probhat 0.2.0

Multivariate Generalized Kernel Smoothing and Related Statistical Methods

Abby Spurdle

December 10, 2019

Constructs, plots and evaluates probability distributions (probability mass/density functions, cumulative distribution functions and quantile functions) with continuous kernel smoothing, and to a lesser extent, discrete kernel smoothing. Supports univariate, multivariate and conditional distributions, including multivariate-conditional distributions. Also, supports other probability distributions (categorical, frequency and empirical-like) and weighted data, which is possibly useful mixed with fuzzy clustering. Furthermore, there are extensions for computing multivariate probabilities and multivariate random numbers, and for parameter and mode estimation.

*****note that this package is subject to change*****

Pre-Intro

This package is based on self-referencing function objects.

Some functions return objects (here, mostly probability distributions), which are also functions.

The resulting function objects have attributes, which are accessible inside the function body.
(This enables functions to be bundled with data).

In this context, it's equivalent to the {d, p, q, r} approach used in R's stats package.

Introduction

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

Kernel smoothing is generalized, by estimating:

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

Also, there are categorical and empirical-like distributions, both of which, may be weighted.

Specifically, this includes:

- With discrete kernel smoothing (∼DKS):
  (Primarily, for modelling smoothed-frequency distributions).
  - Univariate probability mass function ($PMF_{(UV)}$∼DKS).
  - Univariate cumulative distribution function ($CDF_{(UV)}$∼DKS).
  - Univariate quantile function ($QF_{(UV)}$∼DKS).
- With continuous kernel smoothing (∼CKS):
  - Univariate probability density function ($PDF_{(UV)}$∼CKS).
  - Univariate cumulative distribution function ($CDF_{(UV)}$∼CKS).
  - Univariate quantile function ($QF_{(UV)}$∼CKS).
  - ■ Multivariate probability density function ($PDF_{(MV)}$∼CKS).
  - ■ Multivariate cumulative distribution function ($CDF_{(MV)}$∼CKS).
  - Conditional probability density function ($PDF_{(C)}$∼CKS).
  - Conditional cumulative distribution function ($CDF_{(C)}$∼CKS).
  - Conditional quantile function ($QF_{(C)}$∼CKS).
  - ■ Multivariate-conditional probability density function ($PDF_{(MVC)}$∼CKS).
  - ■ Multivariate-conditional cumulative distribution function ($CDF_{(MVC)}$∼CKS).
  - ○ Chained quantile function (ChQF∼CKS).
- Categorical distributions (∼CAT):
  - Univariate probability mass function ($PMF_{(UV)}$∼CAT).
  - Univariate cumulative distribution function ($CDF_{(UV)}$∼CAT).
  - Univariate quantile function ($QF_{(UV)}$∼CAT).
  - Conditional probability mass function ($PMF_{(C)}$∼CAT|CKS), conditional on a continuous variable.
- Empirical-like distributions (∼EL):
  - (Univariate) cumulative distribution function (CDF∼EL).
  - (Univariate) quantile function (QF∼EL).
- Distribution sets:
  - Marginal sets.
  - Categorical sets.
  - Conditional sets.

Here, univariate and multivariate models refer to unconditional distributions, unless stated otherwise. Conditional models refers to univariate-conditional distributions, unless stated otherwise. And multivariate-conditional models refer to the special case where a probability distribution is both multivariate and conditional.

By default, ∼DKS models are lower-bounded and have a bandwidth parameter of one, which results in an (unsmoothed) frequency distribution. By default, univariate and conditional PDF/CDF ∼CKS models use a cubic Hermite spline as an intermediate model.

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

empirical cumulative distribution functions. There's a small modification to the (initial) formula, and the resulting points are interpolated by a cubic Hermite spline, in a similar way to ~CKS models.

Also, ~CKS/~CAT/~EL models can be weighted, and I've provided an example of modelling a fuzzy cluster with weighted multivariate kernel density estimation, in an appendix, at the end of this vignette.

There are plot methods for all univariate models and distribution sets, and for multivariate models but only with two 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, including:

- Computation of probabilities, from the CDF.
- Random number generation, from QFs.
- Computation of the mean, standard deviation, variance, skewness and kurtosis, from the PMF or continuous CDF.
- Computation of the median or quantiles, from QFs.
- Mode estimation, using the PMF or PDF.

Noting that all of the above apply to conditional distributions, too.

In principle, there's no multivariate quantile function, however, I've created a (novel) chained quantile function, to support the computation of multivariate random numbers. (i.e. Synthetic data).

Note that currently, this package doesn't support automatic bandwidth selection.

This feature is likely to be added in the next update.

One solution, is to plot marginal PMFs/PDFs, and select the smallest bandwidth that doesn't look like it causing overfitting.

Also note that most of the models in this package work best with matrix objects (not data.frame objects), which have column names.

Preliminary Code

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

```
> library (probhat)
> library (fclust)
> library (scatterplot3d)
```

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

I will set the theme for default colors to green:

```
> use.ph.theme ("green")
```

And I will construct some data objects:

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

This function emulates a script, the contents of which, are given later, in an appendix.

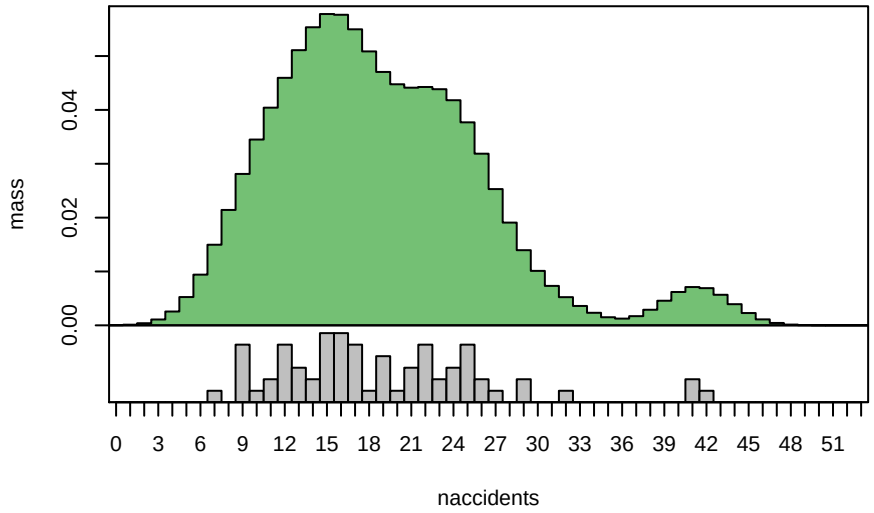
Discrete Kernel Smoothing

We can construct a $\text{PMF}_{(\text{UV})}$~DKS object, using the **pmfuv.dks** constructor.

I will use traffic data, derived from the "Traffic" data in the MASS package:

```
> fh = pmfuv.dks (traffic.x, traffic.h, bw=23, lower=0)
```

```
> plot (fh, TRUE)
```



The "x" variable is number of accidents, and the "y" variable is the frequency.

Refer to the help file, for how to construct these objects, with other combinations of the first two arguments.

The resulting object is a function, which maps an integer vector (of integer quantiles) to a numeric vector (of masses):

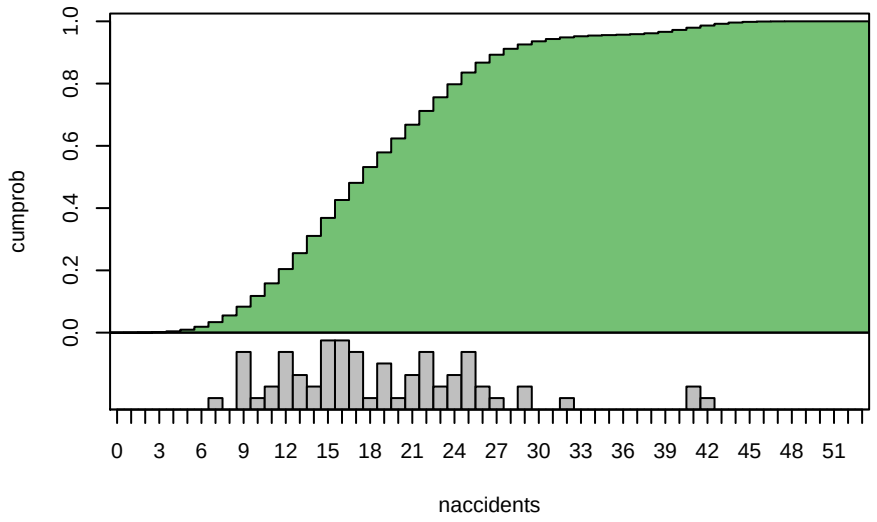
```
> fh (10)
```

```
[1] 0.03449809
```

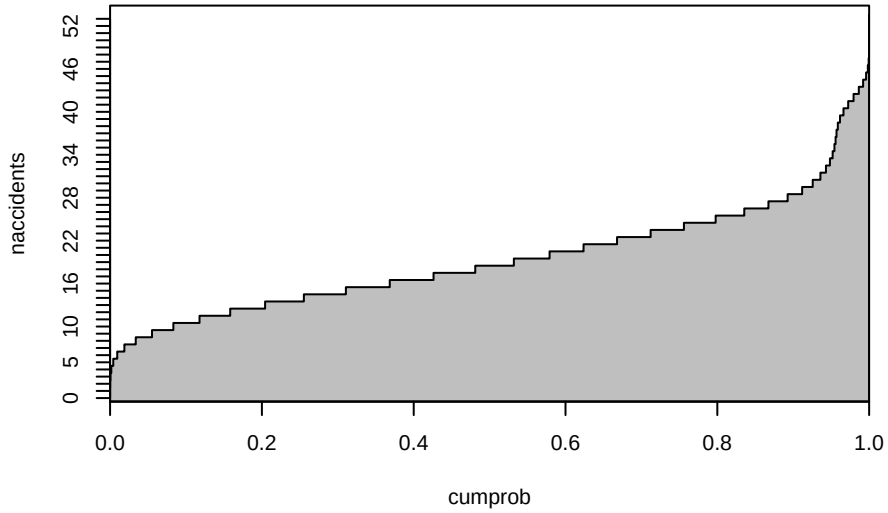
Likewise, we can construct $\text{CDF}_{(\text{UV})}$~DKS and $\text{QF}_{(\text{UV})}$~DKS objects, using the **cdfuv.dks** and **qfuv.dks** constructors:

```
> Fh = cdfuv.dks (traffic.x, traffic.h, bw=23, lower=0)
> Fh.inv = qfuv.dks (traffic.x, traffic.h, bw=23, lower=0)
```

```
> plot (Fh, TRUE)
```




```
> plot (Fh.inv)
```



Discrete quantile functions, are defined in the same way as R's stats package, and map a numeric vector (in the interval [0, 1]) to an integer vector.

Continuous Kernel Smoothing: Univariate Probability Distributions

By default, univariate (and conditional) PDF/CDF ~CKS models use a cubic Hermite spline as an intermediate model, which is more efficient, if the probability distribution needs to be evaluated many times.

$QF_{(UV)}$~CKS (and $QF_{(C)}$~CKS) models use either a cubic Hermite spline or a nested spline.

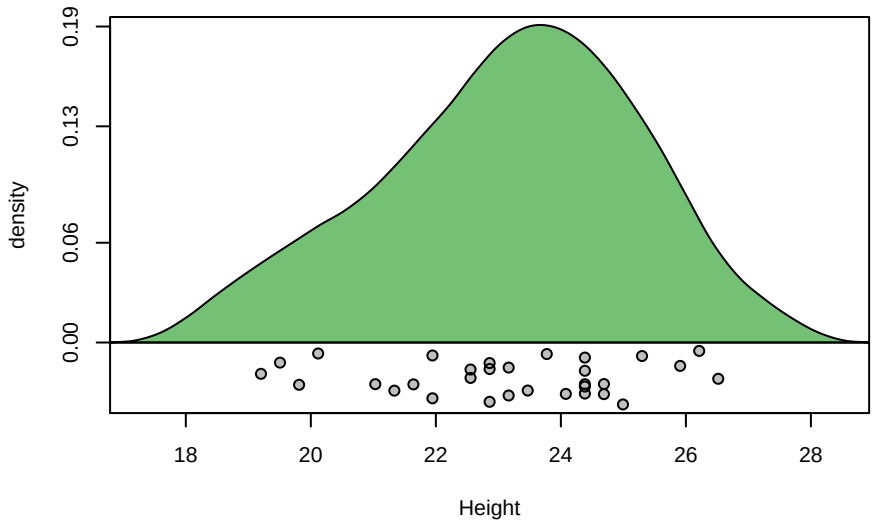
In principle, $QF_{(UV)}$~CKS models are constructed by transposing the CDF, however, if there are level sections in the CDF (i.e. zero-density regions in the corresponding PDF),

then a nested spline is constructed, with a separate cubic Hermite spline for each increasing section of the CDF.

We can construct a $\text{PDF}_{(\text{UV})}$~CKS object, using the **pdfuv.cks** constructor.

I will use the height variable derived from the "trees" data in the datasets package:

```
> fh = pdfuv.cks (Height)
> plot (fh, TRUE)
```



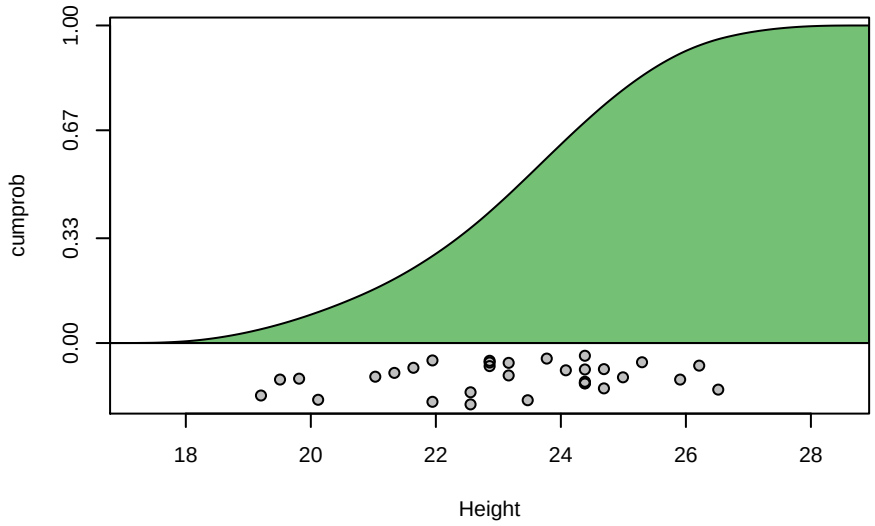
Note that this data has been converted to metric.

The resulting object is a function, which maps a numeric vector (of one or more quantiles) to a numeric vector (of one or more densities):

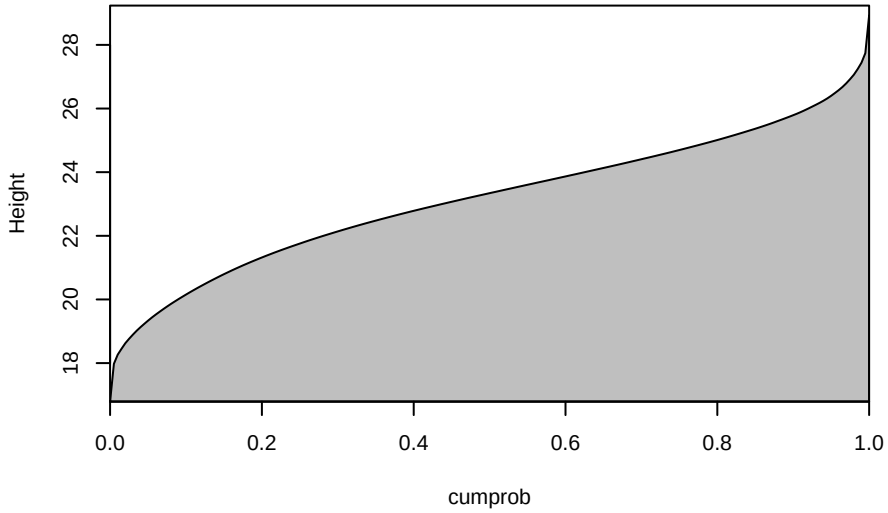
```
> fh (22)
[1] 0.1334647
```

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

```
> Fh = cdfuv.cks (Height)
> Fh.inv = qfuv.cks (Height)
> plot (Fh, TRUE)
```




```
> plot (Fh.inv)
```



Continuous quantile functions, map a numeric vector (in the interval [0, 1]) to a numeric vector.

Note that here, they're not the exact inverse of the CDF, however, they become closer to the CDF, if the number of control points in the spline is increased.

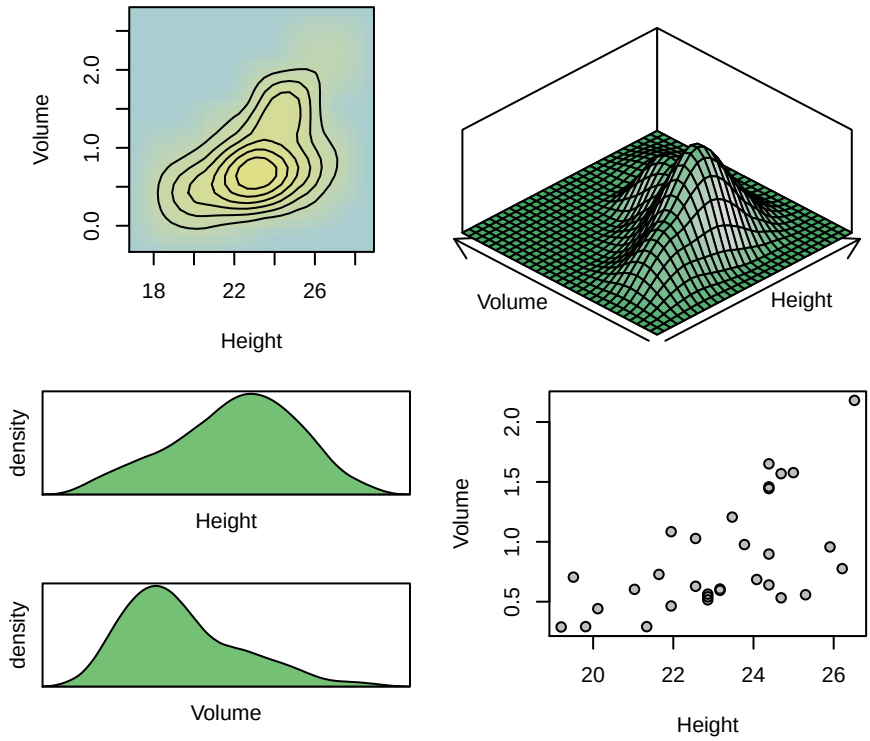
```
> p1 = 0.5
> p2 = Fh (Fh.inv (p1) )

> p1 == p2

[1] FALSE

> abs (p1 - p2)

[1] 2.646609e-05
```

Continuous Kernel Smoothing: Multivariate Probability Distributions

We can construct a $\text{PDF}_{(\text{MV})}$~CKS object, using the **pdfmv.cks** constructor.

Again, I will use the "trees" data:

```
> fh = pdfmv.cks (trees [,2:3])
> plot (fh, all=TRUE)
```

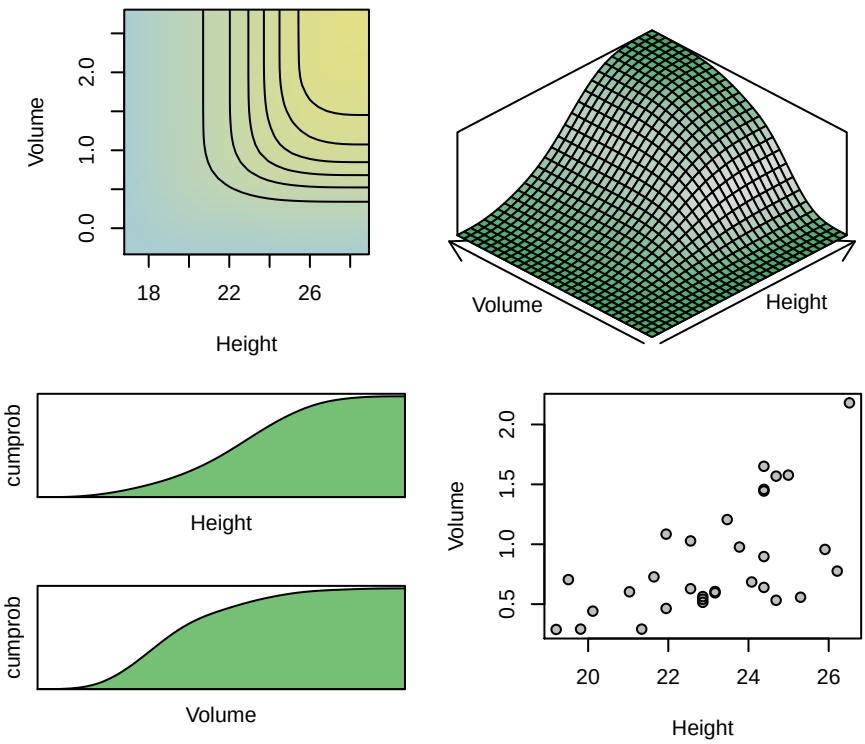


The resulting object is a function, which maps a numeric vector (implying a single row matrix) or matrix to a numeric vector:

```
> fh (c (22, 0.8) )
[1] 0.134061
```

Likewise, we can construct a $\text{CDF}_{(\text{MV})}$~CKS object, using the **cdfuv.cks** constructor:

```
> Fh = cdfmv.cks (trees [,2:3])
> plot (Fh, all=TRUE)
```

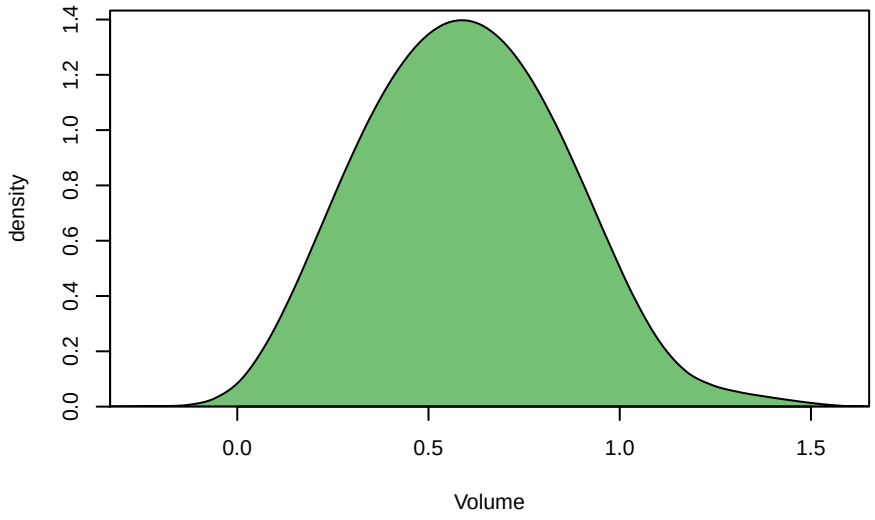


Also, it's possible to compute what I refer to as chained quantile functions, discussed later.

Continuous Kernel Smoothing: Conditional Probability Distributions

We can construct a $\text{PDF}_{(\text{C})}\sim$CKS object, using the **pdfc.cks** constructor:

```
> conditions = c (Girth=30, Height=22)
> cfh = pdfc.cks (trees, conditions=conditions)

> plot (cfh)
```



As with previous objects, it's a function which can be evaluated:

```
> #density of volume (volume=0.85), given girth=30 and height=22
> cfh (0.85)

[1] 0.9716484
```

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

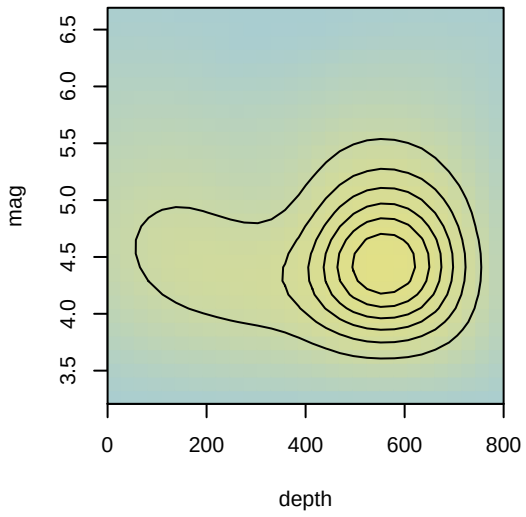
The resulting functions are almost identical to univariate functions, so I will bypass the examples.

Continuous Kernel Smoothing: Multivariate-Conditional Distributions

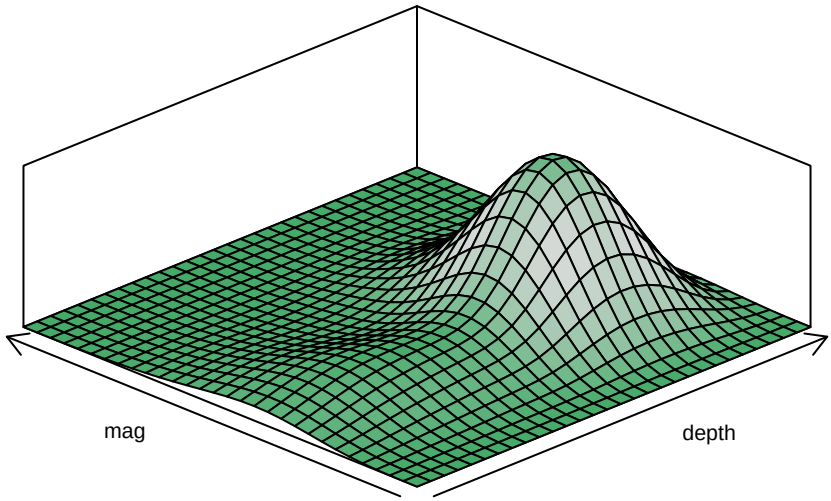
We can construct a $\text{PDF}_{(\text{MVC})}$~CKS object, using the **pdfmvc.cks** constructor.

I will use four variables from the "quakes" data in the datasets package:

```
> conditions = c (lat=-20, long=180)
> cfh = pdfmvc.cks (quakes, conditions=conditions)

> plot (cfh, xlim = c (0, 800) )
```




```
> plot (cfh, TRUE, xlim = c (0, 800) )
```



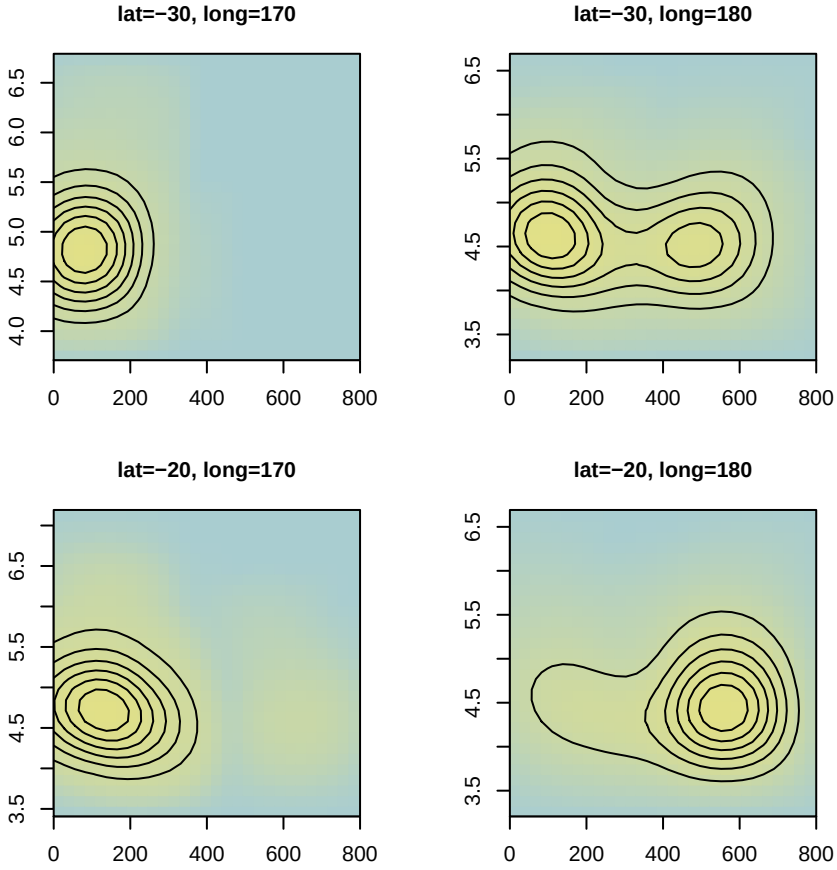
The model above, gives the bivariate density estimate of depth and magnitude, conditional on the near-mean values of latitude and longitude.

I found this interesting, because it suggests that, for a given location, there are two depth-based clusters of earthquakes.

So, here's some similar models, but using different locations:

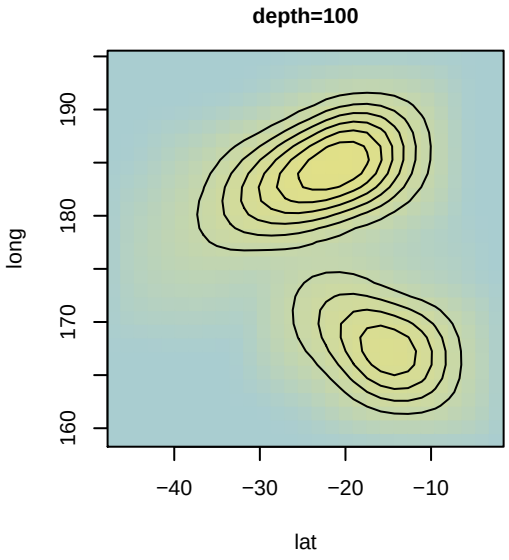
```
> cfh.AA = pdfmvc.cks (quakes, conditions = c (lat=-30, long=170) )
> cfh.AB = pdfmvc.cks (quakes, conditions = c (lat=-30, long=180) )
> cfh.BA = pdfmvc.cks (quakes, conditions = c (lat=-20, long=170) )
> cfh.BB = cfh
```

```
> plot_2x2 (cfh.AA, cfh.AB, cfh.BA, cfh.BB,
     "lat=-30, long=170", "lat=-30, long=180",
     "lat=-20, long=170", "lat=-20, long=180",
     xlim = c (0, 800) )
```

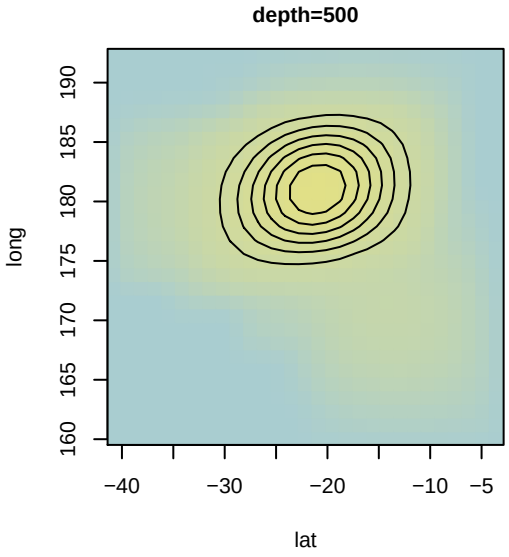


And what if we reverse the relationship (and model latitude and longitude conditional on depth), and ignore magnitude...

```
> cfh.A = pdfmvc.cks (quakes [,-4], conditions = c (depth=100) )
> cfh.B = pdfmvc.cks (quakes [,-4], conditions = c (depth=500) )

> plot (cfh.A, main="depth=100")
```



```
> plot (cfh.B, main="depth=500")
```



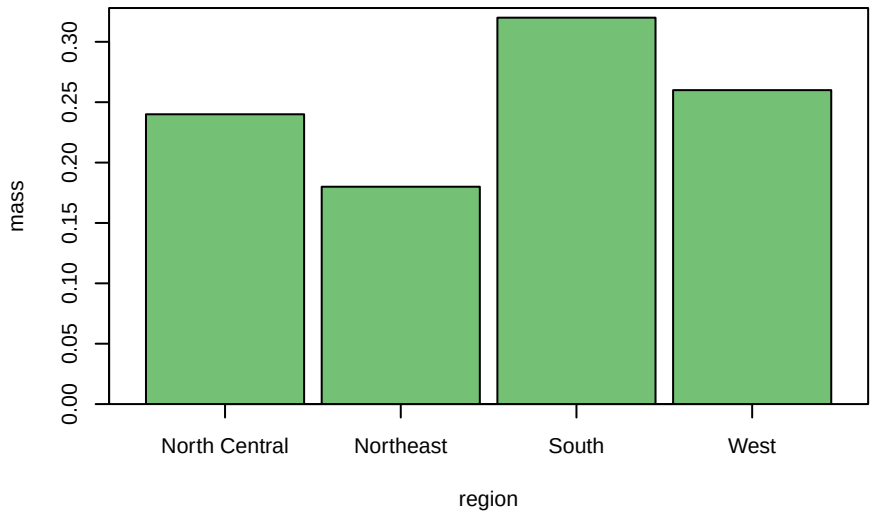
Likewise, we can construct a $\text{CDF}_{(\text{MVC})}$∼CKS object, using the **cdfmvc.cks** constructor, however, I'm going to bypass the example.

Categorical Distributions

We can construct a $\text{PMF}_{(\text{UV})}$∼CAT object, using the **pmfuv.cat** constructor.

I will use the state region variable from the datasets package:

```
> fh = pmfuv.cat (region)
> plot (fh)
```



The resulting object is a function, which maps an integer or character vector to a numeric vector:

```
> fh (1)
```

```
[1] 0.24
```

```
> fh ("North Central")
```

```
[1] 0.24
```

Likewise, we can construct $\text{CDF}_{(\text{UV})}$~CAT and $\text{QF}_{(\text{UV})}$~CAT objects, using the **cdfuv.cat** and **qfuv.cat** constructors. However, as the categorical variable is not clearly ordinal, I will bypass the examples.

Categorical Distributions Conditional on A Continuous Variable

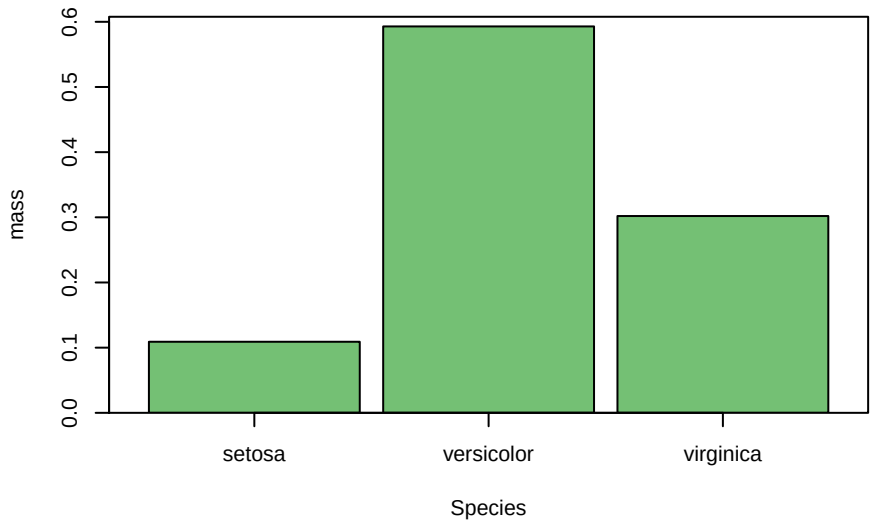
It's possible to construct a $\text{PMF}_{(\text{C})}$~CAT|CKS object, using the **pmfc.cat.cks** constructor.

This gives a categorical distribution (currently, univariate PMF only) conditional on a continuous variable.

I will use the "iris" data from the datasets package:

```
> mean.Sepal.Length = mean (iris.Sepal.Length)
> fh = pmfc.cat.cks (iris.Species, iris.Sepal.Length, at=mean.Sepal.Length)
```

```
> plot (fh)
```



Currently, $\text{PMF}_{(\text{C})}$~CAT|CKS models are derived by using Bayes Theorem with a categorical set, described later.

This in turn, can be used for classification purposes.

Empirical-Like Distributions

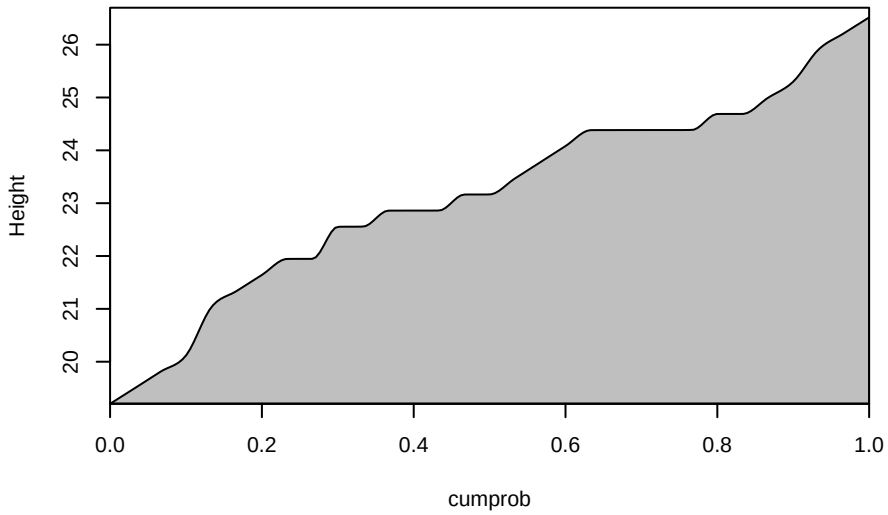
We can construct a CDF~EL object, using the **cdf.el** constructor:

```
> Fh = cdf.el (Height)
```

```
> plot (Fh)
```

Likewise, we can construct a QF∼EL object, using the **qf.el** constructor:

```
> Fh.inv = qf.el (Height)
> plot (Fh.inv)
```



Unlike continuous kernel smoothing, empirical-like models don't "Smooth" the model, they're completely nonparametric. They compute a set of points, representing cumulative probabilities, and interpolate the points with a cubic Hermite spline. However, the resulting functions don't necessarily appear smooth.

∼EL models may be preferable to ∼CKS models, if you want to compute the median or other quantiles, without any smoothing. These quantiles can be regarded as quantiles of the data, itself, rather than estimates derived from a model.

As with $QF_{(UV)}$∼CKS models, QF∼EL models are not the exact inverse their corresponding CDF.

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

Distribution Sets

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

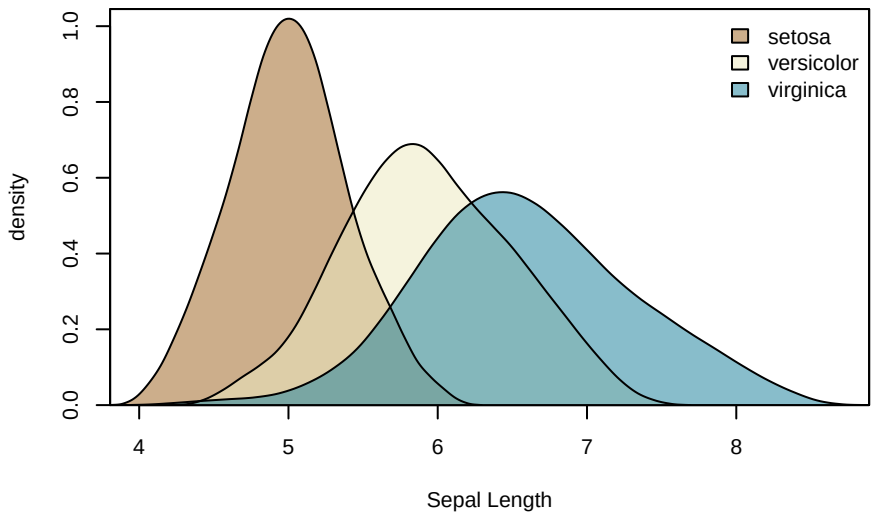
Currently, there are three types, and this may be changed significantly, in the future:

- **Categorical Set**
  One univariate probability distribution for each (categorical) level, out of many (categorical) levels.
- **Conditional Set**
  Similar to a categorical set, except that there's one univariate-conditional probability distribution for each set of conditions, rather than each level.
- **Marginal Set**
  One univariate probability distribution for each variable, out of many variables.

We can construct categorical sets, using any of the univariate constructors, given so far.

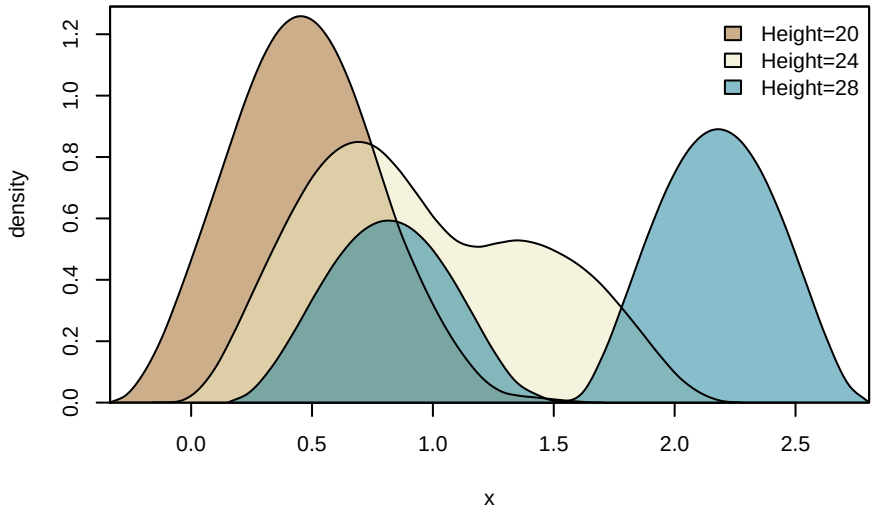
Lets construct $\text{PDF}_{(\text{UV})}$~CKS models of sepal length, grouped by species:

```
> cs = categorical.set (pdfuv.cks, iris.Sepal.Length, group.by=iris.Species)
> plot (cs)
```



Now, lets construct a conditional set, of the density of volume, conditional on different heights:

```
> conditions = cbind (Height = c (20, 24, 28) )
> cond.set = conditional.set (pdfc.cks, trees [,2:3], group.by=conditions)
> plot (cond.set)
```



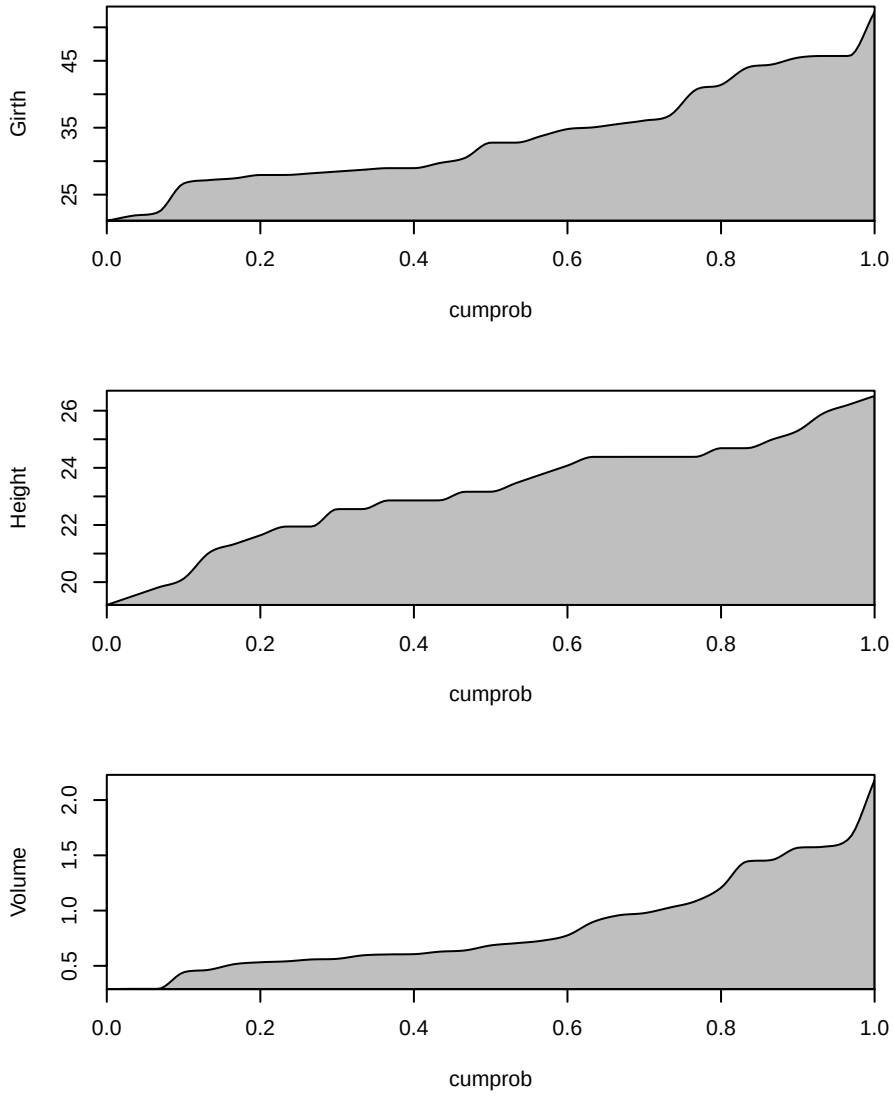
Note that it's possible that the bandwidth is too low, and increasing the bandwidth may minimize the bimodal effect, above.

We can construct marginal sets using any of the univariate constructors, given so far.

Lets construct marginal QF∼EL objects:

```
> ms = marginal.set (qf.el, trees)
```

```
> plot (ms)
```



Multivariate Probabilities

Here, multivariate probability refers to the probability of observing multiple random variables between pairs of lower and upper limits. In theory, such probabilities could be computed from the multivariate PMF or PDF, however (here at least), it's more efficient to compute them from the multivariate CDF.

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

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
> Fh = cdfmv.cks (trees)

> #approximate first and third quartiles
> a = c (28, 22, 0.55)
> b = c (38, 24, 1.05)
```

```
> cbind (lower=a, upper=b)

     lower upper
[1,] 28.00 38.00
[2,] 22.00 24.00
[3,]  0.55  1.05

> #multivariate probability
> probmv (Fh, a, b)

[1] 0.08509282
```

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.

Chained Quantile Functions And Random Number Generation

In addition to the probability distributions presented earlier, it's also possible to construct what I refer to as chained quantile functions.

Standard quantile functions can be used to compute univariate random numbers via (standard) inversion sampling. And chained quantile functions (currently, for continuous kernel smoothing only) can be used to compute multivariate random numbers, via nonstandard inversion sampling.

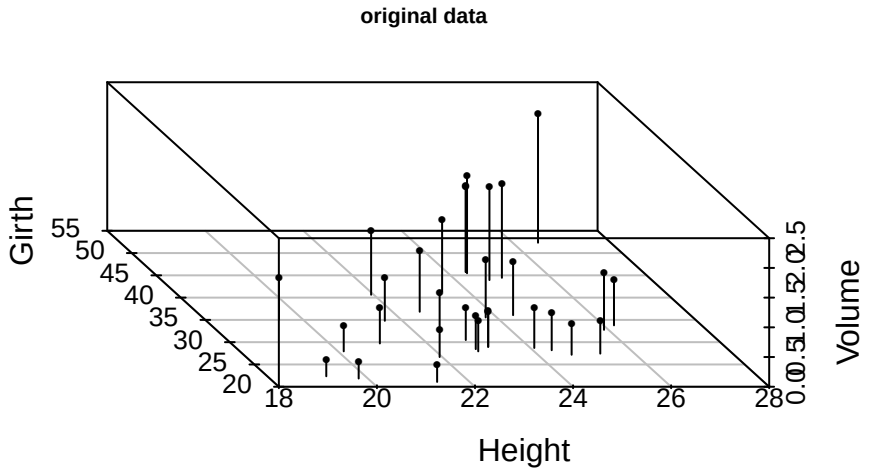
Chained quantile functions work by:

1. Fitting a standard quantile function, to the first variable's observations, $\mathbf{x}^*_{[,1]}$.
2. Using that quantile function to map the first variable's (input) probabilities, $\mathbf{p}_{[,1]}$, to (output) quantiles $\mathbf{q}_{[,1]}$.
3. (Assuming that there are two or more variables).
   Iterating over each evaluation point and each subsequent variable.
   For each (ith) evaluation point and for each (jth) subsequent variable:
   (a) Fitting a conditional quantile function to a subset of variables, $\mathbf{x}^*_{[i,1:j]}$, conditional on the previous variables, $\mathbf{q}_{[i,1:(j-1)]}$.
   (b) And then using that conditional quantile function to evaluate the current point, for the next variable.

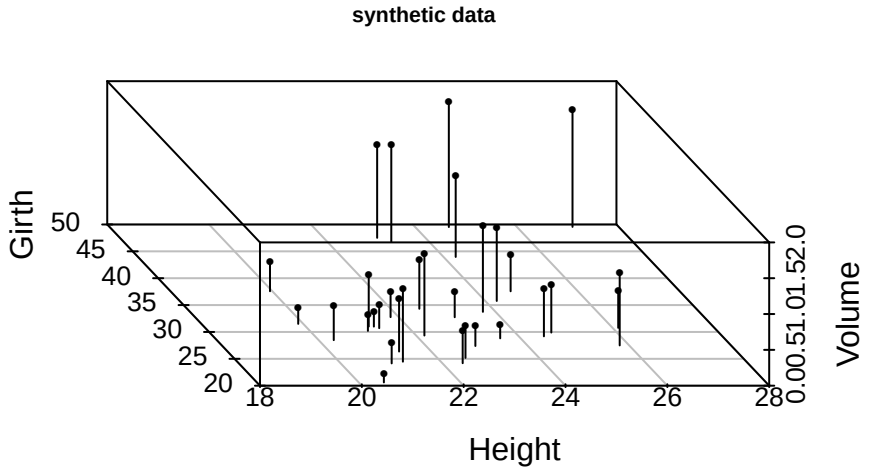
The convenience function, **ph.rng**, takes two arguments, the univariate or chained quantile function, and the number of random numbers to generate, then evaluates the quantile function, using a vector or matrix of uniform random numbers.

```
> chF.inv = chqf.cks (trees)
> synthetic.data = ph.rng (chF.inv, 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 (trees, "original data")
```




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



Note that I've used the same sample size, however, you could use any sample size.

Parameter Estimation

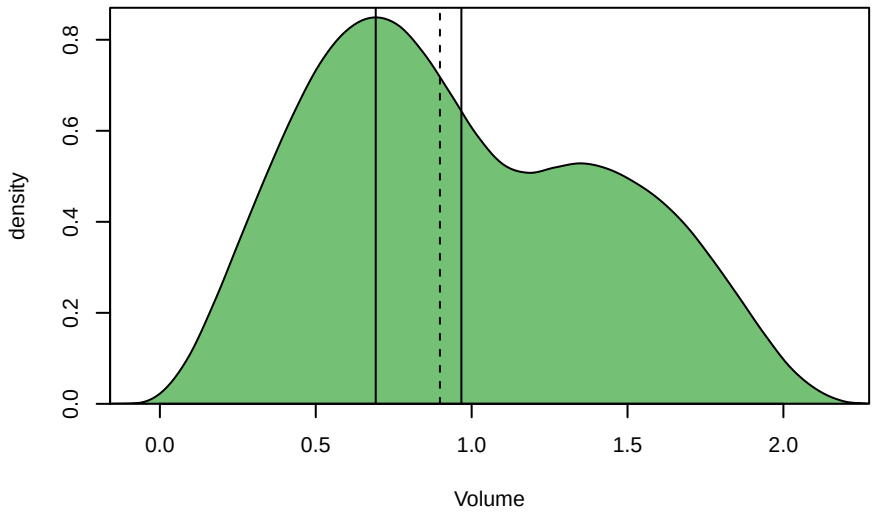
Both univariate and conditional probability distributions can be used to compute probabilities, parameter estimates and other parameter-like estimates. Parameter (and parameter-like) estimates, include the mean, median, mode, other quantiles, standard deviation, variance, skewness and kurtosis.

The mean, standard deviation, variance and higher moments can be computed from the PMF or continuous CDF. The mode can be computed from the PMF or PDF. Probabilities can be computed from the CDF, and the QF can be used for the median and other quantiles.

In the future, I may allow automatic conversion between the PMF/PDF, CDF and QF, however, I note that such conversion may be less efficient.

I'm going to use a conditional distribution, and compute its mean, median and mode:

```
> #conditional distributions
> conditions = c (Height=24)
> cfh = pdfc.cks (trees [,2:3], conditions=conditions)
> cFh = cdfc.cks (trees [,2:3], conditions=conditions)
> cFh.inv = qfc.cks (trees [,2:3], conditions=conditions)

> #mean, median and mode
> mean.Volume = ph.mean (cFh)
> median.Volume = cFh.inv (0.5)
> mode.Volume = ph.mode (cfh)

> cbind (statistic = c ("mean", "median", "mode"),
     value = c (mean.Volume, median.Volume, mode.Volume) )

     statistic value
[1,] "mean"    "0.967117448076443"
[2,] "median"  "0.898280046366254"
[3,] "mode"    "0.6923001776467"

> plot (cfh)
> abline (v = c (mean.Volume, mode.Volume) )
> abline (v=median.Volume, lty=2)
```




```
> #and just as an example, the variance, skewness and kurtosis...
> ph.var (cFh)

[1] 0.224956

> ph.skewness (cFh)

[1] 0.2920473

> ph.kurtosis (cFh)

[1] 2.157884
```

Note that currently, standard deviation, variance and higher moments, should be regarded as unreliable (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.

References

My main sources are:

R's stats Package

Wikipedia

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

intoo: Object Oriented Extensions
Spurdle, A.

barsurf: Bar, Surface and Related Plots
Spurdle, A.

bivariate: Bivariate Probability Distributions
Spurdle, A.

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

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.

kubik: Cubic Hermite Splines
Spurdle, A.

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

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

Appendix A: Multivariate Probabilities

We can compute the probability that a single (continuous) random variable is between a pair of values as:

$$\mathbb{P}(a \leq X \leq b) = F_X(b) - F_X(a)$$

Where a is the lower limit and b is the upper limit.

This is the area under a univariate PDF.

Likewise, we can compute the probability that two (continuous) random variables are between two pairs of values as:

$$\begin{aligned}\mathbb{P}(a_1 \leq X_1 \leq b_1, \quad a_2 \leq X_2 \leq b_2) &= \sum P^{\binom{2}{2}} - \sum P^{\binom{2}{1}} + \sum P^{\binom{2}{1}} \\ &= F_{(X1,X2)}(b_1, b_2) \\ &- [F_{(X1,X2)}(a_1, b_2) + F_{(X1,X2)}(b_1, a_2)] \\ &+ F_{(X1,X2)}(a_1, a_2)\end{aligned}$$

Where $\sum P^{\binom{m}{k}}$ is shorthand for the sum of the m-variate CDF evaluated with each possible combination of k b-terms and (m - k) a-terms.

And where a is a vector of lower limits and b is a vector of upper limits.

This is the volume under the bivariate PDF.

For three and four variables we have:

$$\begin{aligned}&\mathbb{P}(a_1 \leq X_1 \leq b_1, \quad a_2 \leq X_2 \leq b_2, \quad a_3 \leq X_3 \leq b_3) \\ &\quad = \sum P^{\binom{3}{3}} - \sum P^{\binom{3}{2}} + \sum P^{\binom{3}{1}} - \sum P^{\binom{3}{0}} \\ &\quad = F_{(X1,X2,X3)}(b_1, b_2, b_3) \\ &\quad - [F_{(X1,X2,X3)}(a_1, b_2, b_3) + F_{(X1,X2,X3)}(b_1, a_2, b_3) + F_{(X1,X2,X3)}(b_1, b_2, a_3)] \\ &\quad + [F_{(X1,X2,X3)}(a_1, a_2, b_3) + F_{(X1,X2,X3)}(a_1, b_2, a_3) + F_{(X1,X2,X3)}(b_1, a_2, a_3)] \\ &\quad - F_{(X1,X2,X3)}(a_1, a_2, a_3)\end{aligned}$$

$$\begin{aligned}&\mathbb{P}(a_1 \leq X_1 \leq b_1, \quad a_2 \leq X_2 \leq b_2, \quad a_3 \leq X_3 \leq b_3, \quad a_4 \leq X_4 \leq b_4) \\ &\quad = \sum P^{\binom{4}{4}} - \sum P^{\binom{4}{3}} + \sum P^{\binom{4}{2}} - \sum P^{\binom{4}{1}} + \sum P^{\binom{4}{0}}\end{aligned}$$

More generally (given a continuous multivariate CDF, $F_{(X1,X2,...,Xm)}$, for m random variables), we have:

$$\mathbb{P}(a_1 \leq X_1 \leq b_1, \quad a_2 \leq X_2 \leq b_2, \ ..., \ a_m \leq X_m \leq b_m) = \sum_{k \in [0,m]} \left((-1)^{m-k} \sum P^{\binom{m}{k}} \right)$$

Appendix B: Conditional Formulae

We can compute univariate-conditional (continuous) distributions, with one random variable conditional on one other variable, using:

$$\begin{aligned} f_Y(y) &= f_{(Y|X=x)}(y) \\ &= \frac{f_{(X1,X2)}(x_1 = x, x_2 = y)}{f_{X1}(x_1 = x)} \end{aligned}$$

$$\begin{aligned} F_Y(y) &= F_{(Y|X=x)}(y) \\ &= \int_{-\infty}^{y} \frac{f_{(X1,X2)}(x_1 = x, x_2 = u)}{f_{X1}(x_1 = x)} du \end{aligned}$$

We can compute univariate-conditional (continuous) distributions, with one random variable conditional on multiple other variables, using:

$$\begin{aligned} f_Y(y) &= f_{(Y|X1=x1,X2=x2,...,X[m-1]=x[m-1])}(y) \\ &= \frac{f_{(X1,X2,...,Xm)}(\$X, \$Y_{\text{uv}})}{f_{(X1,X2,...,X[\text{ncon}])}(\$X)} \\ &= \frac{f_{(X1,X2,...,Xm)}(x_1 = x_1, x_2 = x_2, ..., x_{[\text{ncon}]} = x_{[\text{ncon}]}, \quad x_m = y)}{f_{(X1,X2,...,X[\text{ncon}])}(x_1 = x_1, x_2 = x_2, ..., x_{[\text{ncon}]} = x_{[\text{ncon}]})} \\ &= \frac{f_{(X1,X2,...,Xm)}(x_1 = x_1, x_2 = x_2, ..., x_{[m-1]} = x_{[m-1]}, \quad x_m = y)}{f_{(X1,X2,...,X[m-1])}(x_1 = x_1, x_2 = x_2, ..., x_{[m-1]} = x_{[m-1]})} \end{aligned}$$

$$\begin{aligned} F_Y(y) &= F_{(Y|X1=x1,X2=x2,...,X[m-1]=x[m-1])}(y) \\ &= \int_{-\infty}^{y} \frac{f_{(X1,X2,...,Xm)}(\$X, \$U_{\text{uv}})}{f_{(X1,X2,...,X_{[\text{ncon}]})}(\$X)} du \\ &= \int_{-\infty}^{y} \frac{f_{(X1,X2,...,Xm)}(x_1 = x_1, x_2 = x_2, ..., x_{[\text{ncon}]} = x_{[\text{ncon}]}, \quad x_m = u)}{f_{(X1,X2,...,X[\text{ncon}])}(x_1 = x_1, x_2 = x_2, ..., x_{[\text{ncon}]} = x_{[\text{ncon}]})} du \\ &= \int_{-\infty}^{y} \frac{f_{(X1,X2,...,Xm)}(x_1 = x_1, x_2 = x_2, ..., x_{[m-1]} = x_{[m-1]}, \quad x_m = u)}{f_{(X1,X2,...,X[m-1])}(x_1 = x_1, x_2 = x_2, ..., x_{[m-1]} = x_{[m-1]})} du \end{aligned}$$

Note that the convention in this package, is that conditioning variables are enumerated first.

In the univariate-conditional case, ncon (the number of conditions) is equal to m (the total number of variables) minus one.

This can be further generalized to compute multivariate-conditional (continuous) distributions, with M random variables conditional on multiple other variables, using:

$$\begin{aligned} f_{Y1,Y2,...YM}(y_1, y_2, ..., y_M) &= f_{(Y1,Y2,...,YM|X1=x1,X2=x2,...,X[\text{ncon}]=x[\text{ncon}])}(y_1, y_2, ..., y_M) \\ &= \frac{f_{(X1,X2,...,Xm)}(\$X, \$Y_{\text{mv}})}{f_{(X1,X2,...,X[\text{ncon}])}(\$X)} \end{aligned}$$

$$\begin{aligned} F_{Y1,Y2,...,YM}(y_1, y_2, ..., y_M) &= F_{(Y1,Y2,...,YM|X1=x1,X2=x2,...,X[\text{ncon}]=x[\text{ncon}])}(y_1, y_2, ..., y_M) \\ &= \int_{-\infty}^{y_1} \int_{-\infty}^{y_2} ... \int_{-\infty}^{y_M} \frac{f_{(X1,X2,...,Xm)}(\$X, \$U_{\text{mv}})}{f_{(X1,X2,...,X[\text{ncon}])}(\$X)} du_M, ..., du_2, du_1 \end{aligned}$$

Where the subexpressions expand as follows:

$\$X : \{x_1 = x_1, x_2 = x_2, ..., x_{[\text{ncon}]} = x_{[\text{ncon}]}\}$

$\$Y_{\text{mv}} : \{x_{[\text{ncon}+1]} = y_1, x_{[\text{ncon}+2]} = y_2, ..., x_{[\text{ncon}+M]} = y_M\}$

$\$U_{\text{mv}} : \{x_{[\text{ncon}+1]} = u_1, x_{[\text{ncon}+2]} = u_2, ..., x_{[\text{ncon}+M]} = u_M\}$

Note that these formulae do not use all the data. A conditional window is computed, and observations outside the window are discarded. There needs to be at least one observation within the conditional window, otherwise, the denominator is undefined.

Note that it's not necessary to compute all of the expression, in each evaluation of the PDF or CDF. Rather, the denominator (which is a multivariate PDF) and the first part of the numerator (given later), can be computed when the object is constructed.

Also, note that it's not necessary to integrate the expression, as such. The algorithms for computing the multivariate PDFs and CDFs via kernel smoothing (also, given later), can be combined.

Appendix C (1): Discrete Kernel Smoothing Formulae

Unstandardized discrete kernels, 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 (or kernel mapping functions), 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 $i \in [1, n]$.

In general, frequencies are integer-valued, however, there's no requirement for this.

Lower-bounded PMFs and CDFs can be computed by modifying the expressions above:

1. Mass-estimates are truncated (i.e. equal to zero) below the lower limit.
2. Remaining estimates are scaled based on the truncated area, such that the remaining mass-estimates sum to one.
3. The x and h values are reflected about the lower limit, prior to smoothing, otherwise, mass-estimates near the lower limit tend to be too small.

Appendix C (2): Continuous Kernel Smoothing Formulae

Standardized continuous kernels, 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 (or kernel mapping functions), 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 the additive-component distributions, 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.

The numerator of univariate-conditional PDFs and CDFs can be computed, as follows:

$$\hat{g}_Y(\mathbf{x}; k, \mathbf{bw}, n, m, \mathbf{x}^*) = \frac{\sum_i \left((\$f_1)) \times ((\$f_2)) \times ... \times ((\$f_{(m-1)})) \times \$f_m\right)}{n}$$
$$\hat{G}_Y(\mathbf{x}; k, K, \mathbf{bw}, n, m, \mathbf{x}^*) = \frac{\sum_i \left((\$f_1)) \times ((\$f_2)) \times ... \times ((\$f_{(m-1)})) \times \$F_m\right)}{n}$$

Where, in general, the values of the subexpressions inside double brackets are precomputed. (i.e. They're computed when objects are constructed, not when top-level functions are evaluated).

The numerator of multivariate-conditional PDFs and CDFs can be computed, as follows:

$$\hat{g}_{\mathbf{Y}}(\mathbf{x}; M, \text{ncon}, k, \mathbf{bw}, n, \mathbf{x}^*)$$
$$= \frac{\sum_i \left((\$f_1)) \times ((\$f_2)) \times ... \times ((\$f_{\text{ncon}})) \times \$f_{(\text{ncon}+1)} \times \$f_{(\text{ncon}+2)} \times ...\$f_{(\text{ncon}+M)} \right)}{n}$$

$$\hat{G}_{\mathbf{Y}}(\mathbf{x}; M, \text{ncon}, k, K, \mathbf{bw}, n, \mathbf{x}^*)$$
$$= \frac{\sum_i \left((\$f_1)) \times ((\$f_2)) \times ... \times ((\$f_{\text{ncon}})) \times \$F_{(\text{ncon}+1)} \times \$F_{(\text{ncon}+2)} \times ...\$F_{(\text{ncon}+M)} \right)}{n}$$

Where M is the number of random variables and ncon is the number of conditions.

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:

$$\sum_i w_i = 1$$

Appendix D:
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:

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

Expanding on an earlier point, 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 E:
Data Preparation

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

```
data (Traffic, package="MASS")
traffic.table = table (Traffic$y [Traffic$limit=="yes"])
traffic.x = as.integer (names (traffic.table) )
traffic.x = COL (traffic.x, "naccidents")
traffic.h = as.vector (traffic.table)

region = COL (datasets::state.region, "region")

iris = datasets::iris
iris.Species = as.character (iris$Species)
iris.Species = COL (iris.Species, "Species")
iris.Sepal.Length = iris$Sepal.Length
iris.Sepal.Length = COL (iris.Sepal.Length, "Sepal Length")

quakes = datasets::quakes
quakes = as.matrix (quakes)[,-5]

trees = datasets::trees
trees = as.matrix (trees)
#Girth (-> cm)
trees [,"Girth"] = 2.54 * trees [,"Girth"]
#Height (-> m)
trees [,"Height"] = 0.3048 * trees [,"Height"]
#Volume (-> m ^ 3)
trees [,"Volume"] = 0.0283168 * trees [,"Volume"]

Height = COL.of (trees, "Height")

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

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 row weights, not column weights, 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 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
```

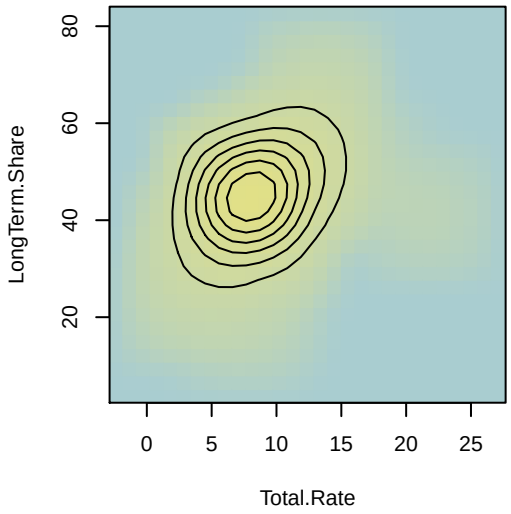
I'm going to extract the weights of the first cluster, and transform them, so that they sum to one:

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

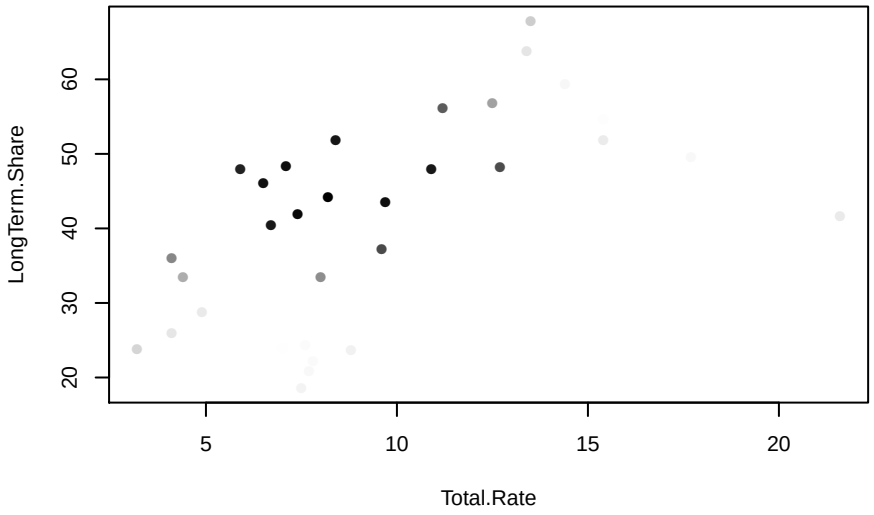
And a weighted model:

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

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




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

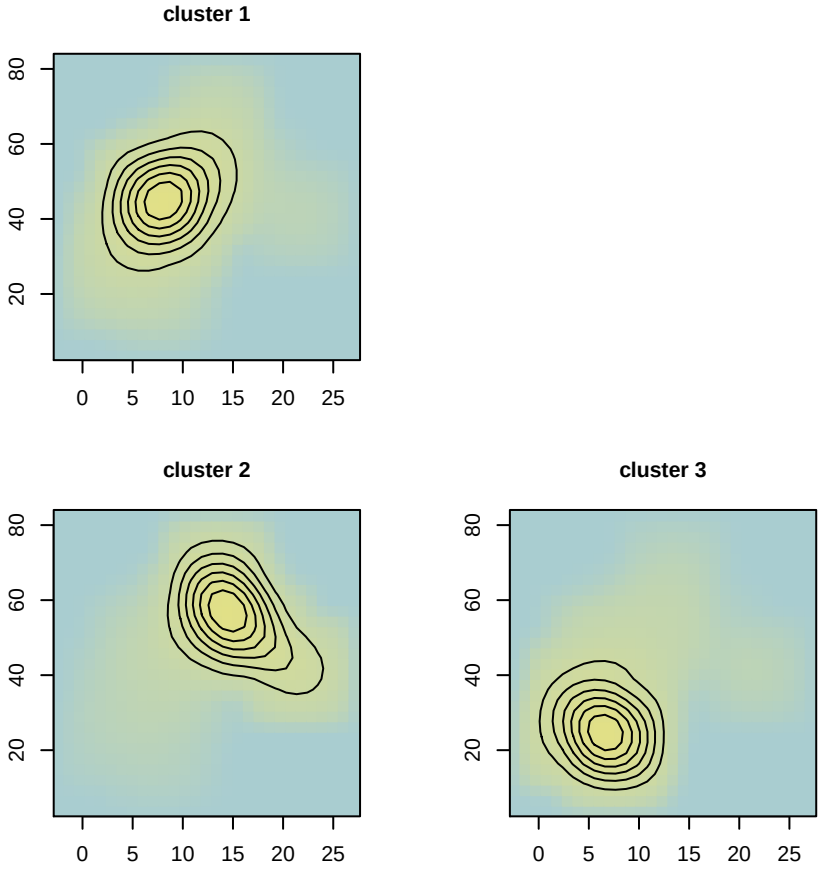


And for the other two clusters:

```
> 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_2x2 (wfh.1,, wfh.2, wfh.3, "cluster 1",, "cluster 2", "cluster 3")
```


cluster 1
cluster 2
cluster 3
20
40
60
80
0
5
10
15
20
25