probhat 0.4.1

Multivariate Generalized Kernel Smoothing and Related Statistical Methods

Abby Spurdle

May 12, 2021

Probability mass functions (PMFs), probability density functions (PDFs), cumulative distribution functions (CDFs) and quantile functions, mainly via (optionally bounded/truncated) kernel smoothing. In the continuous case, there's support for univariate, multivariate and conditional distributions, including distributions that are both multivariate and conditional. Refer to the book "Kernel Smoothing" by Wand and Jones (1995), whose methods are generalized by the methods here. Also, supports categorical distributions, mixed conditional distributions (with mixed input types) and smooth empirical-like distributions, some of which, can be used for statistical classification. There are extensions for computing distance matrices (between distributions), multivariate probabilities, multivariate random numbers, moment-based statistics and mode estimates.

Introduction

This is an R package for multivariate generalized kernel smoothing, as per the title.

Kernel smoothing is generalized, by estimating:

- Bounded random variables.
- Both discrete and continuous probability distributions.
- Probability mass functions (PMFs), probability density functions (PDFs), cumulative distribution functions (CDFs) and quantile functions (QFs).
- In the continuous case, multivariate and conditional distributions, including multivariate-conditional distributions.

Also, there are categorical distributions (univariate and conditional), smooth empirical-like distributions (univariate), and conditional distributions with mixed input types.

Discrete kernel smoothing, could also be described as frequency smoothing.

By default, univariate continuous kernel smoothing models use cubic Hermite splines as intermediate models. Corresponding quantile functions (which are spline only) are constructed by fitting a CDF using a spline, then transposing the control points.

Categorical variables are assumed to be ordinal, however, this assumption is only relevant to the interpretation of the CDF and QF. Empirical-like models are derived from empirical

cumulative distribution functions. There's a small modification to the standard formula, and the resulting points are interpolated by a cubic Hermite spline, in a similar way to univariate continuous kernel smoothing models.

A list of models and some notes on terminology, are given in appendices A and B.
Also, there are appendices with formulae, but they need some polishing.

All models can be given frequencies or weights, and there are examples of modelling fuzzy clusters with weighted multivariate kernel density estimation, in an appendix.

There are plot methods for all univariate models, and for multivariate models but only with two or three random variables.

Often the goal of kernel smoothing is simply to plot the distribution, as an exploratory tool, however, these models can be used for a variety of purposes.

Conditional categorical distributions with mixed input types, can be used for statistical classification purposes.

Also, there are extensions for computing distance matrices (between distributions), computing probabilities, random number generation and parameter-like estimation, including moment-based and mode estimates.

Notes:

- This package is based on S3-based self-referencing function objects.
  (Which are equivalent to the {d, p, q, r} approach used in R's stats package).
- IT'S LIKELY THAT THESE MAY CHANGE TO S4-BASED FUNCTION OBJECTS, IN THE NEAR FUTURE.
  (In principle, you should not access attributes/slots, directly).
- Bivariate/trivariate plotting functions use the barsurf package, which uses the base graphics system.
- Variable names are taken from column names (most cases) or list names (categorical distributions).
- This package gives precedence to vectors and matrices, over lists and data frames.
  (One partial exception is categorical data which allow a list of integer/factor/character vectors).
- This vignette contains a small amount of non-visible R code, to produce multi-panel plots.

Currently, there are some limitations:

- Bandwidth selection methods are simplistic.
- Parameter estimates, mixed conditional models (with mixed input), distribution sets and random number generation need improvement.

Preliminary Code

I'm going to load (and attach) the probhat, fclust and scatterplot3d packages:

```
> library (probhat)       #required
> library (fclust)        #optional, used in appendix
> library (scatterplot3d) #optional
```

Note that the probhat package imports the barsurf and kubik packages.

I will set global options for viewing PDF documents electronically, and default colors to

green:

```
> set.ph.options (rendering.style="pdf", theme="green")
```

And I will construct some data objects:

```
> prep.ph.data ()
```

This function emulates a script.
The script and the resulting datasets are given in the last two appendices.

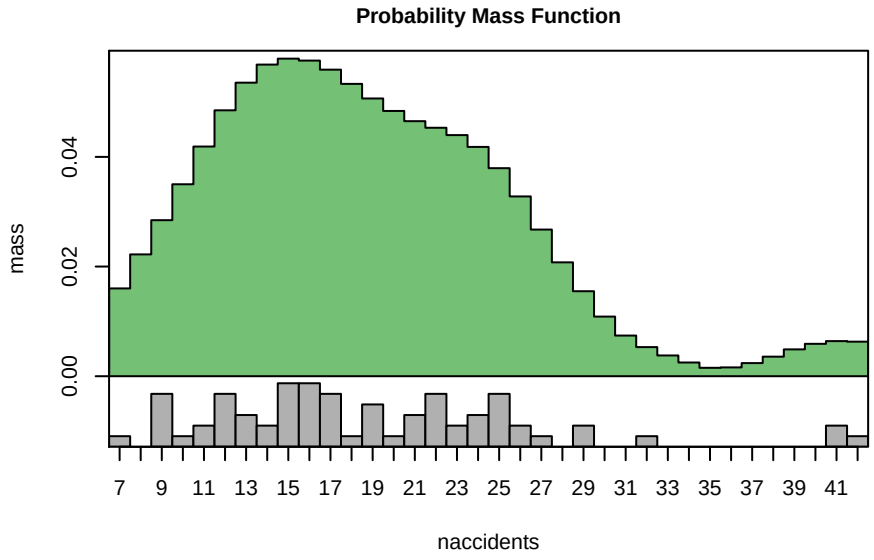
Discrete Kernel Smoothing

We can construct a $\text{PMF}_{(\text{UV})}$~DKS object, using the **pmfuv.dks** constructor.
Likewise, we can construct $\text{CDF}_{(\text{UV})}$~DKS and $\text{QF}_{(\text{UV})}$~DKS objects, using the **cdfuv.dks** and **qfuv.dks** constructors.

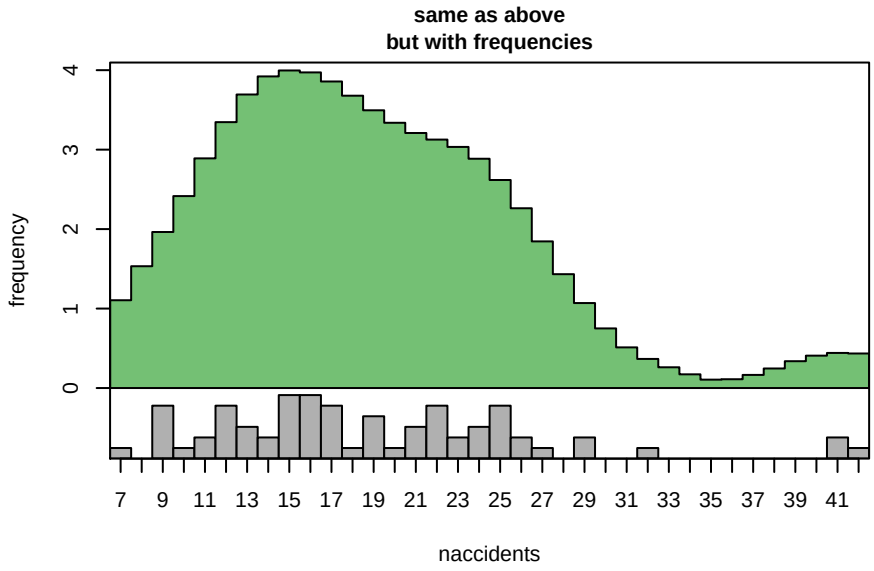
Here are examples, using traffic accident data, derived from the "Traffic" dataset in the MASS package:

```
> dfh <- pmfuv.dks (traffic.bins, traffic.freq)
> dFh <- cdfuv.dks (traffic.bins, traffic.freq)
> dFht <- qfuv.dks (traffic.bins, traffic.freq)
```

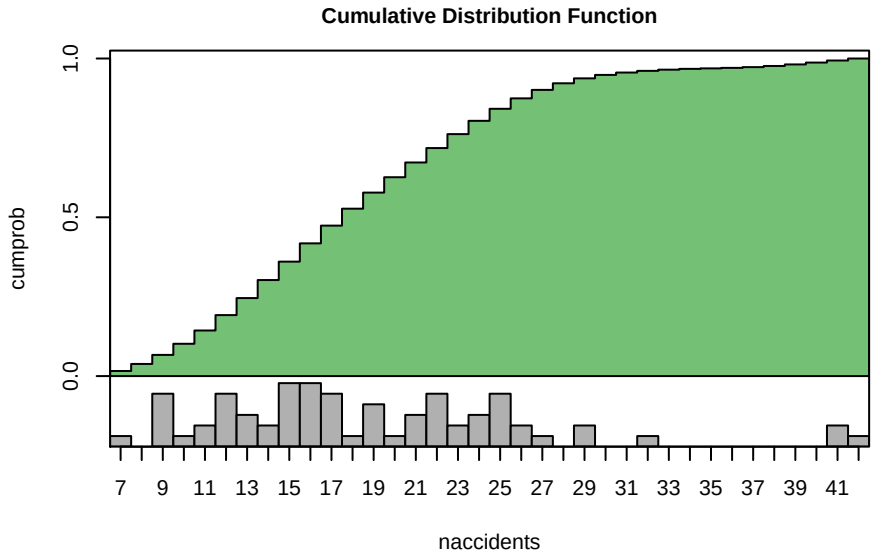
```
> plot (dfh, TRUE, main="Probability Mass Function")
```



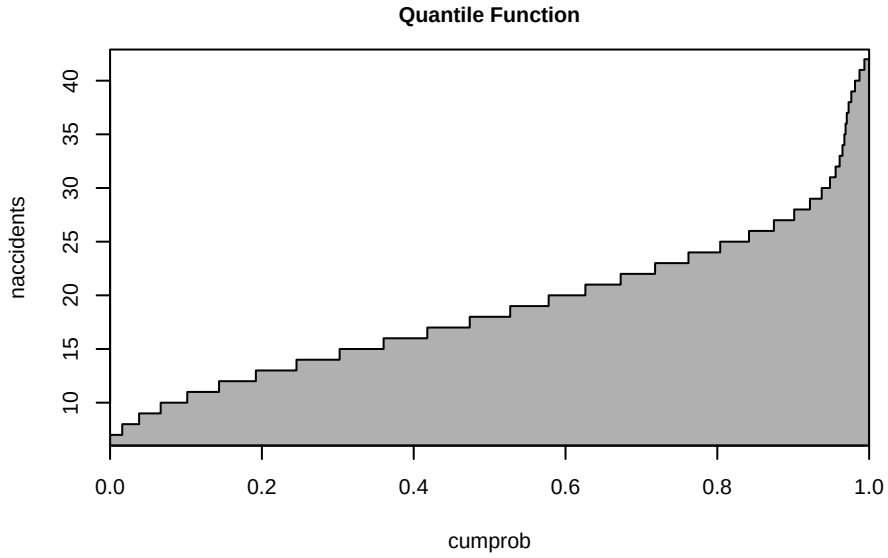

```
> plot (dfh, TRUE, freq=TRUE, main="same as above\nbut with frequencies")
```




```
> plot (dFh, TRUE, main="Cumulative Distribution Function")
```




```
> plot (dFht, main="Quantile Function")
```



Here, traffic.bins is a one-column matrix of integer-indexed bins, and traffic.freq is a vector of counts.

The resulting objects are function objects.
By default, the PMF maps an integer vector (of quantiles) to a numeric vector (of probabilities), the CDF maps quantiles to cumulative probabilities, and the QF maps probabilities to quantiles:

```
> dfh (10)

[1] 0.03500366

> dfh (10, freq=TRUE)

[1] 2.415253

> dFht (c (0.25, 0.5, 0.75) )

[1] 14 18 23
```

Note that PMFs can be constructed, plotted and evaluated with freq=TRUE, which results in frequencies.
Likewise, if constructed with freq=TRUE, they can be plotted and evaluated with freq=FALSE.

Also note that this dataset is not an ideal example.
(Essentially, it gives frequencies of frequencies).

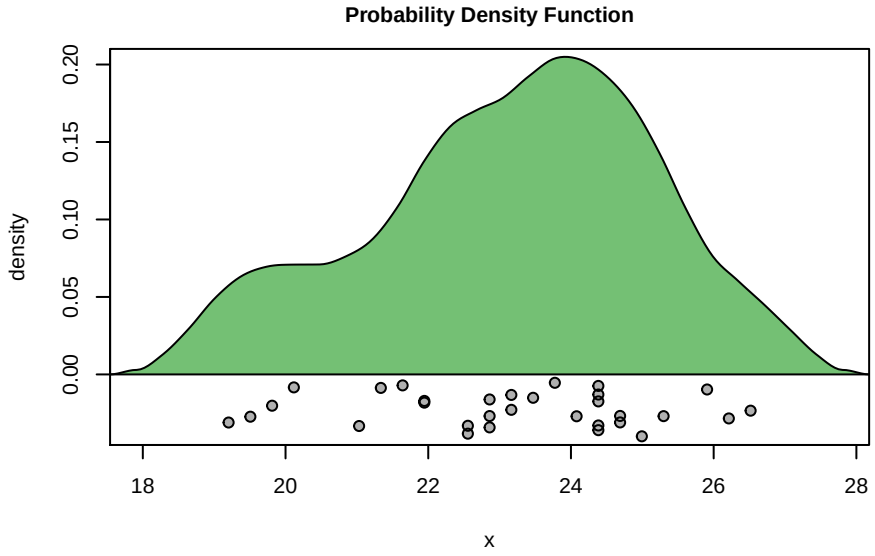
Continuous Kernel Smoothing: Univariate Probability Distributions

We can construct a $\text{PDF}_{(\text{UV})}$~CKS object, using the **pdfuv.cks** constructor.
Likewise, we can construct $\text{CDF}_{(\text{UV})}$~CKS and $\text{QF}_{(\text{UV})}$~CKS objects, using the **cdfuv.cks** and **qfuv.cks** constructors.

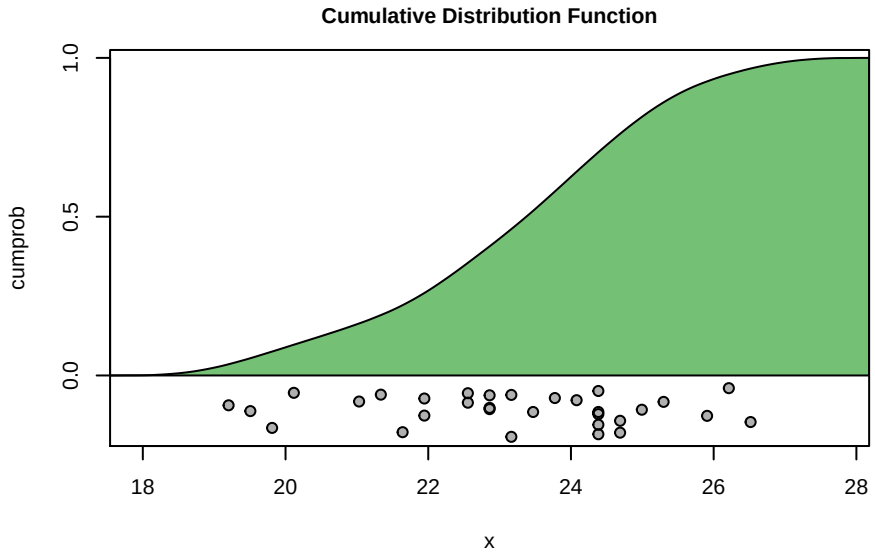
Here are examples, using height data, derived from the "trees" data in the datasets package:
(This in metric).

```
> cfh <- pdfuv.cks (height)
> cFh <- cdfuv.cks (height)
```

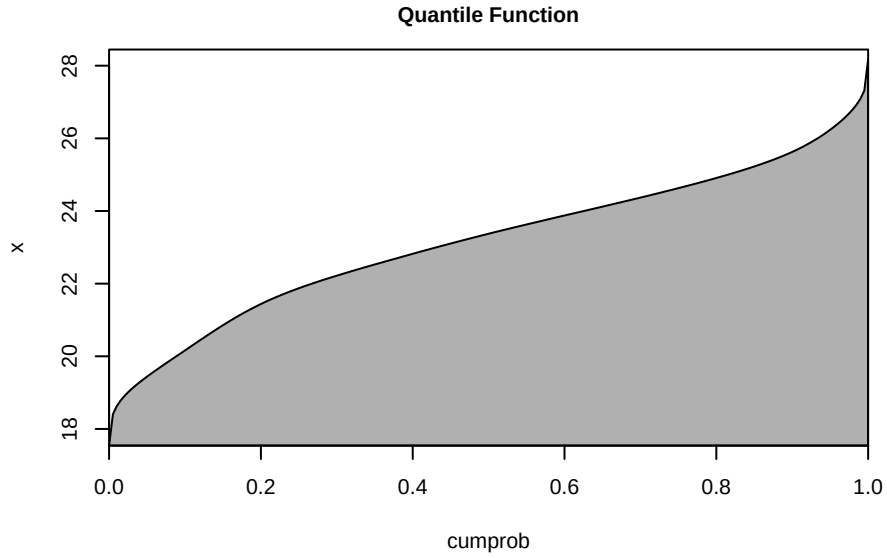
```
> cFht <- qfuv.cks (height)
> plot (cfh, TRUE, main="Probability Density Function")
```




```
> plot (cFh, TRUE, main="Cumulative Distribution Function")
```




```
> plot (cFht, main="Quantile Function")
```



The resulting objects are function objects.
The PDF maps a numeric vector (of quantiles) to a numeric vector (of probabilities), the CDF maps quantiles to cumulative probabilities, and the QF maps probabilities to quantiles.

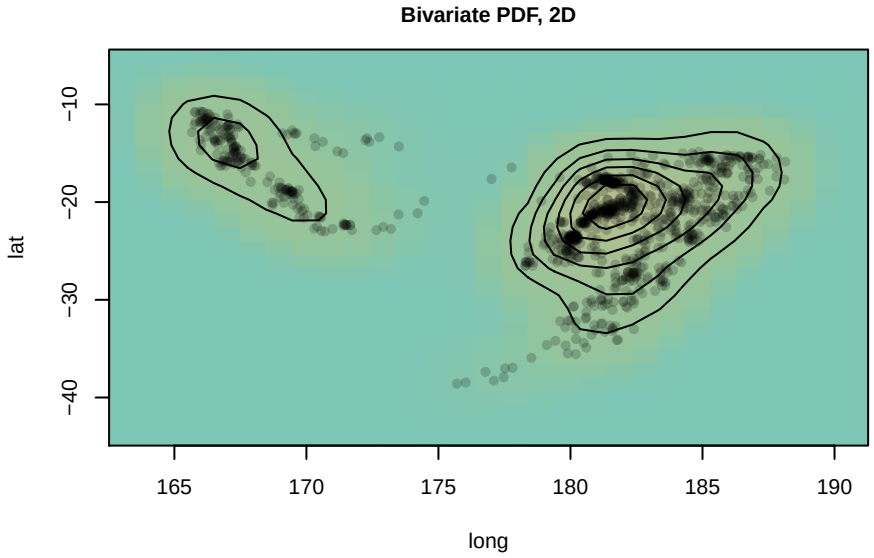
```
> cfh (22)

[1] 0.1416708

> cFht (c (0.25, 0.5, 0.75) )

[1] 21.87344 23.37409 24.62905
```

Continuous Kernel Smoothing: Multivariate Probability Distributions

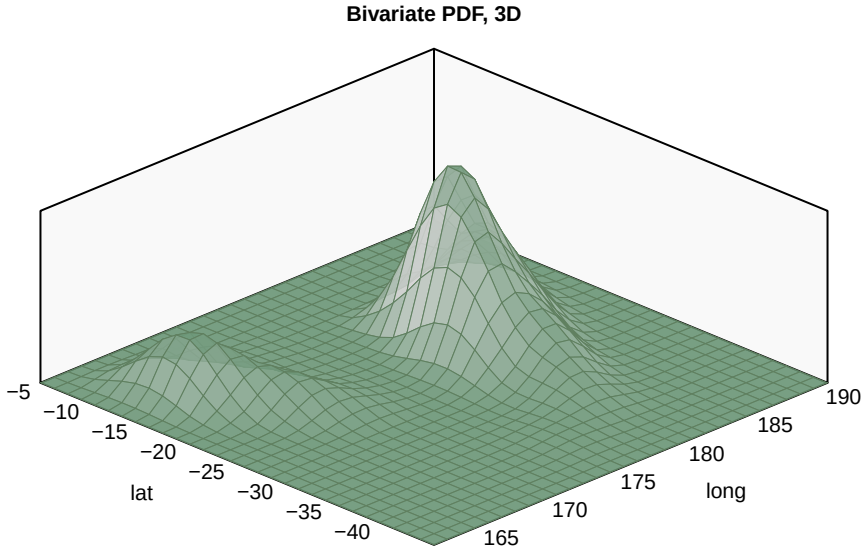
We can construct PDF$_{(MV)}$~CKS and CDF$_{(MV)}$~CKS objects, using the **pdfmv.cks** and **cdfuv.cks** constructors:

Here are examples, using earthquake data, derived from the "quakes" data, in the datasets package:

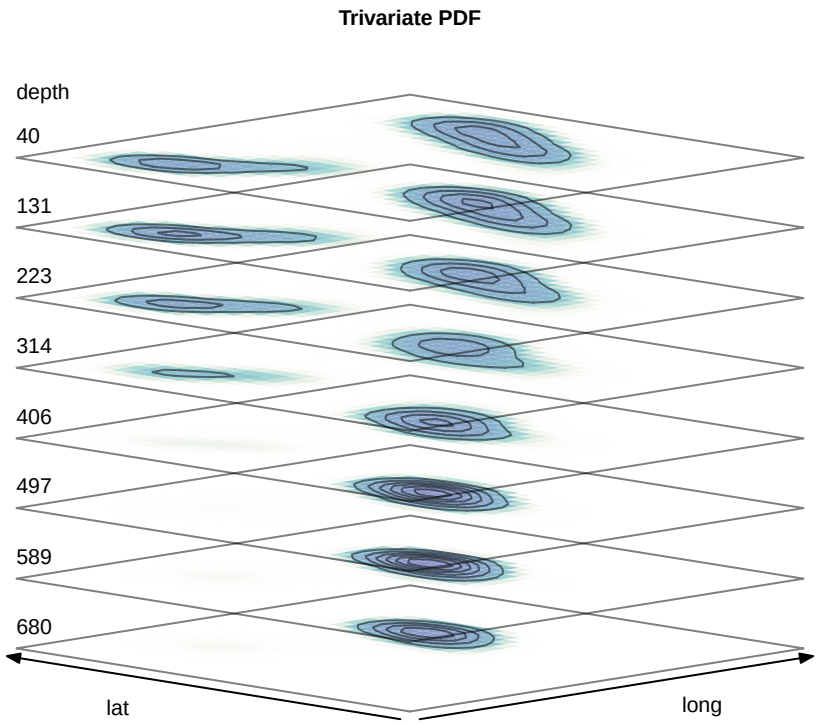
```
> #note, depth is third variable
> #(will fit a semi-bounded model)
> head (quakes2, 2)

      long    lat depth mag
[1,] 181.62 -20.42   562 4.8
[2,] 181.03 -20.62   650 4.2

> cfh2 <- pdfmv.cks (quakes2 [,1:2], smoothness = c (0.35, 1) )
> cfh3 <- pdfmv.cks (quakes2 [,1:3], smoothness = c (0.35, 1, 1),
     a = c (-Inf, -Inf, 0) )

> plot (cfh2,, TRUE,
     main="Bivariate PDF, 2D")
```




```
> plot (cfh2, TRUE, main="Bivariate PDF, 3D")
```




```
> plot (cfh3, main="Trivariate PDF",
    nslides=8, zlim = c (680, 40) )
```



The resulting objects are function objects.
The PDF maps a numeric vector or matrix (of quantiles) to a numeric vector (of probabilities) and the CDF maps quantiles to cumulative probabilities.
In PDFs and CDFs, a standard vector is equivalent to a single-row matrix.

```
> cfh2 (c (180, -20) )

[1] 0.007990662

> cfh3 (c (180, -20, 300) )

[1] 2.829838e-06
```

Note that an example of a multivariate CDF is given in the section on multivariate probabilities, later.

Also note that longitude maps to the "x" variable and latitude maps to the "y" variable. (And longitude is the first variable in the derived dataset, but is the second variable in the original dataset).

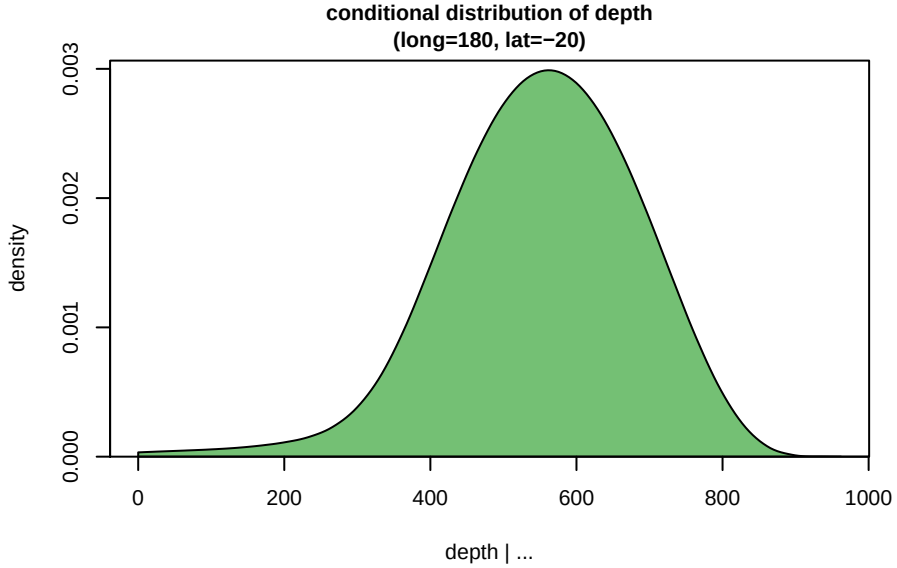
Continuous Kernel Smoothing: Conditional Probability Distributions

We can construct a $\text{PDF}_{(C)}$~CKS object, using the **pdfc.cks** constructor.
Likewise, we can construct $\text{CDF}_{(C)}$~CKS and $\text{QF}_{(C)}$~CKS objects, using the **cdfc.cks** and **qfc.cks** constructors.

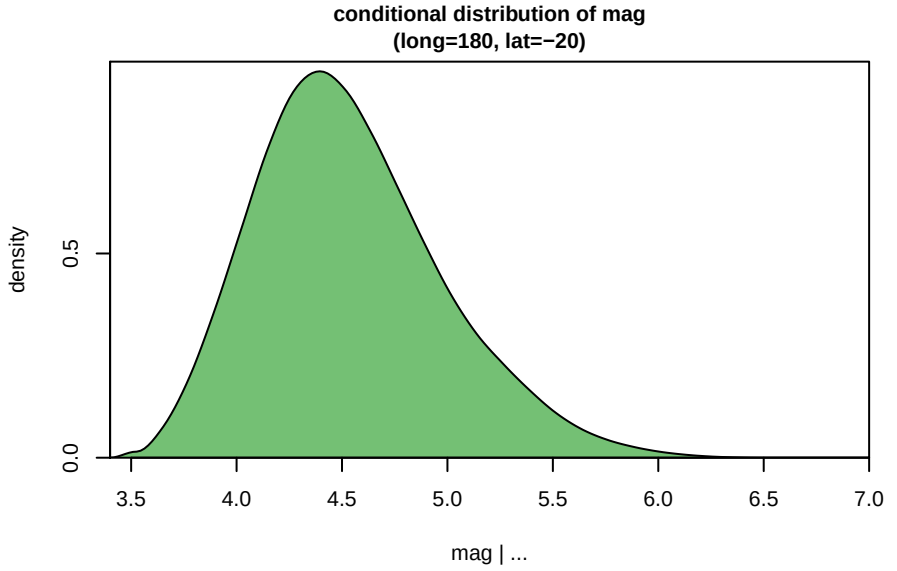
Here are examples, using the quakes data, from the previous section:

```
> conditions <- c (long=180, lat=-20)

> depth.fhc <- pdfc.cks (quakes2 [,-4], smoothness = c (0.35, 1, 1),
```

```
      conditions=conditions, preserve.range=TRUE,
      a = c (-Inf, -Inf, 0) )
> mag.fhc <- pdfc.cks (quakes2 [,-3], smoothness = c (0.35, 1, 1),
      conditions=conditions, preserve.range=TRUE)

> plot (depth.fhc, main="conditional distribution of depth\n(long=180, lat=-20)")
```




```
> plot (mag.fhc, main="conditional distribution of mag\n(long=180, lat=-20)")
```



The resulting objects are function objects, similar to univariate models, given earlier.

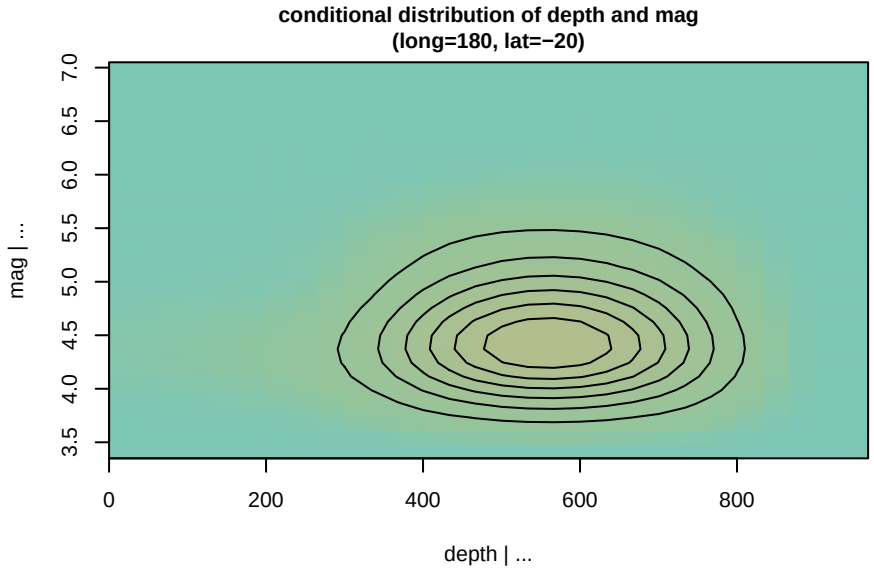
Continuous Kernel Smoothing: Multivariate-Conditional Distributions

We can construct $\text{PDF}_{(\text{MVC})}$∼CKS and $\text{CDF}_{(\text{MVC})}$∼CKS objects, using the **pdfmvc.cks** and **cdfmvc.cks** constructors.

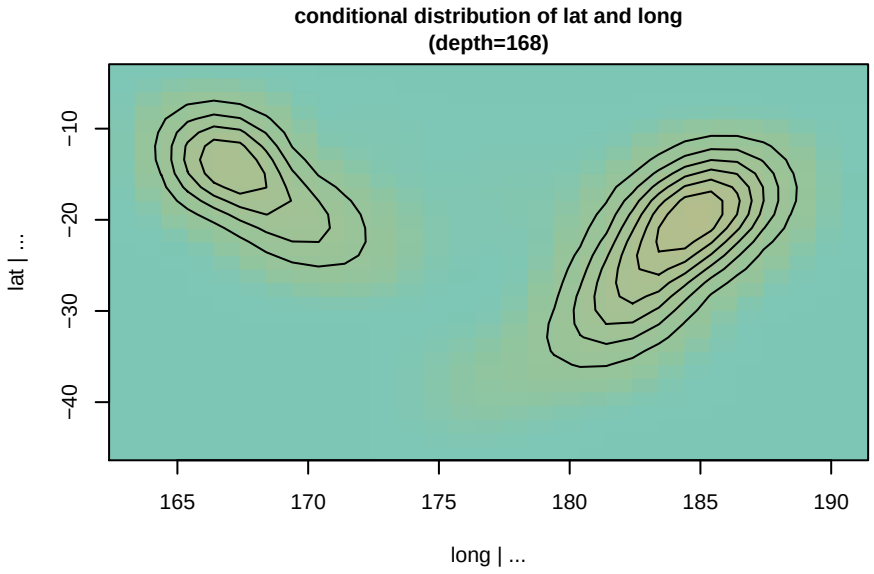
Here are examples, using the quakes data, from the previous sections:

```
> depth.mag.fhc <- pdfmvc.cks (quakes2, smoothness = c (0.35, 1, 1, 1),
     conditions = c (long=180, lat=-20), preserve.range=TRUE,
     a = c (-Inf, -Inf, 0, -Inf) )
> lat.long.fhc <- pdfmvc.cks (quakes2 [,-4], smoothness = c (0.35, 1, 1),
     conditions = c (depth=168), preserve.range=TRUE,
     a = c (-Inf, -Inf, 0) )
```

```
> plot (depth.mag.fhc,
     main="conditional distribution of depth and mag\n(long=180, lat=-20)")
```




```
> plot (lat.long.fhc,
     main="conditional distribution of lat and long\n(depth=168)")
```



The resulting objects are function objects, similar to multivariate models, given earlier.

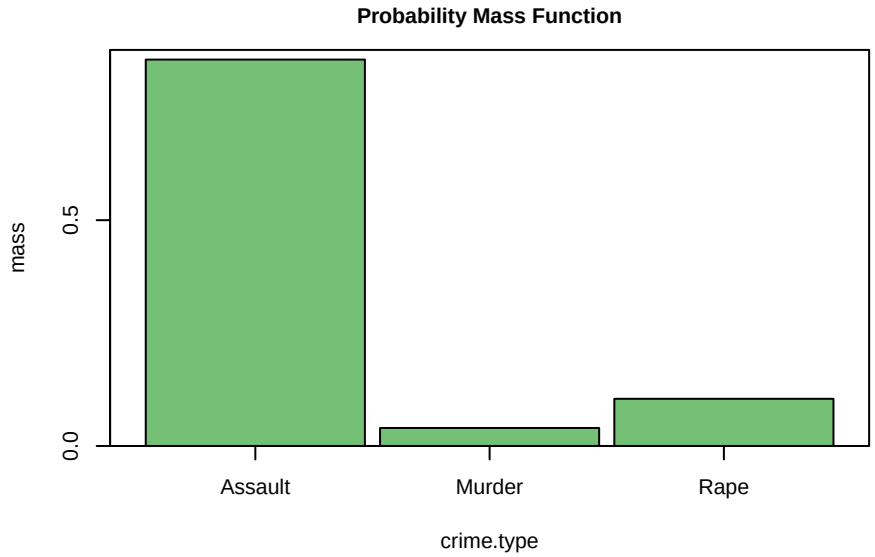
Categorical Distributions

We can construct a $\text{PMF}_{(\text{UV})}$~CAT object, using the **pmfuv.cat** constructor.
Likewise, we can construct $\text{CDF}_{(\text{UV})}$~CAT and $\text{QF}_{(\text{UV})}$~CAT objects, using the **cdfuv.cat** and **qfuv.cat** constructors.

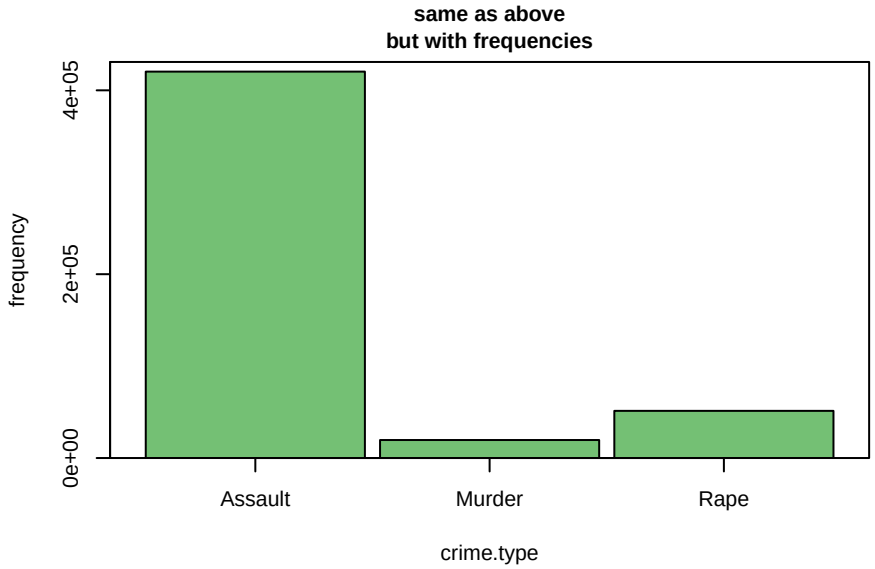
Here are examples, using the number arrests per crime type, derived from the state.x77 and USArrests datasets in the datasets package:

```
> gfh <- pmfuv.cat (crime.type, n.arrests)
```

```
> plot (gfh, main="Probability Mass Function")
```




```
> plot (gfh, freq=TRUE, main="same as above\nbut with frequencies")
```



The resulting objects are function objects, similar to discrete kernel smoothing.
Categorical PMFs and CDFs can be evaluated using integers, factors or characters.

```
> levels (crime.type$crime.type)
```

```
[1] "Assault" "Murder"  "Rape"
> gfh (1)
[1] 0.8556499
> gfh ("Assault")
[1] 0.8556499
> gfh ("Assault", freq=TRUE)
[1] 420356
```

Note that the integer above (1) represents the index of the (first) category, and not the corresponding category name (Assault).
This is obvious here, but caution is required if a categorical distribution is constructed from integer-valued categories.

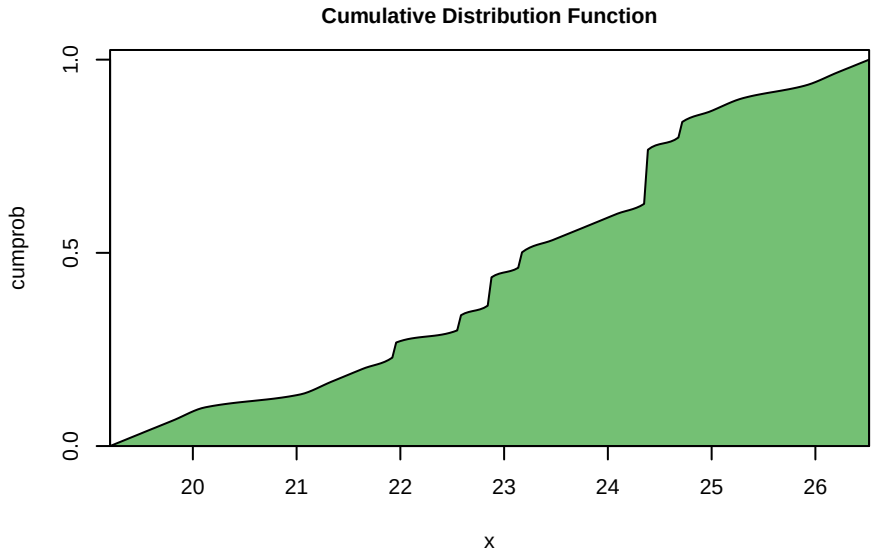
Also, it's possible to construct conditional categorical distributions from categorical data, however, I'm bypassing the examples.

Empirical-Like Distributions

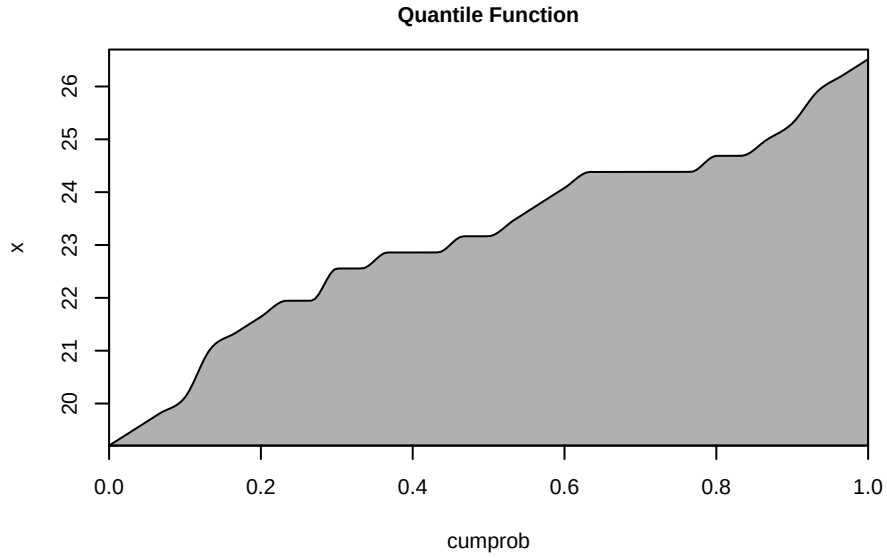
We can construct a $\text{CDF}_{(\text{UV})}$~EL object, using the **cdf.el** constructor.
Likewise, we can construct a $\text{QF}_{(\text{UV})}$~EL object, using the **qf.el** constructor.

Here's an example, using the same height data, as earlier:

```
> eFh <- cdf.el (height)
> eFht <- qf.el (height)
> plot (eFh, main="Cumulative Distribution Function")
```




```
> plot (eFht, main="Quantile Function")
```



These models compute a set of points, representing cumulative probabilities, and interpolate the points with a cubic Hermite spline.

The resulting functions are smooth (the property), but don't necessarily appear smooth.

Unlike continuous kernel smoothing, empirical-like models don't smooth (the method) the model.

Empirical-like models require unique x values, and a small amount of random variation is automatically added, if they're not unique.

Conditional Distributions with Mixed Input Types (And Statistical Classification)

In addition to the conditional probability distributions listed so far, it's also possible to construct categorical and continuous distributions, conditional on both categorical and continuous variables, or conditional on the opposite type.

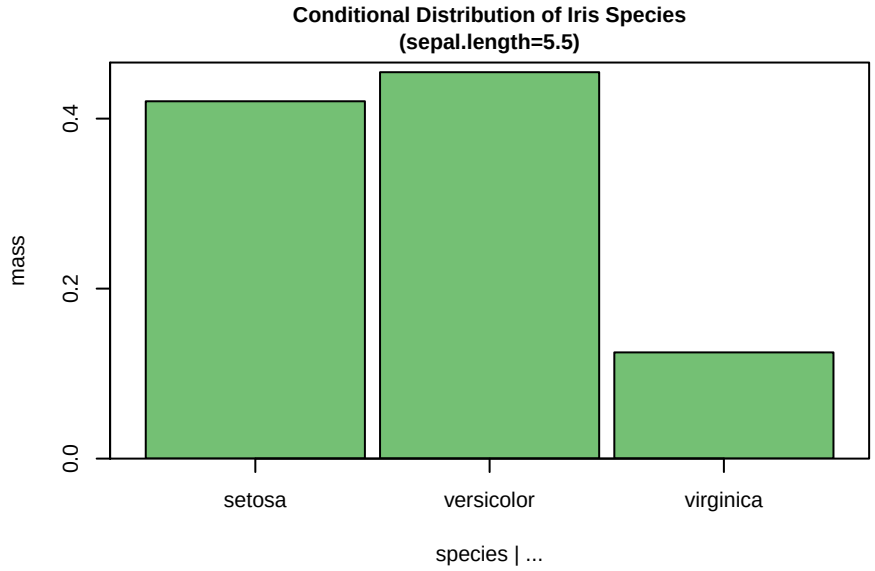
Currently, I refer to these as conditional distributions with mixed input types, and give them the extensions gMIX and xMIX.

A gMIX model, fits a categorical distribution (with a categorical conditional variable) and at least one continuous conditioning variable.
A xMIX model, fits a continuous distribution (with a continuous conditional variable) and at least one categorical conditioning variable.

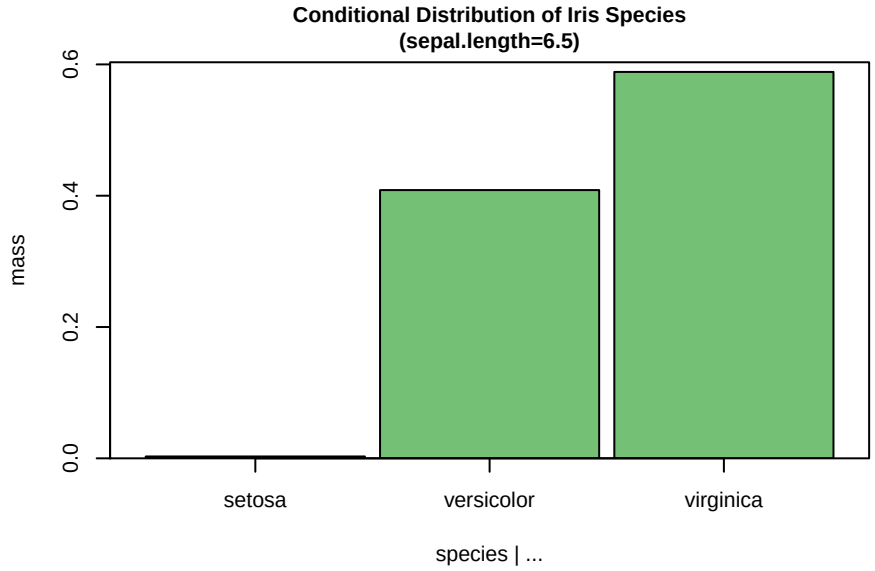
A gMIX model is computed by fitting one model for each category, along with two non-conditional models, and then uses Bayes' Theorem, to invert the models.

Currently, xMIX models (and gMIX models, if there are both categorical and continuous conditions) use a subset of data, where the categorical conditions match the categorical variables, and then fit a model to that subset, however, this may change in the future.

```
> fh1a.gmix <- ph4.pmfc.gmix (species, sepal.length,
     conditions = c (sepal.length=5.5) )
> plot (fh1a.gmix,
     main="Conditional Distribution of Iris Species\n(sepal.length=5.5)")
```

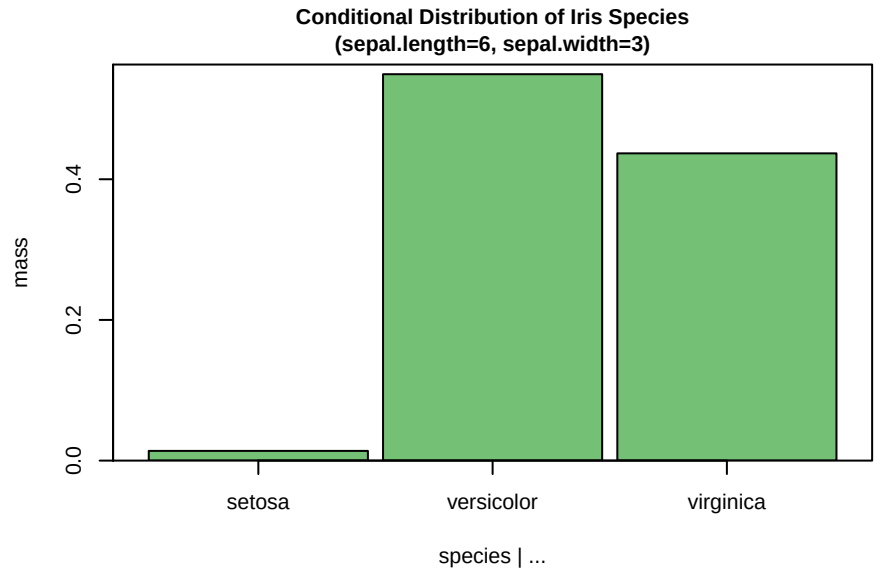



```
> fh1b.gmix <- ph4.pmfc.gmix (species, sepal.length,
     conditions = c (sepal.length=6.5) )
> plot (fh1b.gmix,
     main="Conditional Distribution of Iris Species\n(sepal.length=6.5)")
```



We can use multiple categorical or continuous variables:
(This example uses one conditional categorical variable and two conditioning continuous variables).

```
> fh2.gmix <- ph4.pmfc.gmix (species, cbind (sepal.length, sepal.width),
     conditions = c (sepal.length=6, sepal.width=3) )
> plot (fh2.gmix,
     main = paste (
         "Conditional Distribution of Iris Species",
         "(sepal.length=6, sepal.width=3)",
         sep="\n")
     )
```



Note that the next section, on distribution sets, contains a categorical set, which is equivalent to three xMIX models.

The models above can be used for statistical classification purposes:
(Using the last model):

```
> ph.mode (fh2.gmix)

[1] 2

> ph.mode (fh2.gmix, level.names=TRUE)

[1] "versicolor"
```

The distribution set objects, should be wrapped inside a call to **as.list**, with the exception of using functions specifically designed for them.

Distribution Sets

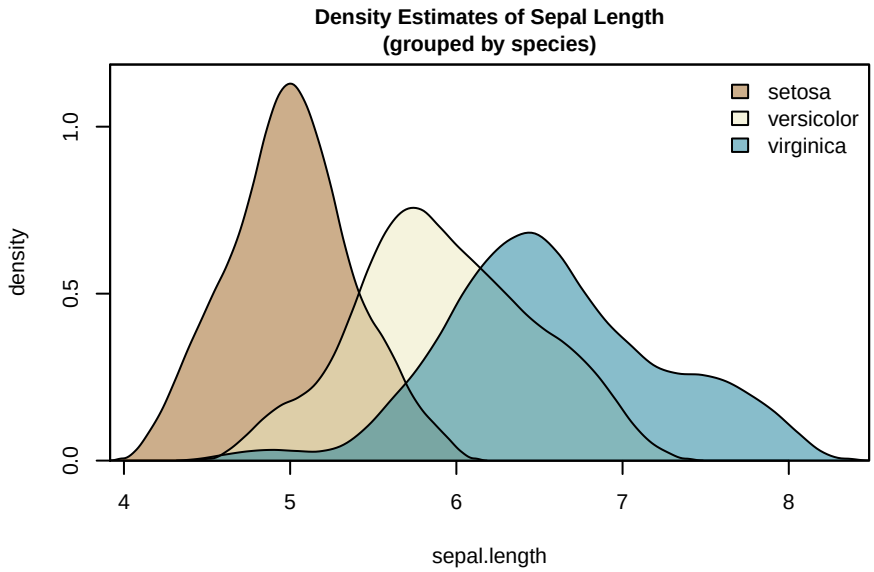
Here, a distribution set is a set of one or more probability distributions.

Currently, there are two types:

- **Categorical Set**
  One probability distribution for each (categorical) level, out of many (categorical) levels.
- **Marginal Set**
  One univariate probability distribution for each variable, out of many variables.
- **Conditional Set**
  One probability distribution for each set of conditions.
  (Refer to the constructor for the conditional CKS models).

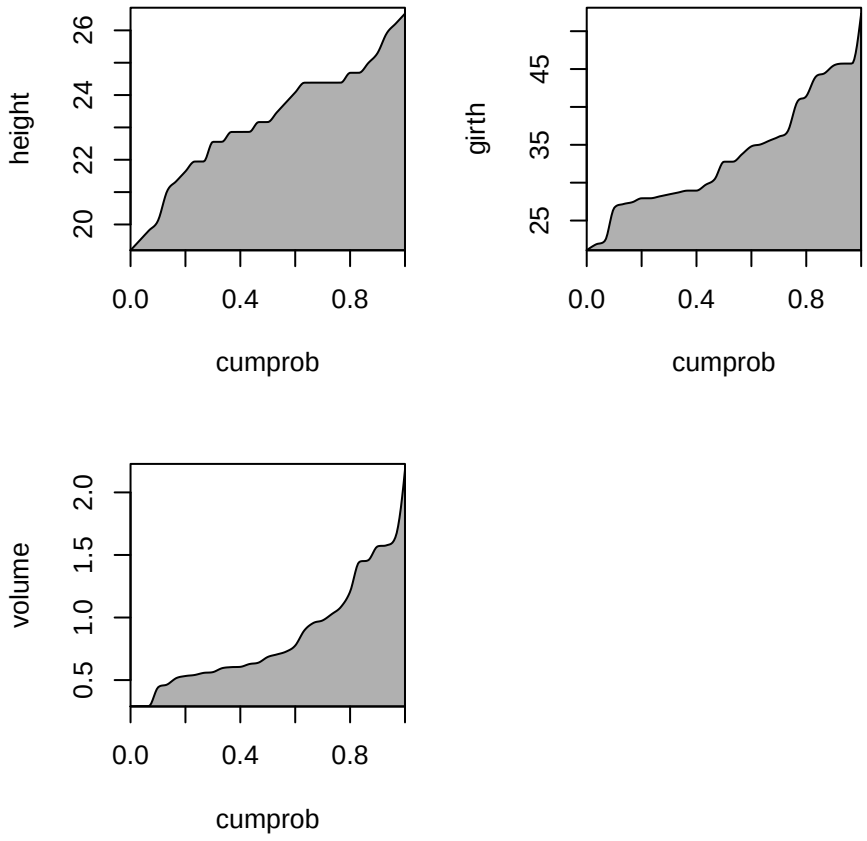
Lets construct $\text{PDF}_{(\text{UV})}$~CKS models of sepal length, grouped by species:

```
> gset <- ph4.pdfuv.gset.cks (species, sepal.length)

> plot (gset, main="Density Estimates of Sepal Length\n(grouped by species)")
```



Lets construct marginal $\mathrm{QF}_{(\mathrm{UV})}$~EL objects:

```
> mset = ph4.qfuv.mset.el (trees2)
> plot (mset, nr=2, nc=2)
```



Note that gset objects can be used to compute distance matrices, discussed in the section after the next.

Multivariate Probabilities

Here, multivariate probability refers to the probability of observing multiple random variables between pairs of lower and upper limits.

Currently, there are two sets of functions for this purpose.

The **probmv** function can be applied to almost any continuous CDF, but it's deprecated. The **pwith** functions use a modified kernel smoothing algorithm, which is more efficient.

Using the trees data, we can compute the probability that height, girth and volume are all between certains pairs of limits.

We can use the **probmv** function, which has three arguments, the multivariate CDF, a vector of lower limits and a vector of upper limits:

```
> #multivariate cdf
> #note pwith function can use pdf or cdf, but only nonconditional
> cFh3 <- cdfmv.cks (trees2)

> xlim <- matrix (c (
     22, 24,    #height in 22 to 24
     28, 38,    #girth  in 28 to 38
     0.55, 1.05 #volume in 0.55 to 1.05
     ),, 2, byrow=TRUE, dimnames = list (colnames (trees2), c ("a", "b") ) )
> xlim

           a     b
height 22.00 24.00
girth  28.00 38.00
volume  0.55  1.05

> #multivariate probability
> probmv (cFh3, xlim [,1], xlim [,2])

[1] 0.07481826

> pwith (cFh3, xlim=xlim)

[1] 0.07481826
```

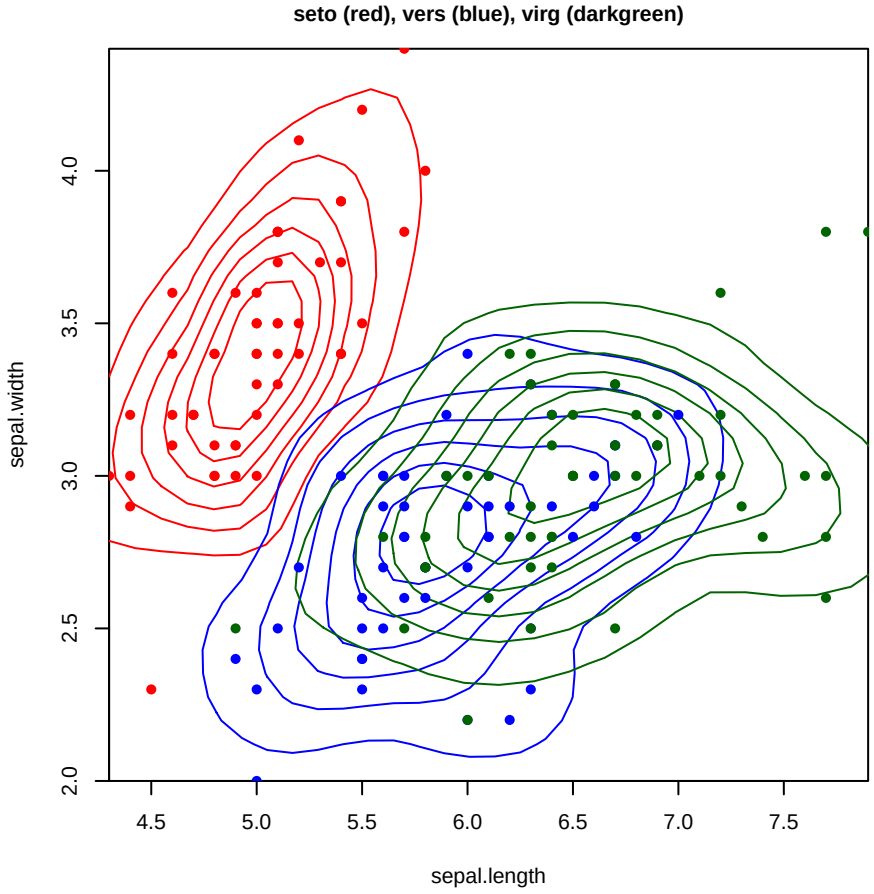
Note that it's possible to compute multiple regions at once by making a and b matrices with each row representing one region. Also note that currently, variables names are ignored, so they must be in the same order as the variables used to construct the CDF.

Distance Matrices

We can compute a distance (or dissimilarity) matrix based on a gset object:

```
> gset1a <- ph4.pdfuv.gset.cks (species, sepal.length)
> gset1b <- ph4.pdfuv.gset.cks (species, sepal.width)
> gset2 <- ph4.pdfmv.gset.cks (species, cbind (sepal.length, sepal.width) )
> dists1a <- pdist (gset1a)
> dists1b <- pdist (gset1b)
> dists2 <- pdist (gset2)

> names <- names (gset2)
> cols <- c ("red", "blue", "darkgreen")
> main <- paste0 (substring (names, 1, 4), " (", cols, ")", collapse=", ")
> I1 <- (names [1] == species [[1]])
```

```
> I2 <- (names [2] == species [[1]])
> I3 <- (names [3] == species [[1]])

> plot (gset2 [[1]], contour.color = cols [1],
     main=main,
     xlim = range (sepal.length), ylim = range (sepal.width),
     heatmap=FALSE)
> plot (gset2 [[2]], contour.color = cols [2],
     add=TRUE, heatmap=FALSE)
> plot (gset2 [[3]], contour.color = cols [3],
     add=TRUE, heatmap=FALSE)
> points (sepal.length [I1], sepal.width [I1], pch=16, col = cols [1])
> points (sepal.length [I2], sepal.width [I2], pch=16, col = cols [2])
> points (sepal.length [I3], sepal.width [I3], pch=16, col = cols [3])
```



We can see from the above plot that versicolor and virginica have the closest distributions.

The following numerical output is consistent with that:

```
> #distance matrix
> #(both variables)
> dists2

             setosa versicolor virginica
setosa     0.0000000  0.7014478 0.6726867
versicolor 0.7014478  0.0000000 0.2699587
virginica  0.6726867  0.2699587 0.0000000
```

```
> #pairs, ranked
> ph4.rdist (dists2)  #both

                  comb      dist
1 versicolor:virginica 0.2699587
2     setosa:virginica 0.6726867
3    setosa:versicolor 0.7014478

> ph4.rdist (dists1a) #length

                  comb      dist
1 versicolor:virginica 0.2963271
2    setosa:versicolor 0.6072892
3     setosa:virginica 0.6580087

> ph4.rdist (dists1b) #width

                  comb      dist
1 versicolor:virginica 0.2259573
2     setosa:virginica 0.6063917
3    setosa:versicolor 0.6867466

>
```

The details are discussed in the help file for pdist.

Note that these methods are relatively new.

I can not give any guarantee of optimality.
(This applies to the whole package, but in particular to these functions).

Chained Quantile Functions (And Multivariate Random Number Generation)

Standard quantile functions can be used to compute univariate random numbers via standard inversion sampling

I've created novel chained quantile functions, to compute multivariate random numbers.

It works by fitting a quantile function to the first variable's observations, and then evaluating that quantile function at the first variable's evaluation points.

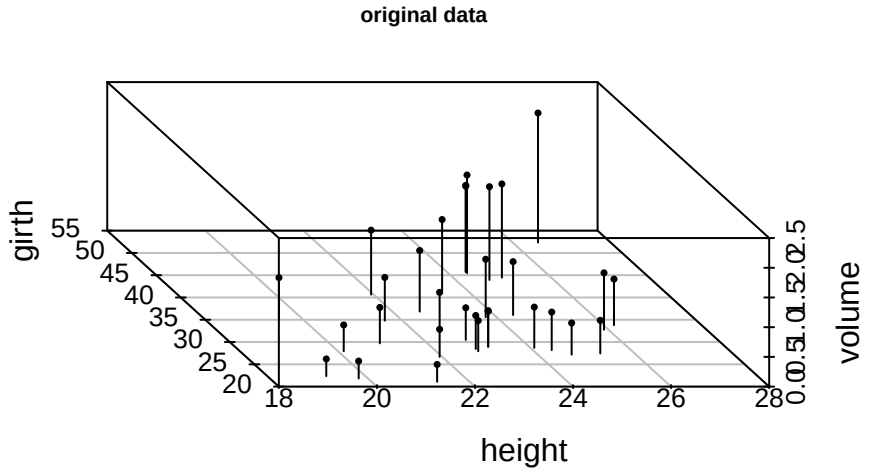
Then, assuming that there are two of more variables, a sequence of conditional quantile functions are fitted to incrementing sets of variables, conditional on the previous variables' evaluations, evaluating one new variable, each time.
This is done for each evaluation point.

The convenience function, **rng**, takes two arguments, the univariate or chained quantile function, and the number of random numbers to generate, then evaluates the quantile function, using a vector or matrix of uniform random numbers.

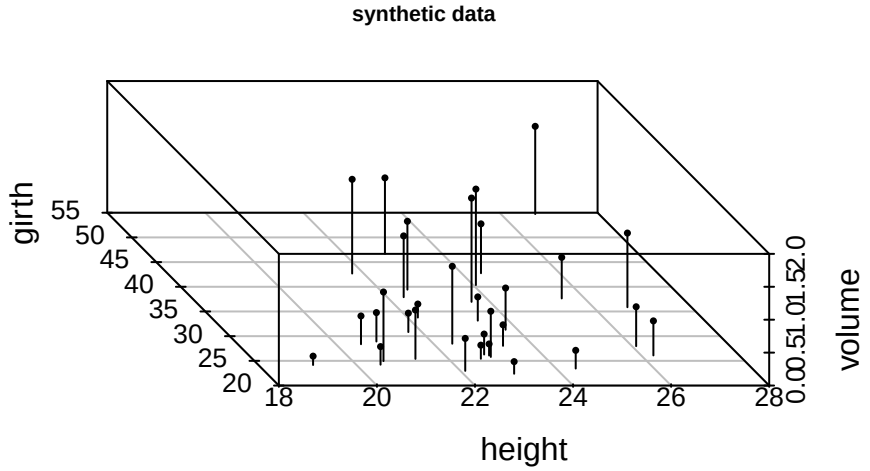
```
> chFht <- chqf.cks (trees2)
> synthetic.data <- rng (chFht, 31)

> #convenience function
> plot.trees.data <- function (x, main)
 {   height <- x [,"height"]
     girth <- x [,"girth"]
     volume <- x [,"volume"]
```

```
    scatterplot3d (height, girth, volume,
        main=main, type="h", angle=112.5, pch=16)
}
```

```
> #original data
> plot.trees.data (trees2, "original data")
```




```
> #synthetic data
> plot.trees.data (synthetic.data, "synthetic data")
```



Note that this is computationally expensive.

For 3 variables and 31 evaluation points, the algorithm needs to fit:
$1 + 31 * (3 - 1) = 63$ distributions.

Parameter Estimates

Overview

Univariate probability distributions (including univariate-conditional probability distributions) can be used to compute probabilities and parameter-like estimates, including the mean, standard deviation, variance, skewness and kurtosis, (arbitrary) moments, median, (arbitrary) quantiles and the mode.

Probabilities can be computed from the CDF.
(i.e. In the univariate continuous case, $\hat{F}(b) - \hat{F}(a)$, no example is given).

The mean, standard deviation, variance and higher moments, require the PMF or spline-based CDF.

The median and quantiles, require the quantile function.

The mode, requires the PMF or spline-based PDF.

In the future, I may allow automatic conversion between the PMF/PDF, CDF and QF.

Moment-Based Statistics

We can compute moment-based estimates using the high-level functions the **ph.mean**, **ph.sd**, **ph.var**, **ph.skewness** and **ph.kurtosis**.

They require a PMF or spline-based CDF.

```
> ph.mean (cFh)

[1] 23.1648

> ph.var (cFh)

[1] 4.044737

> ph.skewness (cFh)

[1] -0.3213694

> ph.kurtosis (cFh)

[1] 2.530339
```

Note that currently, standard deviation, variance and higher moments, should not be regarded as accurate because the smoothing algorithm tends to inflate their values, however, they can still be used as an exploratory tool, especially for the purpose, of comparing different conditional probability distributions.

Order-Based Summary Statistics

We can compute order-based summary statistics, using the **quartiles**, **deciles** and **ntiles** functions.

All of which require a numeric vector, quantile function, or an object than can be coerced to a numeric vector.

```
> quartiles (cFht)

     min       Q1     (Q2)       Q3      max
17.54124 21.87344 23.37409 24.62905 28.17876

> deciles (cFht)
```

```
     min       D1       D2       D3       D4     (D5)       D6       D7
17.54124 20.16155 21.43919 22.21897 22.82123 23.37409 23.87581 24.36876
      D8       D9      max
24.90875 25.62579 28.17876
```

```
> ntiles (8, cFht)
```

```
     min       q1       q2       q3     (q4)       q5       q6       q7
17.54124 20.51468 21.87344 22.67502 23.37409 23.99723 24.62905 25.41074
     max
28.17876
```

```
> ntiles (8, cFht, prob=TRUE)
```

```
       0    0.125     0.25    0.375      0.5    0.625     0.75    0.875
17.54124 20.51468 21.87344 22.67502 23.37409 23.99723 24.62905 25.41074
       1
28.17876
```

An $QF_{(UV)}$~EL object is automatically created, if a numeric vector is used, hence the last two results should be identical (for unique input values) or almost identical (for non-unique input values).

Quantile-Based Statistics

We can compute quantile-based statistics, using the **ph.median**, **ph.quantile** and **iqr** functions.

All of which require a numeric vector, quantile function, or an object than can be coerced to a numeric vector.

```
> ph.median (cFht)
```

```
[1] 23.37409
```

```
> ph.quantile (cFht, c (0.25, 0.5, 0.75) )
```

```
[1] 21.87344 23.37409 24.62905
```

```
> #inter-quartile range
> iqr (cFht)
```

```
[1] 2.755612
```

```
> #inter-quantile ranges
> iqr (cFht, 2/3)
```

```
[1] 4.046585
```

These function work in similar way to the order-based functions in the previous section, which the exception that they're less summary focused.

Note that calling these functions with a quantile function, is equivalent to evaluating the quantile function, directly.
So, the only gain is possible readability.

```
> cFht (0.5)
```

```
[1] 23.37409
```

```
> ph.median (cFht)
```

```
[1] 23.37409
```

Like the order-based functions, they can also be used on a numeric vector.

```
> ph.median (height)

[1] 23.16656
```

In theory, this should be more accurate than the median and quantile functions from the stats package, because these use smooth interpolation, rather than linear interpolation.

Mode Estimates

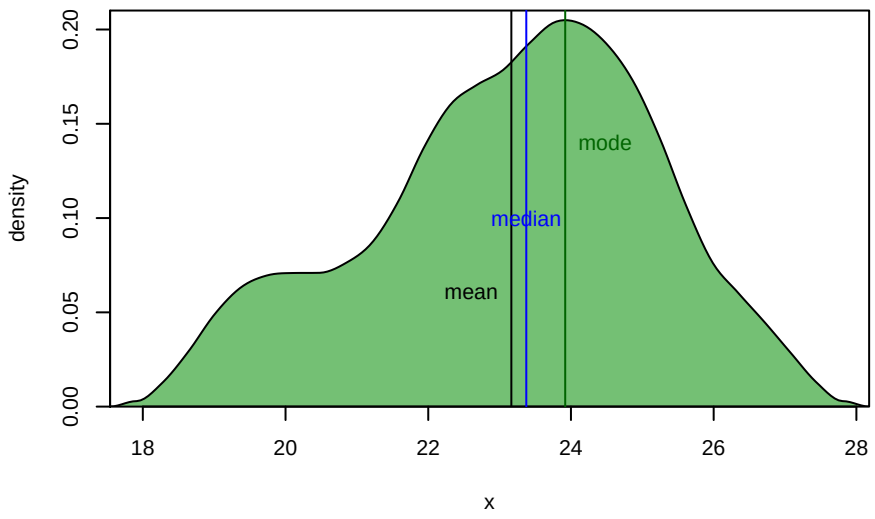
We can compute mode estimates using the **ph.mode** and **ph.modes** functions.

The first requires a PMF or spline-based PDF, and the second which computes multiple modes, requires a spline-based PDF only.

```
> ph.mode(cfh)

[1] 23.91958

> ph.mode(cfh, TRUE)

[1] 0.2049598
```

Putting it All Together

```
> height.summary <- c (
      mean = ph.mean (cFh),          #CDF
      sd = ph.sd (cFh),              #CDF
      variance = ph.var (cFh),       #CDF
      skewness = ph.skewness (cFh), #CDF
      kurtosis = ph.kurtosis (cFh), #CDF
                                     #
      median = ph.median (cFht),     #QF
      mode = ph.mode (cfh) )         #PDF

> strs <- c (c ("mean", "median", "mode") )
> x <- height.summary [strs]
> y <- c (0.06, 0.1, 0.14)
> colors <- c ("black", "blue", "darkgreen")
> adjv <- c (1.25, 0.5, -0.25)

> plot (cfh)
> abline (v=x, col=colors)
> for (i in 1:3)
      text (x [i], y [i], strs [i], adj = adjv [i], col = colors [i])
```




```
> height.summary
```

mean	sd	variance	skewness	kurtosis	median	mode
23.1647993	2.0111531	4.0447368	-0.3213694	2.5303394	23.3740852	23.9195751

References

My main sources are:

R's stats Package

Wikipedia

Also, this package uses the following R packages, or has been influenced by them:

barsurf: Contour Plots, 3D Plots, Vector Fields and Heatmaps
Spurdle, A.

bivariate: Bivariate Probability Distributions
Spurdle, A.

mvtnorm: Multivariate Normal and t Distributions
Genz, A., Bretz, F., Miwa, T., Mi, X. & Hothorn, T.

kubik: Cubic Hermite Splines and Related Root Finding Methods
Spurdle, A.

KernSmooth: Functions for Kernel Smoothing Supporting Wand & Jones (1995)
Wand, M. & Ripley, B.

mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation
Wood, S.

fclust: Fuzzy Clustering
Giordani, P., Ferraro, M.B. & Serafini, A.

scatterplot3d: 3D Scatter Plot
Ligges, U., Maechler, M. & Schnackenberg, S.

MASS: Support Functions and Datasets for Venables and Ripley's MASS
Ripley, B.

Further references:

Wand, M.P. & Jones, M.C. (1995). Kernel Smoothing.

Appendix A:
List of Probability Distributions

Discrete kernel smoothing models (DKS):

- Univariate probability mass function ($\text{PMF}_{(\text{UV})}$~DKS).
- Univariate cumulative distribution function ($\text{CDF}_{(\text{UV})}$~DKS).
- Univariate quantile function ($\text{QF}_{(\text{UV})}$~DKS).

Continuous kernel smoothing models (CKS):

- Univariate probability density function ($\text{PDF}_{(\text{UV})}$~CKS).
- Univariate cumulative distribution function ($\text{CDF}_{(\text{UV})}$~CKS).
- Univariate quantile function ($\text{QF}_{(\text{UV})}$~CKS).

- Multivariate probability density function ($\text{PDF}_{(\text{MV})}$~CKS).
- Multivariate cumulative distribution function ($\text{CDF}_{(\text{MV})}$~CKS).

- Conditional probability density function ($\text{PDF}_{(\text{C})}$~CKS).
- Conditional cumulative distribution function ($\text{CDF}_{(\text{C})}$~CKS).
- Conditional quantile function ($\text{QF}_{(\text{C})}$~CKS).

- Multivariate-conditional probability density function ($\text{PDF}_{(\text{MVC})}$~CKS).
- Multivariate-conditional cumulative distribution function ($\text{CDF}_{(\text{MVC})}$~CKS).

- Chained quantile function (ChQF~CKS).

Categorical models (CAT):

- Univariate probability mass function ($\text{PMF}_{(\text{UV})}$~CAT).
- Univariate cumulative distribution function ($\text{CDF}_{(\text{UV})}$~CAT).
- Univariate quantile function ($\text{QF}_{(\text{UV})}$~CAT).

- Conditional probability mass function ($\text{PMF}_{(\text{C})}$~CAT).
- Conditional cumulative distribution function ($\text{CDF}_{(\text{C})}$~CAT).
- Conditional quantile function ($\text{QF}_{(\text{C})}$~CAT).

Empirical-like models (EL):

- Univariate cumulative distribution function ($\text{CDF}_{(\text{UV})}$~EL).
- Univariate quantile function ($\text{QF}_{(\text{UV})}$~EL).

Conditional categorical distributions with mixed input types (gMIX):

- Conditional probability mass function ($\text{PMF}_{(\text{C})}$~gMIX).
- Conditional cumulative distribution function ($\text{CDF}_{(\text{C})}$~gMIX).
- Conditional quantile function ($\text{QF}_{(\text{C})}$~gMIX).

Conditional continuous distributions with mixed input types (xMIX):

- Conditional probability density function ($\text{PDF}_{(\text{C})}$~xMIX).
- Conditional cumulative distribution function ($\text{CDF}_{(\text{C})}$~xMIX).

- Conditional quantile function ($QF_{(C)}$∼xMIX).

Distribution sets:

- Marginal sets.
- Categorical sets.

Appendix B:
Notation, Terminology and Related Notes

When used as a prefix, the letters “d” and “c” mean discrete and continuous, respectively. Likewise, DPD and CPD mean discrete probability distribution and continuous probability distribution.

PMF, PDF, CDF and QF refer to probability mass function (or just mass function), probability density function (or just density function), cumulative distribution function (or just distribution function) and quantile function, respectively.

Note that I sometimes use the notation, Fht or Fh.inv, to describe the quantile function. However, quantile functions are not necessarily the exact inverse of CDFs.
Firstly, CDFs may have level (non-increasing) sections which are non-invertible, and secondly, the algorithm for constructing quantile splines, only transposes the control points, it does not compute an exact inverse.

When used as a suffix, the letters “uv”, “mv”, “c” and “mvc”, mean univariate, multivariate, conditional and multivariate-conditional.

In publicly visible constructors (e.g. **pmfuv.dks**):
Univariate and multivariate distributions, refer to non-conditional univariate and multivariate distributions only, and conditional distributions refer to univariate conditional distributions only.
This also applies to the corresponding class names, and to most of the section headings in this vignette.

In other contexts, univariate and multivariate distributions, can be conditional or non-conditional, and conditional distributions can be univariate or multivariate, unless stated otherwise.

Currently, conditional distributions need at least one condition.
(i.e. A model with no conditions is not regarded as conditional).

The term ”range”, refers to the range of a random variable, unless stated otherwise.

In this package, there are three subtypes of such ranges:

- The model range.
- The mechanistic or theoretic limits, labelled Xlim.
- The observed range.

Both the model and observed ranges need to be inside the theoretic limits. The same applies to conditions, eval bins/points and the main input data.

Bounded and unbounded random variables, have finite and infinite theoretic limits, respectively. Noting the option of being semi-bounded, which is the default for DKS models.

In the unbounded discrete case, the the model range is observed range +/- half the decremented bandwidth (which needs to be odd), at each end. In the unbounded continuous case, the the model range is observed range +/- half the bandwidth, at each end.

In the bounded case, the model range is the same except that it is truncated, if it would fall outside the theoretic limits, which also means that a truncated smoothing algorithm is used.

Appendix C:
Discrete Kernel Smoothing Formulae, and Related Notes

Discrete kernels (without standardized intervals), take the form:

$$k(x; \text{bw}) = ...$$
$$K(x; \text{bw}) = ...$$

$$\text{hbw} = \frac{\text{bw} - 1}{2}$$

Where k and K are the kernel's PMF and CDF, respectively.
And where bw is the (odd positive) bandwidth parameter.

Unstandardized discrete kernels have zero mass outside the interval [-hbw, +hbw].

We can define additive component-distributions as:

$$k^*(x; k, x_i^*, \text{bw}) = k(x - x_i^*; \text{bw})$$
$$K^*(x; K, x_i^*, \text{bw}) = K(x - x_i^*; \text{bw})$$

Where x_i^* is the (integer-valued) center of the of each additive component-distribution, and x is the (integer-valued) point on the x-axis, where the function is evaluated.

Unbounded PMFs and CDFs can be computed, as follows:

$$\hat{f}_X(x; k, \text{bw}, n, \mathbf{x}^*, \mathbf{w}) = \sum_i w_i k^*(x; k, x_i^*, \text{bw})$$
$$\hat{F}_X(x; K, \text{bw}, n, \mathbf{x}^*, \mathbf{w}) = \sum_i w_i K^*(x; K, x_i^*, \text{bw})$$

Where:

$$w_i = \frac{h_i}{\sum_i h_i}$$

And where $\mathbf{x}^*$ is a vector of (integer-valued) bins and $\mathbf{h}$ is a vector of frequencies, both of which, are of length n, and $i \in [1, n]$.

If truncated discrete smoothing is required, then the data is reflected about the truncation bins.

This truncated algorithm may change in future, to match the continuous case, refer to the end of the next section.

Appendix D: Continuous Kernel Smoothing Formulae, and Related Notes

Continuous kernels (with standardized intervals), take the form:

$$k(x) = ...$$
$$K(x) = ...$$

Where k and K are the kernel's PDF and CDF, respectively.

Standardized continuous kernels have zero density outside the interval [-1, 1].

We can define additive component-distributions using:

$$k^*(x; k, x_i^*, \text{bw}) = \frac{2}{\text{bw}} k(\frac{2}{\text{bw}}(x - x_i^*))$$
$$K^*(x; K, x_i^*, \text{bw}) = K(\frac{2}{\text{bw}}(x - x_i^*))$$

Where bw is the bandwidth, x_i^* is the center of each additive component-distribution, and x is a point on the x-axis, where the function is evaluated.

Univariate PDFs and CDFs can be computed, as follows:

$$\hat{f}_X(x; k, \text{bw}, n, \mathbf{x}^*) = \frac{\sum_i k^*(x; k, \text{bw}, x_i^*)}{n}$$
$$\hat{F}_X(x; K, \text{bw}, n, \mathbf{x}^*) = \frac{\sum_i K^*(x; K, \text{bw}, x_i^*)}{n}$$

Where $\mathbf{x}^*$ is a vector of length n, and $i \in [1, n]$.

Multivariate PDFs and CDFs can be computed, as follows:

$$\hat{f}_{\mathbf{X}}(\mathbf{x}; k, \mathbf{bw}, n, m, \mathbf{x}^*) = \frac{\sum_i \left(\$f_1 \times \$f_2 \times ... \times \$f_m \right)}{n}$$
$$= \frac{\sum_i \left(k^*(x_1; k, \text{bw}_1, x_{[i,1]}^*) \times k^*(x_2; k, \text{bw}_2, x_{[i,2]}^*) \times ... \times k^*(x_m; k, \text{bw}_m, x_{[i,m]}^*) \right)}{n}$$

$$\hat{F}_{\mathbf{X}}(\mathbf{x}; K, \mathbf{bw}, n, m, \mathbf{x}^*) = \frac{\sum_i \left(\$F_1 \times \$F_2 \times ... \times \$F_m \right)}{n}$$
$$= \frac{\sum_i \left(K^*(x_1; K, \text{bw}_1, x_{[i,1]}^*) \times K^*(x_2; K, \text{bw}_2, x_{[i,2]}^*) \times ... \times K^*(x_m; K, \text{bw}_m, x_{[i,m]^*}) \right)}{n}$$

Where **bw** is a bandwidth vector, $\mathbf{x}^*$ is a matrix with n rows (observations) and m columns (variables), and $\mathbf{x}$ is a vector of points on the x-plane, where the function is evaluated.

Continuous PDFs use the ratio of two multivariate expressions, with the n term cancelling out.
Continuous CDFs are similar, but rather than numerically integrate the ratio of two multivariate expressions, the top expression is modified to produce a hybrid algorithm between the PDF and CDF algorithms.

Weighted versions of these formulae are created by substituting:

$$\frac{\sum_i (\$\text{SUB-EXPRESSION})}{n}$$

With:

$$\sum_i w_i (\$\text{SUB-EXPRESSION})$$

Subject to:

$$\sum_i w_i = 1$$

In the univariate case, there are three options for truncated kernel smoothing, simple, local (the default) and reflect.

In the multivariate case, there's only the local option.

In the simple truncation algorithm, it computes the area inside the truncation points. For the PDF, initial density values (taken from the untruncated fit) are multiplied by a scaling factor, which is the reciprocal of the inside area. For the CDF, the approach is the same, except that a lower limit probability value (also taken the untruncated fit) is subtracted from the initial probability values prior to scaling.

In the local truncation algorithm, there's a vector of scaling factors, and a matrix of offsets (for subtraction). This is designed to emulate the effect of locally truncating the kernels themselves. Currently, in the conditional case, only data related to random variables (not variables used in conditions) is used to compute the scaling factors and the offsets.

The reflect algorithm is the same as the simple algorithm, except the the data is reflected about the truncation points.

Appendix E: Empirical-Like Formulae

An empirical cumulative distribution function, which is a step function, can be computed by:

$$\mathbb{P}(X \leq x) = \hat{F}_X(x; n, \mathbf{x}^*) = \frac{\sum_i I(x_i^* \leq x)}{n}$$

Where I is an indicator function, which equals 1, if the enclosed logical expression is true, and equals 0, if false.

A proto-empirical-like distribution, which is also a step function, can be computed by modifying the formula above, to give:

$$\mathbb{P}(X \leq x) = \hat{G}_X(x; n, \mathbf{x}^*) = \frac{\left(\sum_i I(x_i^* \leq x)\right) - 1}{n - 1}$$

This function can be used to generate a sequence of points:

$$\{\left(x_1^*, \hat{G}_X(x_1^*; n, \mathbf{x}^*)\right), \left(x_2^*, \hat{G}_X(x_2^*; n, \mathbf{x}^*)\right), ..., \left(x_n^*, \hat{G}_X(x_n^*; n, \mathbf{x}^*)\right)\}$$

An empirical-like distribution, which is a continuous function, can be computed by using a cubic Hermite spline to interpolate this sequence.

Appendix F:
Fuzzy Clustering
(And Weighted Multivariate Kernel Smoothing)

Fuzzy clustering computes a membership matrix, from some data.

The values in the membership matrix represent the membership of each data point in each cluster, with each row representing one data point and each column representing one cluster.
(Note that rows, not columns, sum to one).

In some situations, is may be of interest to identify the clusters, only. In other situations, it may be of interest to identify the clusters, and model the properties of one or more of those clusters.

It's possible to model each cluster using weighted kernel smoothing.

The following computes the membership matrix for three clusters:

```
> membership <- FKM.gk (unemployment, k=3, seed=2)$U
```

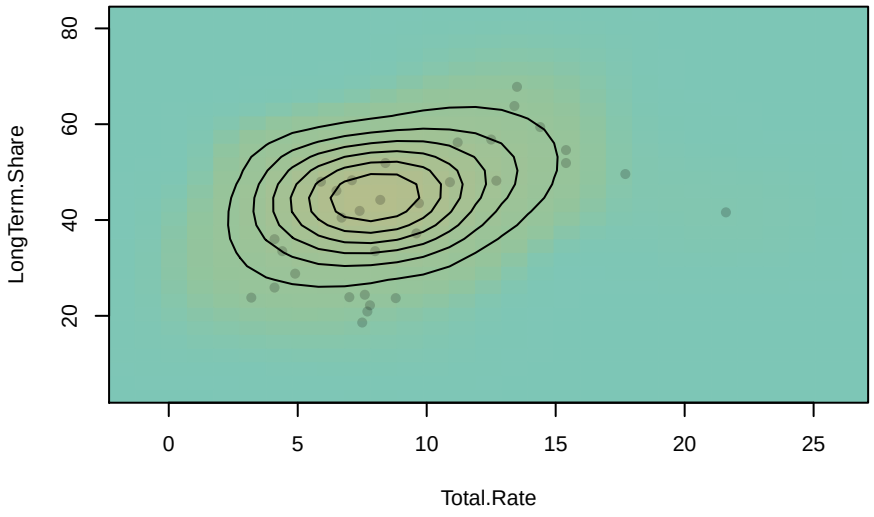
I'm going to extract the weights of the first cluster, and transform them, so that they sum to one:

```
> w <- membership [,1]
> w <- w / sum (w)
```

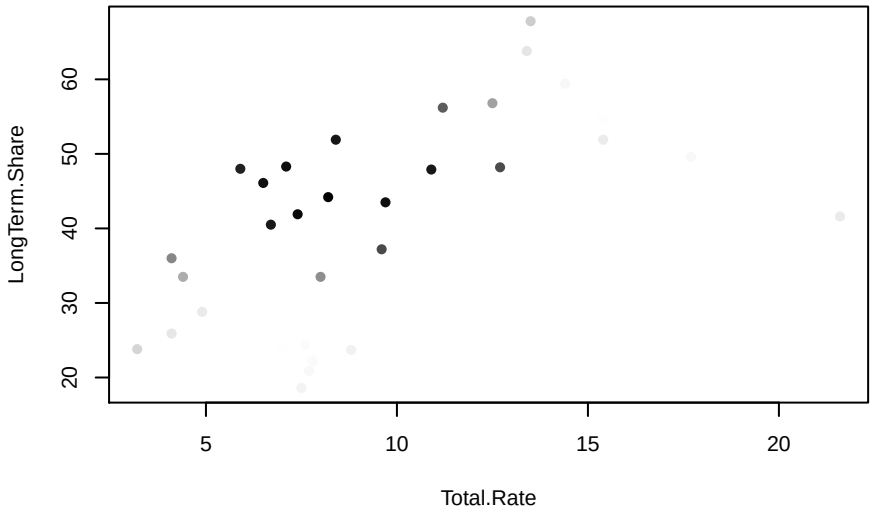
And a weighted model:

```
> wfh.1 <- pdfmv.cks (unemployment, w=w)
```

```
> plot (wfh.1,, TRUE)
```




```
> k = 1 - w / max (w)
> plot (unemployment, pch=16, col=rgb (k, k, k) )
```

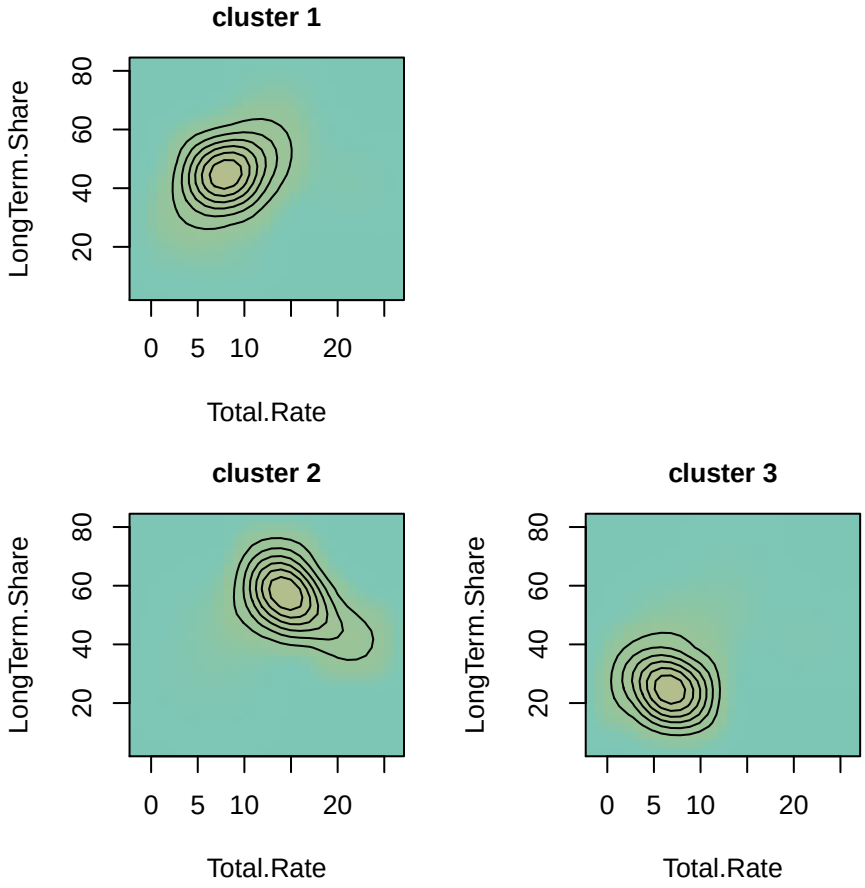


And for the other two clusters:

```
> w <- membership [,2]
> wfh.2 = pdfmv.cks (unemployment, w = w / sum (w) )
> w <- membership [,3]
> wfh.3 = pdfmv.cks (unemployment, w = w / sum (w) )
```

All three:

```
> plot (wfh.1, main="cluster 1")
> plot (wfh.2, main="cluster 2")
> plot (wfh.3, main="cluster 3")
```


cluster 1
LongTerm.Share
20
40
60
80
0
5
10
20
Total.Rate
cluster 2
LongTerm.Share
20
40
60
80
0
5
10
20
Total.Rate
cluster 3
LongTerm.Share
20
40
60
80
0
5
10
20
Total.Rate

Appendix G:
Data Preparation

```
> prep.ph.data (eval=FALSE, echo=TRUE)
```

```
data (Traffic, package="MASS")
traffic.table <- table (Traffic$y [Traffic$limit=="yes"])
traffic.bins <- as.integer (names (traffic.table) )
traffic.bins <- cbind (naccidents=traffic.bins)
traffic.freq <- as.vector (traffic.table)

trees2 <- as.matrix (datasets::trees)[,c (2, 1, 3)]
colnames (trees2) <- tolower (colnames (trees2) )
#Height (-> m)
trees2 [,"height"] <- 0.3048 * trees2 [,"height"]
#Girth (-> cm)
trees2 [,"girth"] <- 2.54 * trees2 [,"girth"]
#Volume (-> m ^ 3)
trees2 [,"volume"] <- 0.0283168 * trees2 [,"volume"]

height <- sort (trees2 [,"height", drop=FALSE])

quakes2 <- as.matrix (datasets::quakes)[,c (2, 1, 3:4)]

crimes <- cbind (state.x77 [,1, drop=FALSE] * 1e3, as.matrix (USArrests [,-3]) / 1e5)

crime.type <- as.factor (rep (colnames (crimes)[-1], each=50) )
crime.type <- list (crime.type=crime.type)
n.arrests <- as.vector (round (crimes [,1] * crimes [,-1]) )

species <- list (species=iris$Species)
sepal.length <- cbind (sepal.length = iris$Sepal.Length)
sepal.width <- cbind (sepal.width = iris$Sepal.Width)

data (unemployment, package="fclust")
unemployment <- as.matrix (unemployment)[,-2]
```

Appendix H: Datasets

```
> ht <- function (object)
 {   print (head (object, 2) )
     print (tail (object, 2) )
 }

> ht (cbind (traffic.bins, freq=traffic.freq) )

     naccidents freq
[1,]          7    1
[2,]          9    5
      naccidents freq
[23,]         41    2
[24,]         42    1

> ht (trees2)

     height  girth   volume
[1,] 21.336 21.082 0.291663
[2,] 19.812 21.844 0.291663
       height  girth   volume
[30,] 24.3840 45.720 1.444157
[31,] 26.5176 52.324 2.180394

> ht (quakes2)

       long    lat depth mag
[1,] 181.62 -20.42   562 4.8
[2,] 181.03 -20.62   650 4.2
          long    lat depth mag
 [999,] 187.80 -17.40    40 4.5
[1000,] 170.56 -21.59   165 6.0

> ht (crimes)

        Population   Murder Assault     Rape
Alabama    3615000 0.000132 0.00236 0.000212
Alaska      365000 0.000100 0.00263 0.000445
          Population  Murder Assault     Rape
Wisconsin    4589000 2.6e-05 0.00053 0.000108
Wyoming       376000 6.8e-05 0.00161 0.000156

> ht (data.frame (crime.type=crime.type$crime.type, n.arrests) )

  crime.type n.arrests
1     Murder       477
2     Murder        36
    crime.type n.arrests
149       Rape       496
150       Rape        59

> ht (data.frame (species, sepal.length, sepal.width) )

  species sepal.length sepal.width
1  setosa          5.1         3.5
2  setosa          4.9         3.0
      species sepal.length sepal.width
```

```
149 virginica          6.2         3.4
150 virginica          5.9         3.0
```

```
> ht (unemployment)
```

```
         Total.Rate LongTerm.Share
BELGIUM         7.1           48.3
BULGARIA       11.2           56.2
        Total.Rate LongTerm.Share
CROATIA       13.4           63.8
TURKEY         8.8           23.7
```