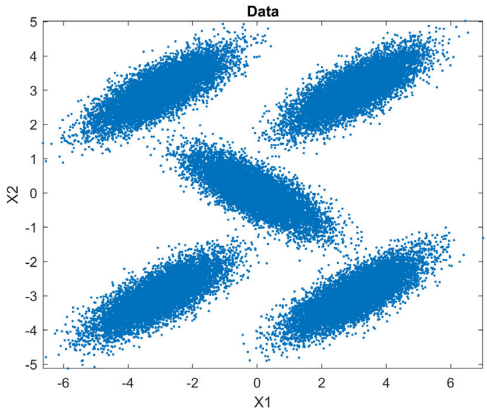
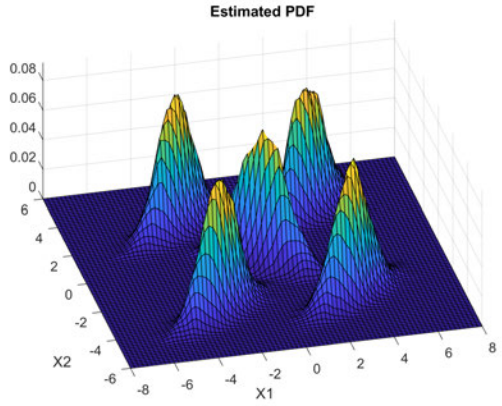
bsspdfest: A MATLAB toolbox for nonparametric probability function estimation using normalized B-splines Version 3.1.0





A manual prepared by

Biometrics Northwest LLC
6215 225th Avenue NE
Redmond, WA 98053
Email: info@biometricsnw.com

May 15, 2023

License

Abstract

A MATLAB toolbox implementing nonparametric probability density function estimation in one and several dimensions using normalized B-splines has been developed. The toolbox uses a series of normalized B-splines in 1-dimension and a series of tensor products of normalized B-splines for multiple dimensions to estimate the probability functions. (Gehringer, 1990, Gehringer and Redner, 1992, Redner and Gehringer, 1994, Redner, 1999, 2000).

The toolbox supports separate estimation and evaluation of the probability density function, the cumulative distribution function, the inverse cumulative distribution function, the survivor function, and the cumulative hazard function for one-dimensional data. For data having two or more dimensions the toolbox supports separate estimation and evaluation of the probability density function, the cumulative distribution function, and the survivor function. The estimation and evaluation operations are separate to support situations where a single estimation step is performed but a large number of evaluations is desired; there is no need to perform the estimation step on each evaluation of a probability function. In addition, a linear interpolant may be created for the probability density function and the cumulative distribution function to provide very fast approximate evaluation of the probability functions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The bsspdfest toolbox implements 1-, 2-, 3-, and N-dimensional nonparametric probability density function (PDF) estimation procedures using a B-spline series for one-dimensional data and a tensor product B-spline series for multi-dimensional data (Gehringer, 1990, Gehringer and Redner, 1992, Redner and Gehringer, 1994, Redner, 1999, 2000). The 1-, 2-, and 3-dimensional specific functions take advantage of the direct addressing of MATLAB arrays and various vectorization approaches to speed up the computations. Two functions, one for estimation (`bsspdfest`) and one for evaluation (`bsseval`), of a B-spline series representation for a PDF are provided. In addition, the `bsseval` function supports evaluation of the CDFs and survivor functions for all dimensions as well as the inverse cumulative distribution function (ICDF) and the cumulative hazard function for one dimensional data.

This User's Guide serves two purposes. First, it describes the use of the `bsspdfest` MATLAB toolbox implementing nonparametric probability density function estimation procedures for one and several dimensions. Second, it provides a demonstration of the correct usage of the toolbox through its use in a variety of examples. The data sets used in the examples are simulated and the code necessary to generate them is provided with the toolbox.

How to install and use the `bsspdfest` toolbox, its functions, and their inputs and outputs are described in Chapter 2. Examples demonstrating the use of the `bsspdfest` toolbox are provided in Chapter 3. Changes and improvements to the `bsspdfest` package are briefly described next in Section 1.1.

1.1 What's New

Version 2.4.0 and later

See the file `changes.txt`.

Version 2.3.1

- Fixed a typo that was not caught in other testing for some reason. Should have done one more test after packaging.

Version 2.3.0

- Changed the method for dealing with boundary corrections from using the histogram value to reflection through any active boundaries. In addition, for 2- and 3-dimensional data the reflections are also performed for corners and edges where multiple active boundaries intersect.
- Added boundary corrections for N-dimensional data via reflection through the active boundaries. No additional corrections are made for multiple intersecting active boundaries.
- Fixed an off by 1 error in bsspdfestnd(). The error produced linear index values that were out of bounds when performing computations with an active boundary. When there was no active boundary this error would have caused a shift in the coefficients for each dimension since the number of partition points was used not the number of partition intervals.
- Fixed a bug in the extrapolation of the values for the 3-dimensional CDF in do_cdf. There was an incorrect test that caused the extrapolated values to be zero in one of several extrapolation cases.
- Changed calls to repmat() to calls to bsxfun() to improve performance and reduce memory requirements for intermediate results.

- Changed the way the coefficient matrix was updated from a point by point summation to a chunked summation based on a sorted list of the unique indices into the coefficient matrix. This provided a significant speed improvement for 2- and 3-dimensional estimation functions.
- Rearranged the computations for the 2- and 3-dimensional estimation functions to further improve performance.
- Moved the bssdeleteinterpolant() function out of the private folder so that it could be used.
- Cleaned up the comments for several functions and fixed typos in several error messages.
- Fixed a bug in bsscreatestruct() where an unassigned variable was passed to bsscheckmemory(). The variable name used was 'ngrid' which is now a MATLAB function.
- Changed the variable name 'ngrid' in bsscreateinterpolant to 'tmpngrid' to avoid any potential problems with the MATLAB function ngrid.
- Added tests to see if the storage requirements were less than intmax('uint32') bytes. If so, everything is assumed to be OK and bsscheckmemory() just returns. Otherwise check the memory. The call to memory() was taking a huge amount of time.
- Added a new private function bssuniqueidx() to compute the unique index values, break points, and an order vector.
- Updated the User's Guide for the new boundary correction method, fixed typos, misspellings, etc. Several new examples were also added.

Version 2.2.0

- Updated the Users Guide to add an examples section demonstrating the estimation for bounded domains. Removed the N-dimensional examples; it is essentially identical to the 3-d example.
- Added a check for active bounds on the estimation interval. This is detected by checking to see if any of the B-spline series coefficients that overlap a boundary are greater than zero. If so, a histogram estimate is generated for the bin(s) adjacent to the boundary and used to specify the PDF value at the boundary. The B-spline series coefficients are then modified to produce this value at the boundary.

- Data points that are not in the estimation interval are now ignored. Previously they were evaluated, but always returned zero values, that would stack up on the boundaries. This caused several unnecessary complications when computing boundary values, leading to the change in behavior.
- Changed values of the multipliers in the bsspartitionsize() function that computes the default partition sizes to improve the visual smoothness of continuous distributions slightly. There are still too many partition intervals, but erring on the side of undersmoothing is the right way to go.
- If the number of inbounds points to one of the bsseval*() functions was zero an error occurred. The output values should all have had a value of zero. If all points are out of bounds zero values are returned.

Version 2.1.1

- Fixed a bug when checking the B-spline order. The bsspartitionsizeok() function was called with the partition size by mistake. It should have been the bssorderok() function with the B-spline order.
- Fixed a bug that occurred when a single 1-dimensional evaluation point was given to bsseval and the B-spline series order was greater than one. This caused a row vector of index values to be used to index a column vector producing a column vector, not a row vector, due to the 1-dimensional indexing semantics of MATLAB.
- Consolidated the input checking for all of the bsspdfest functions.
- Removed the 3-dimensional mixture distribution movie files to reduce the overall package size. The movies may be found at the following web page Mixture density 3-dimensional visualizations.

Version 2.1.0

- Updated the User's Guide to reflect changes in the testing and example scripts and their versions that do not use the MATLAB Statistics and Machine Learning Toolbox.
- Added a new test script `test_bsspdfest_nostats`. This script is identical to the test script `test_bsspdfest` except for not requiring the MATLAB Statistics and Machine Learning Toolbox.

- Added two support functions `normpdftestbss` and `mvnpdftestbss` for the `_nostats` version the test script. The functions are simple implementations the evaluate the normal PDF and multivariate normal PDF for testing the `bsspdfest` toolbox.
- Deleted references to the example script `mk_example_figs_nostats` from the User's Guide. This file is no longer provided with the `bsspdfest` Toolbox.

Version 2.0.0

- Algorithmic improvements were made to enhance the performance of the `bsspdfest` Toolbox.
- Replaced outer products with vectors of ones used to replicate data with calls to REPMAT. Since REPMAT is now a built-in function it is significantly faster than using the outer product to do replication.
- Removed unnecessary reshaping for B-splines of order 1.
- Changed the matrix orientation from m by n to n by m, where m is the B-spline order and n is the total number of evaluation points specified by the input X. This removed several transposes of potentially large matrices.
- Removed unnecessary temporary variables.
- Explicitly created indexes and B-spline evaluation matrices for each dimension in the 2-D and 3-D specific functions eliminating looping over the number of dimensions.
- Added the ability to compute probability based functions in addition to the PDF: the CDFs and survivor functions in all dimensions as well as the inverse CDFs and cumulative hazard function in one dimension.
- The ability to generate a gridded interpolant for linear interpolation and extrapolation of the PDF and CDFs has also been added. When the underlying probability functions are to be estimated once and evaluated many times the gridded interpolant can be significantly faster than the B-spline evaluation procedures for B-spline orders greater than 1.
- For the 1-D estimation, sorted the partition indexes derived from the input data, computing a permutation vector to the data and partition break points to speed up the computations. This replaced using a logical index of values that was computed for each partition sequentially to identify the data points it contained.

- Added a `private` folder to isolate the supporting functions from the two main functions `bsspdfest` and `bsseval`.

Version 1.1.0

Fixed an issue with the license file. No other changes.

Version 1.0.0

The initial release of the bsspdfest package.

Chapter 2

Using the bsspdfest toolbox

This chapter describes how to install and use the bsspdfest MATLAB toolbox. The MATLAB functions and their arguments are described, as well as test scripts and a script that was used to generate the figures in this user manual.

2.1 Installing bsspdfest

To install the bsspdfest toolbox, uncompress the file bsspdfest.zip into a folder called bsspdfest in the local MATLAB toolbox folder and add this folder to the MATLAB path. Type 'test_bsspdfest' at the MATLAB prompt to then test the bsspdfest toolbox.

Two versions of the test script are included with the toolbox: a version that is independent of the MATLAB Statistics and Machine Learning Toolbox having a file name ending in _nostats, and a version that uses the functions normpdf, normcdf, mvnpdf, mvncdf, and cholcov from the MATLAB Statistics and Machine Learning Toolbox. The _nostats version of the test script use simple implementations of functions that evaluate the normal PDF and multivariate normal PDF for testing the bsspdfest toolbox, normpdftestbss and mvnpdftestbss, and the Cholesky function chol.

2.2 Using bsspdfest

Using the bsspdfest toolbox to estimate a PDF from data and evaluate it for a set of points consists of two steps. The first step is the estimation step that creates a B-spline series data structure representing the nonparametric probability density function from a data matrix whose rows represent the data points. This step is accomplished by using the function bsspdfest. The second step uses a B-spline series data structure created in an estimation step with a set of evaluation points, again as rows in a matrix, to evaluate the PDF using bsseval. As a simple example, the following MATLAB code generates 1000 data points from a 2-dimensional standard normal distribution, estimates the PDF from the data, evaluates the estimated function at the data points, and plots the likelihood values for each data point as Figure 2.1.

```
% Generate data from a 2-D standard normal distribution

data = randn(1000,2);

% Estimate a pdf from data

bss1 = bsspdfest(data);

% Evaluate the estimated PDF at the data points to get
% their likelihood values

lh = bsseval(data,bss1);

% Plot the likelihood values in 3-D to show the
% approximation to the PDF.

figure;
plot3(data(:,1),data(:,2),lh,'+');
grid
xlabel('X1')
ylabel('X2')
zlabel('Likelihood')
```

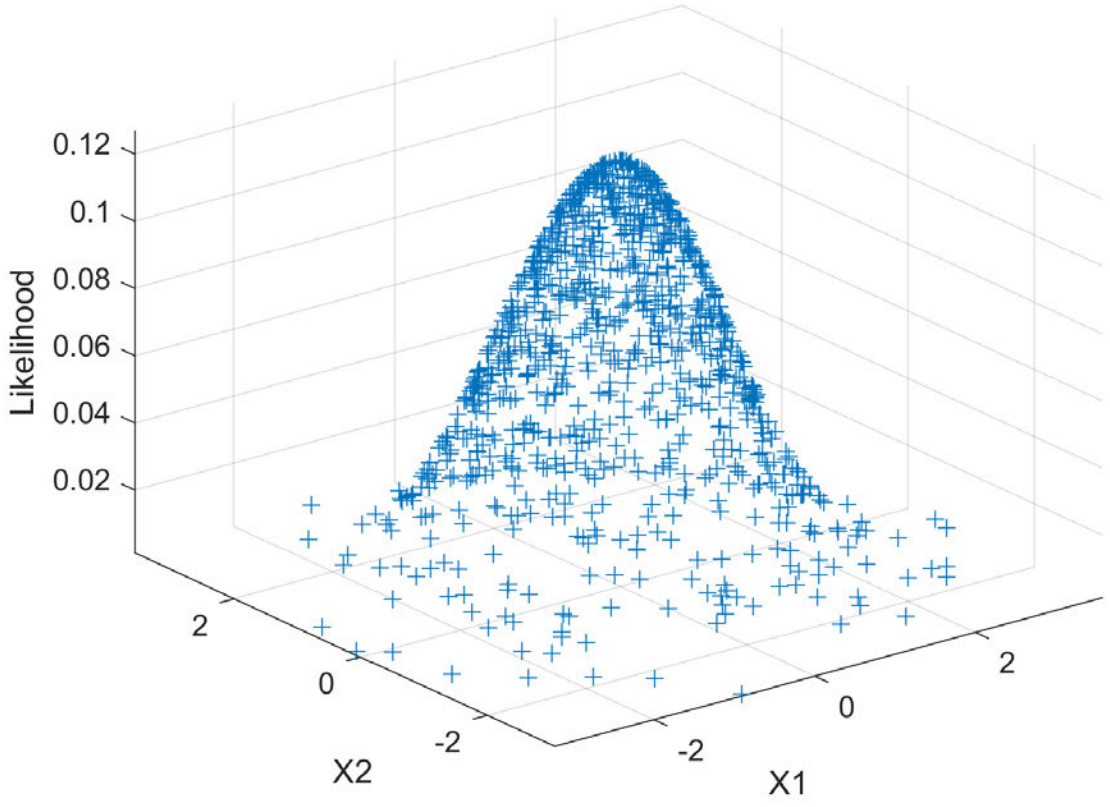


Figure 2.1: A simple example using the `bsspdfest` toolbox to estimate a 2-dimensional standard normal distribution and plot likelihood values for the data points.

2.3 bsspdfest toolbox functions

The MATLAB functions implementing the bsspdfest toolbox, their inputs, and their outputs are now described. The functions used to estimate and evaluate a B-spline series representation of a PDF are described in Section 2.3.1 and the scripts used to generate example figures using the bsspdfest toolbox are described in Section 2.3.2.

2.3.1 Probability density function estimation and evaluation

A nonparametric estimate of a PDF is computed from data using the function bsspdfest.

`bss = bsspdfest(x, bnds, n, m, bss)` Estimate a probability density function using a B-spline series.

INPUTS:

x The data to be used for the B-spline density estimation. This must be a column oriented array with each row representing a data point, X(NDATA,NDIM) in size. Values of X outside the estimation bounds are ignored.

bnds The lower and upper bounds of the estimation interval for each dimension. This is an NDIM by 2 matrix, specifying the minimum and maximum values used to create the B-spline series partition for each dimension. If empty or not present, the estimation bounds are generated from the data.

If any of the bounds are active, a boundary correction is made using reflection through each active boundary. For 2- and 3-dimensional data reflection based corrections are also made for corners and edges where multiple boundaries are active.

n The nominal partition size or the number of subintervals to use for the partition in each dimension. The partition size for each dimension does NOT include the extra subintervals that are created based on the order of the B-splines that are used. This is a row or column vector with NDIM elements. NDIM must be ≥ 1. If empty or not present, a default value based on the number of data points is used for each dimension.

m The order of the B-spline basis functions to be used. This must be an integer scalar value greater than or equal to one ($M \geq 1$). If not present or empty a default value of $m = 4$ is used to generate a cubic approximation.

bss An existing B-spline series structure that is to have more data points added to it. This allows one to build a B-spline density estimate a little bit at a time. If this argument exists, then BNDS, N, and M are ignored and the values in BSS are used. Empty values `[]` should be used as place holders for BNDS, N, and M when BSS is used.

OUTPUTS:

bss A B-spline series data structure with the following fields.

m The order of the B-spline basis functions. This is a scalar value that is the same for all dimensions.

ndim The number of dimensions for the B-spline series.

npts The number of data points used to compute the coefficients.

bnds The estimation bounds for each dimension. The endpoints do not take into account the extra partition subintervals that are necessary if the B-spline order is greater than one.

partition The partition structure containing fields X, N, NPART, and H. The fields represent the partition of the estimation interval for each dimension, with X defining the boundary points for the partition subintervals, N defining the nominal partition size, NPART defining the number of boundary points including any extra boundary points needed for $M > 1$, and H defining the width of the subintervals for each dimension.

c The coefficient matrix for the B-spline series

The function `bsseval` is used to evaluate a B-spline series representation of a function using a B-spline series data structure obtained from `bsspdfest`. The only function type currently supported is a nonparametric estimate of a PDF and associated probability functions including: the CDFs and survivor functions for all dimensions and the ICDF and the cumulative hazard function for one dimensional data. Extensions to permit the nonparametric estimation of arbitrary functions are planned for a future version.

`[y,bssout] = bsseval(x,bss,evalfun,zerotol)` Evaluate a B-spline series.

INPUTS:

x The data to be used for evaluating the B-spline series. This is an NPOINTS by NDIM matrix with the rows representing the points. This value may be empty

if a gridded interpolant is being created. In this situation the output Y will be assigned an empty value. This feature allows the creation of a gridded interpolant without the need to evaluate it.

bss The B-spline series data structure containing the series to be used for evaluation.

evalfun The function that is to be evaluated.

If the B-spline series represents a probability density function, the following output functions are available 'pdf', 'cdf', 'icdf', 'survivor', 'cumhazard'. If no evaluation function is specified the B-spline series is simply evaluated without transformations. An evaluation function value of 'bss' (default) is also allowed, and indicates that the B-spline series is simply to be evaluated. If the B-spline series represents a PDF then using 'bss' is equivalent to using 'pdf'.

Allowed evaluation functions are:

bss Evaluate the B-spline series (default).

pdf Evaluate the probability density function, $f(x)$.

cdf Evaluate the cumulative distribution function, $F(x)$.

icdf Evaluate the inverse cumulative distribution function, $F^{-1}(p)$.

survivor Evaluate the survivor function, $S(x) = 1 - F(x)$.

cumhazard Evaluate the cumulative hazard function, $-\log(1 - F(x))$ or equivalently $-\log(S(x))$.

The evalution functions 'icdf' and 'cumhazard' are only available for 1-dimensional distributions. The 'pdf','cdf', and 'survivor' functions are available for all dimensions.

If the B-spline series does not represent a probability density function, then the output values cannot be interpreted as being from one of the probability based functions.

zerotol A zero tolerance for finding flat spots in the CDFs when computing the ICDF. It is also used to determine a minimal support interval for the CDFs for interpolation of the ICDF. If a value is not supplied a default value of sqrt(eps) is used. A value of zero is not recommended. A nearly exact zero tolerance may be specified by using the string value 'exact', indicating that a zero tolerance of 2048*eps will be used. The string value 'default' will use the default value of sqrt(eps). Large values of the zero tolerance may give inaccurate or incorrect results.

The zero tolerance is only used for the 1-dimensional 'icdf' evaluation function.

OUTPUTS:

y The values for the B-spline series evaluated at the points defined by X.

bssout If present, this output argument causes a gridded interpolant to be produced and stored in the output B-spline series data structure. The gridded interpolant may be used for fast approximate evaluation of the function represented by the B-spline series.

Three gridded interpolant functions are allowed depending on the evaluation function:

bss The B-spline series. This is the default if no evaluation function is specified.

pdf The probability density function.

cdf The cumulative distribution function.

Using any other evaluation function with this output argument present is an error.

2.3.2 Example scripts and supporting functions

Brief descriptions of the example script, the test scripts, and supporting functions are provided below.

`test_bsspdfest` Test script for the `bsspdfest` toolbox that uses the functions `normpdf`, `normcdf`, `mvnpdf`, `mvncdf`, and `cholcov` from the MATLAB Statistics and Machine Learning Toolbox.

`test_bsspdfest_nostats` Test script for the `bsspdfest` toolbox that does not use uses the functions `normpdf`, `normcdf`, `mvnpdf`, `mvncdf`, and `cholcov` from the MATLAB Statistics and Machine Learning Toolbox.

`mk_example_figs.m` Script used to generate the example figures in this document. This script requires the Statistics and Machine Learning Toolbox due to its use of the functions `normpdf`, `normcdf`, `mvnpdf`, `mvncdf`, and `cholcov`.

`y = normpdftestbss(x,mu,sigma)` Evaluate the one-dimensional normal distribution at the values in X using mean MU and standard deviation SIGMA. This function is used in the example script ending in `_nostats` to eliminate the dependency on the Statistics Toolbox functions `normpdf` and `normcdf`. This function does no error checking.

INPUTS:

x A column vector of points at which to evaluate the one-dimensional normal PDF.

mu The mean value for the one-dimensional normal distribution. If not present a value of zero (0) is used.

sigma The standard deviation for the one-dimensional normal distribution. If not present a value of one (1) is used.

OUTPUTS:

y The values of the one-dimensional normal distribution evaluated at the points in X.

`y = mvnpdftestbss(x,mu,sigma)` Evaluate the multivariate normal distribution at the values in X using mean vector MU and covariance matrix SIGMA. This function is used in the test scripts ending in `_nostats` to eliminate the dependency on the Statistics Toolbox functions `mvnpdf` and `mvncdf`. This function does no error checking.

INPUTS:

x The points at which the multivariate normal distribution PDF is to be evaluated. This is an NPOINTS by NDIM matrix with the rows representing the points.

mu The mean vector for the multivariate normal distribution. If not present the NDIM-dimensional zero vector is used.

sigma The covariance matrix for the multivariate normal distribution. If not present an NDIM by NDIM matrix with ones along the diagonal is used.

OUTPUTS:

y The values of the multivariate normal distribution evaluated at the points in X.

2.4 Errors and error messages

The `bsspdfest` toolbox functions detect a variety of potential problems, reporting any errors that are detected using the MATLAB `error` function. All errors that are detected by the `bsspdfest` toolbox are considered to be fatal errors, that is, they halt execution. If an error occurs, an error message is generated and displayed in the MATLAB window. The `bsspdfest` toolbox functions also produce several warning messages using the MATLAB `warning` function to provide feedback in situations that may produce questionable results.

2.5 Bug reporting

In the event that a bug is discovered when using the `bsspdfest` toolbox, a problem or bug report may be sent to `bugs@biometricsnw.com` with a complete description of the problem. Before sending the bug or problem report, please use the following short check-list to help prepare the bug or problem report. This will enable a more rapid resolution of the problem.

1. Be sure that the problem to be reported is real and not simply an erroneous input value or some other simple problem that is not actually a bug or defect in the software.
2. Attempt to fully and concisely document the circumstances under which the problem occurred. Be sure to include the following items in any message reporting a possible bug.
   - Your name and contact information, email address or phone number in particular.
   - The name and version of the software that was used.
   - The specific commands that were used.
   - All input or output files that were used when the problem occurred.
   - Error messages, if any, reported by the software or MATLAB when the problem occurred.
3. The subject line of the email should clearly indicate that the message is a bug report or problem report for the `bsspdfest` MATLAB toolbox.

Contact Biometrics Northwest LLC at the address below to request additional information or to report problems and potential bugs.

Biometrics Northwest LLC
6215 225th Avenue NE
Redmond, WA 98053

email:
Bug reports: `bugs@biometricsnw.com`
Information requests: `info@biometricsnw.com`

Home Page: `http://www.biometricsnw.com`

Chapter 3

Examples

The `bsspdfest` toolbox is demonstrated in this chapter for a variety of probability distributions in one and several dimensions. For each example the area or volume under the PDF is computed for the estimated PDF to verify that it is a probability density function, up to round-off error. Differences between a true PDF and an estimated PDF are characterized using the integrated absolute error (IAE), the root mean squared error (RMSE), and the integrated RMSE (IRMSE). A summary of these basic goodness of fit results is provided at the end of the examples in Table 3.1. In addition to the PDF, the other available probability functions will also be demonstrated for each dimension. Aside from specifying the probability function to evaluate, default values for `bsspdfest` and `bsseval` parameters were used in all examples, except where indicated.

In probability density estimation problems the dimensionality of the problem is determined by the domain of the function, as represented by the data, rather than the combined dimensionality of the range and the domain. So, a one-dimensional PDF estimation problem produces a function in two-dimensions, a two-dimensional problem produces a function in three-dimensions, or a surface, etc. Examples for 1-dimensional problems are presented in Section 3.1, and examples for 2-dimensional and 3-dimensional problems, respectively, in Section 3.2 and Section 3.3. Examples with bounded domains are presented in Section 3.4. Finally, examples using a gridded interpolant for execution speed are presented in Section 3.5. For each example, the MATLAB code used to generate the simulated data sets, the evaluation points, and to perform the probability function estimation and evaluation is included, except for the code used to generate figures. The code for figures was not included to save space, but it may be found in the file `mk_example_figs.m` that generates all of the figures found here, as well as several others.

Table 3.1: Summary of PDF estimation results for all examples. D is the data dimension, N_{data} is the number of sample points, N_{eval} is the number of evaluation points. Areas and volumes were computed using a straightforward numerical integration approach: summing the PDF values and multiplying by the evaluation point spacing in one dimension and the product of the spacings for multiple dimensions.

Distribution	D	N_{data}	N_{eval}	Area/Vol.	IAE	RMSE	IRMSE
$N(0,1)$	1	1000	501	1.000000	0.078149	0.010806	0.035875
$\frac{1}{3}N(0,0.25)+\frac{2}{3}N(3,0.5)$	1	1500	501	1.000000	0.217739	0.043800	0.124010
$\frac{1}{3}N(0,0.25)+\frac{2}{3}N(3,0.5)$	1	7500	501	1.000000	0.088370	0.020792	0.058867
$N(0,I)$	2	5000	6561	1.000001	0.100253	0.021521	0.026632
$N(0,I)$	2	10000	6561	0.999999	0.054992	0.012874	0.015932
$\frac{1}{5}\sum_{i=1}^{5}N(\mu_i,\Sigma_i)$	2	50000	6561	0.999992	0.080439	0.010549	0.016444
$N(0,I)$	3	25000	1771561	1.000000	0.084037	0.037176	0.011349
$\frac{1}{5}\sum_{i=1}^{5}N(\mu_i,\Sigma_i)$	3	25000	3442951	0.999996	0.149321	0.018328	0.010965
$\frac{1}{5}\sum_{i=1}^{5}N(\mu_i,\Sigma_i)$	3	50000	3442951	0.999992	0.132899	0.016711	0.009998
$N(0,I)$	4	100000	6765201	0.999997	0.061645	0.010933	0.003779

The number of B-spline basis functions that should be used is a function of both the support or truncated support in the domain for the underlying true distribution, as represented by the extent of the data, and the desired evaluation region, or domain, for the estimated PDF. For a histogram, a B-spline of order $m = 1$, the optimal number of basis functions or bins is proportional to the cube root of the number of data points, $n^{\frac{1}{3}}$, and twice this value, rounded to the nearest integer, for each dimension typically provides a good starting point in practice. Some adjustment of the partition size is to be expected to obtain the desired smoothness of the result. In addition, for bounded domains the partition size may need to be smaller than the default size, making the partition subintervals larger, for the boundary corrections to be effective. An example demonstrating this may be found in Section 3.4.2.

3.1 One-dimensional examples

3.1.1 Standard normal distribution

For the first example, $n = 1000$ data points are generated from a standard normal distribution $N(0, 1)$ and used to compute estimates of the available probability functions: the PDF, the CDFs, the survivor function, the ICDF, and the cumulative hazard function. For each

function the MATLAB code used to generate the data and perform the probability function estimation is provided. The evaluation points and the PDF estimate `bss1test` are used for each of the probability functions. Results are plotted for each function comparing the estimated function to the actual function along with the estimation error.

Generate the data and evaluation points.

```
nsample  = 1000;
evalbnds = [-6 6];
xx1      = linspace(evalbnds(1),evalbnds(2),501)';
xtest1   = randn(nsample,1);
```

Compute the PDF estimate, evaluate it, and compute the true standard normal PDF.

```
bss1test = bsspdfest(xtest1);
y1       = bsseval(xx1,bss1test,'pdf');
yn       = normpdf(xx1,0,1);
```

Evaluate the CDFsand compute the true standard normal CDFs.

```
y1 = bsseval(xx1,bss1test,'cdf');
yn = normcdf(xx1,0,1);
```

Evaluate the survivor function and compute the true standard normal survivor function.

```
y1 = bsseval(xx1,bss1test,'survivor');
yn = 1-normcdf(xx1,0,1);
```

Evaluate the cumulative hazard function and compute the true standard normal cumulative hazard function.

```
y1       = bsseval(xx1,bss1test,'cumhazard');
yn       = 1-normcdf(xx1,0,1);
ynch     = 1-normcdf(xx1,0,1);
gt0      = yn>0;
yn(~gt0) = nan;
yn(gt0)  = -log(yn(gt0));
```

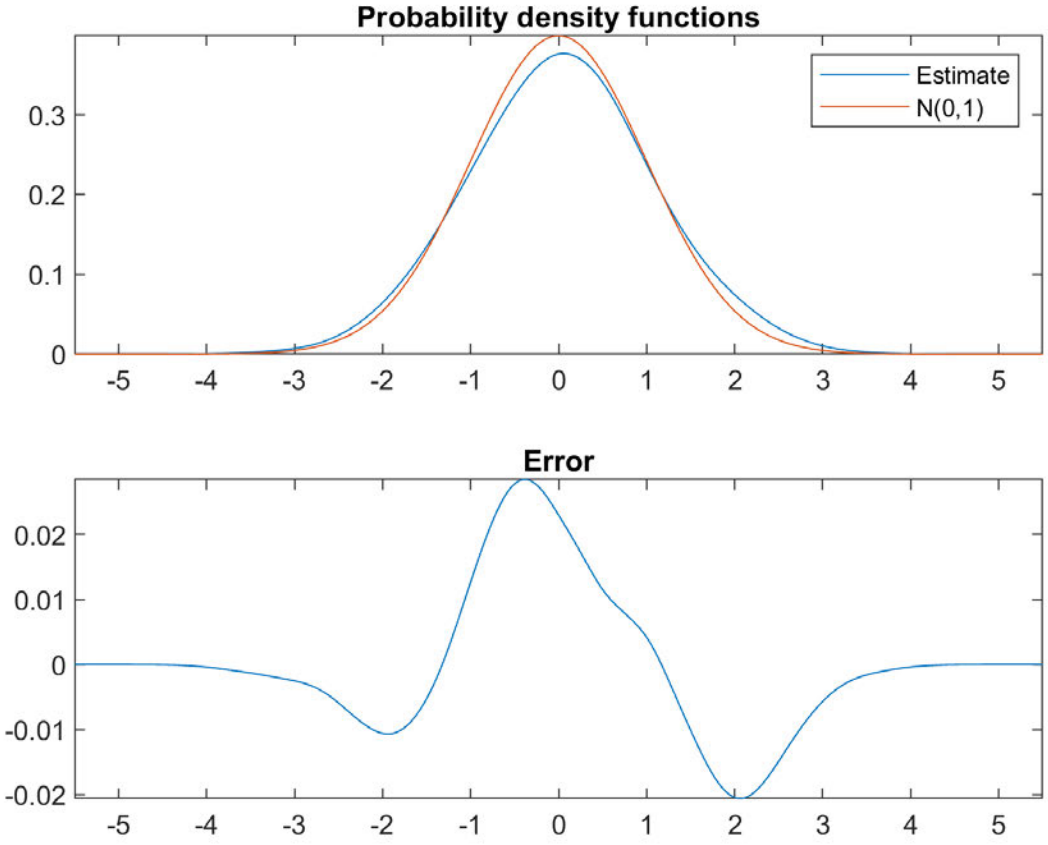


Figure 3.1: Estimate of a 1-dimensional standard normal distribution using $n = 1000$ points. Estimated and true PDF (top) and error (bottom).

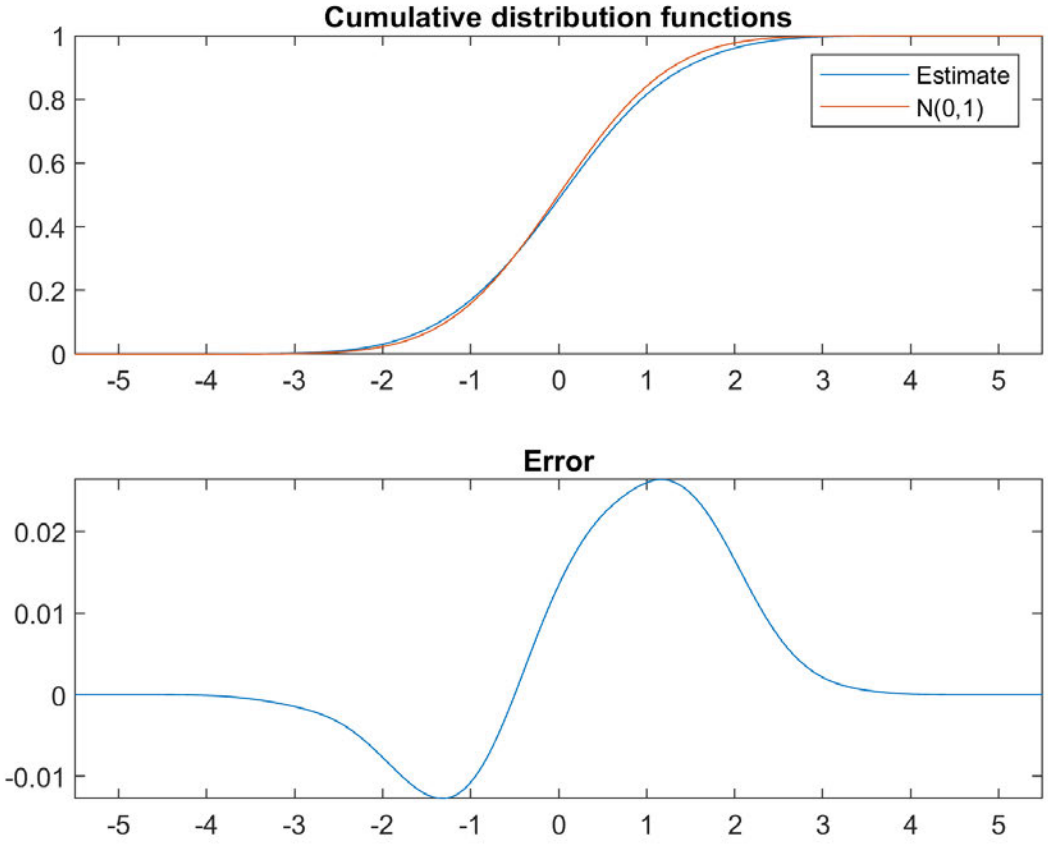


Figure 3.2: Estimate of a 1-dimensional standard normal cumulative distribution function using $n = 1000$ points. Estimated and true CDFs (top) and error (bottom).

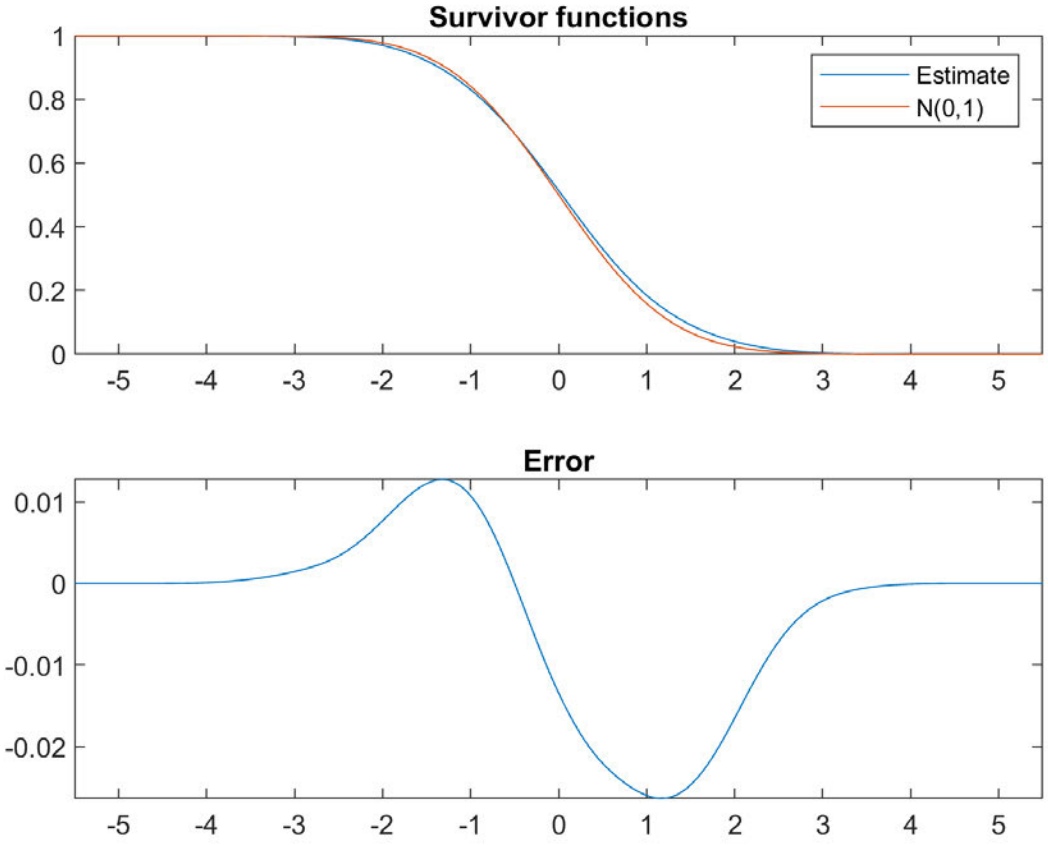


Figure 3.3: Estimate of a 1-dimensional standard normal survivor function using $n = 1000$ points. Estimated and true survivor functions (top) and error (bottom).

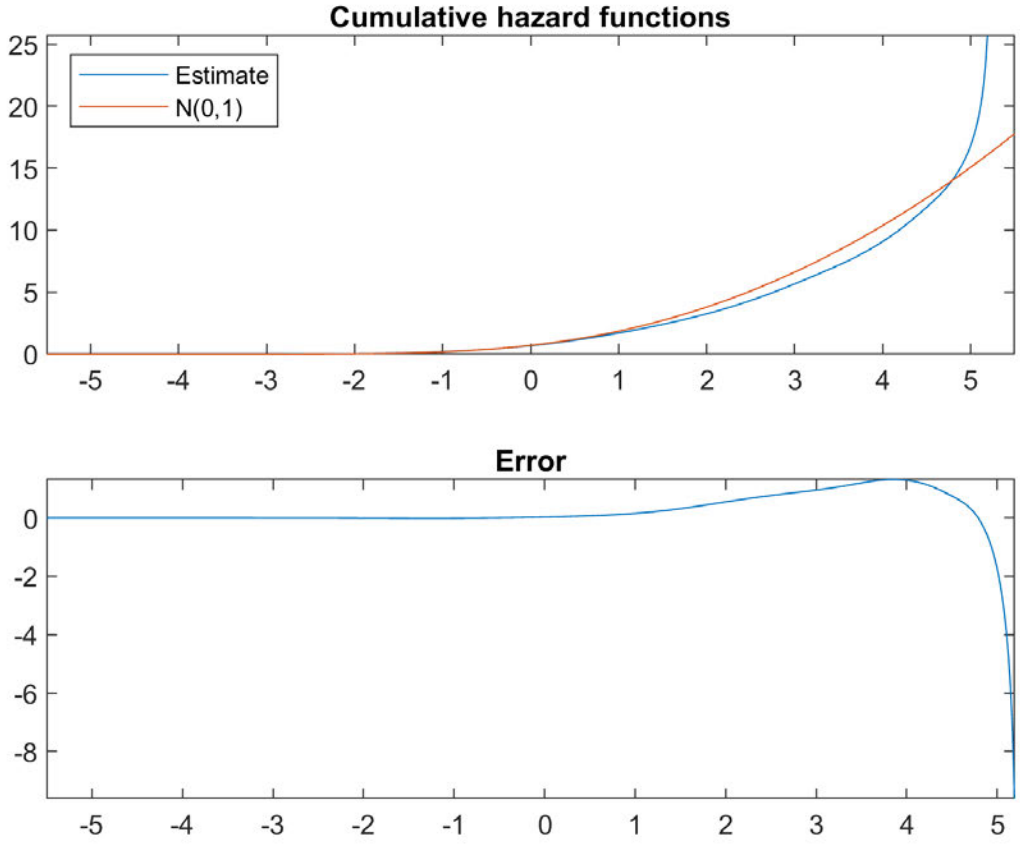


Figure 3.4: Estimate of a 1-dimensional cumulative hazard function using $n = 1000$ points. Estimated and true cumulative hazard functions (top) and error (bottom).

Evaluate the ICDFand compute the true standard normal ICDF.

```
xx1p = linspace(0,1,501)';
y1   = bsseval(xx1p,bss1test,'icdf');
yn   = 1-normcdf(xx1,0,1);
ynch = 1-norminv(xx1p,0,1);
```

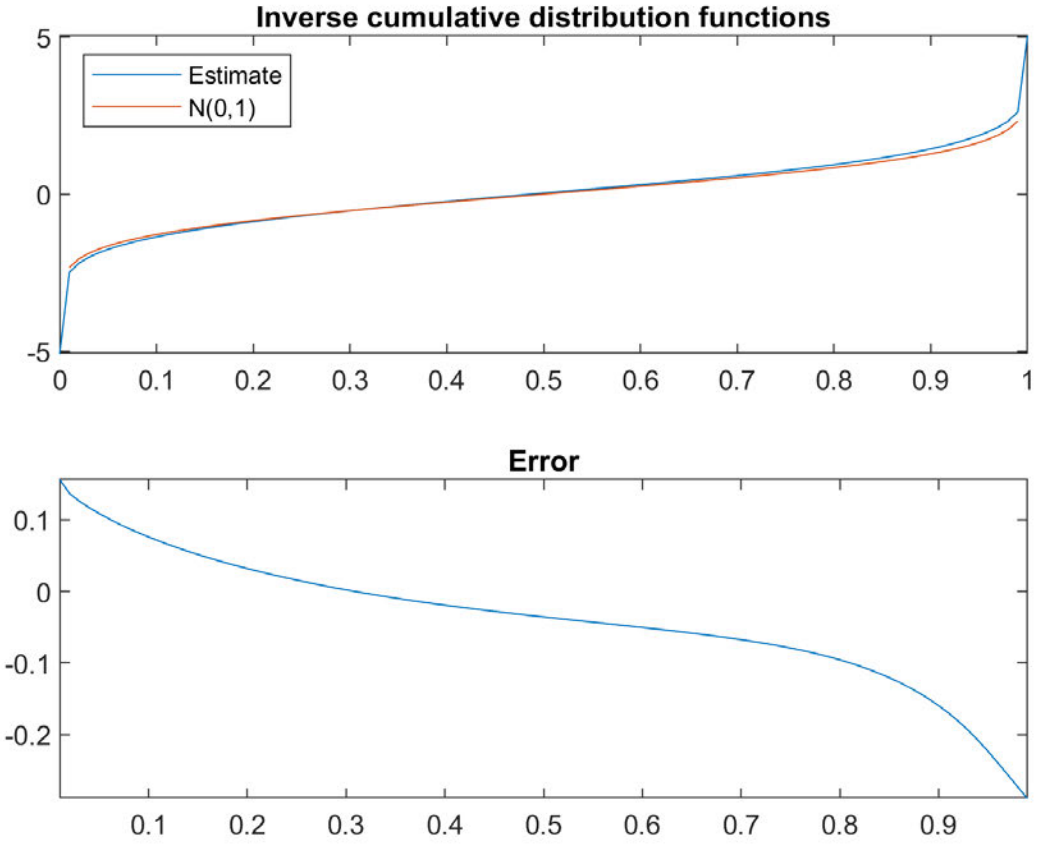


Figure 3.5: Estimate of a 1-dimensional standard normal ICDFusing $n = 1000$ points. Estimated and true ICDF(top) and error (bottom).

3.1.2 Mixture of 2 normal distributions

For this example, $n = 1500$ data points are generated from an unequal mixture of two normal distribution $\frac{1}{3}N(0, 0.25) + \frac{2}{3}N(3, 0.5)$ and used compute estimates of the available probability functions: the PDF, the CDFs, the survivor function, the ICDF, and the cumulative hazard function. For each function the MATLAB code used to generate the data and perform the probability function estimation is provided. The evaluation points and the PDF estimate `bss1test` are used for each of the probability functions. Results are plotted for each function comparing the estimated function to the actual function with the estimation error.

Generate the data and evaluation points.

```
nsample  = 500;
ndata    = 3*nsample;
evalbnds = [-2 6];
xx1      = linspace(evalbnds(1),evalbnds(2),501)';
xtest1   = [0.25*randn(nsample,1); 3+0.5*randn(2*nsample,1)];
```

Compute the PDF estimate, evaluate it, and compute the true mixture PDF.

```
bss1test = bsspdfest(xtest1);
y1       = bsseval(xx1,bss1test,'pdf');
ynmix    = (1/3)*normpdf(xx1,0,0.25) + (2/3)*normpdf(xx1,3,0.5);
```

Evaluate the CDFs and compute the true mixture CDFs.

```
y1    = bsseval(xx1,bss1test,'cdf');
ynmix = (1/3)*normcdf(xx1,0,0.25) + (2/3)*normcdf(xx1,3,0.5);
```

Evaluate the survivor function and compute the true mixture survivor function.

```
y1    = bsseval(xx1,bss1test,'survivor');
ynmix = 1-((1/3)*normcdf(xx1,0,0.25) + (2/3)*normcdf(xx1,3,0.5));
```

Evaluate the cumulative hazard function and compute the true mixture cumulative hazard function.

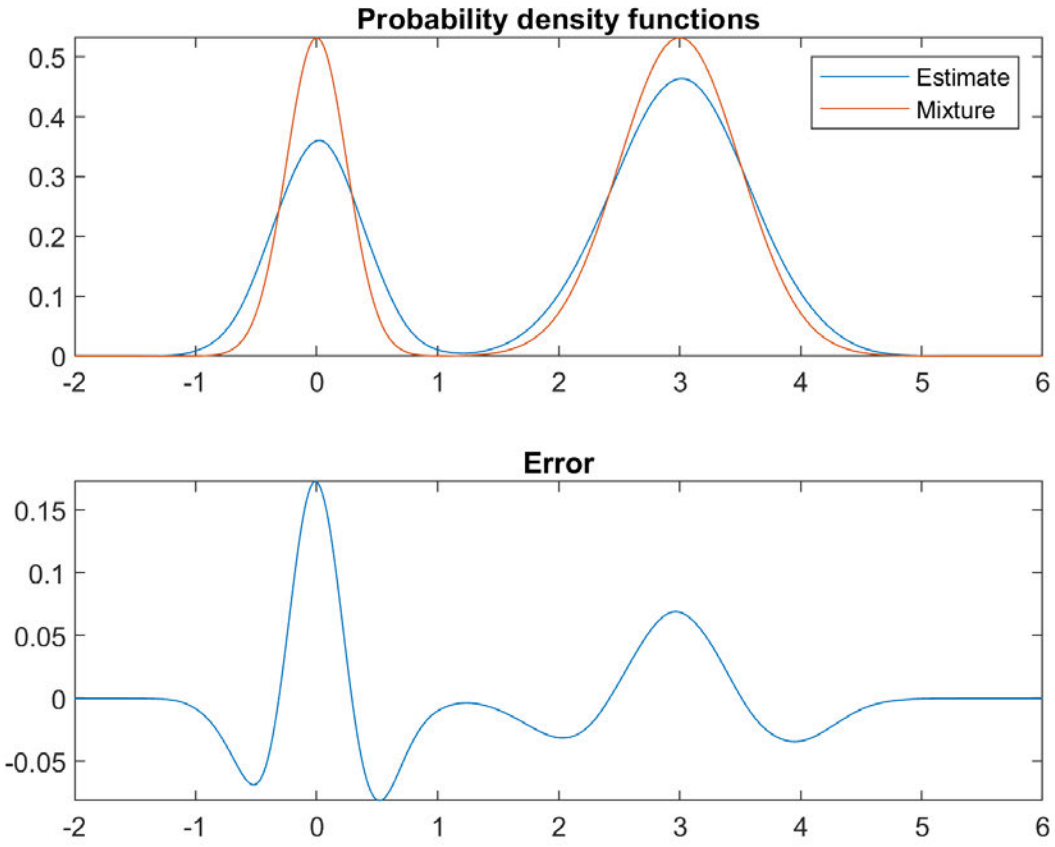


Figure 3.6: Estimating a 1-dimensional mixture of two normal distributions using $n = 1500$ data points. Estimated and true PDF (top) and error (bottom).

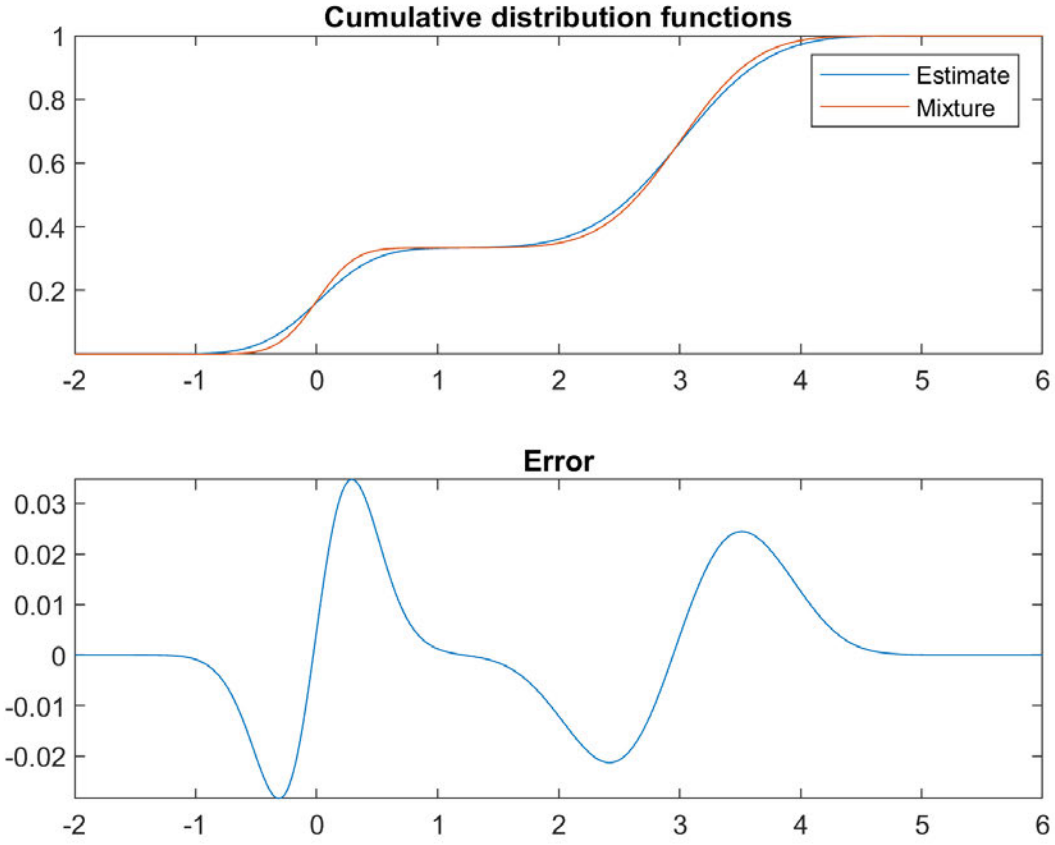


Figure 3.7: Estimate of the cumulative distribution function for a 1-dimensional mixture of two normal distributions. Estimated and true CDFs (top) and error (bottom).

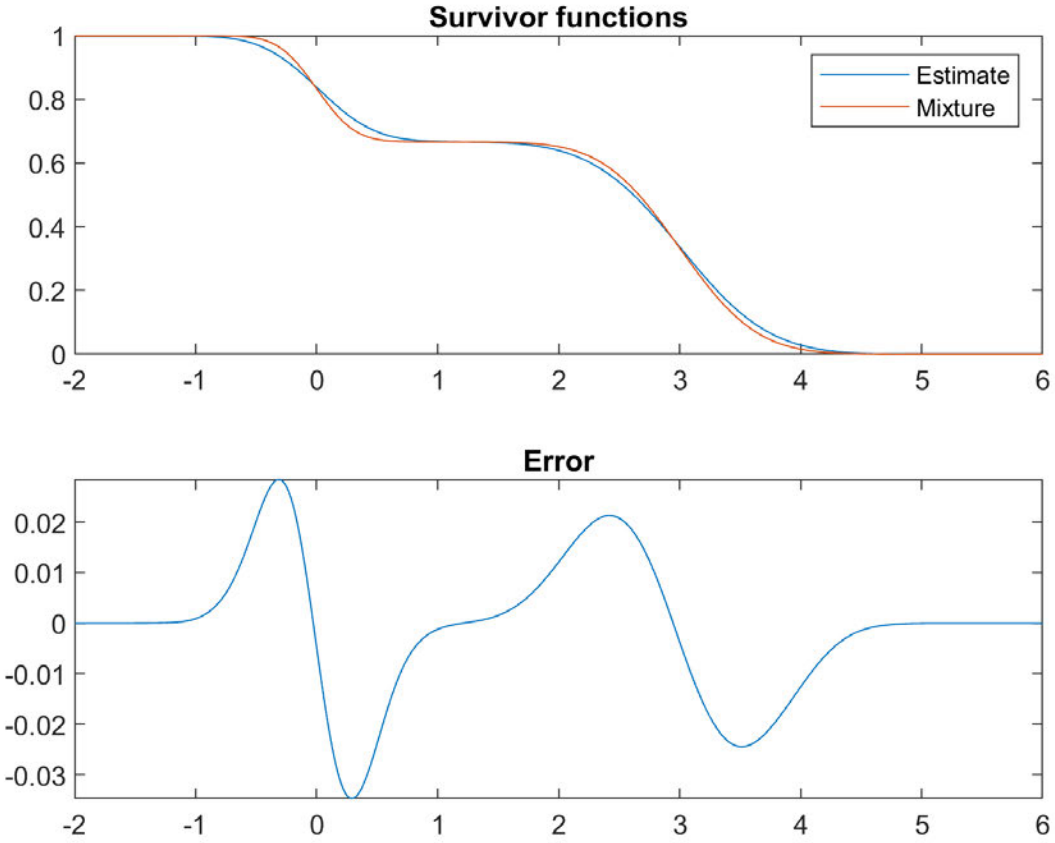


Figure 3.8: Estimate of the survivor function for a 1-dimensional mixture of two normal distributions. Estimated and true survivor functions (top) and error (bottom).

```
y1    = bsseval(xx1,bss1test,'cumhazard');
ynmix =  1-((1/3)*normcdf(xx1,0,0.25) + (2/3)*normcdf(xx1,3,0.5));
gt0   = ynmix>0;
ynmix(~gt0) = nan;
ynmix(gt0)  = -log(ynmix(gt0));
```

Evaluate the ICDF and compute the true mixture ICDF.

```
xx1p    = linspace(0,1,501)';
y1      = bsseval(xx1p,bss1test,'icdf');
xx      = linspace(-2,6,1001)';
ympdf   = (1/3)*normpdf(xx,0,0.25) + (2/3)*normpdf(xx,3,0.5);
ymcdf   = cumtrapz(xx,ympdf);
ztol    = 2*max([min(ymcdf) min(abs(1-ymcdf))]);
idx0    = find(ymcdf<ztol,1,'last');
idx1    = find(abs(1-ymcdf)<ztol,1,'first');
idxbad  = find(diff(ymcdf)==0);
idxbad  = idxbad((idxbad>=idx0 & idxbad <=idx1));
idxgood = (idx0:idx1)';
idxgood = setdiff(idxgood,idxbad);
ymix    = interp1(ymcdf(idxgood),xx(idxgood),xx1p,'linear','extrap');
```

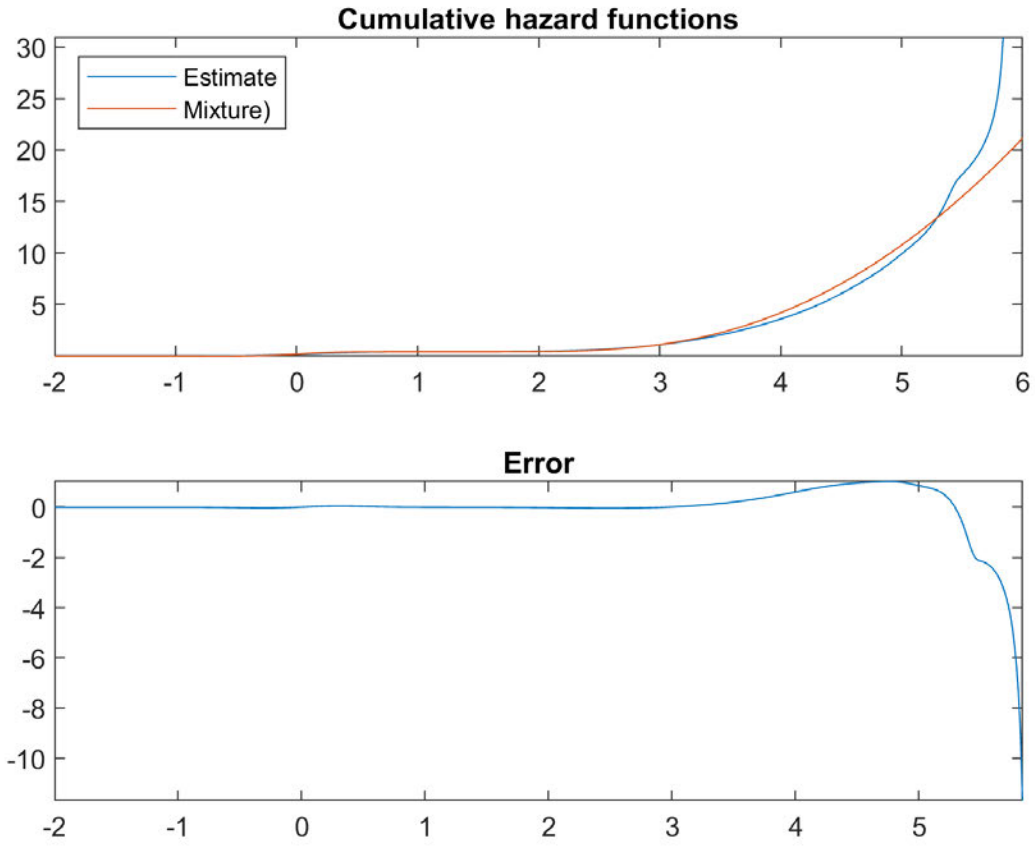


Figure 3.9: Estimate of the cumulative hazard function for a 1-dimensional mixture of two normal distributions. Estimated and true cumulative hazard functions (top) and error (bottom).

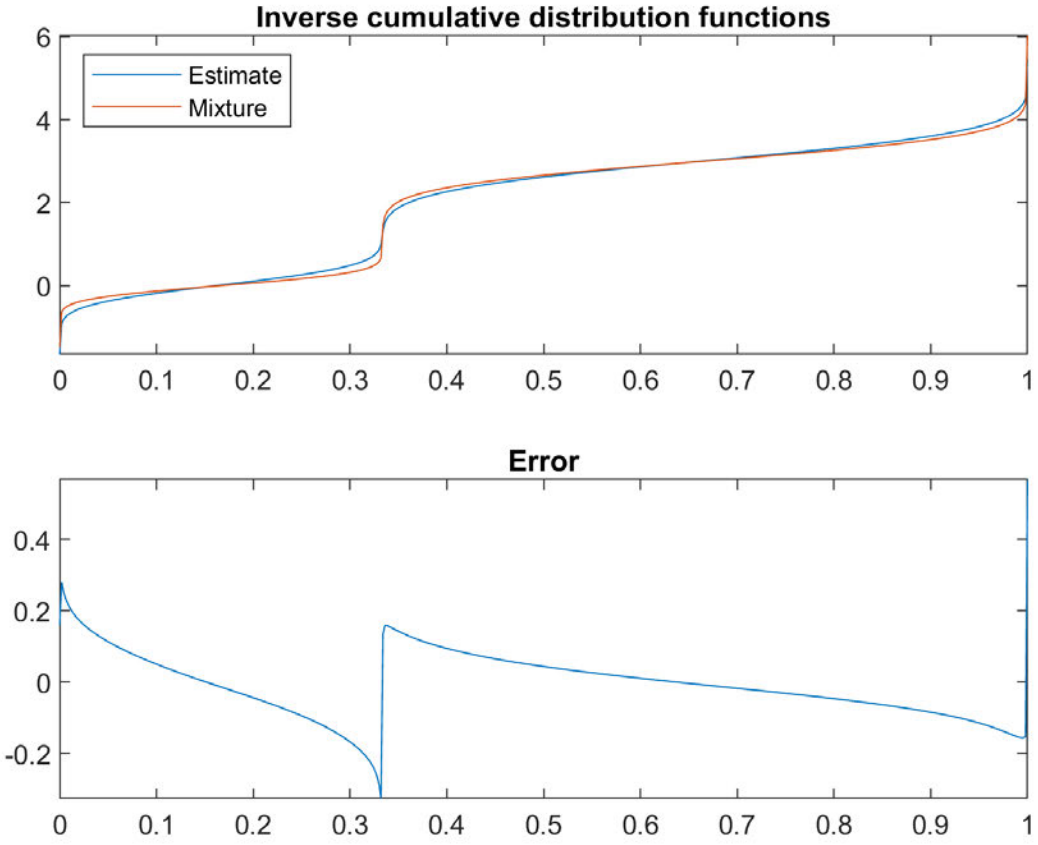


Figure 3.10: Estimate of the ICDF for a 1-dimensional mixture of two normal distributions. Estimated and true ICDF(top) and error (bottom).

The mixture distribution example is repeated using, $n = 7500$ data points from the unequal mixture of two normal distributions $\frac{1}{3}N(0, 0.25) + \frac{2}{3}N(3, 0.5)$.

Generate the data and evaluation points.

```
nsample   = 2500;
ndata     = 3*nsample;
evalbnds  = [-6 8];
xx1       = linspace(evalbnds(1),evalbnds(2),501)';
xtest1    = [0.25**randn(nsample,1); 3+0.5*randn(2*nsample,1)];
```

Compute the PDF estimate, evaluate it and compute the true mixture PDF.

```
bss1test  = bsspdfest(xtest1);
y1        = bsseval(xx1,bss1test,'pdf');
ynmix     = (1/3)*normpdf(xx1,0,0.25) + (2/3)*normpdf(xx1,3,0.5);
```

Evaluate the CDFsand compute the true mixture CDFs.

```
y1    = bsseval(xx1,bss1test,'cdf');
ynmix = (1/3)*normcdf(xx1,0,0.25) + (2/3)*normcdf(xx1,3,0.5);
```

Evaluate the survivor function andcompute the true mixture survivor function.

```
y1    = bsseval(xx1,bss1test,'survivor');
ynmix = 1-((1/3)*normcdf(xx1,0,0.25) + (2/3)*normcdf(xx1,3,0.5));
```

Evaluate the cumulative hazard function and compute the true mixture cumulative hazard function.

```
y1          = bsseval(xx1,bss1test,'cumhazard');
ynmix       =  1-((1/3)*normcdf(xx1,0,0.25) + (2/3)*normcdf(xx1,3,0.5));
gt0         = ynmix>0;
ynmix(~gt0) = nan;
ynmix(gt0)  = -log(ynmix(gt0));
```

Evaluate the ICDFand compute the true mixture ICDF.

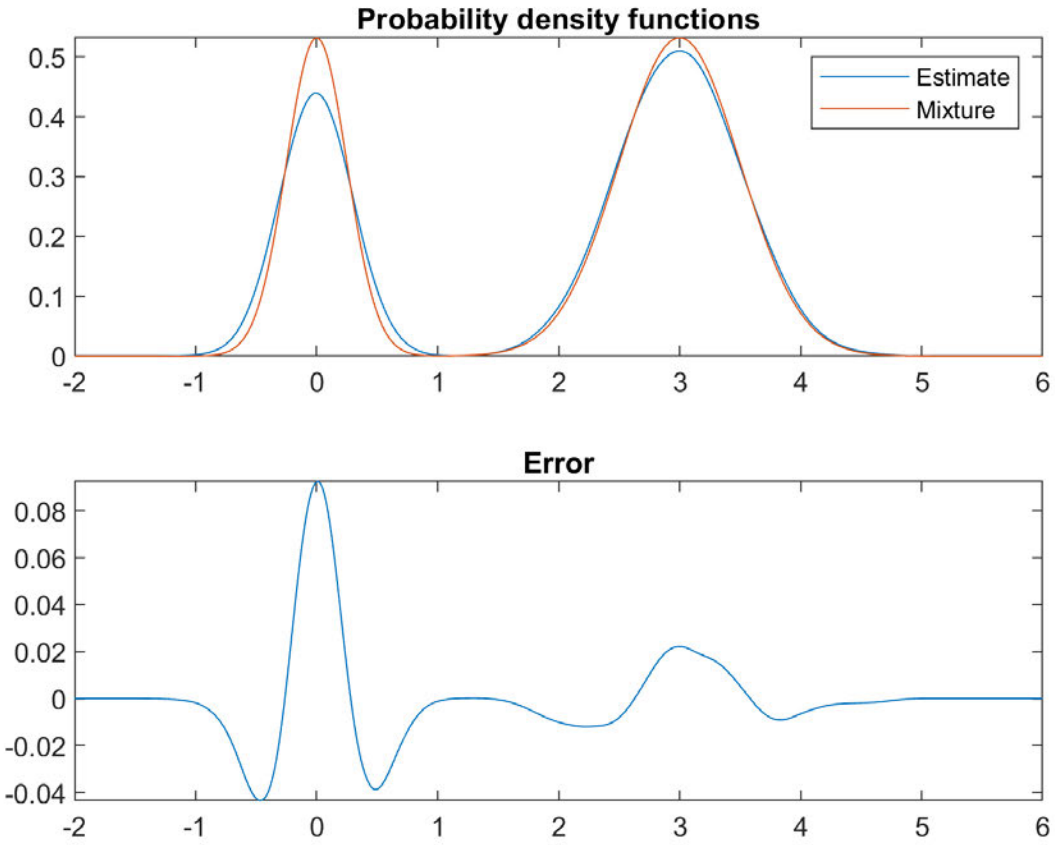


Figure 3.11: Estimating a 1-dimensional mixture of two normal distributions using $n = 1500$ data points. Estimated and true PDF (top) and error (bottom).

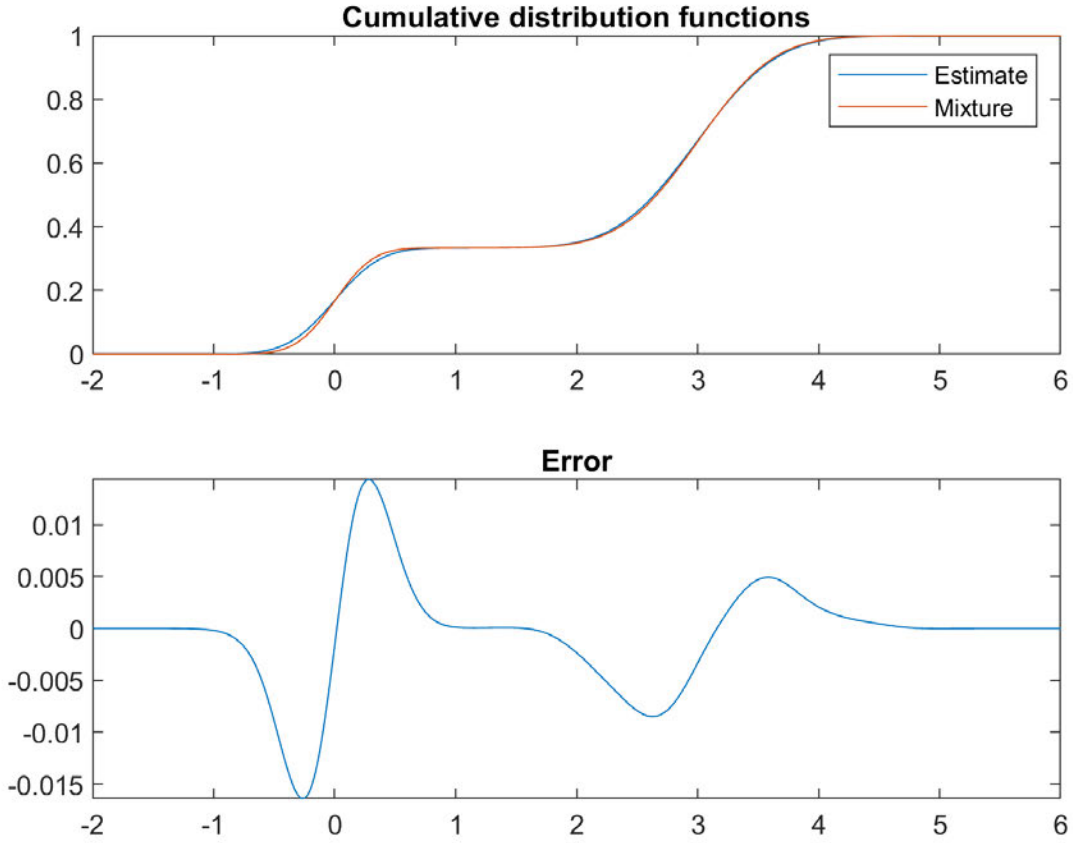


Figure 3.12: Estimate of the cumulative distribution function for a 1-dimensional mixture of two normal distributions. Estimated and true CDFs (top) and error (bottom).

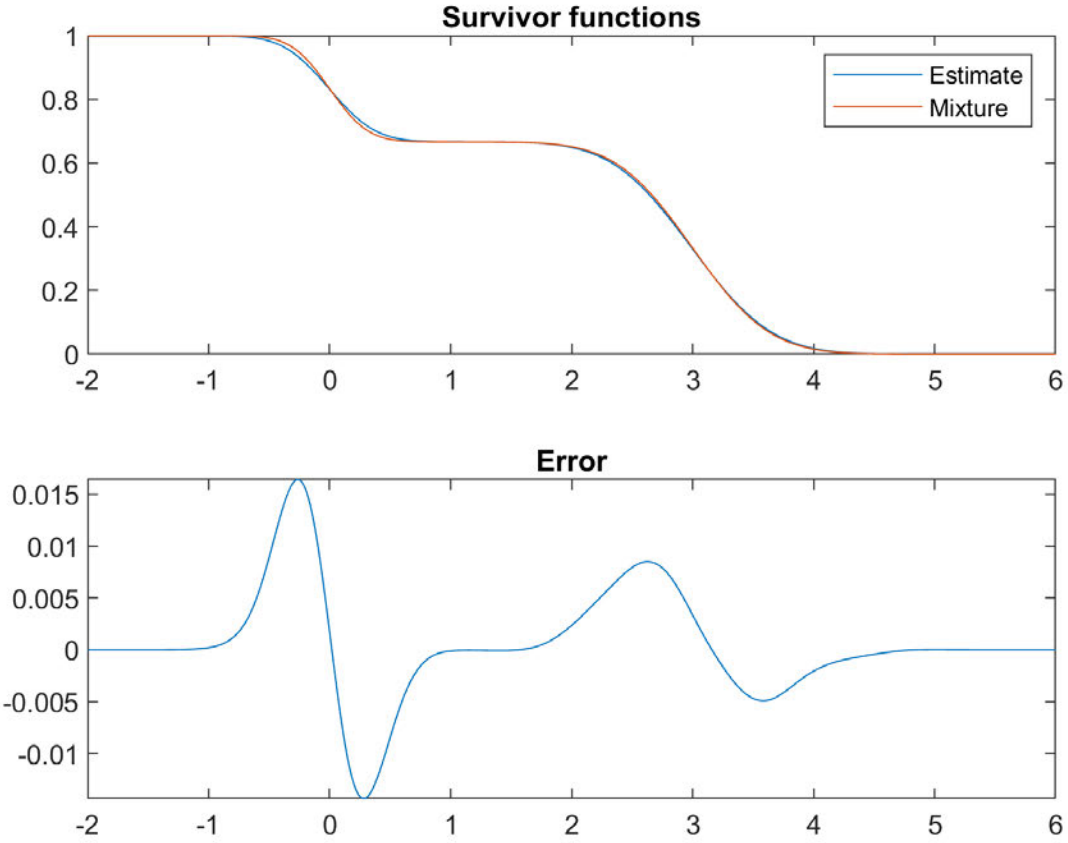


Figure 3.13: Estimate of the survivor function for a 1-dimensional mixture of two normal distributions. Estimated and true survivor functions (top) and error (bottom).

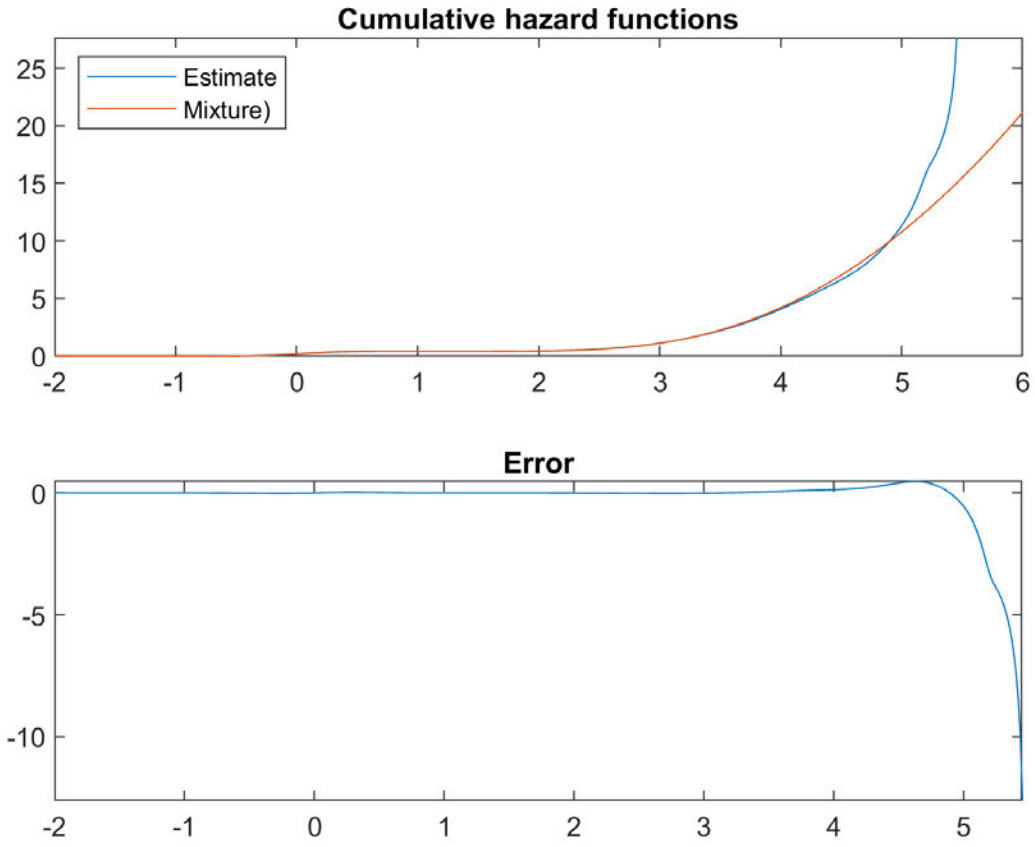


Figure 3.14: Estimate of the cumulative hazard function for a 1-dimensional mixture of two normal distributions. Estimated and true cumulative hazard functions (top) and error (bottom).

```
xx1p    = linspace(0,1,501)';
y1      = bsseval(xx1p,bss1test,'icdf');
xx      = linspace(-2,6,1001)';
ympdf   = (1/3)*normpdf(xx,0,0.25) + (2/3)*normpdf(xx,3,0.5);
ymcdf   = cumtrapz(xx,ympdf);
ztol    = 2*max([min(ymcdf) min(abs(1-ymcdf))]);
idx0    = find(ymcdf<ztol,1,'last');
idx1    = find(abs(1-ymcdf)<ztol,1,'first');
idxbad  = find(diff(ymcdf)==0);
idxbad  = idxbad((idxbad>=idx0 & idxbad <=idx1));
idxgood = (idx0:idx1)';
idxgood = setdiff(idxgood,idxbad);
ymix    = interp1(ymcdf(idxgood),xx(idxgood),xx1p,'linear','extrap');
```

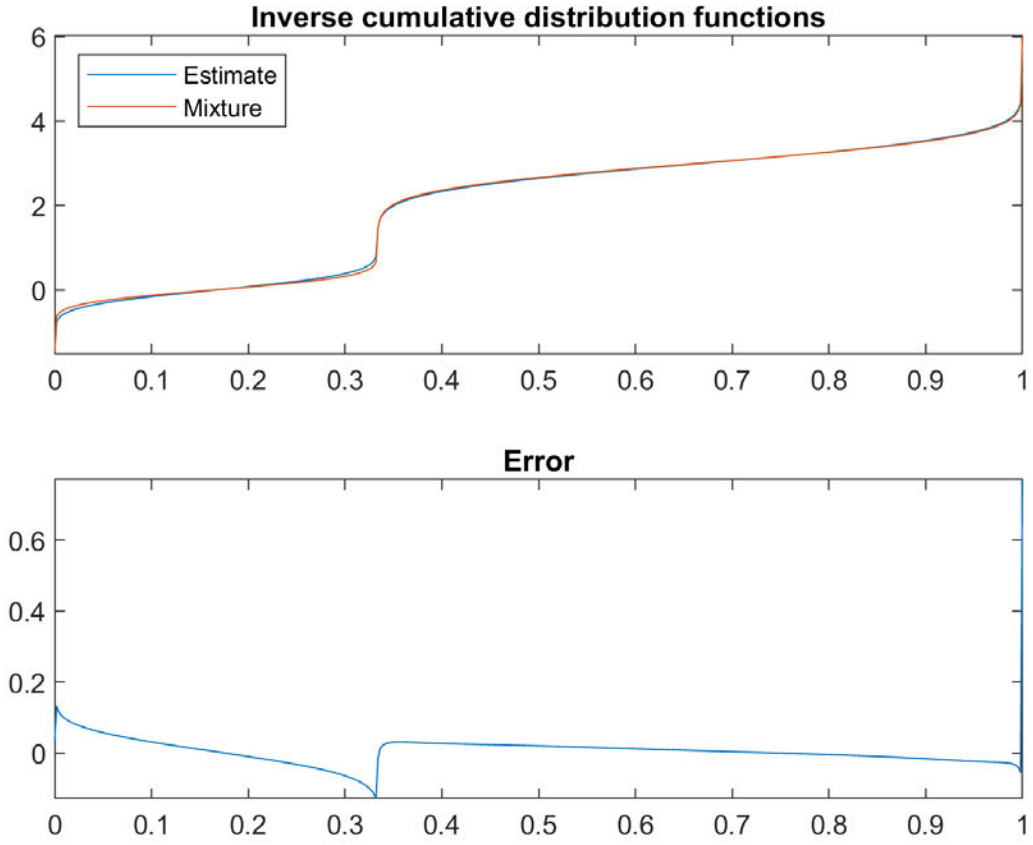


Figure 3.15: Estimate of the ICDF for a 1-dimensional mixture of two normal distributions. Estimated and true ICDF(top) and error (bottom).

3.2 Two-dimensional examples

3.2.1 Standard normal distribution

For the first 2-dimensional example, $n = 5000$ data points are generated from a standard normal distribution $N(0, I_2)$, where 0 is the 2-dimensional zero vector and I_2 is the 2×2 identity matrix. The data are then used to compute an estimate of the PDF, which is used subsequently to compute the CDFs and the survivor function. The MATLAB code used to generate the data and perform the probability function estimation is provided. The evaluation points and the PDF estimate `bss2test` are used for each of the probability functions. Results are plotted for each function comparing the estimated function to the actual function with the estimation error. Notice in the error plot that the largest magnitude errors occur where the curvature of the underlying probability density function is changing most rapidly. These are the regions that are typically the most difficult to fit. The addition of a few more partition subintervals or basis functions may provide a better fit in these regions, but may introduce undesirable wiggles.

Generate the data and evaluation points.

```
nsample  = 5000;
evalbnds = repmat([-6 6],2,1);
xtest2   = randn(nsample,2);
xx2      = [
    linspace(evalbnds(1,1),evalbnds(1,2),101);
    linspace(evalbnds(2,1),evalbnds(2,2),101);
    ]';
[xx2mg1, xx2mg2] = ndgrid(xx2(:,1),xx2(:,2));
xx2mg            = [xx2mg1(:) xx2mg2(:)];
```

Compute the PDF estimate, evaluate it, and compute the true two dimensional standard normal PDF.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'pdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = mvnpdf(xx2mg,[0 0],ones(1,2));
y2n      = reshape(y2nmg,size(xx2mg1));
```

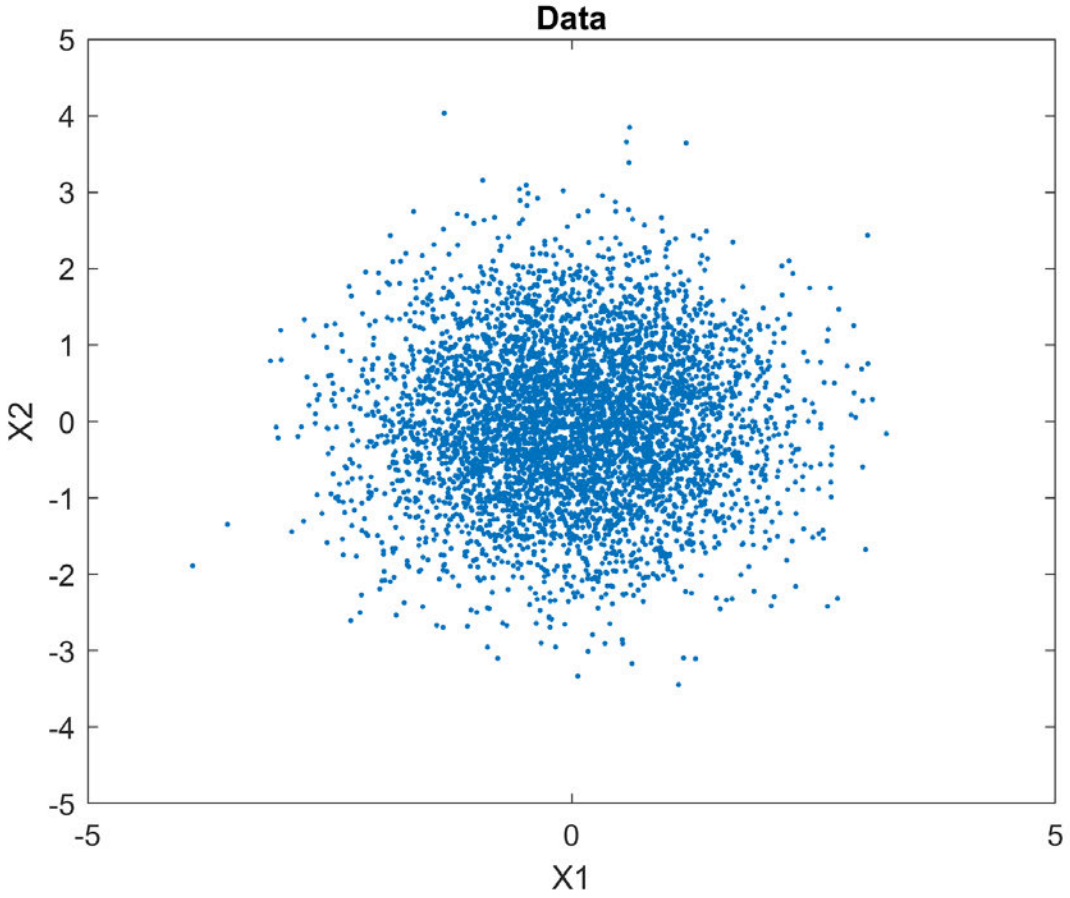


Figure 3.16: 5000 data points generated from a 2-dimensional standard normal distribution.

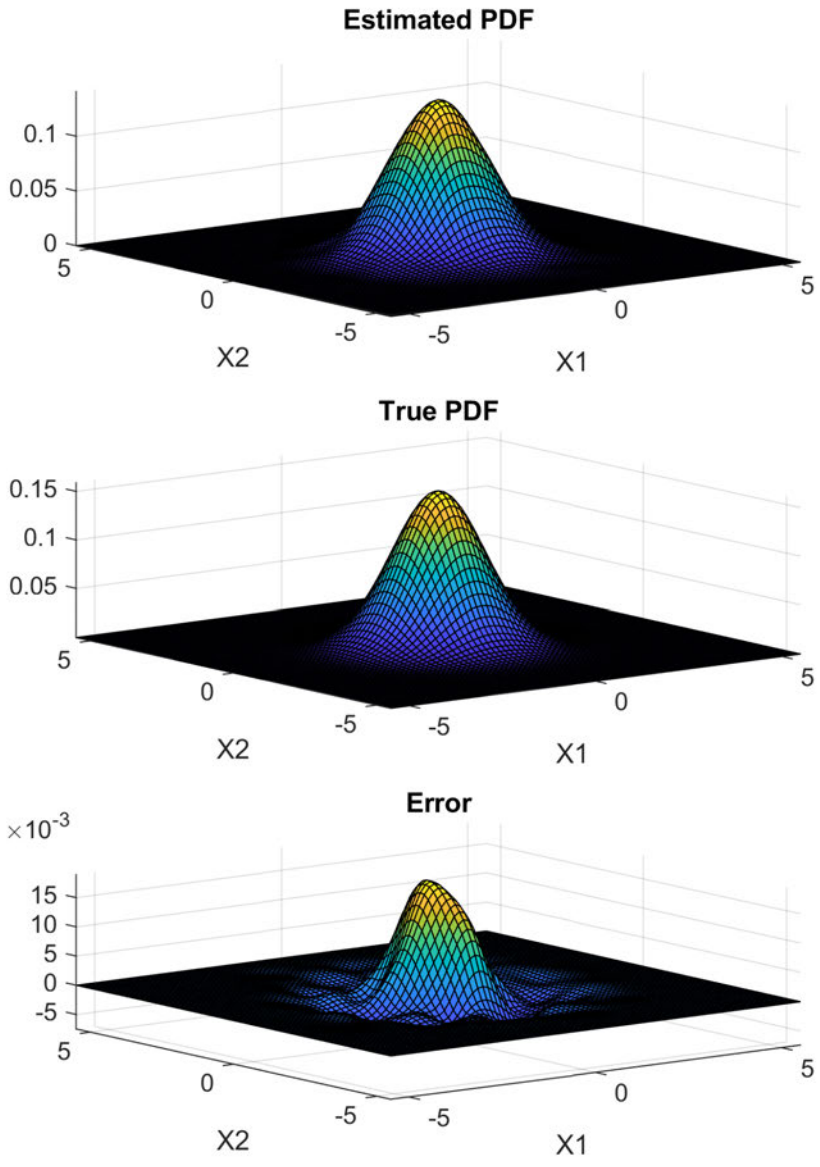


Figure 3.17: Estimate of a 2-dimensional standard normal PDF using $n = 5000$ data points. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

Evaluate the CDFs estimate and compute the true two dimensional standard normal CDFs.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'cdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = mvncdf(xx2mg,[0 0],ones(1,2));
y2n      = reshape(y2nmg,size(xx2mg1));
```

Evaluate the survivor function and compute the true two dimensional standard normal survivor function.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'cdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = 1-mvncdf(xx2mg,[0 0],ones(1,2));
y2n      = reshape(y2nmg,size(xx2mg1));
```

The 2-dimensional standard normal distribution example is repeated using, $n = 10000$ data points.

Generate the data and evaluation points.

```
nsample  = 10000;
evalbnds = repmat([-6 6],2,1);
xtest2   = randn(nsample,2);
xx2      = [
    linspace(evalbnds(1,1),evalbnds(1,2),101);
    linspace(evalbnds(2,1),evalbnds(2,2),101);
    ]';
[xx2mg1, xx2mg2] = ndgrid(xx2(:,1),xx2(:,2));
xx2mg = [xx2mg1(:) xx2mg2(:)];
```

Compute the PDF estimate, evaluate it, and compute the true two dimensional standard normal PDF.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'pdf');
```

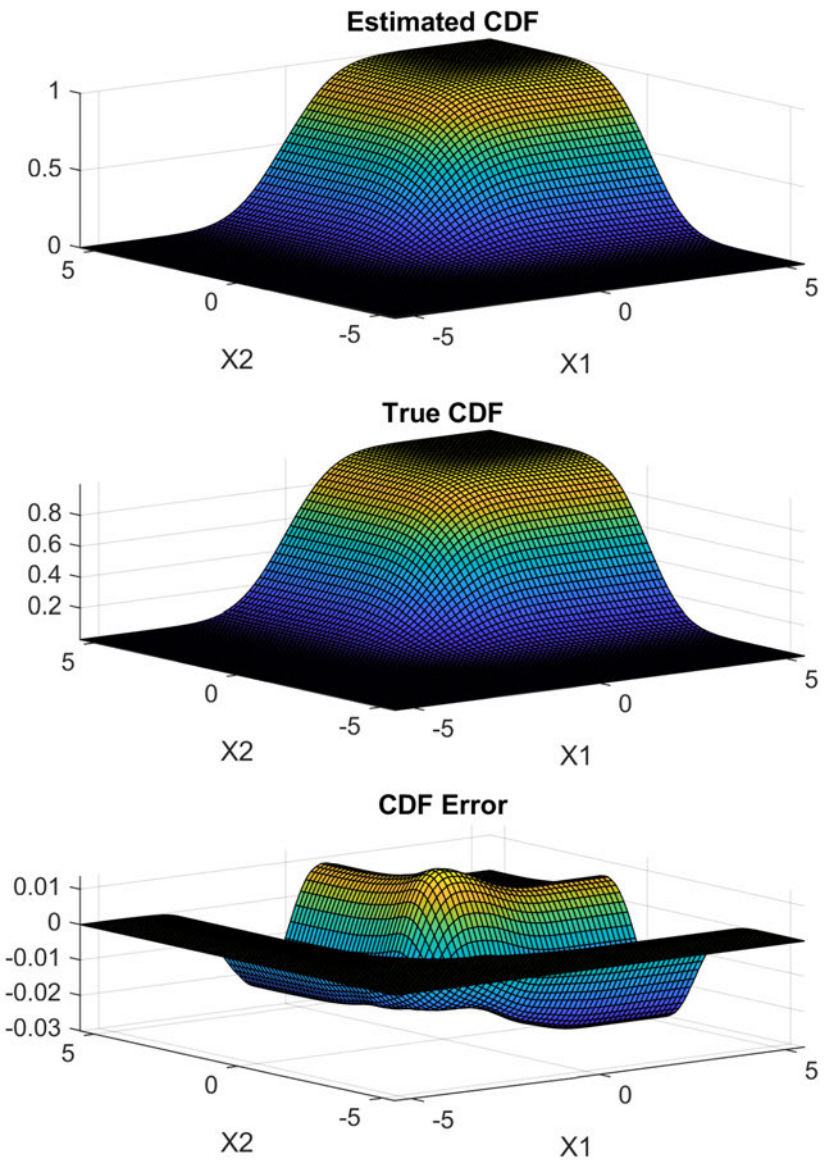


Figure 3.18: Estimate of a 2-dimensional standard normal CDFs using $n = 5000$ data points. Estimated CDFs (top), true CDFs (middle), and estimation error (bottom).

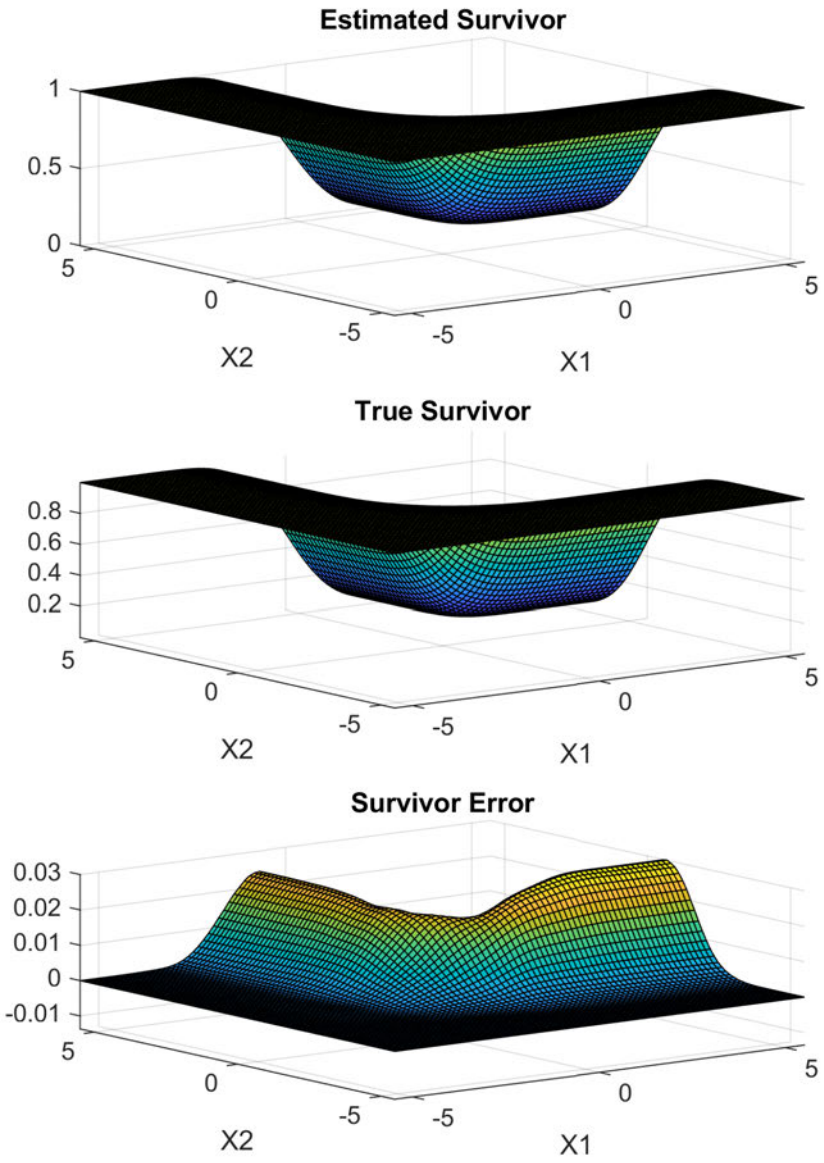


Figure 3.19: Estimate of a 2-dimensional standard normal survivor function using $n = 5000$ data points. Estimated survivor function (top), true survivor function(middle), and estimation error (bottom).

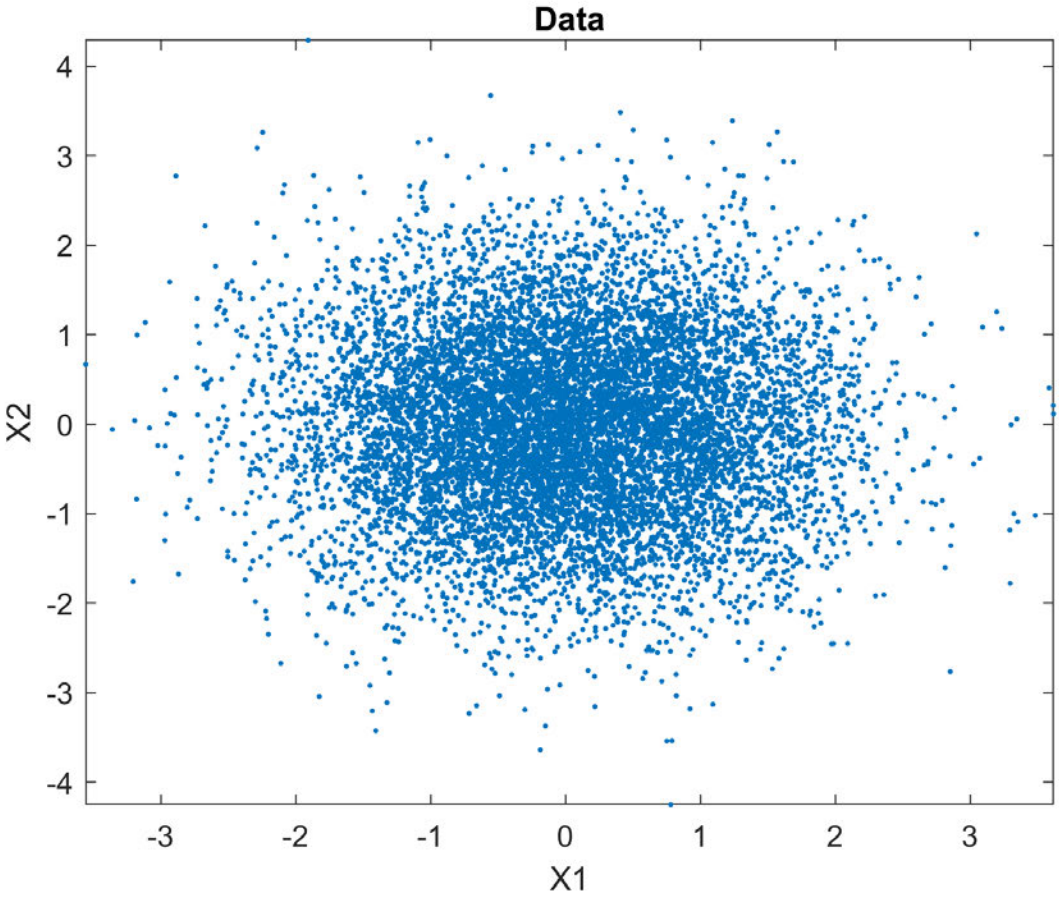


Figure 3.20: 10000 data points generated from a 2-dimensional standard normal distribution.

```
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = mvnpdf(xx2mg,[0 0],ones(1,2));
y2n      = reshape(y2nmg,size(xx2mg1));
```

Evaluate the CDFs estimate and compute the true two dimensional standard normal CDFs.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'cdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = mvncdf(xx2mg,[0 0],ones(1,2));
y2n      = reshape(y2nmg,size(xx2mg1));
```

Evaluate the survivor function and compute the true two dimensional standard normal survivor function.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'cdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = 1-mvncdf(xx2mg,[0 0],ones(1,2));
y2n      = reshape(y2nmg,size(xx2mg1));
```

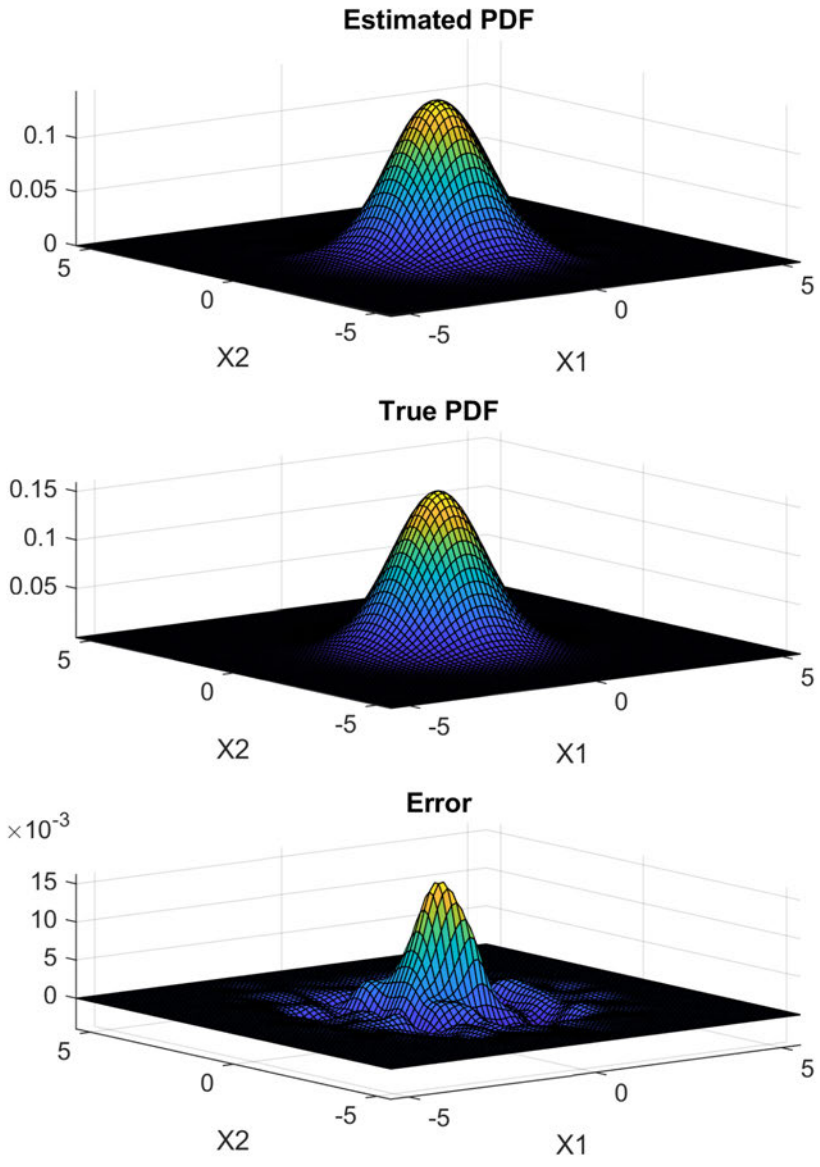


Figure 3.21: Estimate of a 2-dimensional standard normal PDF using $n = 10000$ data points. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

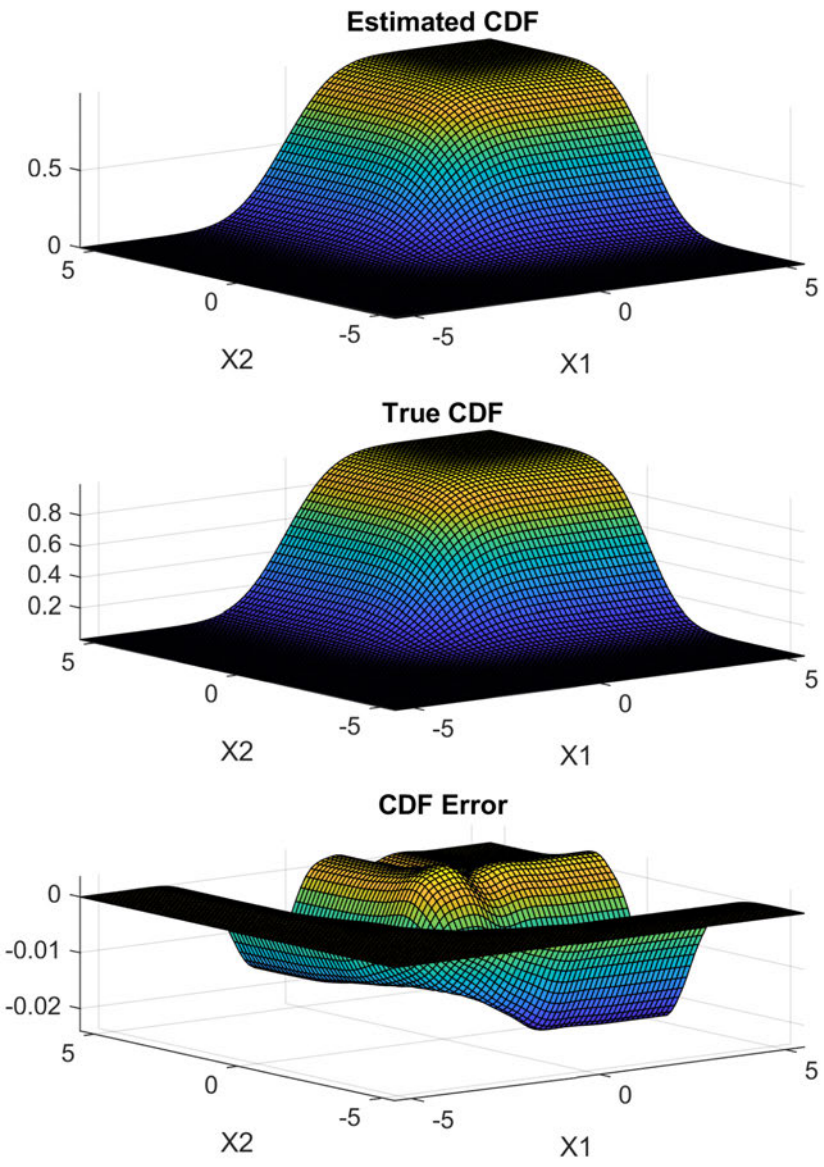


Figure 3.22: Estimate of a 2-dimensional standard normal CDFs using $n = 10000$ data points. Estimated CDFs (top), true CDFs (middle), and estimation error (bottom).

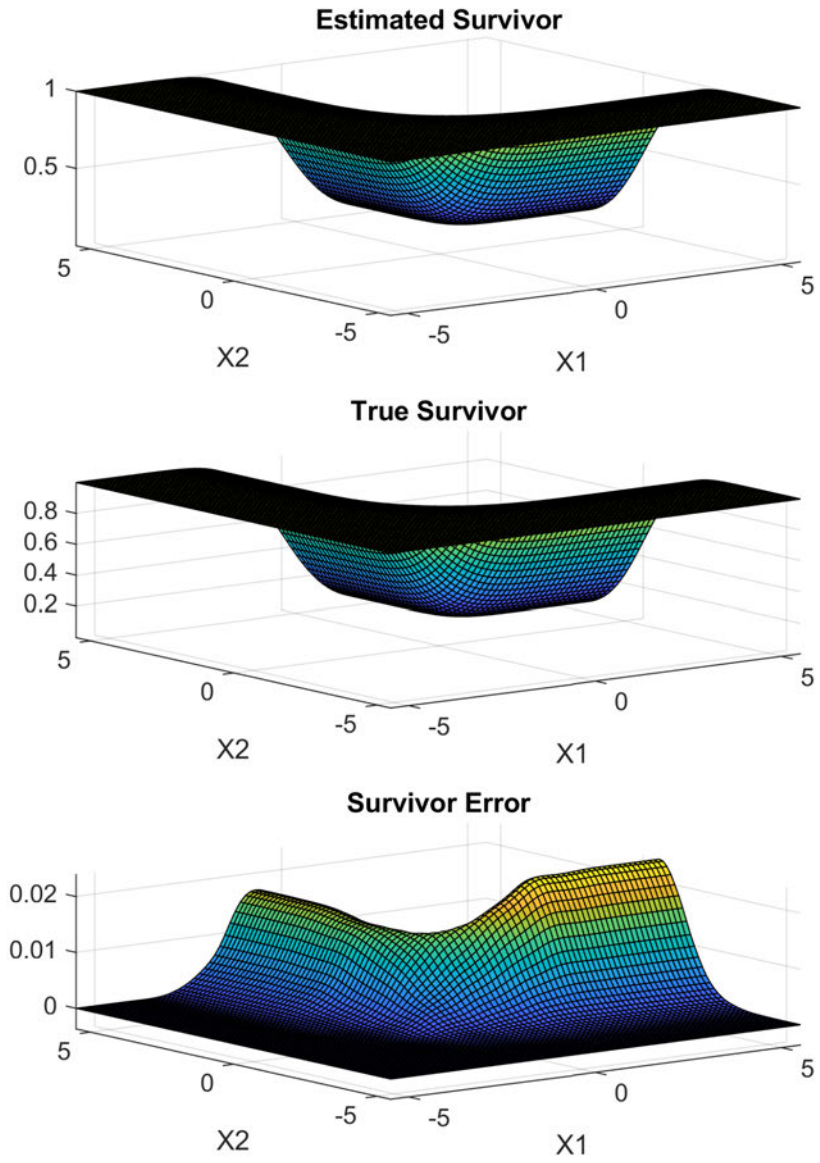


Figure 3.23: Estimate of a 2-dimensional standard normal survivor function using $n = 10000$ data points. Estimated survivor function (top), true survivor function (middle), and estimation error (bottom).

3.2.2 Mixture of 5 2-dimensional normal distributions

For this example, $n = 50000$ data points are generated from an equal mixture of five normal distributions, with 10000 points from each component of the mixture, $f(x) = \frac{1}{5}\sum_{i=1}^{5} N(\mu_i, \Sigma_i)$ and used to compute an estimate of the PDF, where

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mu_2 = \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \mu_3 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \mu_4 = \begin{pmatrix} 3 \\ -3 \end{pmatrix}, \mu_5 = \begin{pmatrix} -3 \\ 3 \end{pmatrix},$$

and

$$\Sigma_1 = \begin{pmatrix} 0.9 & -0.4 \\ -0.4 & 0.9 \end{pmatrix}, \Sigma_i = \begin{pmatrix} 0.9 & 0.4 \\ 0.4 & 0.9 \end{pmatrix}, i = 2, 3, 4, 5$$

Generate the data and evaluation points.

```
nsample  = 10000;
ndata    = 5*nsample;
xmu      = [0 0; -3 -3; 3 3; 3 -3; -3 3];
xcov0    = [0.9 -0.4; -0.4 0.3];
xcov3    = [0.9  0.4;  0.4 0.3];
chxcov0  = cholcov(xcov0);
chxcov3  = cholcov(xcov3);
evalbnds = [-9 9;-7 7];

xtest2 = [
    repmat(xmu(1,:),nsample,1) + randn(nsample,2)*chxcov0;
    repmat(xmu(2,:),nsample,1) + randn(nsample,2)*chxcov3;
    repmat(xmu(3,:),nsample,1) + randn(nsample,2)*chxcov3;
    repmat(xmu(4,:),nsample,1) + randn(nsample,2)*chxcov3;
    repmat(xmu(5,:),nsample,1) + randn(nsample,2)*chxcov3;
    ];

xx2 = [
    linspace(evalbnds(1,1),evalbnds(1,2),101);
    linspace(evalbnds(2,1),evalbnds(2,2),101);
    ]';
[xx2mg1, xx2mg2] = ndgrid(xx2(:,1),xx2(:,2));
xx2mg = [xx2mg1(:) xx2mg2(:)];
```

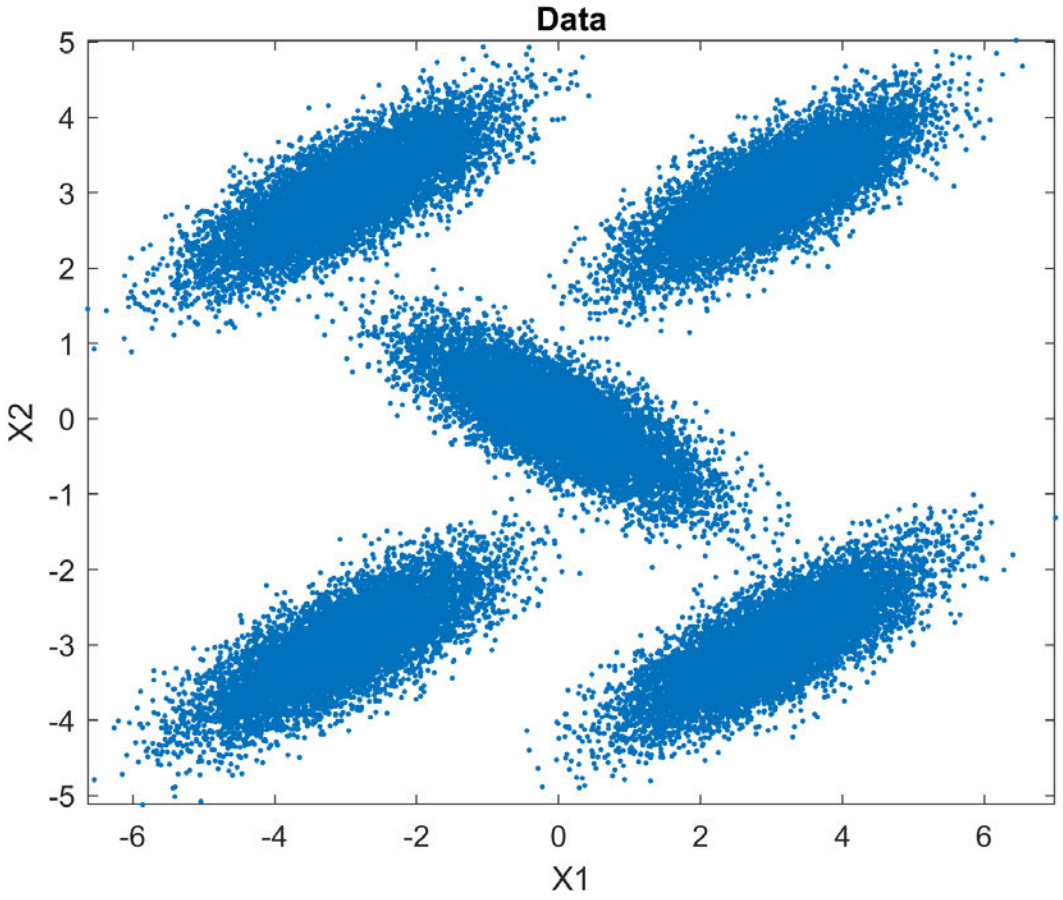


Figure 3.24: 50000 data points generated from a mixture of 5 2-dimensional normal distributions.

Compute the PDF estimate, evaluate it and compute the true two dimensional mixture PDF.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'pdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = zeros(length(xx2mg),1);
for i = 1:5
   if (i == 1)
      y2nmg = y2nmg + (1/5)*mvnpdf(xx2mg,xmu(i,:),xcov0);
   else
      y2nmg = y2nmg + (1/5)*mvnpdf(xx2mg,xmu(i,:),xcov3);
   end
end
y2n = reshape(y2nmg,size(xx2mg1));
```

Evaluate the CDFs estimate and compute the true two dimensional mixture CDFs.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'cdf');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = zeros(length(xx2mg),1);
for i = 1:5
   if (i == 1)
      y2nmg = y2nmg + (1/5)*mvncdf(xx2mg,xmu(i,:),xcov0);
   else
      y2nmg = y2nmg + (1/5)*mvncdf(xx2mg,xmu(i,:),xcov3);
   end
end
y2n = reshape(y2nmg,size(xx2mg1));
```

Evaluate the survivor function estimate and compute the true two dimensional mixture survivor function.

```
bss2test = bsspdfest(xtest2);
y2mg     = bsseval(xx2mg,bss2test,'survivor');
y2       = reshape(y2mg,size(xx2mg1));
y2nmg    = zeros(length(xx2mg),1);
```

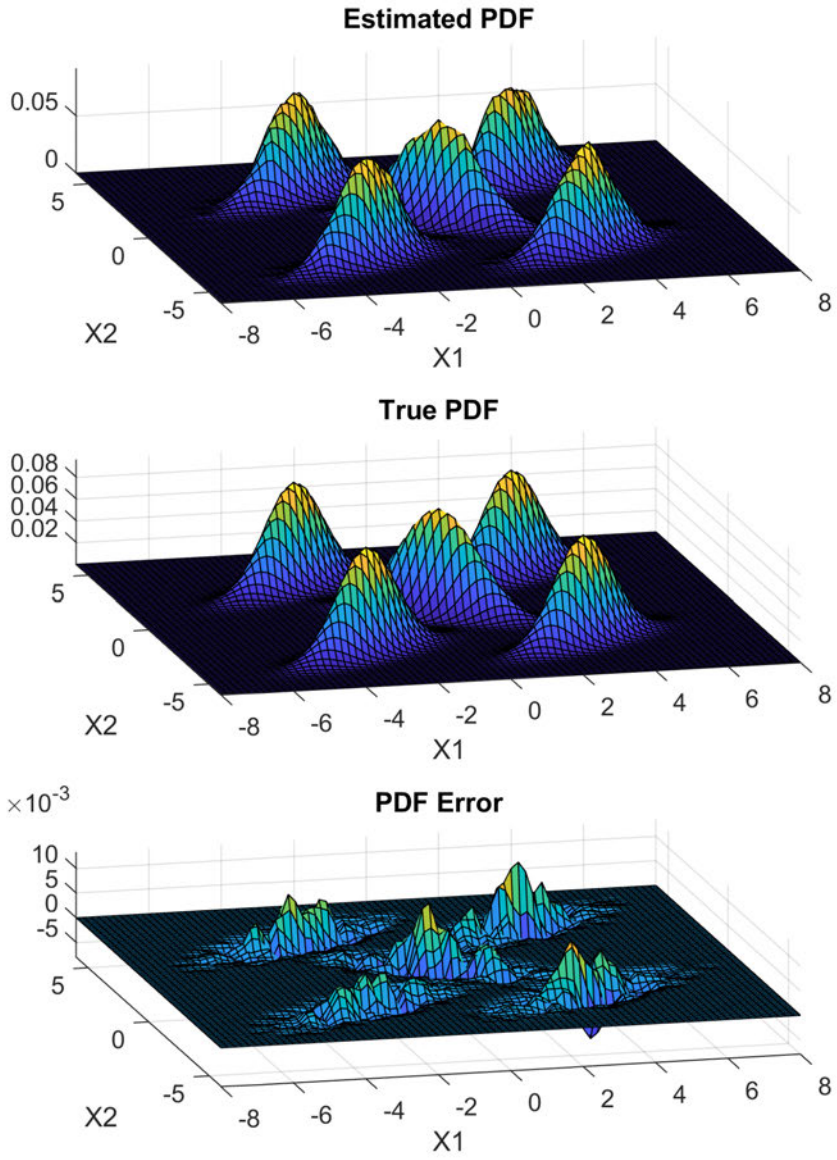


Figure 3.25: Estimating a mixture of 5 2-dimensional normal distributions using $n = 50000$ data points. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

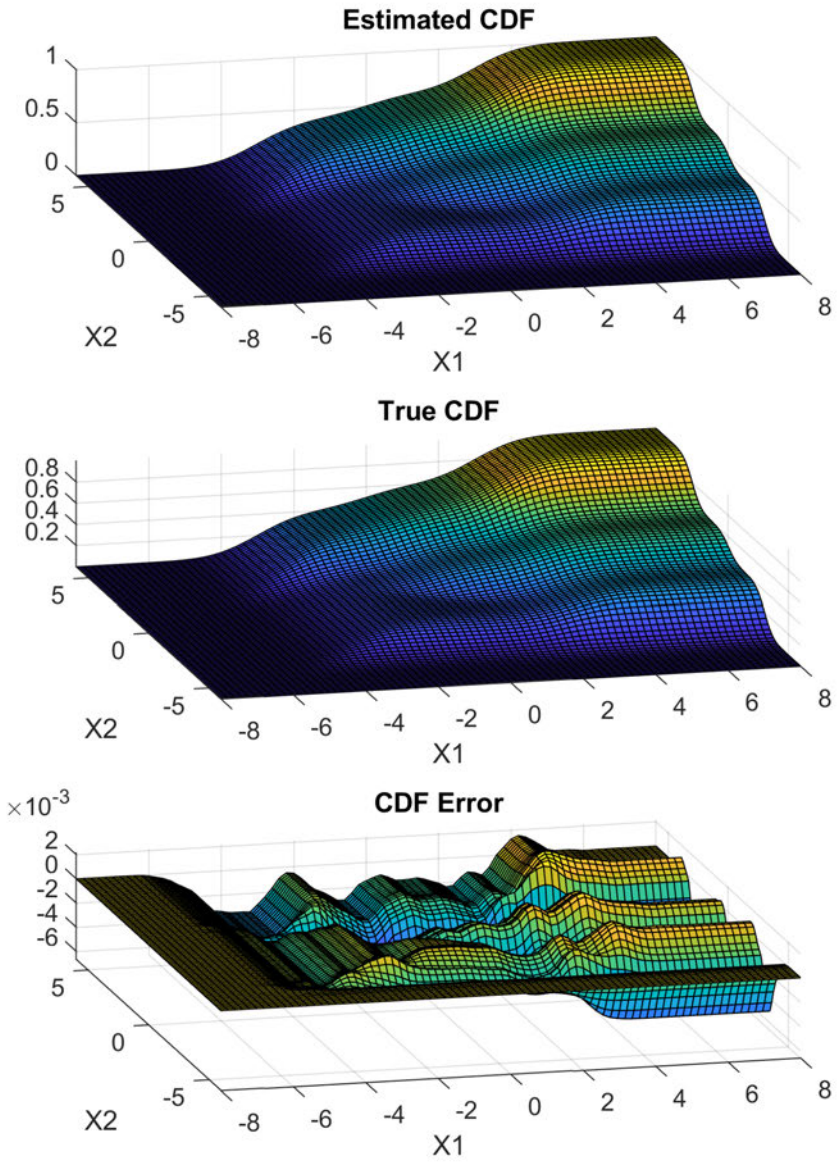


Figure 3.26: Estimated CDFs for a mixture of 5 2-dimensional normal distributions using $n = 50000$ data points. Estimated CDFs (top), true CDFs (middle), and estimation error (bottom).

```
for i = 1:5
   if (i == 1)
      y2nmg = y2nmg + (1/5)*mvncdf(xx2mg,xmu(i,:),xcov0);
   else
      y2nmg = y2nmg + (1/5)*mvncdf(xx2mg,xmu(i,:),xcov3);
   end
end
y2n = 1-reshape(y2nmg,size(xx2mg1));
```

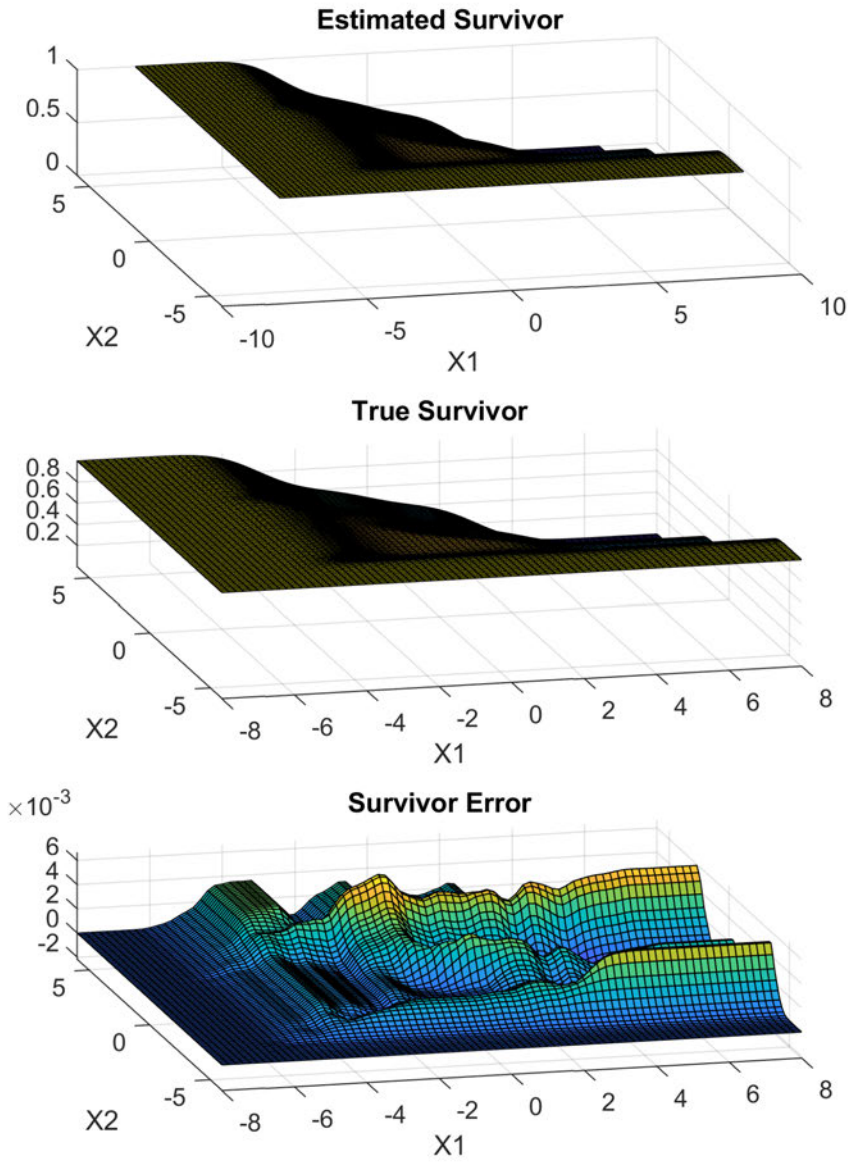


Figure 3.27: Estimated survivor function for a mixture of 5 2-dimensional normal distributions using $n = 50000$ data points. Estimated survivor function (top), true survivor function (middle), and estimation error (bottom).

3.3 Three-dimensional examples

3.3.1 Standard normal distribution

For the first 3-dimensional example, $n = 25000$ data points are generated from a standard normal distribution $N(0, I_3)$, where 0 is the 3-dimensional zero vector and I_3 is the 3×3 identity matrix, and used to compute an estimate of the PDF. The MATLAB code used to generate the data and perform the PDF estimation for the standard 3-dimensional normal distribution appears below. A scatter plot of the data is presented in Figure 3.28, and plots of the marginal density functions obtained using cross sections defined by the xy-plane, the xz-plane, and the yz-plane for the estimated PDF, the true PDF and the estimation error are presented in Figure 3.29,Figure 3.30, and Figure 3.31.

Generate the data and evaluation points.

```
neval    = 51;
nsample  = 25000;
xtest3 = randn(nsample,3);
evalbnds = repmat([-6 6],3,1);
xx3      = [
           linspace(evalbnds(1,1),evalbnds(1,2),neval);
           linspace(evalbnds(2,1),evalbnds(2,2),neval);
           linspace(evalbnds(3,1),evalbnds(3,2),neval);
           ]';
[xx3mg1, xx3mg2, xx3mg3] = ndgrid(xx3(:,1),xx3(:,2),xx3(:,3));
xx3mg = [xx3mg1(:) xx3mg2(:) xx3mg3(:)];
```

Compute the PDF estimate, evaluate it, and compute the true three dimensional standard normal PDF.

```
bss3test = bsspdfest3d(xtest3);
y3mg     = bseval3d(xx3mg,bss3test,'pdf');
y3       = reshape(y3mg,size(xx3mg1));
y3nmg    = mvnpdf(xx3mg,[0 0 0],ones(1,3));
y3n      = reshape(y3nmg,size(xx3mg1));
```

Evaluate the CDFs estimate and compute the true three dimensional standard normal CDFs.

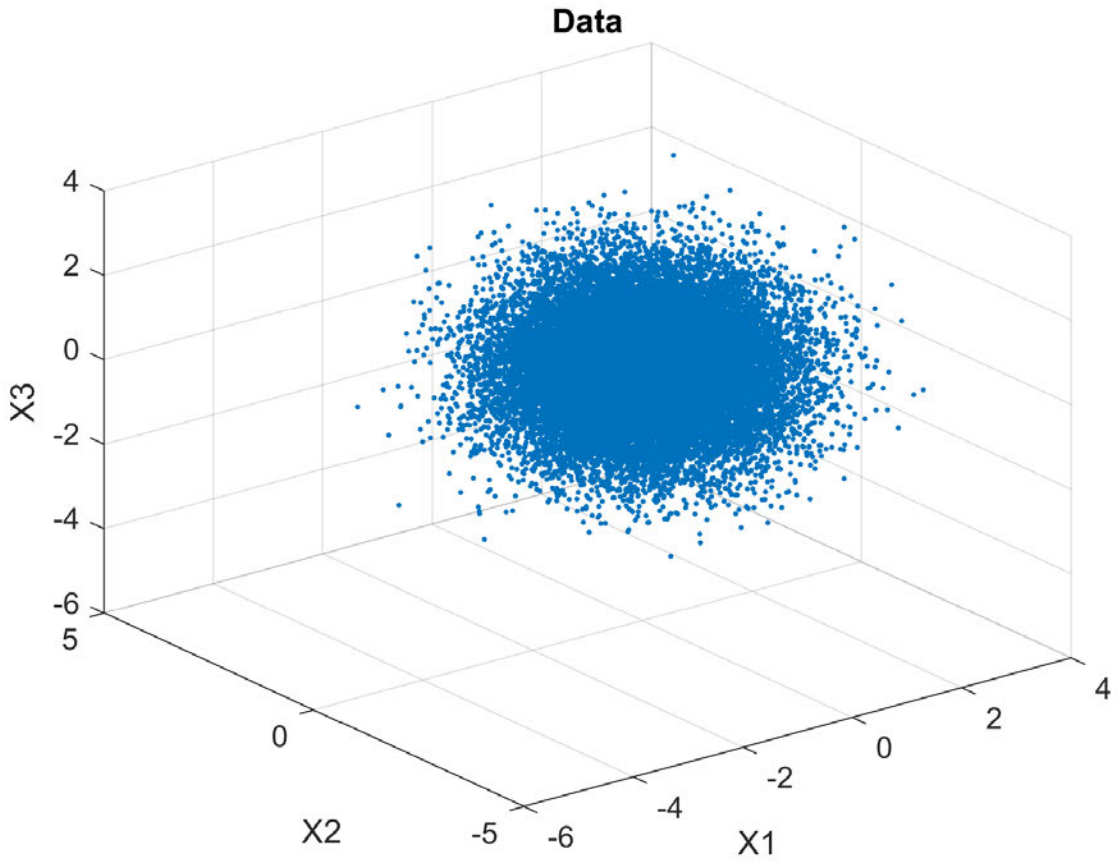


Figure 3.28: 25000 data points generated from a 3-dimensional standard normal distribution.

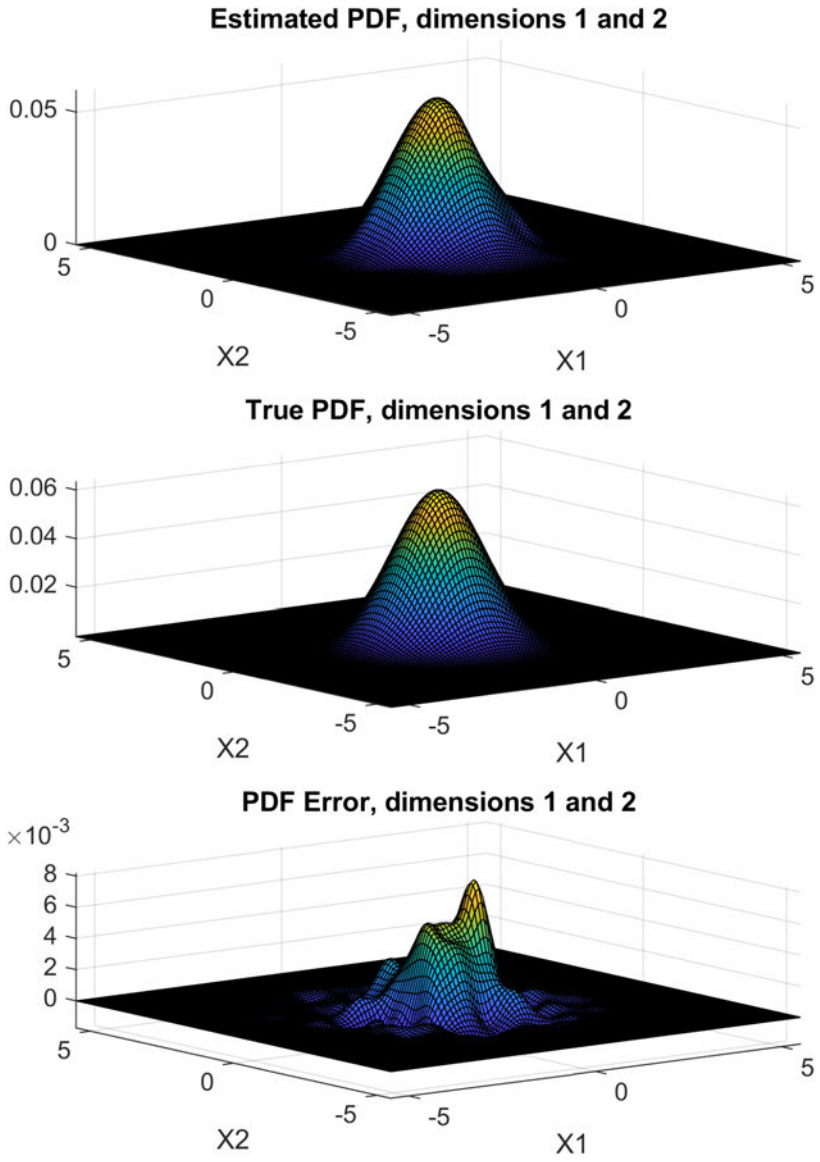


Figure 3.29: Estimating a 3-dimensional standard normal distribution using $n = 25000$ data points. Estimated PDF (top), true PDF (middle), and estimation error (bottom) for dimensions 1 and 2.

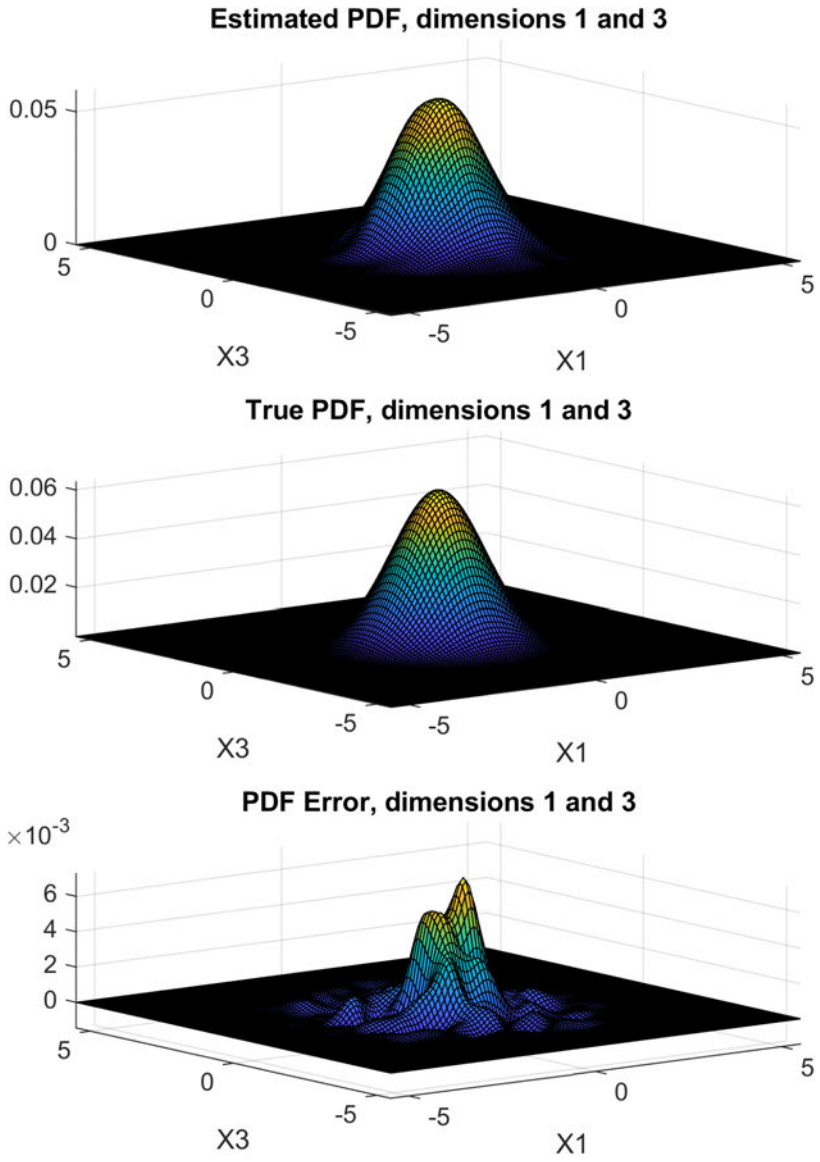


Figure 3.30: Estimating a 3-dimensional standard normal distribution using $n = 25000$ data points. Estimated PDF (top), true PDF (middle), and estimation error (bottom) for dimensions 1 and 3.

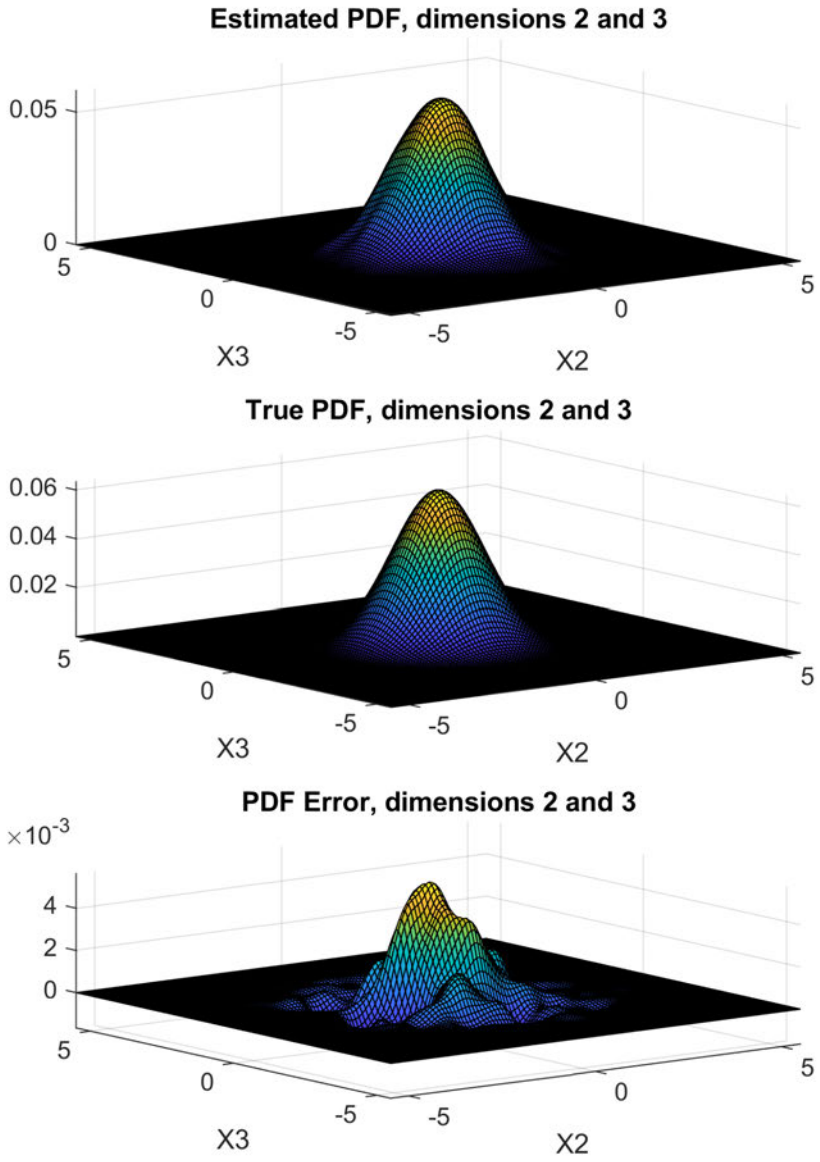


Figure 3.31: Estimating a 3-dimensional standard normal distribution using $n = 25000$ data points. Estimated PDF (top), true PDF (middle), and estimation error (bottom) for dimensions 2 and 3.

```
y3mg  = bsseval3d(xx3mg,bss3test,'cdf');
y3    = reshape(y3mg,size(xx3mg1));
y3nmg = mvncdf(xx3mg,[0 0 0],ones(1,3));
y3n   = reshape(y3nmg,size(xx3mg1));
```

Evaluate the survivor function estimate and compute the true three dimensional standard normal survivor function.

```
y3mg  = bsseval3d(xx3mg,bss3test,'survivor');
y3    = reshape(y3mg,size(xx3mg1));
y3nmg = 1-mvncdf(xx3mg,[0 0 0],ones(1,3));
y3n   = reshape(y3nmg,size(xx3mg1));
```

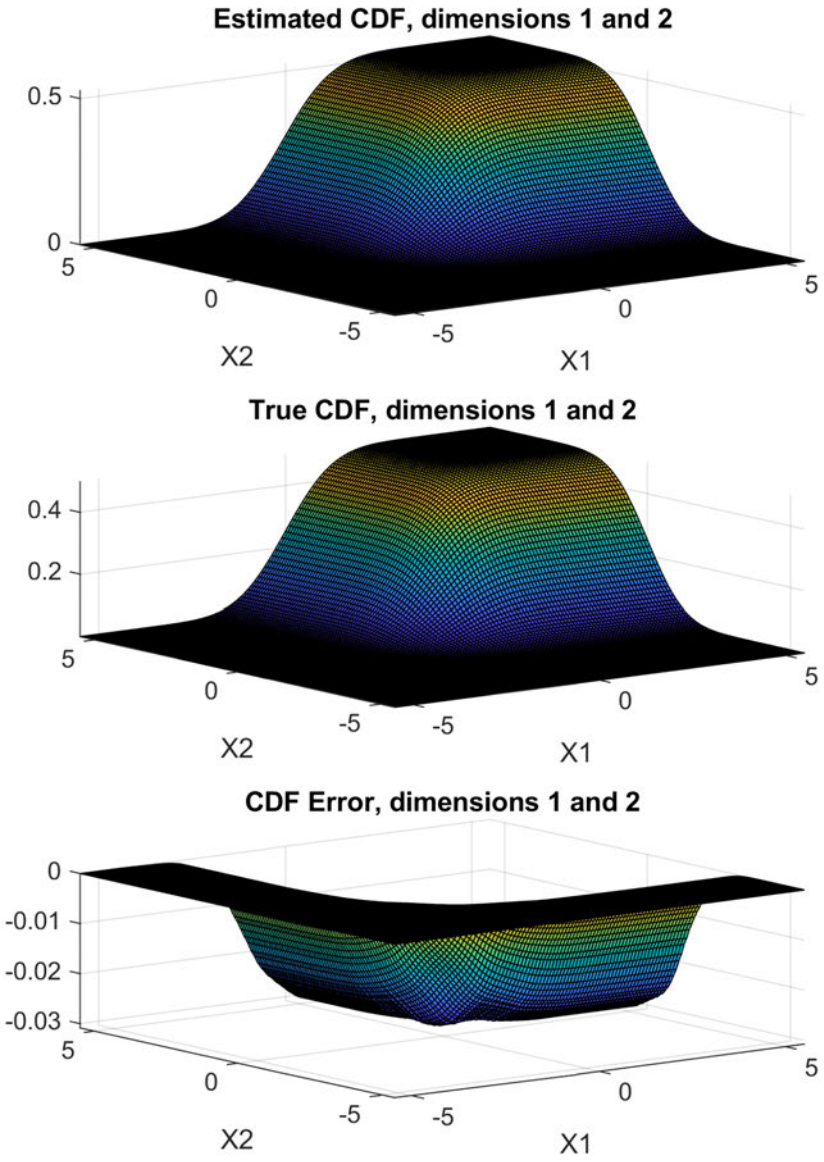


Figure 3.32: Estimating a 3-dimensional standard normal CDFs using $n = 25000$ data points. Estimated CDFs (top), true CDFs (middle), and estimation error (bottom) for dimensions 1 and 2.

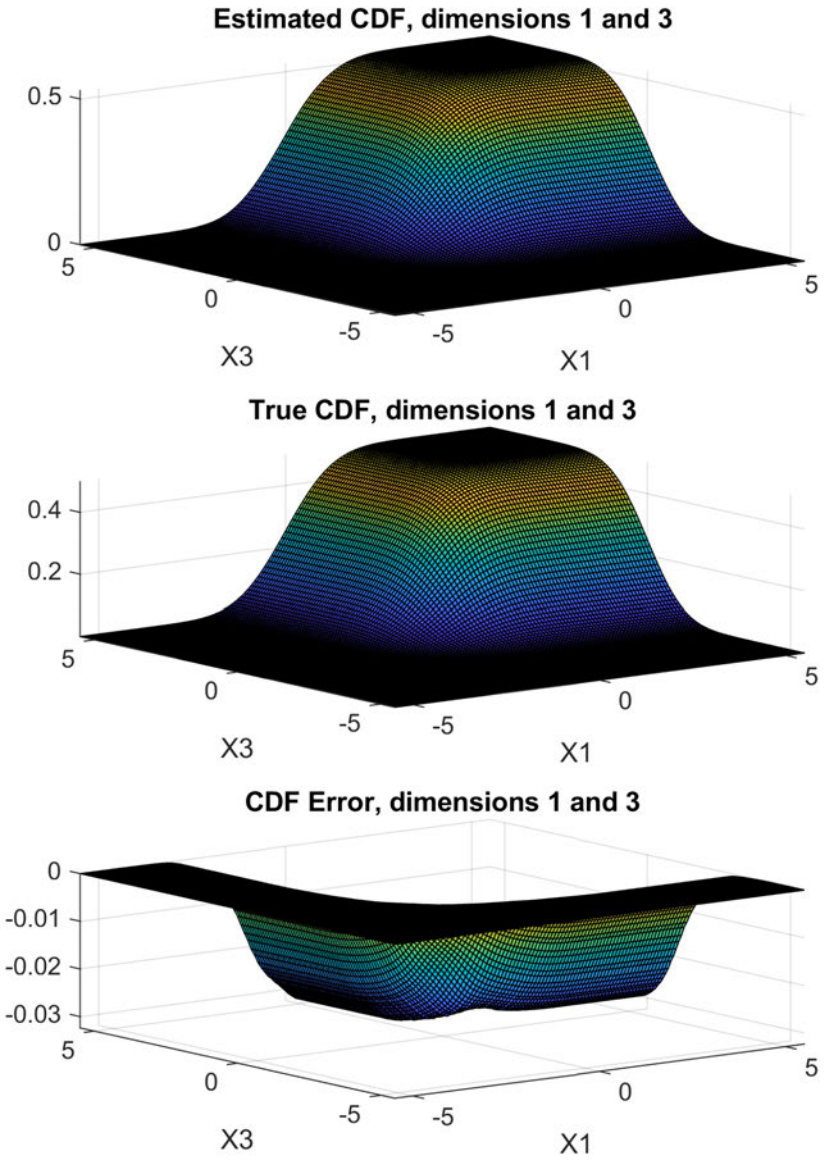


Figure 3.33: Estimating a 3-dimensional standard normal CDFs using $n = 25000$ data points. Estimated CDFs (top), true CDFs (middle), and estimation error (bottom) for dimensions 1 and 3.

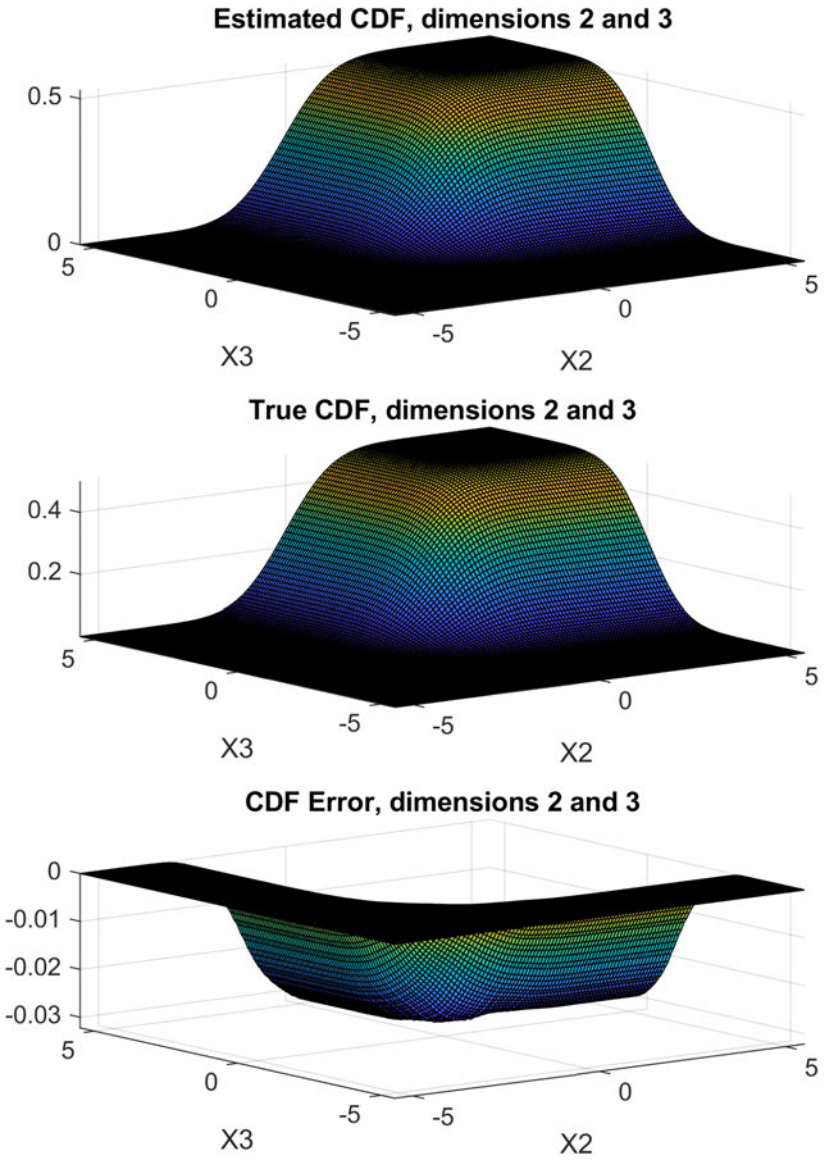


Figure 3.34: Estimating a 3-dimensional standard normal CDFs using $n = 25000$ data points. Estimated CDFs (top), true CDFs (middle), and estimation error (bottom) for dimensions 2 and 3.

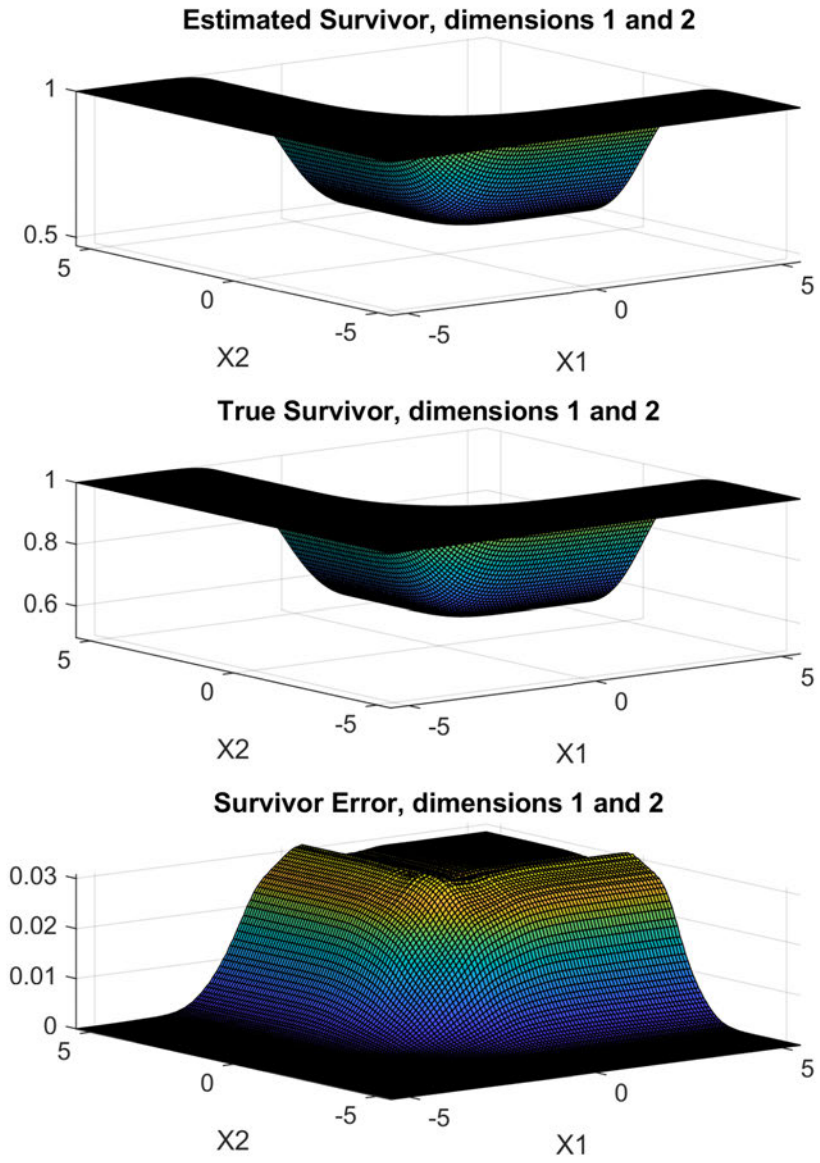


Figure 3.35: Estimating a 3-dimensional standard normal survivor function using $n = 25000$ data points. Estimated survivor function (top), true survivor function (middle), and estimation error (bottom) for dimensions 1 and 2.

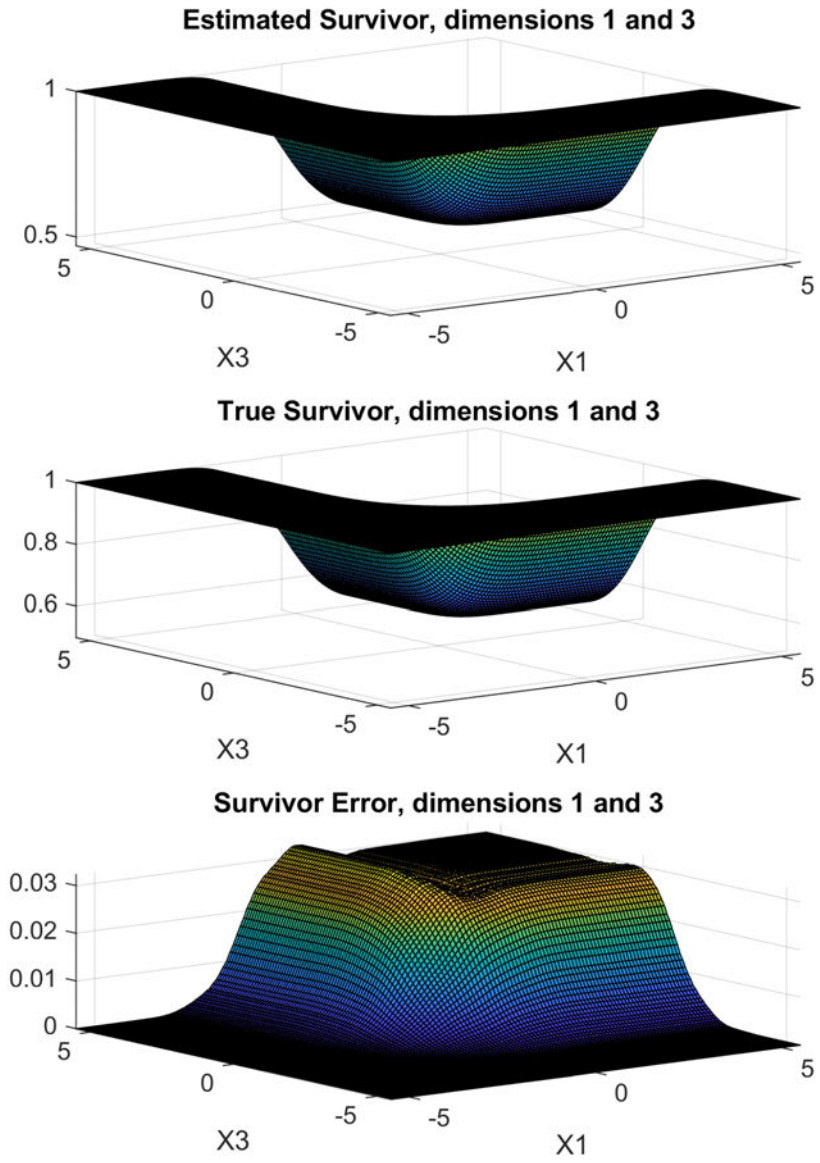


Figure 3.36: Estimating a 3-dimensional standard normal survivor function using $n = 25000$ data points. Estimated survivor function (top), true survivor function (middle), and estimation error (bottom) for dimensions 1 and 3.

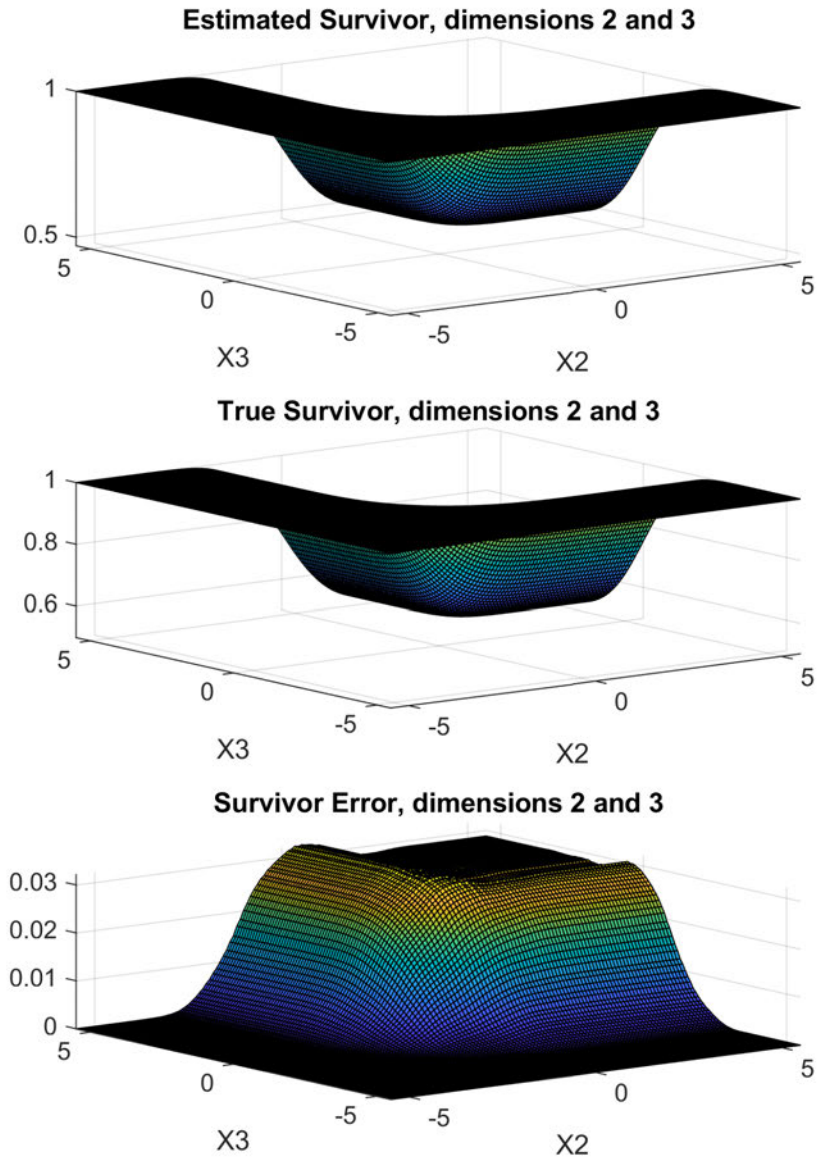


Figure 3.37: Estimating a 3-dimensional standard normal survivor function using $n = 25000$ data points. Estimated survivor function (top), true survivor function (middle), and estimation error (bottom) for dimensions 2 and 3.

3.3.2 Mixture of 5 3-dimensional normal distributions

For this example, $n = 25000$ data points are generated from an equal mixture of five 3-dimensional normal distributions $f(x) = \frac{1}{5}\sum_{i=1}^{5} N(\mu_i, \Sigma_i)$, with 5000 points from each component of the mixture, and used to compute an estimate of the PDF, where

$$\mu_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \mu_2 = \begin{pmatrix} -3 \\ -3 \\ 0 \end{pmatrix}, \mu_3 = \begin{pmatrix} 3 \\ 3 \\ 0 \end{pmatrix}, \mu_4 = \begin{pmatrix} 3 \\ -3 \\ 0 \end{pmatrix}, \mu_5 = \begin{pmatrix} -3 \\ 3 \\ 0 \end{pmatrix},$$

and

$$\Sigma_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \Sigma_i = \begin{pmatrix} 0.9 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 1 \end{pmatrix}, i = 2, 3, 4, 5$$

Unlike the 3-dimensional normal distribution, there is only one convenient plane to use for examining the marginal distribution, the xy-plane. To overcome this handicap, three dimensional visualizations of the marginal distributions for cross sections through the data using planes parallel to the xy-plane and the yz-plane were generated. These visualizations of the estimated PDF used planar cross sections through the data perpendicular to the x-axis and perpendicular to the z-axis. The Visualizations are MPEG-4 video files and were too large to include in this document, but they are available with the `bsspdfest` MATLAB toolbox and on the internet at Mixture density visualization x-axis and Mixture density visualization z-axis.

Generate the data and the evaluation points.

```
neval    = 101;
nsample  = 5000;
ndata    = 5*nsample;
xmu      = [0 0 0; -3 -3 0; 3 3 0; 3 -3 0; -3 3 0];
xcov0    = [0.9 0 0; 0 0.3 0; 0 0 1];
xcov3    = [0.9  0 0;  0 0.3 0; 0 0 1];
chxcov0  = cholcov(xcov0);
chxcov3  = cholcov(xcov3);
evalbnds = repmat([-10 10],3,1);

xtest3  = [
    repmat(xmu(1,:),nsample,1) + randn(nsample,3)*chxcov0;
    repmat(xmu(2,:),nsample,1) + randn(nsample,3)*chxcov3;
```

```
    repmat(xmu(3,:),nsample,1) + randn(nsample,3)*chxcov3;
    repmat(xmu(4,:),nsample,1) + randn(nsample,3)*chxcov3;
    repmat(xmu(5,:),nsample,1) + randn(nsample,3)*chxcov3;
    ];

xx3 = [
    linspace(evalbnds(1,1),evalbnds(1,2),neval);
    linspace(evalbnds(2,1),evalbnds(2,2),neval);
    linspace(evalbnds(3,1),evalbnds(3,2),neval);
    ]';
[xx3mg1, xx3mg2, xx3mg3] = ndgrid(xx3(:,1),xx3(:,2),xx3(:,3));
xx3mg = [xx3mg1(:) xx3mg2(:) xx3mg3(:)];
```

Evaluate the PDF estimate and compute the true three dimensional mixture PDF.

```
bss3test = bsspdfest(xtest3);
y3mg     = bsseval(xx3mg,bss3test,'pdf');
y3       = reshape(y3mg,size(xx3mg1));
y3nmg    = zeros(length(xx3mg),1);
for i = 1:5
   if (i == 1)
      y3nmg = y3nmg + (1/5)*mvnpdf(xx3mg,xmu(i,:),xcov0);
   else
      y3nmg = y3nmg + (1/5)*mvnpdf(xx3mg,xmu(i,:),xcov3);
   end
end
y3n = reshape(y3nmg,size(xx3mg1));
```

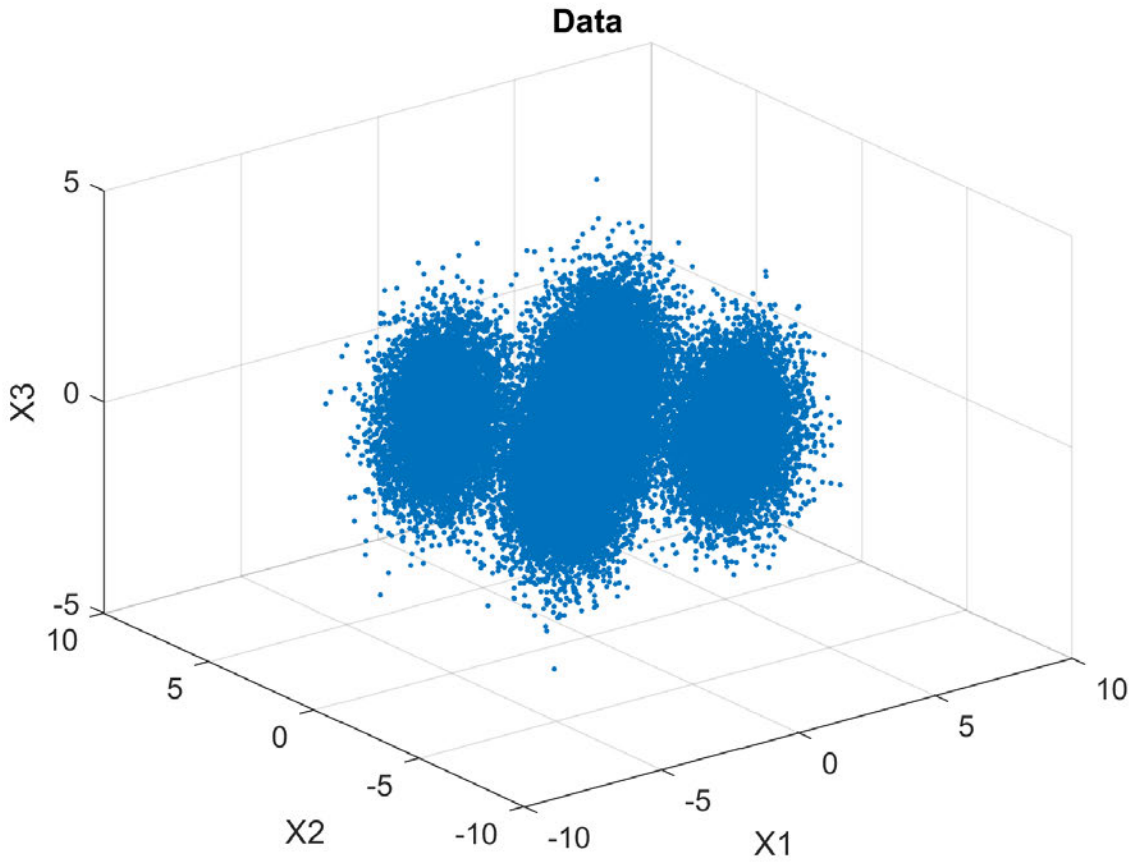


Figure 3.38: 25000 data points generated from a mixture of 5 3-dimensional normal distributions.

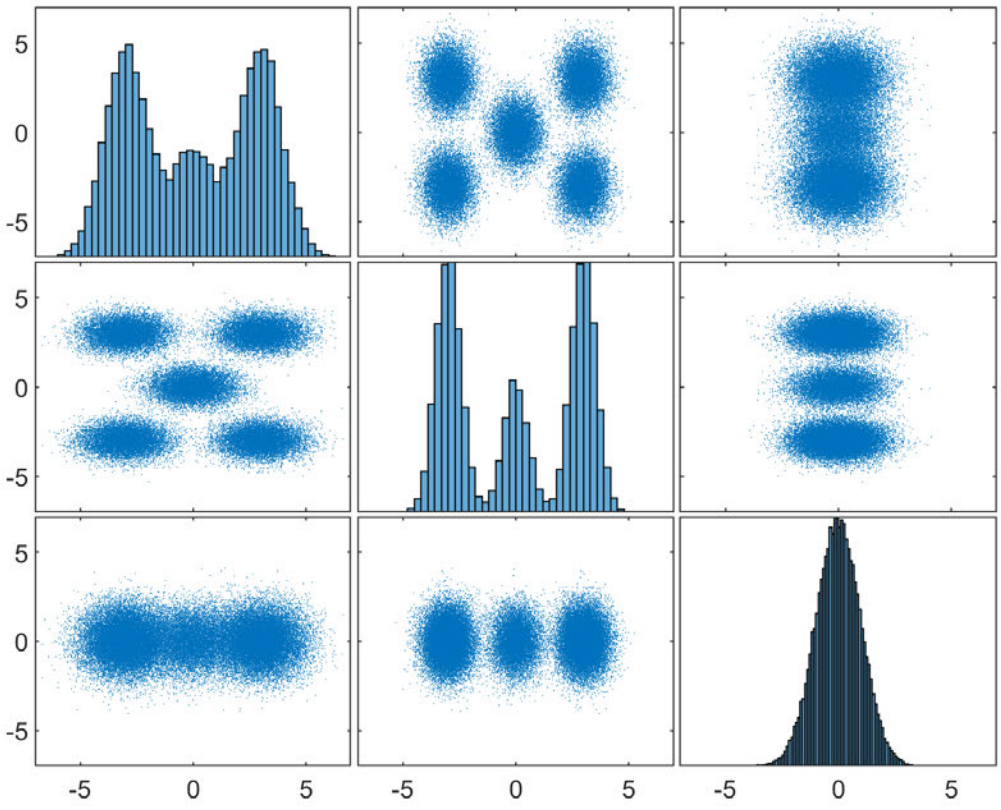


Figure 3.39: MATLAB plotmatrix of the 25000 data points generated from a mixture of 5 3-dimensional normal distributions.

3.4 Bounded domain examples

When using a B-spline series approximation to a probability function, in particular a PDF, a bounding box specifying the rectilinear boundaries of a finite estimation domain must be defined in order to generate the uniformly spaced partition for each dimension. The limits of the bounding box are typically defined so that the estimated PDF gradually becomes zero at a rate controlled by the order m of the B-spline basis functions used in a particular series. For $m > 1$ the support of the estimated PDF, the region where the estimated PDF is positive, can extend beyond the boundaries of the bounding box specifying the domain of the function. For a PDF that approaches zero smoothly as values of x increase or decrease, this poses no difficulties: the limits of a bounding box may be increased or decreased to accommodate the smooth approach to a zero value.

If, however, there is an active boundary where there is a discontinuity of the PDF, being positive inside the bounding box and zero outside the bounding box, having positive values outside the box becomes problematic, for example a uniform distribution or an exponential distribution. For $m = 1$ the B-spline series produces a histogram estimate of a PDF, and the limits of the bounding box exactly specify the domain of the estimated PDF, and all of the boundaries are considered to be active, even if the estimated PDF has a value of zero: this simply means that the support of the underlying PDF may be a strict subset of the domain specified by the bounding box. For a B-spline series having an order $m > 1$ and an active boundary, a boundary correction must be performed to eliminate the positive values outside the active boundary in order to reduce the bias at the boundary.

When using a B-spline series to estimate a function it is possible to automatically detect active boundaries. When an active boundary is detected, reflection through the boundary is used to adjust the B-spline series coefficients at the boundary (Jones, 1993). The automatic detection and correction of active boundaries for a B-spline series PDF estimate has been implemented for all dimensions. For two- and three-dimensional data the reflection based corrections are also made for corners and edges where multiple active boundaries intersect. For dimensions greater than three, the reflection based correction is applied for active boundaries only, without additional corrections for the intersection of multiple active boundaries.

3.4.1 One-dimensional bounded domains

Three one-dimensional examples of probability density function estimation on bounded domains are presented: a uniform distribution in Section 3.4.1, an exponential distribution in Section 3.4.1, and a truncated normal distribution in Section 3.4.1. These examples use functions from the MATLAB Statistics and Machine Learning Toolbox.

Uniform distribution

For the first example, $n = 100000$ data points are generated from a uniform distribution $U(0, 1)$ and used to compute two estimates of the PDF: one with a bounded domain restricted to the interval $[0, 1]$ (`bssu1`) and one with an unbounded domain (`bssunb1`). A large sample size was used to fill the domain of the underlying distribution with data points to demonstrate the edge effects at the boundaries. The MATLAB code used to generate the data and perform the PDF estimation is provided, as well as the code for the evaluation points and the evaluated B-spline series. Results are plotted for each PDF estimate comparing it to the actual function along with the a plot of the estimation errors.

Generate the data and evaluation points.

```
rng('default');
xu1  = rand(100000,1);
xxu1 = linspace(-.1,1.1,5001)';
```

Compute the PDF estimate, evaluate it, and compute the true standard normal PDF.

```
bssunb1 = bsspdfest(xu1);
bssu1   = bsspdfest(xu1,[0 1]);
yut1    = unifpdf(xxu1);
yu      = bsseval(xxu1,bssu1);
yunb1   = bsseval(xxu1,bssunb1);
```

Note the poor fit near the corners of the uniform distribution in Figure 3.40. Since the default B-spline order, $m = 4$, was used to produce cubic B-splines it will be nearly impossible to get a good fit in the corners: The cubic splines must curve. It is possible, however, to improve the fit by using a smaller B-spline order, $m = 1$ for a histogram estimate or $m = 2$ for a linear estimate using "hat" functions. The uniform distribution estimation is repeated using

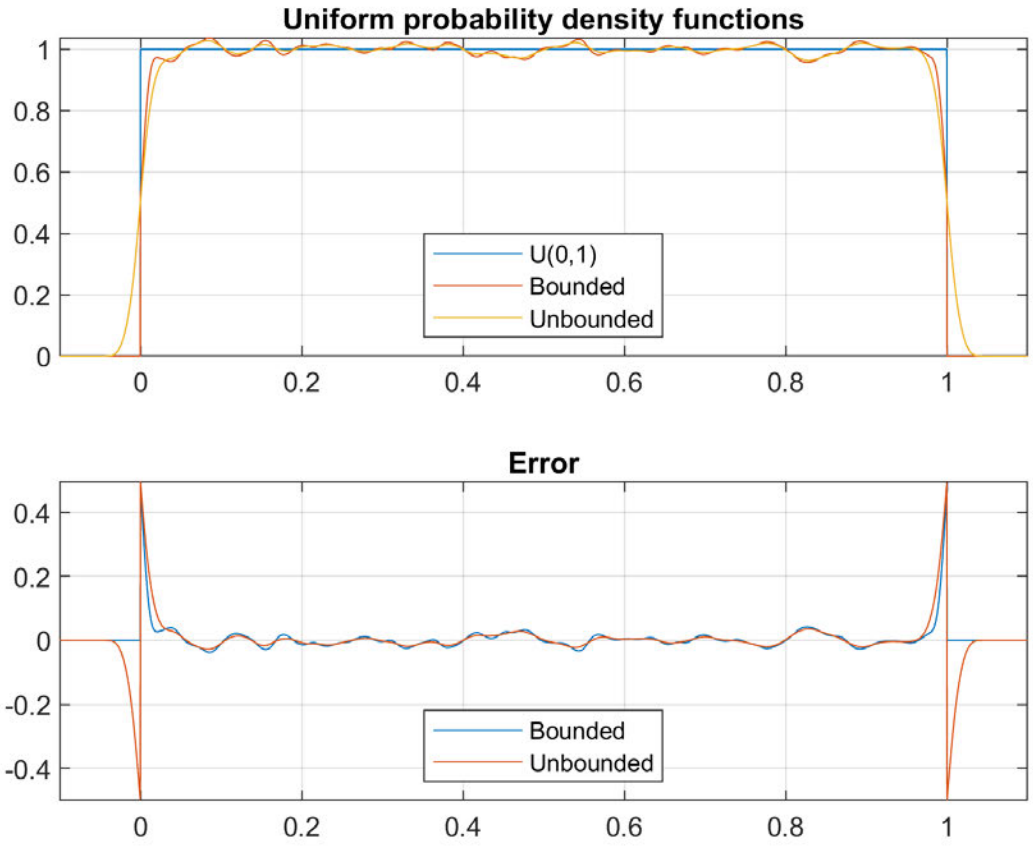


Figure 3.40: Estimate of a 1-dimensional uniform distribution $U(0,1)$. Estimated and true PDFs (top) and errors (bottom).

the B-spline order $m = 1$. The different partition sizes were used to make the partition sub-intervals for the bounded and unbounded estimations nearly equal.

```
bssunb1 = bsspdfest(xu1,[],17,1);
bssu1   = bsspdfest(xu1,[0 1],11,1);
yut1    = unifpdf(xxu1);
yu      = bsseval(xxu1,bssu1);
yunb1   = bsseval(xxu1,bssunb1);
```

The results of using the histogram estimate are given in Figure 3.41, and clearly indicate the better performance of the histogram and the better fit in the corners.

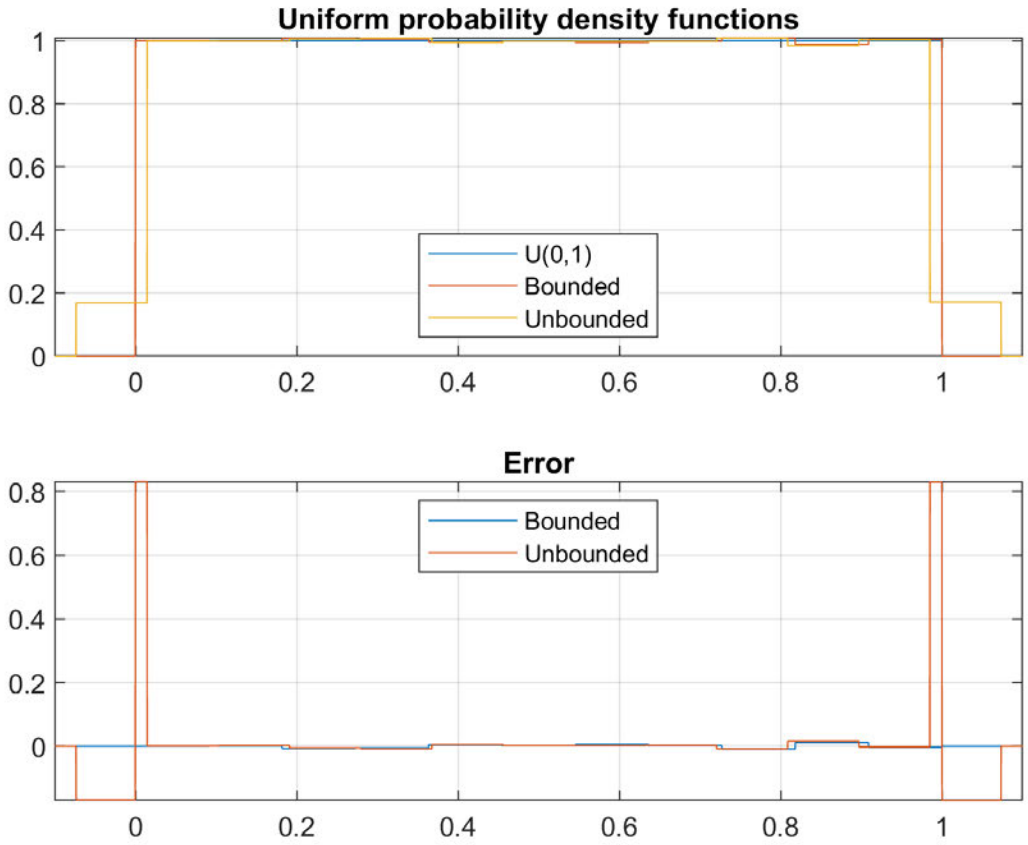


Figure 3.41: Estimate of a 1-dimensional uniform distribution $U(0,1)$ using a B-spline order of $m = 1$ to produce a histogram. Estimated and true PDFs (top) and errors (bottom).

Exponential distribution

For the second example, $n = 100000$ data points are generated from an exponential distribution with mean $\mu = 2$, $f(x) = \frac{1}{2}\exp(-x/2)$, and used to compute two estimates of the PDF: one with a bounded domain restricted to the interval $[0, \infty)$ (`bsse1`) and one with an unbounded domain (`bssenb1`). A large sample size was again used to fill the domain of the underlying distribution with data points to demonstrate the edge effects at the boundary. The MATLAB code used to generate the data and perform the PDF estimation is provided, as well as the code for the evaluation points and the evaluated B-spline series. Results are plotted for each PDF estimate comparing it to the actual function along with the a plot of the estimation errors.

Generate the data and evaluation points.

```
rng('default');
xe1  = exprnd(2,100000,1);
xxe1 = linspace(-1,22,5001)';
```

Compute the PDF estimate, evaluate it, and compute the true standard normal PDF.

```
bsse1   = bsspdfest(xe,[0 22]);
bssenb1 = bsspdfest(xe,[-1 22]);
yet1    = exppdf(xxe,2);
ye1     = bsseval(xxe,bsse);
yenb1   = bsseval(xxe,bssenb);
```

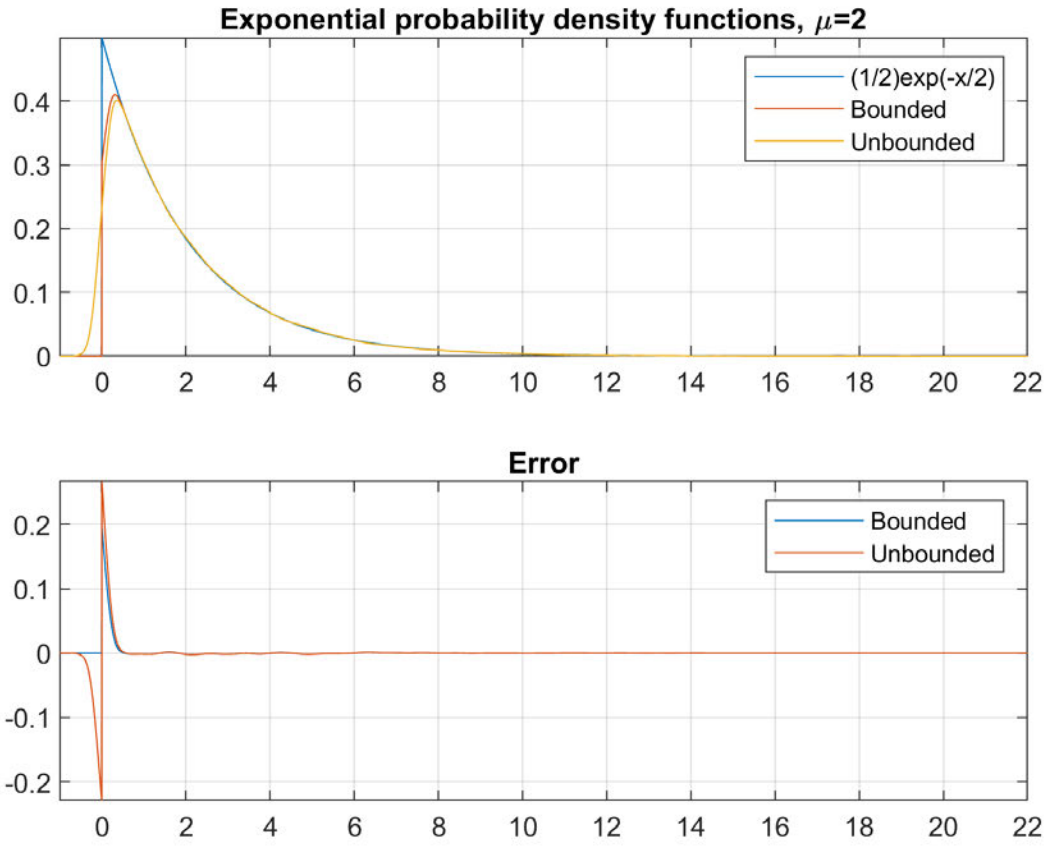


Figure 3.42: Estimate of a 1-dimensional exponential distribution with mean $\mu = 2$ and n = 100000. Estimated and true PDFs (top) and errors (bottom).

Truncated normal distribution

For the third example, $n = 100000$ data points are generated from a standard normal distribution and filtered to obtain 91113 points from a truncated normal distribution on the interval $[-2, 1.5]$, $N(0, 1, -2, 1.5)$. The 91113 points were then used to compute two estimates of the PDF: one with a bounded domain restricted to the interval $[-2, 1.5]$ (`bsstn1`) and one with an unbounded domain (`bsstnnb1`). A large sample size was used to fill the domain of the underlying distribution with data points to demonstrate the edge effects at the boundary. The MATLAB code used to generate the data and perform the PDF estimation is provided, as well as the code for the evaluation points and the evaluated B-spline series. Results are plotted for each PDF estimate comparing it to the actual function along with the a plot of the estimation errors.

Generate the data and evaluation points.

```
rng('default');
xtn1     = randn(100000,1);
xtn1     = xtn1(xtn1>-2.0&xtn1<1.5);
xxtn1    = linspace(-2.1,1.6,5001)';
inbounds = xxtn1>-2.0 & xxtn1<1.5;
xxtnt1   = xxtn1(inbounds);
```

Compute the PDF estimate, evaluate it, and compute the true standard normal PDF.

```
bsstn1          = bsspdfest(xtn1,[-2.0 1.5]);
bsstnnb1        = bsspdfest(xtn1);
ytnt1           = zeros(size(xxtn1));
ytnt1(inbounds) = normpdf(xxtnt1);
ytnt1(inbounds) = ytnt1(inbounds)./trapz(xxtnt1,ytnt1(inbounds));
ytn1            = bsseval(xxtn1,bsstn1);
ytnnb1          = bsseval(xxtn1,bsstnnb1);
```

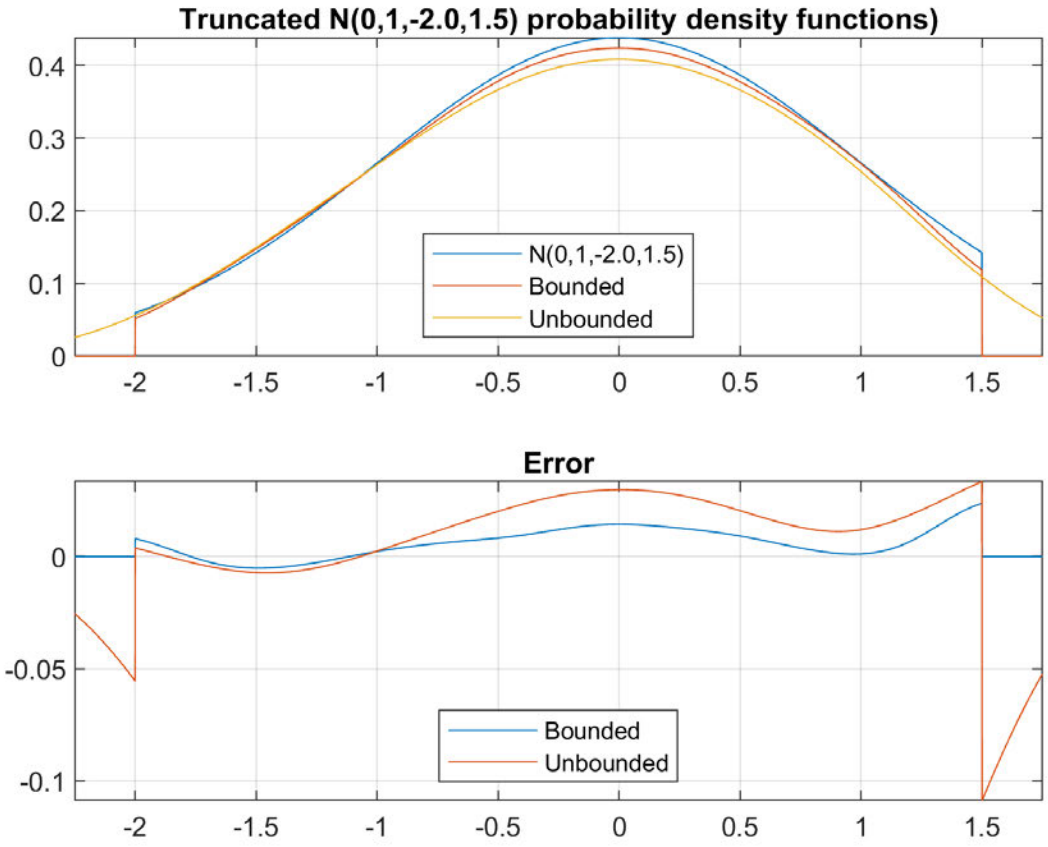


Figure 3.43: Estimate of a 1-dimensional truncated normal distribution with interval $[-2, 1.5]$, $n = 91113$. Estimated and true PDFs (top) and errors (bottom).

3.4.2 Two-dimensional bounded domains

Two two-dimensional examples of probability density function estimation on bounded domains are presented: a uniform distribution in Section 3.4.2 and a truncated normal distribution in Section 3.4.2. These examples use functions from the MATLAB Statistics and Machine Learning Toolbox.

Uniform distribution

For the first 2-dimensional example, $n = 200000$ data points are generated from a 2-dimensional uniform distribution on $[0, 1) \times [0, 1)$. The data are then used to compute two estimates of the PDF: one with a bounded domain restricted to the region $[0, 1) \times [0, 1)$ (`bssu2`) and one with an unbounded domain (`bssunb2`). The MATLAB code used to generate the data and perform the probability function estimation is provided. Results are plotted for each estimated PDF comparing the estimated function to the actual function with the estimation error. Notice in the error plots that the largest magnitude errors occur at the boundary and corners for the unbounded PDF estimate and where the curvature of the underlying probability density function is changing most rapidly. These are the regions that are typically the most difficult to fit.

Generate the data and evaluation points.

```
rng('default')
xu2           = rand(200000,2);
xx1u          = linspace(-0.1,1.1,101)';
xx2u          = linspace(-0.1,1.1,101)';
[xx1ug,xx2ug] = ndgrid(xx1u,xx2u);
xxu2          = [xx1ug(:) xx2ug(:)];
```

Compute the PDF estimate, evaluate it, and compute the true two dimensional truncated normal PDF. The partition sizes of 23 and 31 were chosen for the bounded and unbounded PDF estimates to generate similar partition widths for a consistent comparison of the estimation errors.

```
bssu2   = bsspdfest(xu2,[0 1;0 1],[23 23]');
bssunb2 = bsspdfest(xu2,[],[31 31]');
yut2    = double(reshape( ...
```

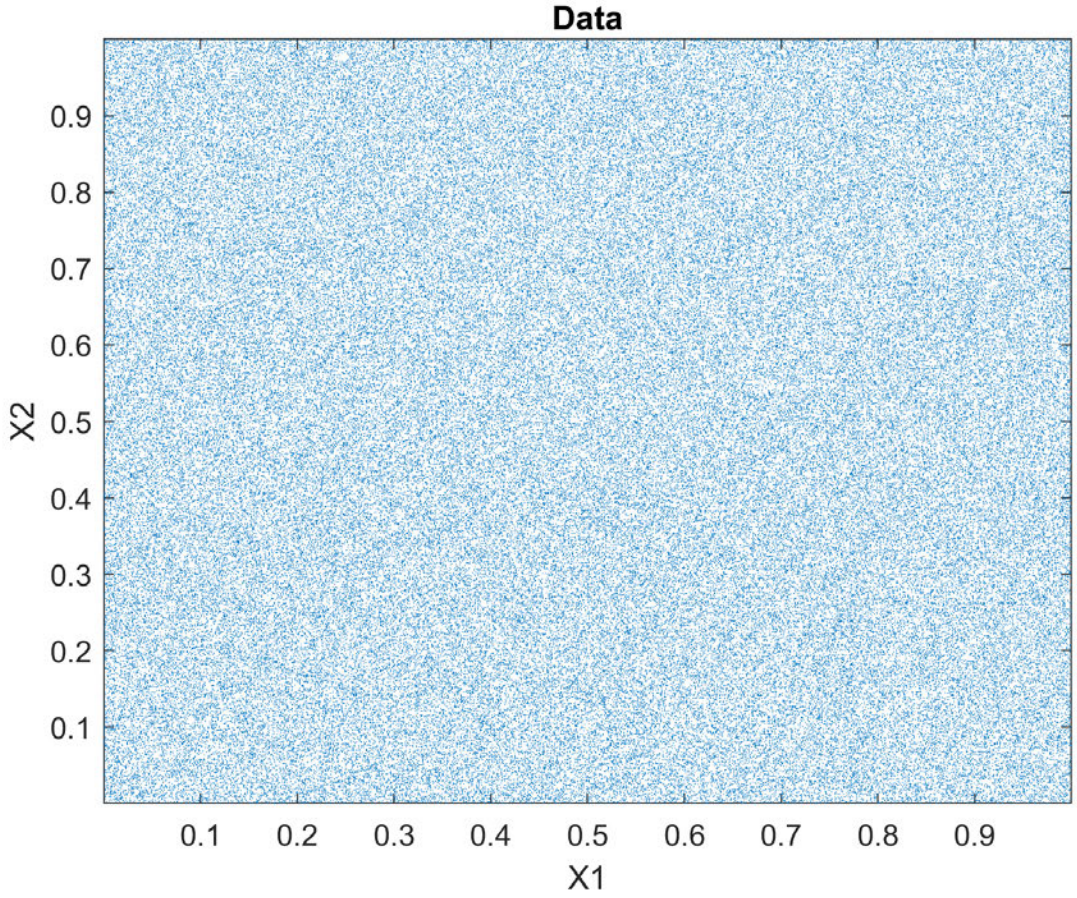


Figure 3.44: 200000 data points generated from a 2-dimensional uniform distribution.

```
         (xx1ug(:)>=0 & xx1ug(:)<1) & ...
         (xx2ug(:)>=0 & xx2ug(:)<1),size(xx1ug)));
yu2     = reshape(bsseval(xxu2,bssu2),size(xx1ug));
yunb2   = reshape(bsseval(xxu2,bssunb2),size(xx1ug));
```

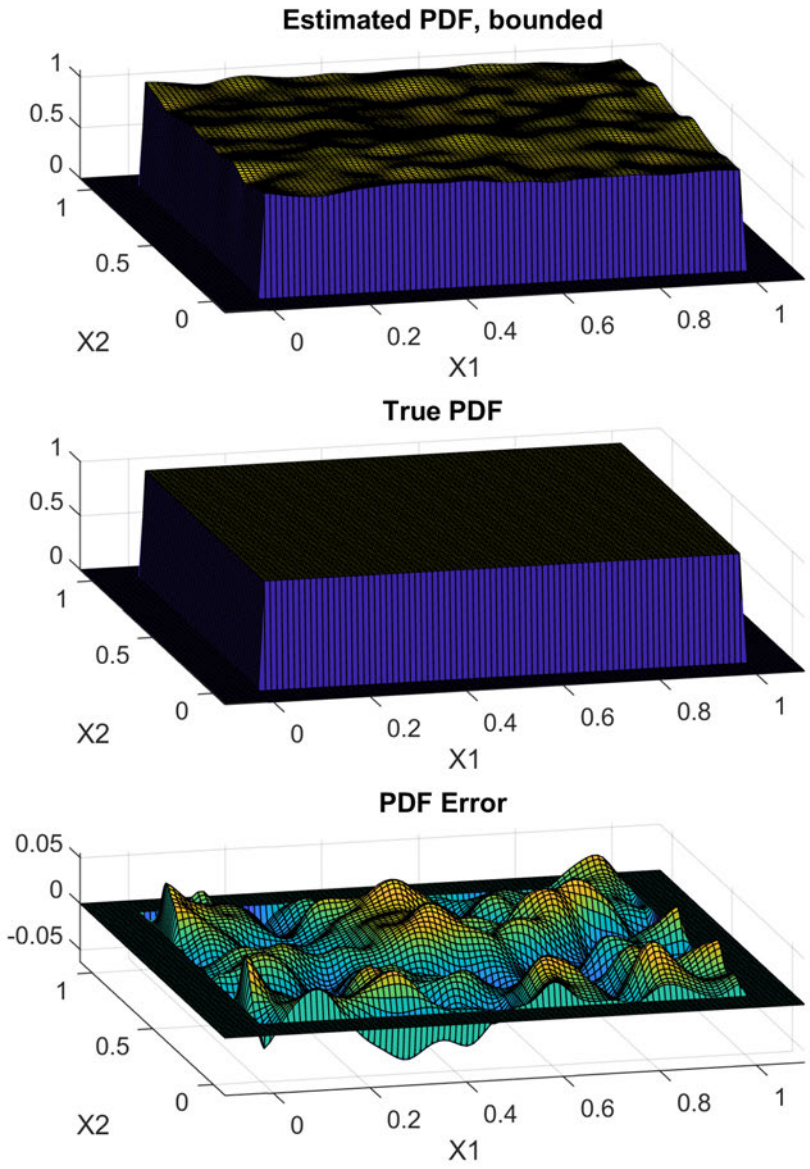


Figure 3.45: Estimate of a 2-dimensional uniform PDF using $n = 200000$ data points and estimation bounds restricted to the semi-infinite rectangle $[0, 1) \times [0, 1)$. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

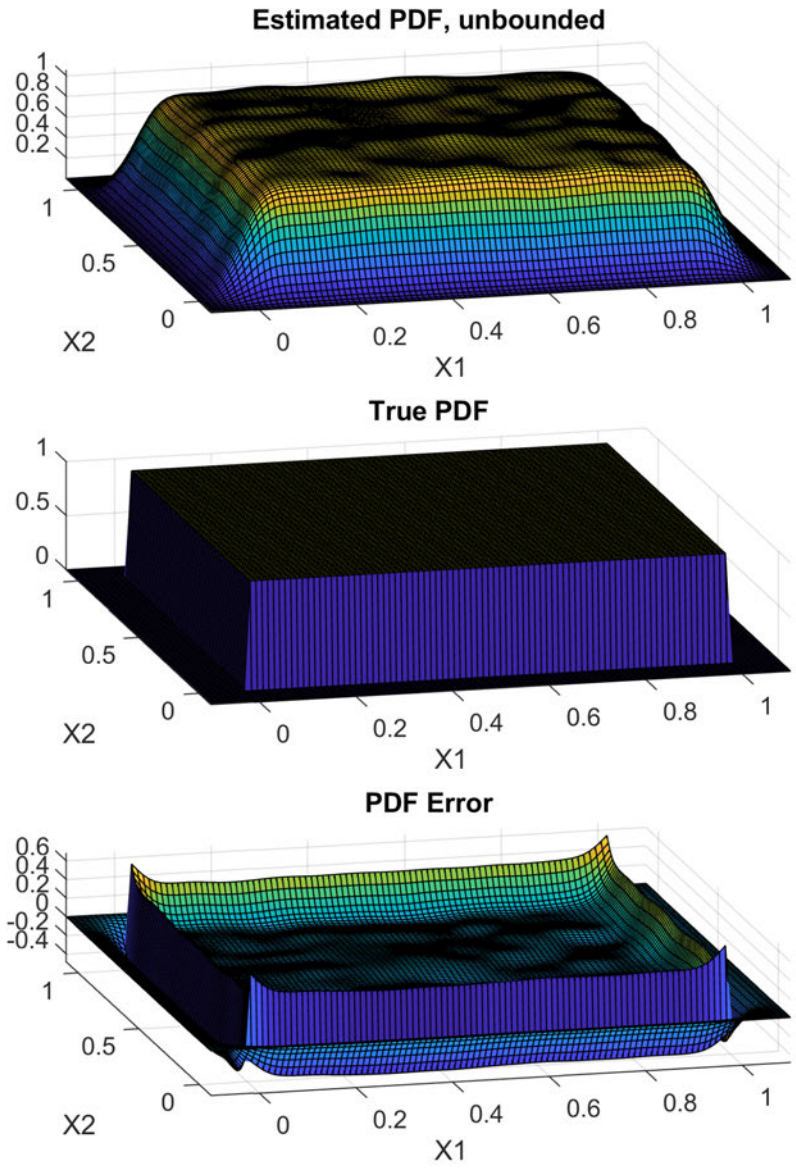


Figure 3.46: Estimate of a 2-dimensional uniform PDF using $n = 200000$ data points and unrestricted estimation bounds. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

Truncated normal distribution

For the second 2-dimensional example, $n = 200000$ data points are generated from a 2-dimensional standard normal distribution $N(0, I_2)$, where 0 is the 2-dimensional zero vector and I_2 is the 2×2 identity matrix, and filtered to obtain 174221 points from a truncated normal distribution having a domain defined by the semi-infinite rectangle $[-1.5, \infty) \times [-1.5, \infty)$. The data are then used to compute two estimates of the PDF: one with a bounded domain restricted to the region $[-1.5, \infty) \times [-1.5, \infty)$ (`bsstn2`) and one with an unbounded domain (`bsstnnb2`). The MATLAB code used to generate the data and perform the probability function estimation is provided. Results are plotted for each estimated PDF comparing the estimated function to the actual function with the estimation error. Notice in the error plots that the largest magnitude errors occur at the boundary for the unbounded PDF estimate and where the curvature of the underlying probability density function is changing most rapidly. These are the regions that are typically the most difficult to fit.

Generate the data and evaluation points.

```
rng('default');
xtn2            = randn(200000,2);
xtn2            = xtn2((xtn2(:,1)>-1.5)&(xtn2(:,2)>-1.5),:);
xx1tn2          = linspace(-2,5,101)';
xx2tn2          = linspace(-2,5,101)';
[xx1tng,xx2tng] = ndgrid(xx1tn2,xx2tn2);
xxtn2           = [xx1tng(:) xx2tng(:)];
inbounds        = (xxtn2(:,1)>-1.5) & (xxtn2(:,2)>-1.5);
xxtnt2          = xxtn2(inbounds,:);
```

Compute the PDF estimate, evaluate it, and compute the true two dimensional truncated normal PDF.

```
bsstn2          = bsspdfest(xtn2,[-1.5 5;-1.5 5],[31 31]');
bsstnnb2        = bsspdfest(xtn2);
ytnt2           = zeros(size(xxtn2,1),1);
ytnt2(inbounds) = mvnpdf(xxtnt2);
ytnt2           = ytnt2./(sum(ytnt2(:) ...
                  *prod([xx1tn2(2)-xx1tn2(1) xx2tn2(2)-xx2tn2(1)])));
ytnt2           = reshape(ytnt2,size(xx1tng));
ytn2            = reshape(bsseval(xxtn2,bsstn2),size(xx1tng));
```

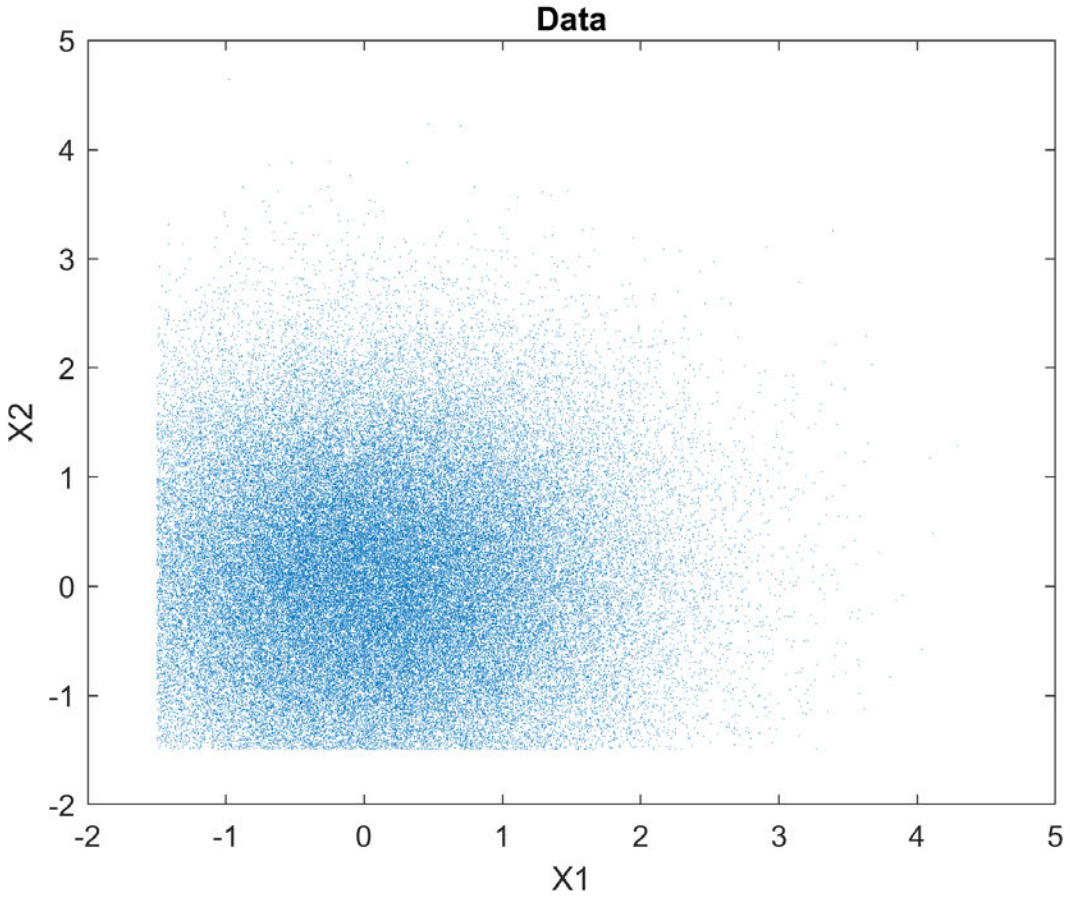


Figure 3.47: 174221 data points generated from a 2-dimensional truncated normal distribution.

```
ytnnb2          = reshape(bsseval(xxtn2,bsstnnb2),size(xx1tng));
```

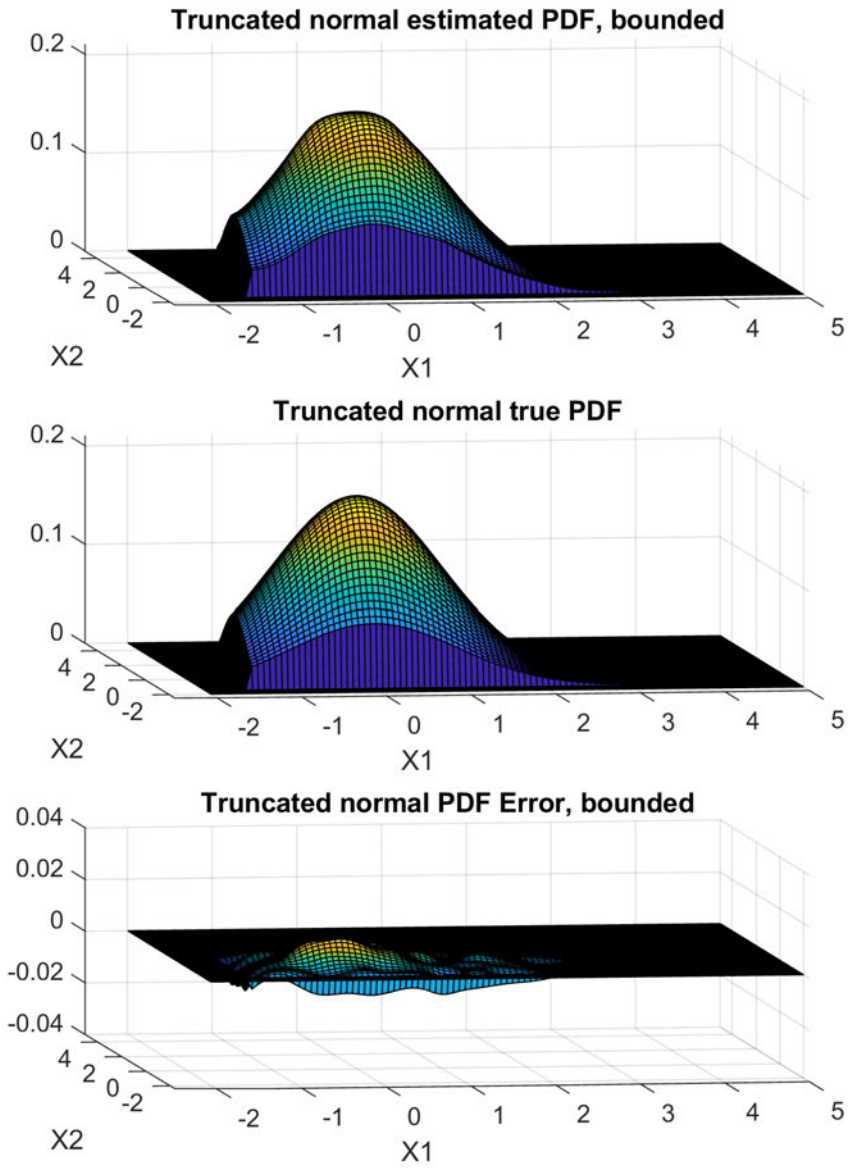


Figure 3.48: Estimate of a 2-dimensional truncated normal PDF using $n = 174221$ data points and estimation bounds restricted to the semi-infinite rectangle $[-1.5, \infty) \times [-1.5, \infty)$. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

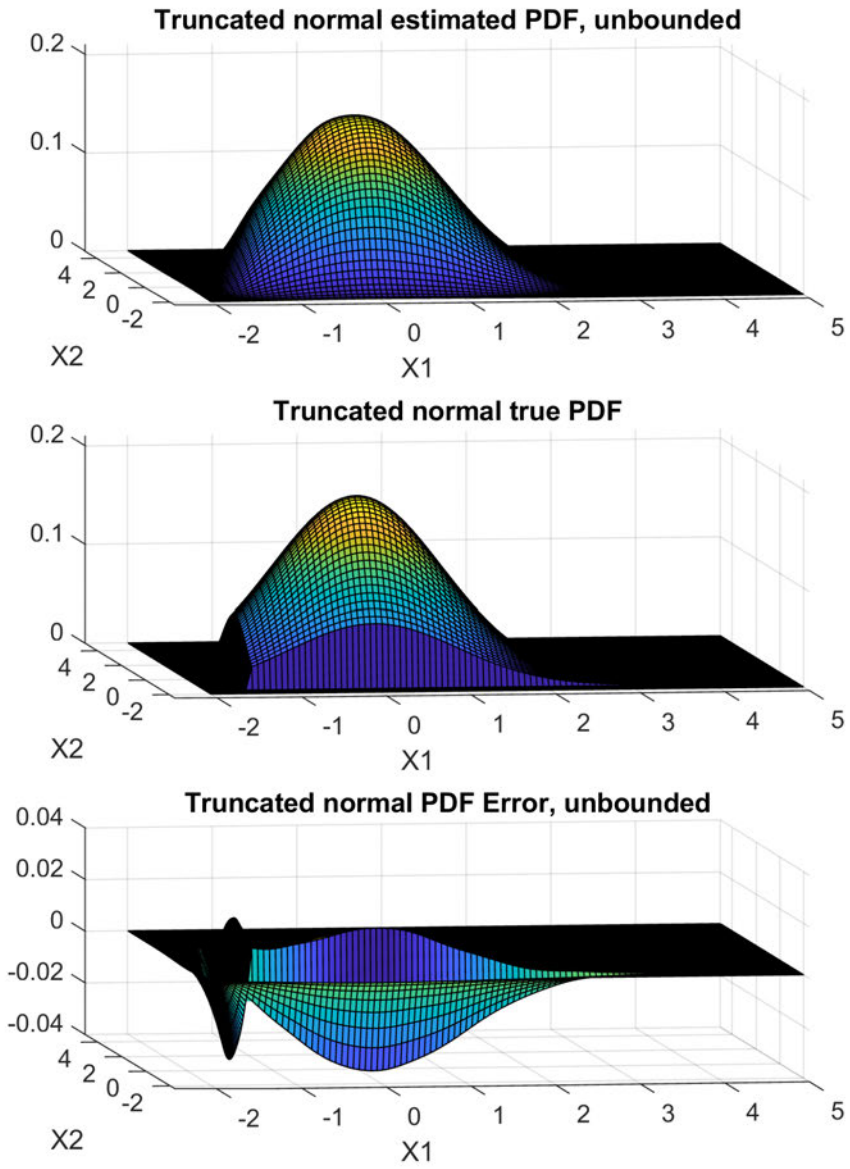


Figure 3.49: Estimate of a 2-dimensional truncated normal PDF using $n = 174221$ data points and unrestricted estimation bounds. Estimated PDF (top), true PDF (middle), and estimation error (bottom).

Reflection *vs.* Histogram correction at the boundaries

Previous versions of the `bsspdfest` toolbox, version 2.2.0 and 2.2.1, used a histogram boundary correction. A brief comparison of the new reflection boundary correction and the histogram boundary correction for a one-dimensional truncated normal distribution is presented to show the differences between the two approaches at the boundaries. First, generate the data and evaluation points and the true truncated normal PDF.

```
rng('default');
xtn1     = randn(100000,1);
xtn1     = xtn1(xtn1>-2.0&xtn1<1.5);
xxtn1    = linspace(-2.1,1.6,5001)';
inbounds = xxtn1>-2.0 & xxtn1<1.5;
xxtnt1   = xxtn1(inbounds);
ytnt1           = zeros(size(xxtn1));
ytnt1(inbounds) = normpdf(xxtnt1);
ytnt1(inbounds) = ytnt1(inbounds)./trapz(xxtnt1,ytnt1(inbounds));
```

Compute the truncated normal PDF estimates with the default partition size, using the new reflection based boundary correction and the old histogram based boundary correction, and evaluate the estimated PDFs.

```
% Estimate with reflection boundary correction
bsstn1          = bsspdfest(xtn1,[-2.0 1.5]);
ytn1            = bsseval(xxtn1,bsstn1);
% Estimate with histogram boundary correction
bsstnh1         = bsspdfest_h(xtn1,[-2.0 1.5]);
ytnh1           = bsseval(xxtn1,bsstnh1);
```

The results presented in Figure 3.50, at first glance, would seem to favor the histogram boundary correction since the reflection boundary correction produces larger magnitude errors at the boundary. However, the default partition size contains 89 subintervals and under-smooths the estimated PDFs, indicated by the large number of wiggles around the true truncated normal distribution. A consequence of this behavior is that there are more subintervals, and hence basis functions, near the boundaries, that allow the PDF estimated using the reflection boundary correction to decline rapidly near the boundary. This decline occurs outside the boundary correction zone, that is, far enough from the boundary that the

boundary has no impact. This is exactly what is supposed to occur: the reflection boundary correction is occurring, but only the partition subintervals, and basis functions, closest to the boundary are affected, and there are simply too many partition subintervals creating a very localized boundary correction. The steep decline outside the boundary correction zone is caused by the smaller amount of data available adjacent to the boundary correction zone.

Reducing the partition size from the default of 89 subintervals to 11 subintervals,

```
% Estimate with reflection boundary correction
bsstn1          = bsspdfest(xtn1,[-2.0 1.5],11);
ytn1            = bsseval(xxtn1,bsstn1);
% Estimate with histogram boundary correction
bsstnh1         = bsspdfest_h(xtn1,[-2.0 1.5],11);
ytnh1           = bsseval(xxtn1,bsstnh1);
```

see figure Figure 3.51, improves the situation for the reflection boundary correction. In fact, the reflection boundary correction even provides a reasonable approximation to the first derivatives of the truncated normal distribution at the boundaries, whereas the histogram boundary correction clearly does not. The reflection boundary correction reduces the bias at the boundary while providing better agreement with the first derivative of the estimated PDF at the boundaries provided a partition size that does not under-smooth is used. This partition size can be readily determined by trial and error.

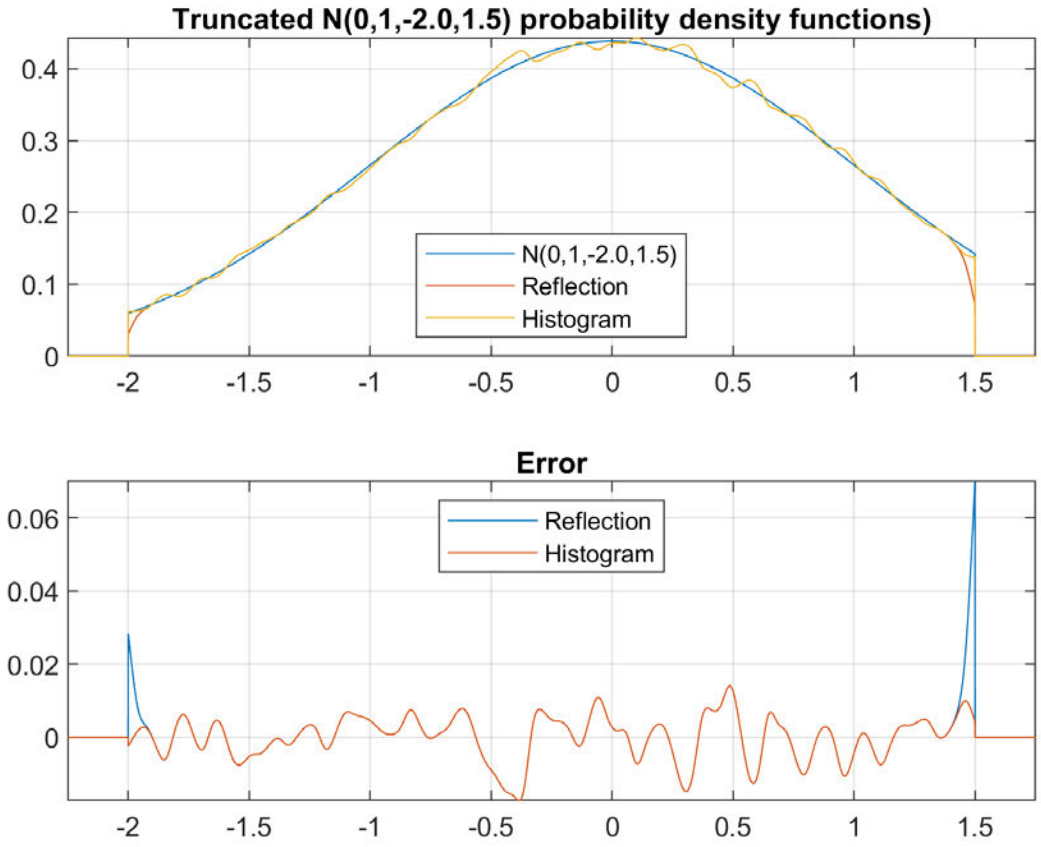


Figure 3.50: Estimate of a 1-dimensional truncated normal distribution with domain $[-2, 1.5]$ using the default partition size. Estimated and true PDFs (top) and errors (bottom).

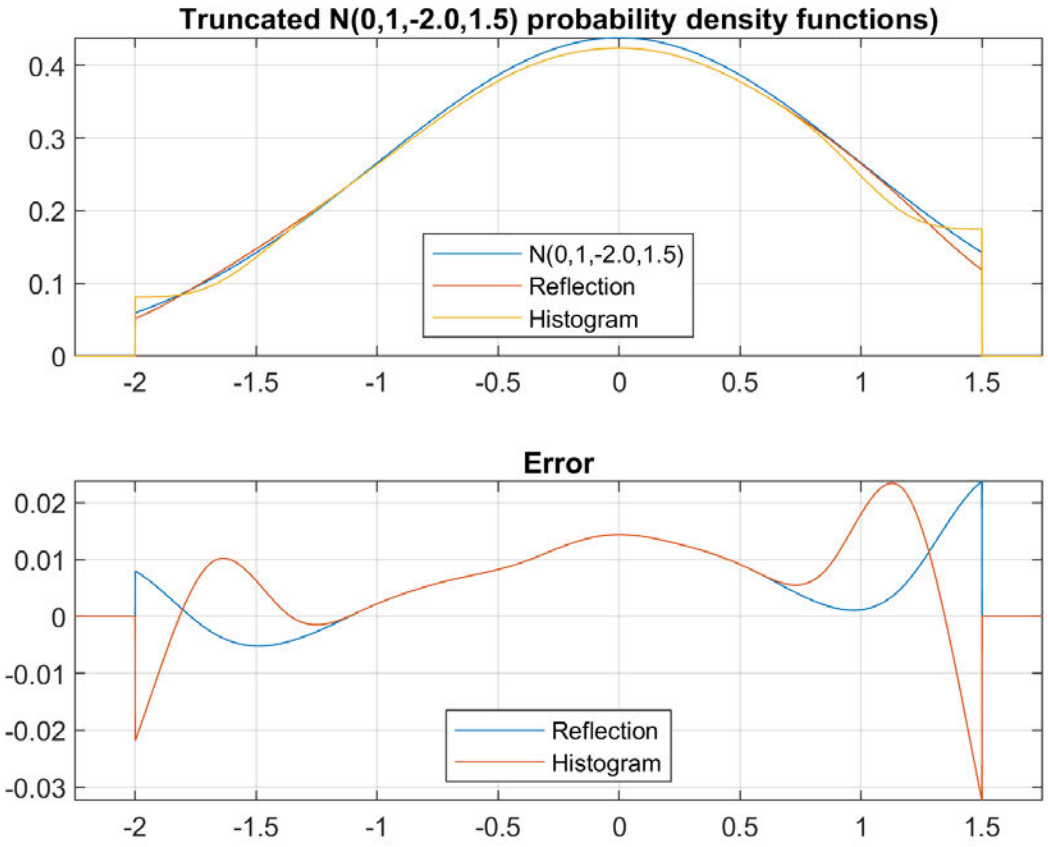


Figure 3.51: Estimate of a 1-dimensional truncated normal distribution with domain $[-2, 1.5]$ using a custom partition size. Estimated and true PDFs (top) and errors (bottom).

3.5 Efficient use of B-spline series estimators

The separate estimation (`bsspdfest`) and evaluation (`bsseval`) functions for B-spline series approximations may be used in several scenarios relating to the number of estimation and evaluation calls that are required to solve a problem. These usage scenarios are designated EE(n,m), where n and m are the numbers of estimation and evaluation calls that are necessary to solve a particular problem. The scenario EE(1,1) indicates that there is a one to one relationship between estimation calls and evaluation calls. This is comparable to the case for kernel based methods of probability density estimation where sample points and evaluation points are provided to a combined estimation/evaluation function. The scenario EE(1,m) indicates that there are m evaluation calls for each estimation call, potentially allowing a significant reduction in the computation time necessary to solve a problem. An example of this usage scenario would be a simulation requiring many likelihood values, but having a fixed, but unknown, underlying probability density function that must be estimated from a sample. The scenario EE(n,m) indicates that there are m evaluation calls for each of n estimation calls. An example of this usage scenario would be a simulation or other computation requiring many likelihood values, but having an unknown underlying probability density function that changes over time, or that is updated periodically with new data, and must be estimated from samples taken at different points in time.

The `bsseval` function also supports the creation of a gridded interpolant that is added to a B-spline series data structure for very fast approximate evaluation of a B-spline series. Three types of gridded interpolants may be created: `bss`, `pdf`, and `cdf`. The `bss` is the default and generates a gridded interpolant for fast evaluation of the function represented by a B-spline series. If the B-spline series represents a PDF then the `pdf` or `cdf` gridded interpolants may be created for fast evaluation of a PDF or CDFs. If a gridded interpolant exists in a B-spline series data structure it is used when evaluating the B-spline series for a specific probability function.

A gridded interpolant is created by using a second output argument with the `bsseval` function. The MATLAB code

```
[y,bssoutp] = bsseval(x,bssin,'pdf')
```

creates a gridded interpolant that is used to evaluate a PDF. For this example, the output B-spline series data structure `bssoutp` will be identical to the input B-spline series data structure `bssin` except for the addition of a new field `bssoutp.pdf`. A gridded interpolant for a CDFs can also be added to a B-spline series data structure by using `bssoutp` as

the input to `bsseval` and `bssoutpc` as the second output argument that will contain both gridded interpolants in the fields `bssoutp.pdf`and `bssoutp.cdf`.

```
[y,bssoutpc] = bsseval(x,bssoutp,'cdf')
```

This is, however, not a memory efficient way to rapidly compute both the PDF and CDFs when using a gridded interpolant. Instead, simply create the `pdf` gridded interpolant which will be used to compute the PDF or the CDFs. If the B-spline series represents a probability density function, the same results may be obtained by using the `bss` gridded interpolant. A gridded interpolant has usage scenarios GI(1,m) and GI(n,m) that are comparable to usage scenarios EE(1,m) and EE(n,m). The cost of creating a gridded interpolant makes a usage scenario equivalent to EE(1,1) impractical: it would take longer to compute than the EE(1,1) usage scenario.

To demonstrate the benefits of using a gridded interpolant, consider the problem of estimating and evaluating a standard normal distribution in one, two, and three dimensions for the EE(1,1) usage scenario repeated 250 times, the EE(10,25) and EE(1,250) usage scenarios, and the GI(10,25) and GI(1,250) usage scenarios. For each data dimension and usage scenario a common random sample was used to estimate the PDF. The PDF was than estimated and evaluated the specified number of times for each scenario. For the gridded interpolant scenarios, the PDF was estimated and a gridded interpolant for the `pdf` was created before the PDF was evaluated. Timing results, in seconds, for a variety of sample sizes ranging from 100 to 100000 points for each data dimension and using 1001, 5041, and 132651 evaluation points for the one, two, and three dimensional data are presented in Table 3.2, Table 3.3, and Table 3.4. Note the dramatic reduction in computation time for the GI(1,250) scenario relative to the EE(n,m) usage scenarios for the two and three dimensional problems. Everything is fast for one dimensional data, so the reductions in computation time for this case are not as dramatic, but are still quite respectable.

The two dimensional PDF produced from the B-spline series and its error are presented in Figure 3.52 and the PDF computed using the gridded interpolant and its error are given in Figure 3.53. Visually these two estimates are identical, but they are different, and Figure 3.54 is a plot of the difference between the B-spline series estimate and the gridded interpolant estimate.

Table 3.2: Times (seconds) to estimate a one dimensional standard normal PDF.

N_{data}	N_{eval}	EE(1,1)	EE(10,25)	EE(1,250)	GI(10,25)	GI(1,250)
100	1001	0.25	0.14	0.11	0.07	0.06
250	1001	0.28	0.12	0.10	0.07	0.06
500	1001	0.26	0.11	0.10	0.07	0.06
1000	1001	0.39	0.12	0.10	0.07	0.06
2500	1001	0.49	0.11	0.10	0.08	0.06
5000	1001	0.54	0.12	0.10	0.08	0.06
10000	1001	1.05	0.14	0.10	0.11	0.06
25000	1001	1.85	0.18	0.11	0.14	0.06
50000	1001	4.47	0.28	0.12	0.25	0.08
100000	1001	8.55	0.46	0.14	0.43	0.10

Table 3.3: Times (seconds) to estimate a two dimensional standard normal PDF.

N_{data}	N_{eval}	EE(1,1)	EE(10,25)	EE(1,250)	GI(10,25)	GI(1,250)
100	5041	3.82	3.09	3.86	0.28	0.12
250	5041	4.35	3.66	3.61	0.40	0.14
500	5041	4.86	4.19	4.90	0.49	0.13
1000	5041	5.29	4.52	4.87	0.70	0.15
2500	5041	6.70	5.40	5.27	1.27	0.22
5000	5041	7.71	5.84	5.73	1.97	0.27
10000	5041	9.88	6.50	6.36	3.07	0.39
25000	5041	14.03	7.28	7.15	5.56	0.63
50000	5041	21.54	7.76	7.09	8.81	0.97
100000	5041	36.84	8.37	7.30	14.32	1.51

Table 3.4: Times (seconds) to estimate a three dimensional standard normal PDF.

N_{data}	N_{eval}	EE(1,1)	EE(10,25)	EE(1,250)	GI(10,25)	GI(1,250)
100	132651	32.56	31.10	38.89	0.59	0.33
250	132651	42.45	42.15	41.89	0.95	0.41
500	132651	52.57	52.30	52.76	1.30	0.46
1000	132651	58.66	58.21	68.44	2.21	0.65
2500	132651	71.58	70.95	75.83	5.54	1.31
5000	132651	81.49	80.39	82.54	10.16	2.23
10000	132651	90.83	87.53	87.43	20.49	4.29
25000	132651	100.27	92.68	92.07	48.29	9.83
50000	132651	109.47	93.64	92.73	99.04	20.02
100000	132651	126.24	95.03	91.99	193.16	38.99

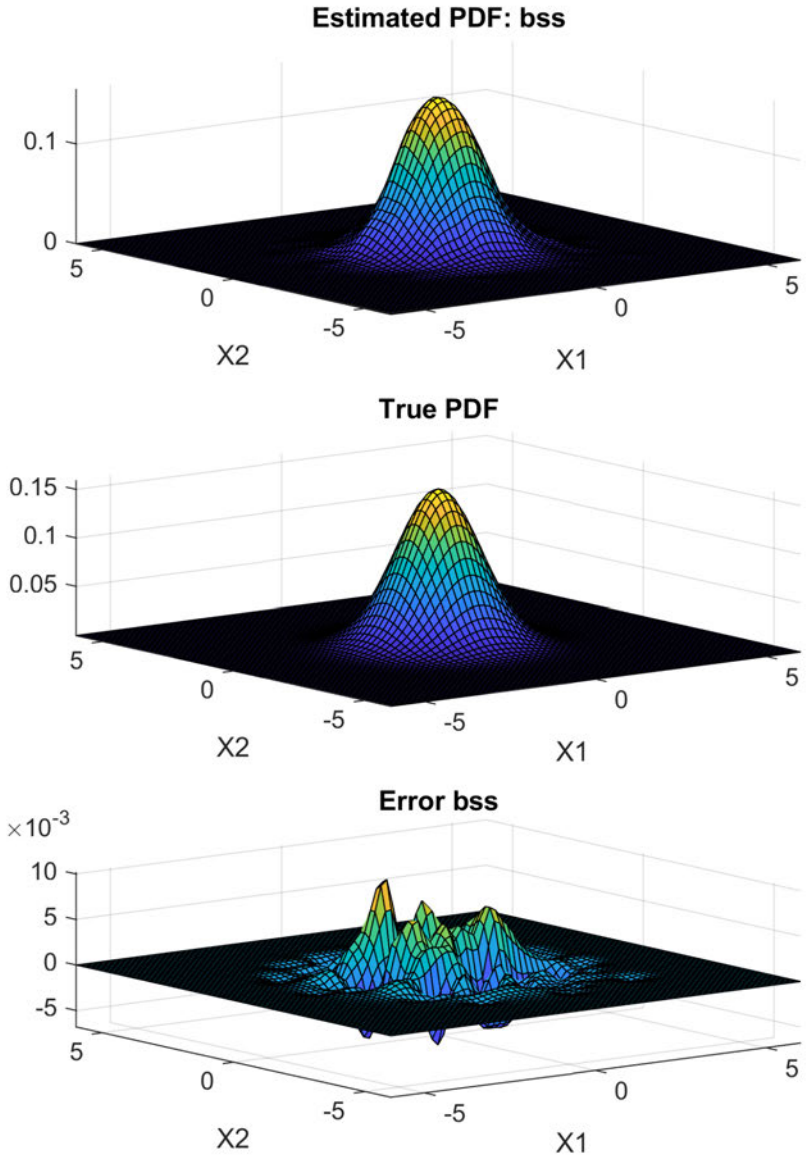


Figure 3.52: B-spline series estimate of a 2-D standard normal distribution, $n = 5000$.

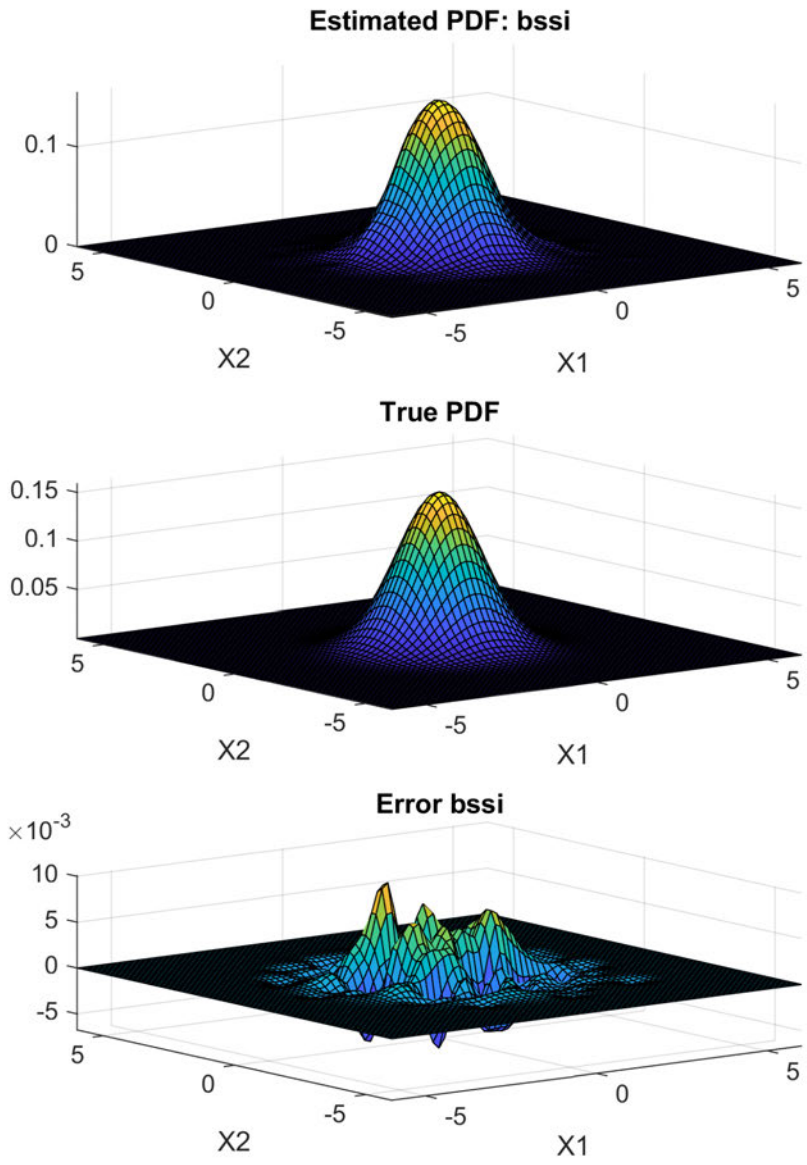


Figure 3.53: Gridded interpolant estimate of a 2-D standard normal distribution, $n = 5000$.

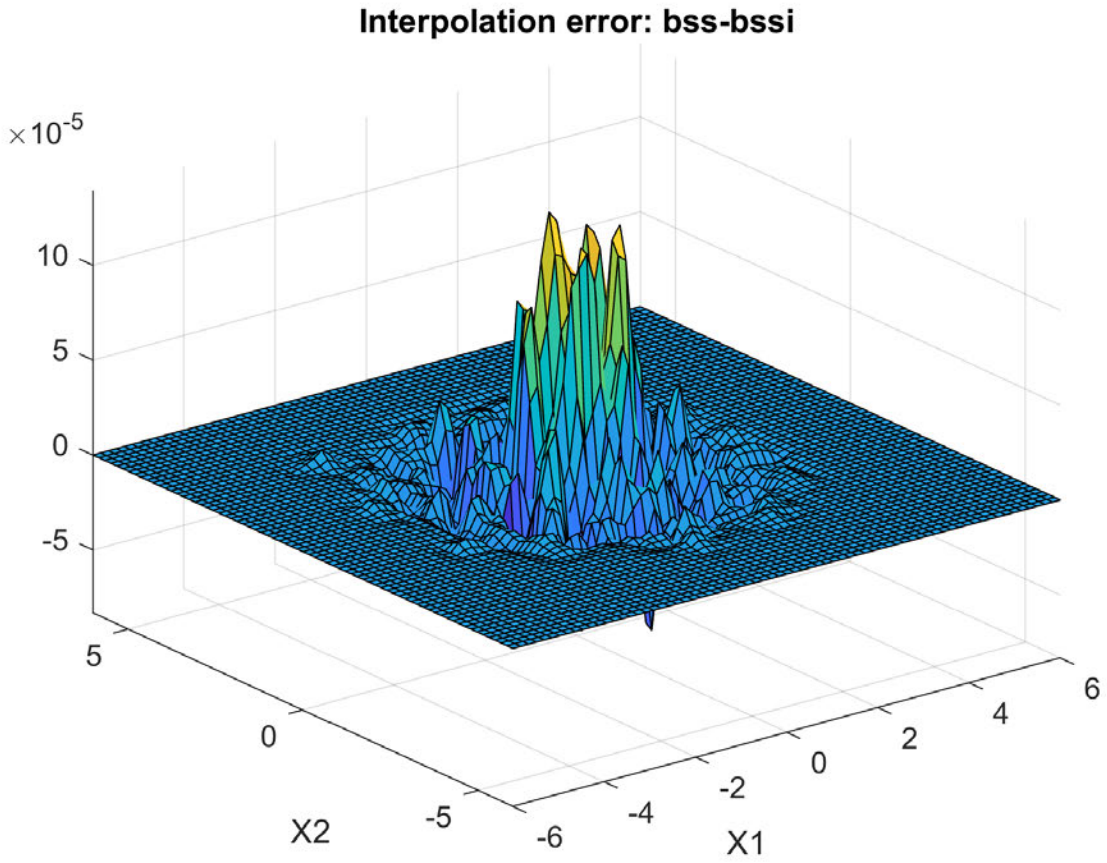


Figure 3.54: Difference between the B-spline series estimate and the gridded interpolant estimate of a 2-D standard normal distribution, $n = 5000$.

Bibliography

Gehringer, K. R. (1990). Nonparametric probability density estimation using normalized B-Splines. Master's thesis, The University of Tulsa.

Gehringer, K. R. and Redner, R. A. (1992). Nonparametric probability density estimation using normalized B-splines. *Comm. Statist. Simulation Comput.*, 21(3):849–878.

Jones, M. (1993). Simple boundary correction for kernel density estimation. *Statistics and Computing*, 3:135–146.

Redner, R. A. (1999). Convergence rates for uniform B-spline density estimators. I. One dimension. *SIAM J. Sci. Comput.*, 20(6):1929–1953 (electronic).

Redner, R. A. (2000). Convergence rates for uniform B-spline density estimators. II. multiple dimensions. *Nonparametric Statistics*, 12:753–777.

Redner, R. A. and Gehringer, K. (1994). Function estimation using partitions of unity. *Comm. Statist. Theory Methods*, 23(7):2059–2078.