This early stopping criterion penalizes deviations from the prior distribution used to start the iterations. However, due to the method nonlinearity, it is difficult to augment this statement with analytical derivations of the penalty applicable to arbitrary response functions.

It should be appreciated that, for these methods, the sample binning itself serves as a part of problem regularization. Just imagine making very wide bins — this leads to a response matrix which is almost diagonal and easily invertible. However, information about small structures within each bin is now lost.

8.2 EMS Unfolding

The unfolding method implemented in NPStat combines smoothing in the $\mathbf{x}$ space with the expectation-maximization iterations performed until convergence. This combination (abbreviated as EMS — expectation-maximization with smoothing) has important advantages over SVD unfolding and expectation-maximization with early stopping:

— *Ad hoc* binning is no longer needed. While the solution is still implemented on a grid, the cell width can be chosen sufficiently fine so that discretization does not affect the results.

[21] In other disciplines, utility of these methods has been discovered earlier [46].

[22] This regularization technique has been reinvented many times. Depending on the problem, is also called the *constrained linear inversion method*, the *Phillips-Twomey method*, the *Tikhonov regularization*, or the *ridge-parameter approach* [47, 48].

— Precise response functions can be employed instead of their coarsely binned versions affected by prior distribution assumptions.

— Problem regularization can be unambiguously described in terms of the parameters of the smoothing filter used (bandwidth, *etc*).

While the empirical success of the EMS unfolding method has already been reported in the statistical literature [51, 52, 50], the procedures implemented in NPStat also address the issues of uncertainty estimation for the reconstructed distribution and of choosing the filter parameters according to an objective criterion.

The method proceeds as follows. The standard expectation-maximization iterations update the reconstructed values of $p(\mathbf{x})$ according to the formula [51, 52]

$$\lambda_j^{(k+1)} = \frac{\lambda_j^{(k)}}{\epsilon_j} \sum_{i=1}^{n} \frac{K_{ij} y_i}{\sum_{\rho=1}^{m} K_{i\rho} \lambda_\rho^{(k)}}. \tag{82}$$

Here, $\lambda_j^{(k)}$ are the unnormalized $p(\mathbf{x})$ values (event counts) discretized on a sufficiently fine grid in the physical process space $\mathbf{x}$ (whose cells are small in comparison with the typical size of response function features), obtained on a k^{th} iteration. The index $j = 1, ..., m$ refers to the *linearized* cell number in this (possibly multidimensional) grid. All $\lambda_j^{(0)}$ values (the starting point for the iterations) can normally be set to the same constant $c = N/(\epsilon\, m)$, where N is the number of observed events and ϵ is the overall detector efficiency for a constant $p(\mathbf{x})$. y_i, $i = 1, ..., n$, denotes the number of observed events inside the cell with linearized index i in the space of observations $\mathbf{y}$. Dimensionalities of the $\mathbf{x}$ and $\mathbf{y}$ spaces can be arbitrary and distinct. K_{ij} is the discretized response matrix. It is the probability that an event from the physical cell j causes an observation in the cell i of the $\mathbf{y}$ space. $\epsilon_j = \sum_{i=1}^{n} K_{ij}$ is the detector efficiency for the physical cell j.

These iterations are modified by introducing a smoothing step. The updating scheme becomes

$$\lambda_j^{*(k+1)} = \frac{\lambda_j^{(k)}}{\epsilon_j} \sum_{i=1}^{n} \frac{K_{ij} y_i}{\sum_{\rho=1}^{m} K_{i\rho} \lambda_\rho^{(k)}}, \tag{83}$$

$$\lambda_r^{(k+1)} = \alpha^{(k+1)} \sum_{j=1}^{m} S_{rj} \lambda_j^{*(k+1)}, \tag{84}$$

where S_{rj} is the *smoothing matrix*, and the smoothing step normalization constant, $\alpha^{(k+1)}$, preserves the overall event count obtained during the preceding expectation-maximization step (so that $\sum_{r=1}^{m} \lambda_r^{(k+1)} = \sum_{j=1}^{m} \lambda_j^{*(k+1)}$). The values $\lambda_j^{(\infty)}$ obtained upon iteration convergence are therefore solutions of the equation

$$\lambda_r^{(\infty)} = \alpha^{(\infty)} \sum_{j=1}^{m} S_{rj} \frac{\lambda_j^{(\infty)}}{\epsilon_j} \sum_{i=1}^{n} \frac{K_{ij} y_i}{\sum_{\rho=1}^{m} K_{i\rho} \lambda_\rho^{(\infty)}}, \tag{85}$$

where $\alpha^{(\infty)} = \sum_{r=1}^{m} \lambda_r^{*(\infty)} / \sum_{r=1}^{m} \sum_{j=1}^{m} S_{rj} \lambda_j^{*(\infty)}$.

The equation for the error propagation matrix, $J_{rs} \equiv \frac{\partial \lambda_r^{(\infty)}}{\partial y_s}$, can be obtained by differentiating Eq. 85 w.r.t. y_s. In the matrix notation, this equation is

$$\mathbf{J} = (\alpha^{(\infty)}\mathbf{S} + \mathbf{A})\,(\mathbf{M} + \mathbf{B}\mathbf{J})\,, \tag{86}$$

where

$$A_{jq} = \frac{\left(1 - \alpha^{(\infty)} \sum_{r=1}^{m} S_{rq}\right) \lambda_j^{(\infty)}}{\sum_{i=1}^{m} \lambda_i^{(\infty)}}, \tag{87}$$

$$B_{jq} = \frac{\delta_{jq}}{\epsilon_j} \sum_{i=1}^{n} \frac{K_{ij} y_i}{\sum_{\rho=1}^{m} K_{i\rho} \lambda_\rho^{(\infty)}} - \frac{\lambda_j^{(\infty)}}{\epsilon_j} \sum_{i=1}^{n} \frac{K_{iq} K_{ij} y_i}{\left(\sum_{\rho=1}^{m} K_{i\rho} \lambda_\rho^{(\infty)}\right)^2}, \tag{88}$$

$$M_{jq} = \frac{\lambda_j^{(\infty)}}{\epsilon_j} \frac{K_{qj}}{\sum_{\rho=1}^{m} K_{q\rho} \lambda_\rho^{(\infty)}}. \tag{89}$$

The NPStat code solves the equivalent equation, $(\mathbf{I} - (\alpha^{(\infty)}\mathbf{S} + \mathbf{A})\mathbf{B})\,\mathbf{J} = (\alpha^{(\infty)}\mathbf{S} + \mathbf{A})\mathbf{M}$, using the LU factorization algorithm as implemented in LAPACK [40], and then runs iterative refinement cycles defined by Eq. 86 until convergence.

For unweighted samples, the covariance matrix of observations, $\mathbf{V}$, can be derived automatically by NPStat according to either Poisson or multinomial distribution assumptions using $\hat{y}_i = \sum_{\rho=1}^{m} K_{i\rho} \lambda_\rho^{(\infty)}$ as mean values. In more complicated situations, the user is expected to provide the covariance matrix of observations[23]. The covariance matrix of unfolded values is then estimated according to $\mathbf{J}\mathbf{V}\mathbf{J}^{\mathrm{T}}$.

While the original expectation-maximization algorithm is agnostic about the dimensionalities of $\mathbf{x}$ and $\mathbf{y}$ spaces, the smoothing step is, of course, dimensionality-specific. The unfolding code is using smoothing filters constructed with the help of facilities described in Section 6. The following classes implement unfolding with smoothing:

SmoothedEMUnfold1D (header file "npstat/stat/SmoothedEMUnfold1D.hh") — unfolds one-dimensional distributions using objects of LocalPolyFilter1D class as smoothing filters.

SmoothedEMUnfoldND (header file "npstat/stat/SmoothedEMUnfoldND.hh") — unfolds multivariate distributions. Dimensionalities of the $\mathbf{x}$ and $\mathbf{y}$ spaces can be arbitrary and distinct. Dimensionality of the smoothing filter is, of course, expected to be consistent with the structure of $\mathbf{x}$. Typically, the filters will be objects of either LocalPolyFilterND or SequentialPolyFilterND class, adapted for unfolding use by the UnfoldingFilterND template. SmoothedEMUnfoldND code is employing a more efficient implementation of the response matrix than the Matrix class used by the rest of NPStat code, but Eq. 86 is still solved using normal dense matrices. Therefore, practically usable number of cells in the discretization of the $\mathbf{x}$ space will be at most a few thousands, as the computational complexity of the algorithm based on dense matrices is proportional to this number cubed.

[23]This may require running the unfolding code twice, first to obtain $\lambda_\rho^{(\infty)}$, and then, when the covariance matrix is constructed externally, to propagate the uncertainties. Note that unfolding with the expectation-maximization algorithm intrinsically assumes Poisson distribution of the observed counts. If the covariance matrix of observations is highly inconsistent with this assumption, it will be impossible to interpret the result as a maximum likelihood estimate.

8.3 Choosing the Smoothing Parameters

For likelihood-based inference, a useful model selection principle is provided by the Akaike information criterion (AIC) [53]. The AIC criterion adjusted for the finite sample size is [54]

$$AIC_c = -2\ln L + 2k + \frac{2k(k+1)}{N-k-1}, \tag{90}$$

where L is the maximized value of the model likelihood, N is the sample size, and k the number of parameters in the model. Selecting a model by minimizing AIC_c avoids overfitting by reaching a compromise between complexity of the model and goodness-of-fit.

Application of the AIC_c criterion to the EMS unfolding procedure is, however, not completely straightforward. While calculation of the likelihood can be accomplished assuming Poisson distribution in the space of observations, it is not immediately obvious how to count the number of parameters in the model. To overcome this difficulty, NPStat assumes that the number of model parameters can be estimated as the *effective rank* of the $\mathbf{KJJ}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}$ matrix. This assumption is based on the following reasoning[24]. The covariance matrix of the fitted folded values (*i.e.*, $\hat{y}_i$) is $\mathbf{V}_{\hat{y}}(\mathbf{V}) = \mathbf{KJVJ}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}$. If, using polynomial series, one fits multiple independent samples of random points taken from the uniform distribution, with the number of points per sample varying according to the Poisson distribution, the rank of the covariance matrix of the fitted unnormalized density values calculated over these samples will be equal to the degree of the fittted polynomial plus one. This is precisely the number of parameters of the fitted model. It doesn't matter how many abscissae are used to construct the covariance matrix of the fitted values as long as the number of abscissae exceeds the degree of the polynomial and the average number or points in a sample is "sufficiently large". While the model fitted to the observed values by the EMS unfolding method isn't polynomial, we can still identify some measure of the rank of $\mathbf{V}_{\hat{y}}(\mathbf{I}) = \mathbf{KJJ}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}$ with the number of model parameters.

Two different definitions of the effective rank of a symmetric positive-semidefinite matrix (say, $\mathbf{Q}$) are implemented. The first one is the exponent of the von Neumann entropy of $\mathbf{Q}/\mathrm{tr}(\mathbf{Q})$. In terms of the $\mathbf{Q}$ eigenvalues, e_i, it is expressed as[25]

$$\mathrm{erank}_1(\mathbf{Q}) = \exp\left\{-\sum_{i=1}^{n}\frac{e_i}{\|e\|}\ln\left(\frac{e_i}{\|e\|}\right)\right\}, \quad \|e\| = \sum_{i=1}^{n} e_i. \tag{91}$$

The second is the ratio of the matrix trace to the largest eigenvalue:

$$\mathrm{erank}_2(\mathbf{Q}) = \frac{\mathrm{tr}(\mathbf{Q})}{\max_{1\le i\le n} e_i} = \frac{\|e\|}{\max_{1\le i\le n} e_i}. \tag{92}$$

These effective ranks are calculated by the symPSDefEffectiveRank method of the NPStat Matrix class. From some initial experimentation with simple models, it appears that setting

[24] One can also argue that the $\mathbf{KJ}$ matrix plays similar role to the hat matrix in linear regression problems. This leads to the same conclusion about the number of model parameters.

[25] Naturally, $\mathrm{erank}_1(\mathbf{Q})$ is also the exponent of the Shannon entropy of the normalized eigenspectrum.

k in Eq. 90 to either $\mathrm{erank}_1(\mathbf{K}\mathbf{J}\mathbf{J}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}})$ or $\mathrm{erank}_2(\mathbf{K}\mathbf{J}\mathbf{J}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}})$ works well, as both of these ranks have similar derivatives w.r.t. filter bandwidth. The corresponding AIC_c criteria will be called $EAIC_c$ (E in this abbreviation stands for "entropic") and $TAIC_c$ (T stands for "trace").

For any smoothing filter, the most important parameter regulating its functionality is the bandwidth. NPStat provides the following classes which facilitate the studies of AIC_c criteria and unfolding performance vs. filter bandwidth:

UnfoldingBandwidthScanner1D (header "npstat/stat/UnfoldingBandwidthScanner1D.hh") — simplifies and speeds up studies of 1-d EMS unfolding results obtained with multiple bandwidth values. Filters constructed for different bandwidth settings are stored internally and do not have to be recomputed when multiple observed samples are processed. Effective ranks of the $\mathbf{K}\mathbf{J}\mathbf{J}^{\mathrm{T}}\mathbf{K}^{\mathrm{T}}$ matrices are memoized as well. The bandwidth which optimizes either $EAIC_c$ or $TAIC_c$ criterion can be automatically searched for by calling the processAICcBandwidth method of this class. Typical class usage is illustrated by the "ems_unfold_1d.cc" example program located in the "examples/C++" subdirectory of the NPStat package.

UnfoldingBandwidthScannerND (header "npstat/stat/UnfoldingBandwidthScannerND.hh") — assists in studies of multivariate EMS unfolding results obtained with multiple bandwidth settings. SequentialPolyFilterND class is used internally to generate multivariate filters according to the user-provided bandwidth values in each dimension of the $\mathbf{x}$ space. The filters and the effective ranks are memoized, but there is no automatic procedure to determine the optimal bandwidth set[26].

Use of an effective rank to determine the number of model parameters leads to the requirement that the number of discretization cells in the observation space $\mathbf{y}$ should be substantially larger than this rank. This condition should be verified once the EMS unfolding is performed with the optimal filter.

If the unfolded values of $p(\mathbf{x})$ are expected to become close to 0 somewhere in the $\mathbf{x}$ region considered, it is important to choose a type of filter that guarantees the non-negativity of the smoothed result[27]. All LOrPE filters of degree zero have this property.

8.4 Large Problems with Sparse Matrices

The standard matrix multiplication methods and the LU factorization algorithm used to solve Eq. 86 have $O(m^3)$ computational complexity, where m is the number of cells used to discretize the physical process space. For programs running on a single processor, this complexity effectively limits m to a few thousands. However, in many practically important applications the response function is sufficiently short-ranged so that most elements of its

[26]The UnfoldingBandwidthScannerND class is designed to speed up repetitive mutidimensional grid scans. For more sophisticated algorithms searching for a function minimum in multiple dimensions, the strategy of memoizing a lot of intermediate results for each point considered will not be optimal.

[27]The code will automatically truncate negative filtered values but this truncation will lead to distortions in the covariance matrix of unfolded results not taken into account by linear error propagation formulae. A highly inappropriate choice of filter can even break the expectation-maximization iterations by producing $\hat{y}_i = 0$ corresponding to a positive y_i value.

discretized representation K_{ij} are zeros. In such situations it may be advantageous to utilize techniques and algorithms developed for sparse matrices in order to increase m.

NPStat provides a header-only EMS unfolding implementation based on sparse matrix arithmetic in the namespace "emsunfold". In order to take advantage of this code, two additional software packages have to be installed by the user. NPStat relies on Eigen [55] for basic operations with sparse matrices and for solving sparse linear systems. The package TRLAN [56] is used for calculating leading eigenvalues and eigenvectors of large covariance matrices[28]. The following classes implement EMS unfolding with sparse matrices:

SmoothedEMSparseUnfoldND (header "npstat/emsunfold/SmoothedEMSparseUnfoldND.hh") — parallels the functionality of the SmoothedEMUnfoldND class, with a few peculiarities related to the sparseness of various matrices. In particular, only Poisson uncertainties on the observed data can be calculated automatically, as multinomial covariances would not be sparse. User-provided uncertainties are restricted to diagonal matrices. The smoothing matrix should be doubly stochastic[29], otherwise Eq. 87 will inject a dense matrix into the system.

SparseUnfoldingBandwidthScannerND — parallels the functionality of UnfoldingBandwidthScanner1D, with additional input parameters and diagnostic outputs needed to drive determination of eigenvalues and eigenvectors of large covariance matrices. This class is declared in the header file "npstat/emsunfold/SparseUnfoldingBandwidthScannerND.hh".

9 Pseudo- and Quasi-Random Numbers

The C++11 Standard [3] defines an API for generating pseudo-random numbers. Unfortunately, this API suffers from a disconnect between modeling of statistical distributions (including densities, cumulative distributions, *etc*) and generation of random numbers. Moreover, quasi-random numbers [57] useful for a large variety of simulation and data analysis purposes are not represented by the Standard, and there is no meaningful support for generating genuinely multivariate random sequences.

NPStat adopts a different approach towards generation of pseudo-random, quasi-random, and non-random sequences — the one that is more appropriate in the context of a statistical analysis package. A small number of high-quality generators is implemented for producing such sequences on a unit d-dimensional cube. All such generators inherit from the same abstract base class AbsRandomGenerator (header file "npstat/rng/AbsRandomGenerator.hh"). Generators developed or ported by users can be seamlessly added as well. Conversion of uniformly distributed sequences into other types of distributions and to different support regions is performed by the classes that represent statistical distributions — in particular,

[28] I am not aware of any method for calculating *all* eigenvalues and eigenvectors of a symmetric positive-semidefinite matrix, sparse or not, with computational complexity better than $O(m^3)$.

[29] That is, each row and column of the smoothing matrix should sum to 1, within the tolerance parameter of the unfolding algorithm. Construction of doubly stochastic filters with NPStat is discussed in Section 6.3. Double stochasticity is not enforced by the code itself, as it is often useful to benchmark the results obtained with sparse matrices against an implementation utilizing dense matrices.

by those classes which inherit from AbsDistribution1D and AbsDistributionND bases. Both of these bases have a virtual method random which takes a sequence generator instance as an input and produces correspondingly distributed numbers (random or not) on output. By default, transformation of sequences is performed by the quantile method for one-dimensional distributions and by unitMap method for multivariate ones. However, it is expected that the random method itself will be overriden by the derived classes when it is easier, for example, to make a random sequence with the desired properties by the acceptance-rejection technique. The following functions and classes are implemented:

MersenneTwister (header file "npstat/rng/MersenneTwister.hh") — generates pseudo-random numbers using the Mersenne Twister algorithm [58].

SobolGenerator (header file "npstat/rng/SobolGenerator.hh") — generates Sobol quasi-random sequences [59].

HOSobolGenerator (header file "npstat/rng/HOSobolGenerator.hh") — generates higher order scrambled Sobol sequences [60].

RandomSequenceRepeater (header file "npstat/rng/RandomSequenceRepeater.hh") — this class can be used to produce multiple repetitions of sequences created by other generators whenever an instance of AbsRandomGenerator is needed. The whole sequence is simply remembered (which can take a significant amount of memory for large sequences) and extended as necessary by calling the original generator.

WrappedRandomGen (header file "npstat/rng/AbsRandomGenerator.hh") — a simple adaptor class for "old style" random generator functions like "drand48()". Implements AbsRandomGenerator interface.

CPP11RandomGen (header file "npstat/rng/CPP11RandomGen.hh") — a simple adaptor class for pseudo-random generator engines defined in the C++11 standard. Implements AbsRandomGenerator interface.

EquidistantSampler1D (header file "npstat/rng/EquidistantSampler1D.hh") — generates a sequence of equidistant points, similar to bin centers of a histogram with axis limits at 0 and 1.

RegularSampler1D (header file "npstat/rng/RegularSampler1D.hh") — generates a sequence of points by splitting the $[0, 1]$ interval by 2, then splitting all subintervals by 2, *etc.* The points returned are the split locations. Useful for generating $2^k - 1$ points when integer k is not known in advance.

convertToSphericalRandom (header file "npstat/rng/convertToSphericalRandom.hh") — converts a multivariate random number from a unit d-dimensional cube into a random direction in d dimensions and one additional random number between 0 and 1. Useful for generating random numbers according to spherically symmetrical distributions.

10 Algorithms Related to Combinatorics

The NPStat package implements several functions and algorithms related to permutations of integers and combinatorics:

factorial (header file "npstat/rng/permutation.hh") — this function returns an exact

factorial if the result does not exceed the largest unsigned long (up to 12! on 32-bit systems and up to 20! on 64-bit ones).

ldfactorial (header file "npstat/rng/permutation.hh") — this function returns an approximate factorial up to 1754! as a long double.

logfactorial (header file "npstat/rng/permutation.hh") — natural logarithm of a factorial, up to $\ln((2^{32}-1)!)$, as a long double.

binomialCoefficient (header file "npstat/nm/binomialCoefficient.hh") — calculates the binomial coefficients $C_N^M = \frac{N!}{M!(N-M)!}$ using an algorithm which avoids overflows.

orderedPermutation (header file "npstat/rng/permutation.hh") — this function can be used to iterate over permutations of numbers $\{0, 1, ..., N-1\}$ in a systematic way. It generates a unique permutation of such numbers given a non-negative input integer below $N!$.

permutationNumber (header file "npstat/rng/permutation.hh") — inverse of orderedPermutation: maps a permutation of numbers $\{0, 1, ..., N-1\}$ into a unique integer between 0 and $N!$.

randomPermutation (header file "npstat/rng/permutation.hh") — generates random permutations of numbers $\{0, 1, ..., N-1\}$ in which probability of every permutation is the same.

NMCombinationSequencer (header file "npstat/stat/NMCombinationSequencer.hh") — this class iterates over all possible choices of $j_1, ..., j_M$ from N possible values for each j_k in such a way that all $j_1, ..., j_M$ are distinct and appear in the sequence in the increasing order, last index changing most often. Naturally, the total number of all such choices equals to the number of ways to pick M distinct items out of N: it is the binomial coefficient C_N^M.

11 Numerical Analysis Utilities

The NPStat package includes a menagerie of numerical analysis utilities designed, primarily, to support the statistical calculations described in the previous sections. They are placed in the "nm" directory of the package. A number of these utilities can be used as stand-alone tools. If the corresponding header file is not mentioned explicitly in the descriptions below, it is "npstat/nm/NNNN.hh", where NNNN stands for the actual name of the class or function.

ConvolutionEngine1D and ConvolutionEngineND — These classes encapsulate the NPStat interface to the FFTW package [61]. They can be used to performs DFFT convolutions of one-dimensional and multivariate functions, respectively.

EquidistantInLinearSpace (header file "npstat/nm/EquidistantSequence.hh") — A sequence of equidistant points. For use with algorithms that take a vector of points as one of their parameters.

EquidistantInLogSpace (header file "npstat/nm/EquidistantSequence.hh") — A sequence of points whose logarithms are equidistant.

findRootInLogSpace — templated numerical equation solving for 1-d functions (or for 1-d subspaces of multivariate functions) using interval division. It is assumed that the solution can be represented as a product of some object (*e.g.,* a vector) by a positive real number, and that number is then searched for.

GaussHermiteQuadrature and GaussLegendreQuadrature — templated Gauss-Hermite and Gauss-Legendre quadratures for one-dimensional functions. Internally, calculations are performed in long double precision. Of course, lower precision functions can be integrated as well, with corresponding reduction in the precision of the result.

rectangleIntegralCenterAndSize (header file "npstat/nm/rectangleQuadrature.hh") — Gauss-Legendre cubatures on rectangular and hyperrectangular domains using tensor product integration[30].

goldenSectionSearchInLogSpace (header file "npstat/nm/goldenSectionSearch.hh") — templated numerical search for a minimum of 1-d functions (or for 1-d subspaces of multivariate functions) using the golden section method. It is assumed that location of the minimum can be represented as a product of some object (*e.g.,* a vector) by a positive constant, and that constant is then searched for.

goldenSectionSearchOnAGrid (header file "npstat/nm/goldenSectionSearch.hh") — search for a minimum using coordinates restricted to a user-defined one-dimensional grid. Appropriate for use with functions which are expensive to evaluate and for which the user has some prior idea about their smoothness.

parabolicExtremum (header file "npstat/nm/MathUtils.hh") – determine extremum of a parabola passing through three given points on a plane. Can be used in combination with goldenSectionSearchOnAGrid to refine location of the minimum.

GridAxis — This class can be used to define an axis of a rectangular grid with non-uniform spacing of points. The complementary class UniformAxis works more efficiently for representing equidistant points, while the class DualAxis can be used to represent both uniform and non-uniform grids.

interpolate_linear, interpolate_quadratic, interpolate_cubic (these three functions are declared in the header file "npstat/nm/interpolate.hh") — linear, quadratic, and cubic polynomials with given values at two, three, and four equidistant points, respectively.

LinInterpolatedTable1D — persistent one-dimensional lookup table with linear interpolation between the tabulated values. Useful for representing arbitrary one-dimensional functions in case the full numerical precision is not required. If the table is monotonous, the inverse table can be constructed automatically.

LinInterpolatedTableND — persistent multidimensional lookup table with multilinear interpolation between the tabulated values, as in Eq 25. Extrapolation beyond the grid boundaries is supported as well. This class is useful for representing arbitrary functions in case the full numerical precision is not required. GridAxis, UniformAxis, or DualAxis class (or user-developed classes with similar sets of methods) can be used to define grid point locations. Note that simple location-based lookup of stored values (without interpolation) can be trivially performed with the closestBin method of the HistoND class. Lookup of histogram bin values with interpolation can be performed by the interpolateHistoND function (header file "npstat/stat/interpolateHistoND.hh").

[30] "Tensor product integration" simply means that the locations at which the function is evaluated and corresponding weights are determined by sequential application of Gauss-Legendre quadratures in each dimension.

LinearMapper1d, LogMapper1d — linear and log-linear transformations in 1-d as functors (with "double operator()(const double& x) const" method defined). The CircularMapper1d class works similarly to LinearMapper1d in situations with circular data topologies.

Matrix — A templated matrix class. Useful for standard matrix manipulations, solving linear systems, finding eigenvalues and eigenvectors of symmetric matrices, singular value decomposition, *etc.* Encapsulates NPStat interface to LAPACK [40].

findPeak3by3, findPeak5by5 (header file "npstat/nm/findPeak2D.hh") — utilities which facilitate peak finding for two-dimensional surfaces which can be contaminated by small amounts of noise (*e.g.*, from round-off errors). The user can fit a 2-d quadratic polynomial inside a 3×3 or 5×5 window by least squares[31] and check whether that polynomial has an extremum inside the window. Initially intended for studying 2-d log-likelihoods using sliding windows.

solveQuadratic, solveCubic (header file "npstat/nm/MathUtils.hh") — solutions of quadratic and cubic equations, respectively, by numerically sound methods[32].

ndUnitSphereVolume (header file "npstat/nm/MathUtils.hh") — volume of the n-dimensional unit sphere.

polyAndDeriv (header file "npstat/nm/MathUtils.hh") — monomial series $\sum_{k=0}^{M} c_k x^k$ and its derivative with respect to x, templated on the type of coefficients c_k.

polySeriesSum, legendreSeriesSum, gegenbauerSeriesSum, chebyshevSeriesSum (all of these functions are declared in the header file "npstat/nm/MathUtils.hh") — templated series of one-dimensional monomials, Legendre polynomials, Gegenbauer polynomials, and Chebyshev polynomials, respectively. Numerically sound recursive formulae are used to generate the polynomials.

hermiteSeriesSumProb, hermiteSeriesSumPhys (header file "npstat/nm/MathUtils.hh") — templated series of "probabilist" and "physicist" Hermite polynomials, respectively.

chebyshevSeriesCoeffs (header file "npstat/nm/MathUtils.hh") — utility for approximating mathematical functions with Chebyshev polynomials.

OrthoPoly1D, OrthoPolyND — uni- and multivariate orthogonal polynomials on equidistant rectangular grids with arbitrary weight functions. In addition to producing the polynomials themselves, these classes can be used to calculate polynomial series, polynomial series expansion coefficients for gridded functions, and polynomial filters defined by Eq. 46 (but normalized so that the sum of filter coefficients is 1).

The NPStat package also includes implementations of various special functions needed in statistical calculations (incomplete gamma function and its inverse, incomplete beta function, *etc*). These functions are declared in the header file "npstat/nm/SpecialFunctions.hh".

[31] A fast method utilizing discrete orthogonal polynomial expansion is used internally.

[32] The textbook formula $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ for the roots of quadratic equation $ax^2 + bx + c = 0$ is also a prime example of a numerical analysis pitfall. A minimal modification, $x_1 = -\frac{b + (I(b \geq 0) - I(b < 0))\sqrt{b^2 - 4ac}}{2a}$, $x_2 = \frac{c}{ax_1}$, avoids the subtractive cancellation problem in the $x_{1,2}$ numerator.

References

[1] Minuit2 Minimization Package, http://www.cern.ch/minuit/

[2] Geners — Generic Serialization for C++, http://geners.hepforge.org/

[3] ISO/IEC Standard 14882:2011 *Programming Language C++* (2011).

[4] M.C. Jones, and H.W. Lotwick, "On the Errors Involved in Computing the Empirical Characteristic Function", *Journal of Statistical Computation and Simulation* **17**, 133 (1983).

[5] D.N. Joanes and C.A. Gill, "Comparing measures of sample skewness and kurtosis", *The Statistician* **47**, 183 (1998).

[6] D.V. Hinkley, "On the ratio of two correlated normal random variables", *Biometrika* **56**, 635 (1969).

[7] N.L. Johnson, "Systems of Frequency Curves Generated by Methods of Translation", *Biometrika* **36**, 149 (1949).

[8] W.P. Elderton and N.L. Johnson, "Systems of Frequency Curves", Cambridge University Press (1969).

[9] G.J. Hahn and S.S. Shapiro, "Statistical Models in Engineering", Wiley (1994).

[10] J. Draper, "Properties of Distributions Resulting from Certain Simple Transformations of the Normal Distribution", *Biometrika* **39**, 290 (1952).

[11] I.D. Hill, R. Hill, and R. L. Holder, "Algorithm AS 99: Fitting Johnson Curves by Moments", *Applied Statistics* **25**, 180 (1976).

[12] M.S. Handcock and M. Morris, "Relative Distribution Methods", *Sociological Methodology* **28**, 53 (1998).

[13] O. Thas, "Comparing Distributions", Springer Series in Statistics (2009).

[14] L. Yang and J.S. Marron, "Iterated Transformation-Kernel Density Estimation", *Journal of the American Statistical Association* **94**, 580 (1999).

[15] J. Neyman, "'Smooth Test' for Goodness of Fit", *Skandinavisk Aktuarietidskrift* **20**, 150 (1937).

[16] R.B. Nelsen, "An Introduction to Copulas", 2nd Ed., Springer Series in Statistics (2006).

[17] "Eight queens puzzle", Wikipedia.

[18] "Rook polynomial", Wikipedia.

[19] J. Ma and Z. Sun, "Mutual information is copula entropy", arXiv:0808.0845v1 (2008).

[20] A.L. Read, "Linear interpolation of histograms", *Nucl. Instr. Meth.* **A 425**, 357 (1999).

[21] B.W. Silverman, "Density Estimation for Statistics and Data Analysis", Chapman & Hall (1986).

[22] A.J. Izenman, "Recent Developments in Nonparametric Density Estimation", *Journal of the American Statistical Association* **86**, 205 (1991).

[23] D.W. Scott, "Multivariate Density Estimation: Theory, Practice, and Visualization", Wiley (1992).

[24] M.C. Jones, J.S. Marron and S.J. Sheather, "A Brief Survey of Bandwidth Selection for Density Estimation", *Journal of the American Statistical Association* **91**, 401 (1996).

[25] C.R. Loader, "Bandwidth Selection: Classical or Plug-in?", *The Annals of Statistics* **27**, 415 (1999).

[26] M.P. Wand and M.C. Jones, "Kernel Smoothing", Chapman & Hall (1995).

[27] S.-T. Chiu, "An Automatic Bandwidth Selector for Kernel Density Estimation", *Biometrika* **79**, 771 (1992).

[28] T.-J. Wu and M.-H. Tsai, "Root n bandwidths selectors in multivariate kernel density estimation", *Probab. Theory Relat. Fields* **129**, 537 (2004).

[29] M.C. Jones, "Discretized and Interpolated Kernel Density Estimates", *Journal of the American Statistical Association* **84**, 733 (1989).

[30] C. Yang, R. Duraiswami, N.A. Gumerov and L. Davis, "Improved Fast Gauss Transform and Efficient Kernel Density Estimation", in *Proceedings of the Ninth IEEE International Conference on Computer Vision* (2003).

[31] D. Lee, A.G. Gray, and A.W. Moore, "Dual-Tree Fast Gauss Transforms", arXiv:1102.2878v1 (2011).

[32] J.S. Marron and M.P. Wand, "Exact Mean Integrated Squared Error", *The Annals Of Statistics* **20**, 712 (1992).

[33] I.S. Abramson, "On Bandwidth Variation in Kernel Estimates — A Square Root Law", *The Annals Of Statistics* **10**, 1217 (1982).

[34] P.D.A. Dassanayake, "Local Orthogonal Polynomial Expansion for Density Estimation", M.S. Thesis, Texas Tech University (2012).

[35] D.P.A. Dassanayake, I. Volobouev, and A.A. Trindade, "Local Orthogonal Polynomial Expansion for Density Estimation", *Journal of Nonparametric Statistics* **29**, 806 (2017).

[36] S.X. Chen, "Beta kernel estimators for density functions", *Computational Statistics & Data Analysis* **31**, 131 (1999).

[37] G.J. Babu, A.J. Canty, and Y.P. Chaubey, "Application of Bernstein Polynomials for smooth estimation of a distribution and density function", *Journal of Statistical Planning and Inference* **105**, 377 (2002).

[38] R. Sinkhorn and P. Knopp, "Concerning Nonnegative Matrices and Doubly Stochastic Matrices", *Pacific Journal of Mathematics* **21**, 343 (1967).

[39] R. Khoury, "Closest Matrices in the Space of Generalized Doubly Stochastic Matrices", *Journal of Mathematical Analysis and Applications* **222**, 562 (1998).

[40] LAPACK — Linear Algebra PACKage, http://www.netlib.org/lapack/

[41] T.J. Hastie, "Non-parametric Logistic Regression", SLAC PUB-3160, June 1983.

[42] CRAN - Package locfit, http://cran.r-project.org/web/packages/locfit/

[43] K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares", *Quarterly of Applied Mathematics* **2**, 164 (1944).

D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters", *SIAM Journal on Applied Mathematics* **11**, 431 (1963).

[44] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching", *Communications of the ACM* **18**, 509 (1975).

[45] H.J. Wang and L. Wang, "Locally Weighted Censored Quantile Regression", *Journal of the American Statistical Association* **104**, 487 (2009).

[46] F. Spanò, "Unfolding in particle physics: a window on solving inverse problems", *EPJ Web of Conferences* **55**, 03002 (2013).

[47] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, "Numerical Recipes: the Art of Scientific Computing", 3rd ed., Cambridge University Press (2007).

[48] A. Meister, "Deconvolution Problems in Nonparametric Statistics", Springer Lecture Notes in Statistics, Vol. 193 (2009).

[49] A. Höcker and V. Kartvelishvili, "SVD approach to data unfolding", *Nuclear Instruments and Methods in Physics Research A* **372**, 469 (1996).

[50] G. D'Agostini, "A multidimensional unfolding method based on Bayes' theorem", *Nuclear Instruments and Methods in Physics Research A* **362**, 487 (1995).

[51] D. Nychka, "Some Properties of Adding a Smoothing Step to the EM Algorithm", *Statistics & Probability Letters* **9**, 187 (1990).

[52] B.W. Silverman *et al.*, "A Smoothed EM Approach to Indirect Estimation Problems, with Particular Reference to Stereology and Emission Tomography", *Journal of the Royal Statistical Society B* **52**, 271 (1990).

[53] H. Akaike, "A new look at the statistical model identification", *IEEE Transactions on Automatic Control* **19**, 716 (1974).

[54] K.P. Burnham and D.R. Anderson, "Model Selection and Multi-Model Inference: A Practical Information-Theoretic Approach", 2nd ed., Springer (2004).

[55] Eigen — a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms, http://eigen.tuxfamily.org/

[56] K. Wu and H. Simon, "Thick-Restart Lanczos Method for Large Symmetric Eigenvalue Problems", *SIAM Journal on Matrix Analysis and Applications* **22**, 602 (2000).

[57] H. Niederreiter, "Random Number Generation and Quasi-Monte Carlo Methods", Society for Industrial and Applied Mathematics (SIAM), Philadelphia (1992).

[58] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", *ACM Transactions on Modeling and Computer Simulation* **8**, 3 (1998).

[59] I. Sobol and Y.L. Levitan, "The Production of Points Uniformly Distributed in a Multidimensional Cube", Tech. Rep. 40, Institute of Applied Mathematics, USSR Academy of Sciences, 1976 (in Russian).

[60] J. Dick, "Higher order scrambled digital nets achieve the optimal rate of the root mean square error for smooth integrands", arXiv:1007.0842 (2010).

[61] FFTW (Fastest Fourier Transform in the West), http://www.fftw.org/

Functions and Classes