Introduction to statistical computing in R

Jennifer Wilcock

STAT221 2020 S2 Weeks 7 - 12

The course overall

Covers topics in statistical computing (the use of computers in a statistical context) and computational statistics (computationally intensive statistical methods).

In weeks 1 to 6:

- two topics in statistical computing:
  - programming in R
  - generating pseudo-random numbers from different distributions
- an important area of computational statistics
  - Monte Carlo integration and MC simulation
  - that make use of these random numbers

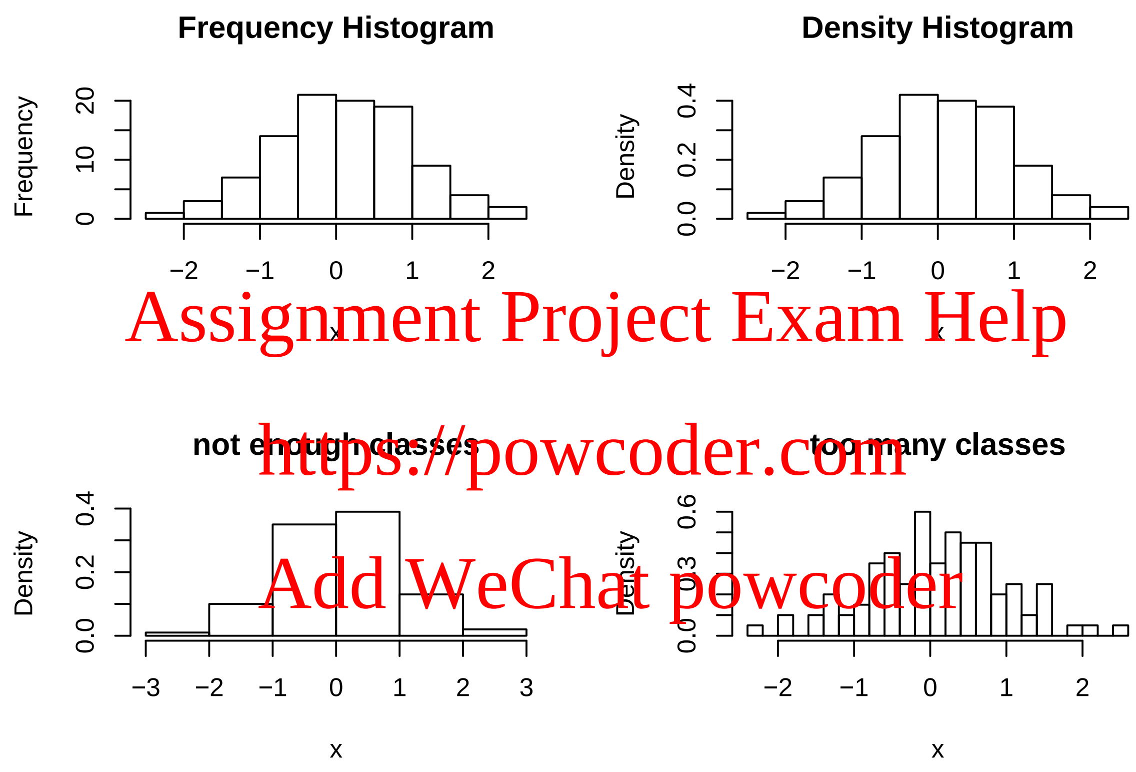
In weeks 7 to 12 the focus will be on working with and understanding the structure of observed data:

- histograms and kernel density estimators for estimating probability densities
- some aspects of statistical graphics in R
- using MC sampling in randomization tests, jack-knife techniques, and bootstrapping, where a given dataset is repeatedly resampled.

Histograms

Some examples in R

```r
set.seed(1)
x = rnorm(100)

# display 4 subgraphs in one (2 rows by 2 columns)
par(mfrow = c(2, 2))
hist(x, main = "Frequency Histogram")
hist(x, freq = F, main= "Density Histogram")
hist(x, prob = T, breaks=5, main = "not enough classes")
hist(x, prob = T, nclass = 30, main = "too many classes")
```



```r
# remember to change back to the default layout afterwards (1 row by 1 column)
par(mfrow = c(1,1))
```

- Notice that the argument `freq = FALSE` can be shortened to `freq = F`
- Also that `freq = F` is same as `prob = T`
- Another way of specifying the number of bins is to use `nclass = 30` instead of `breaks = 30`
- Many functions in R have arguments which overlap in terms of their functionality, so you can use whichever feels most natural to you

We will start by thinking about what's on the y-axis

- **Frequency** histograms count the number of times an observation lands in each bin
- Height of the rectangle is proportional to this number
- Denote the frequency count in bin i as f_i, with the bin width all equal to h
- Scale for the y-axis is commonly:

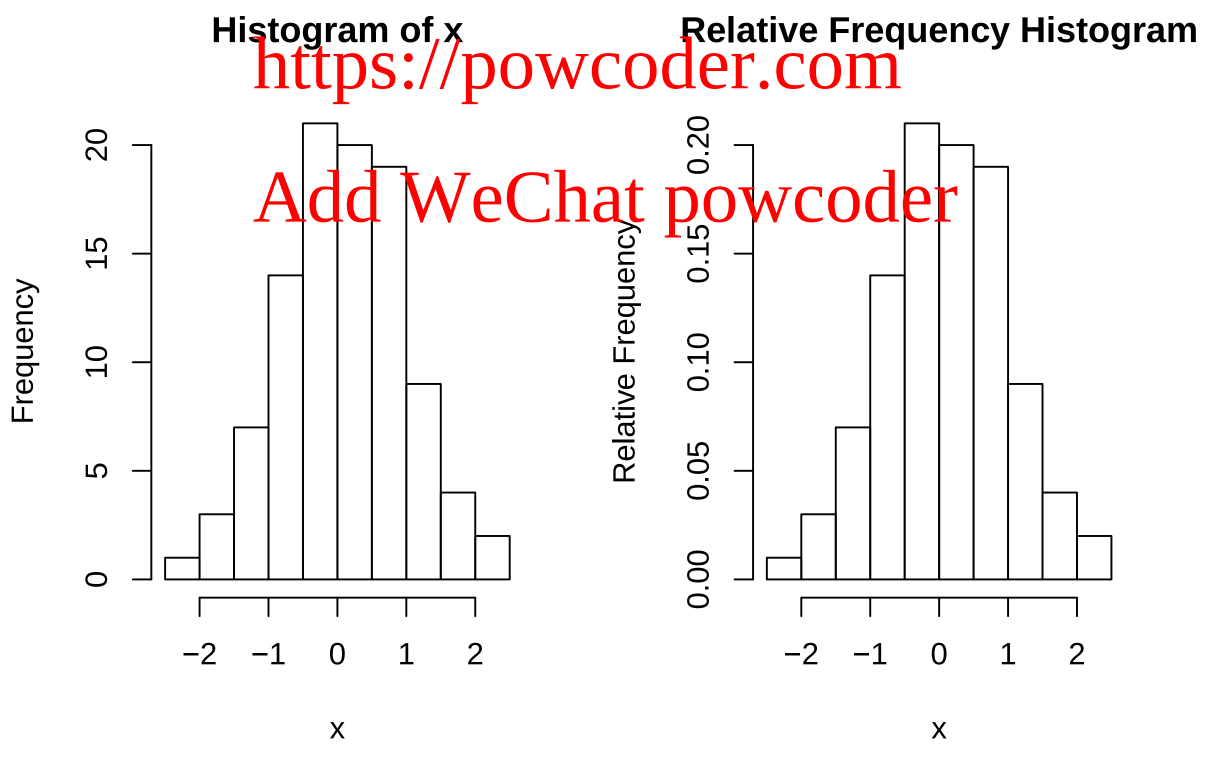
  - frequency count f_i - "**frequency histogram**"
  - proportion $p_i = \frac{f_i}{n}$ - "**relative frequency histogram**"
  - probability density $d_i = \frac{f_i}{n \times h}$ - "**density histogram**"

- We know that the total area of the density histogram bins is equal to 1, just like a probability density
- In this case, the histogram can be thought of as an estimate of the true probability density

Relative Frequency Histograms in R

- Relative frequency histograms cannot be produced directly using the `hist` function
- But you can use the `hist` function to calculate the bin counts, which can then be turned into proportions and plotted separately
- The `plot` function has different "methods" of plotting, depending on what is input
- Here it identifies that an object from the `hist` function is provided so applies the histogram method of plotting

```r
par(mfrow = c(1, 2))
H = hist(x)
H$counts = H$counts / sum(H$counts)
plot(H, freq=TRUE, ylab="Relative Frequency",
     main="Relative Frequency Histogram")
```

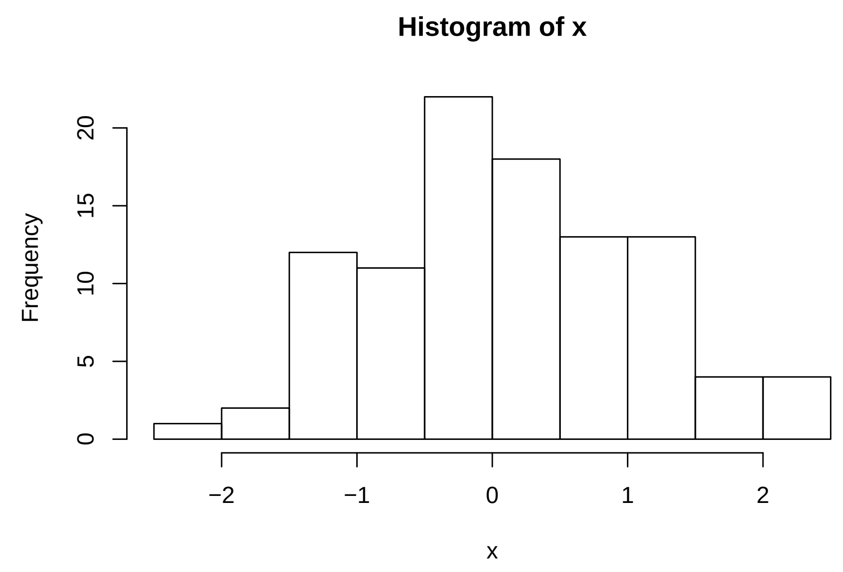


```r
par(mfrow = c(1, 1))
```

What's in a `histogram` object?

When we stored a histogram as an object, we can see many quantities that R is calculating to generate the histogram

```r
set.seed(15)
x = rnorm(100)
H = hist(x)
```



```r
H  # the name of the object we want to look at
```

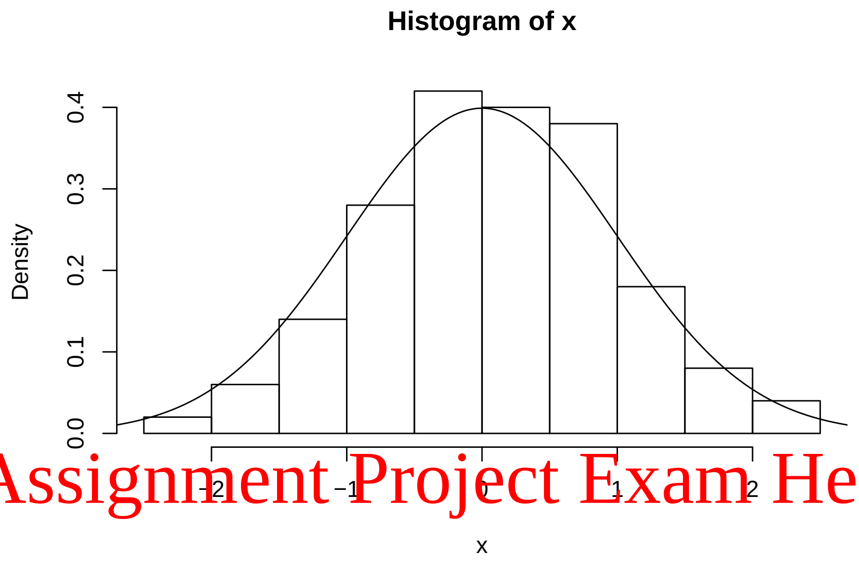
```
## $breaks
##  [1] -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5
##
## $counts
##  [1]  1  2 12 11 22 18 13 13  4  4
##
## $density
##  [1] 0.02 0.04 0.24 0.22 0.44 0.36 0.26 0.26 0.08 0.08
##
## $mids
##  [1] -2.25 -1.75 -1.25 -0.75 -0.25  0.25  0.75  1.25  1.75  2.25
##
## $xname
## [1] "x"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

- `H$breaks` gives the values $t_1, \ldots, t_{b+1}$, where b is the number of bins. Every observation should be at least t_1 and less than t_{b+1}.
- `H$mids` gives the midpoint of each bin interval
- `H$counts` gives the frequency counts in each bin
- `H$density` gives the density height of each bin

Overlaying a density histograms with the true density

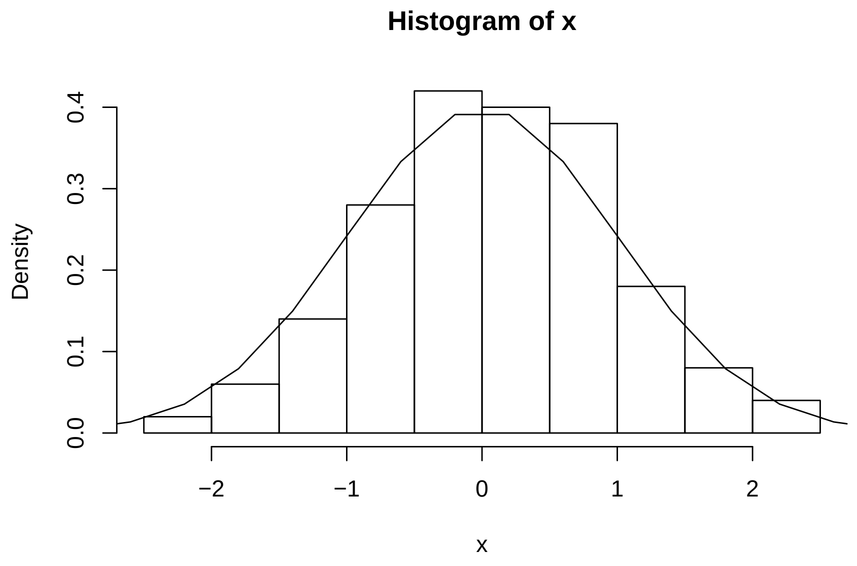
If we know the true density (for example, because we sampled from it), then it is straightforward to overlay the true probability density function over the top of the density histogram

```
set.seed(1)
x = rnorm(100)
hist(x, freq=F)

x.to.plot = seq(-5,5,0.01)
y.to.plot = dnorm(x.to.plot, mean=0, sd=1)
lines(x.to.plot, y.to.plot)
```



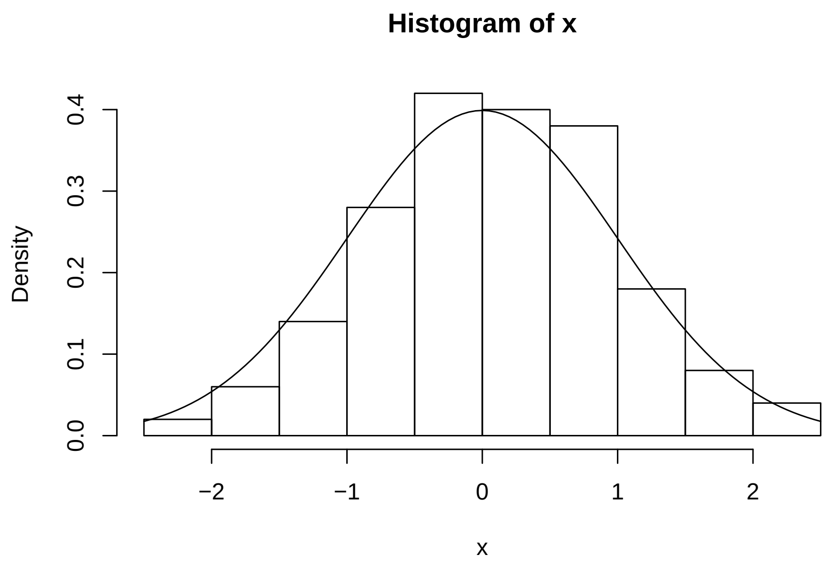
The `lines` command draws straight lines between points, so to draw a 'smooth curve' in R, we just draw straight lines between lots of points that are very close together. Here is the same 'curve' with the points further apart:

```
hist(x, freq=F)

x.to.plot = seq(-5, 5, 0.4)
y.to.plot = dnorm(x.to.plot, mean=0, sd=1)
lines(x.to.plot, y.to.plot)
```



There is also a function `curve` that will do this for you automatically:

```
hist(x, freq=F)
curve(dnorm(x), from = -2.5, to = 2.5, add = TRUE)
```



Now thinking about the x-axis

Plots of the same data can look very different depending on:

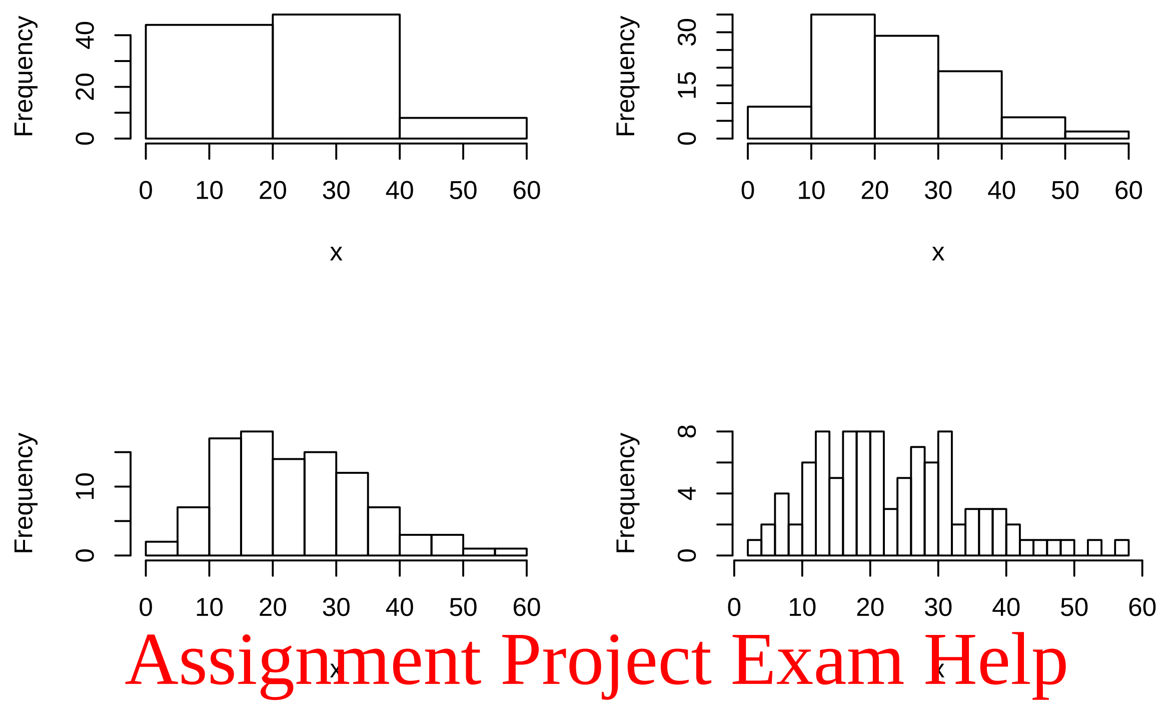
- the bin widths used
- the location of the bin starting points
- whether the bins have equal width or not

We will look at a random sample from a gamma distribution and do some frequency histograms with this dataset:

```
set.seed(1)
x = rgamma(100, shape = 3, scale = 8)
```

and look at the effect of bin width and bin location. First, we look at varying the bin width/number of bins:

```
par(mfrow = c(2, 2))

hist(x, main = "", breaks = 3)
hist(x, main = "", breaks = 4)
hist(x, main = "", breaks = 8 )
hist(x, main = "", breaks = 20)
```



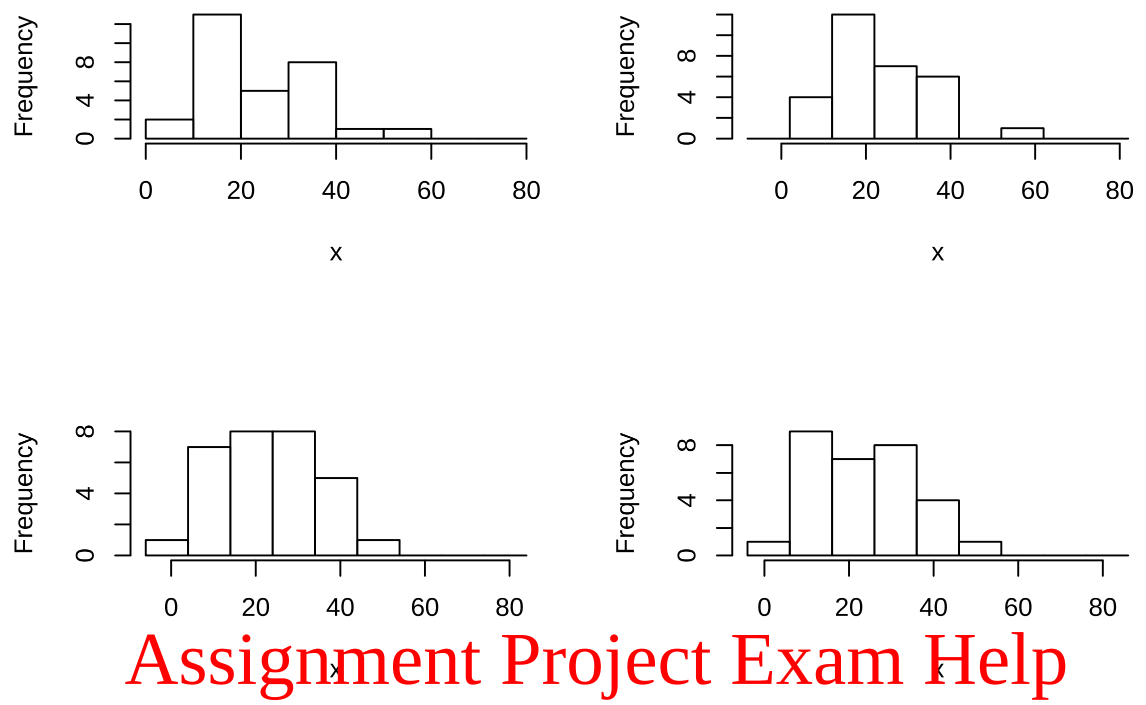
```r
par(mfrow = c(1,1))
```

In general:

- more bins gives a rougher appearance
- either too few or too many bins obscures the structure in the data that we are looking to see.

The appearance can also be affected by the location of the bin cutpoints:

```r
set.seed(1)
x = rgamma(30, shape = 3, scale = 8)

par(mfrow = c(2, 2))

hist(x, main = "", breaks = c(0, 10, 20, 30, 40, 50, 60, 70, 80))
hist(x, main = "", breaks = c(-8, 2, 12, 22, 32, 42, 52, 62, 72, 82))
hist(x, main = "", breaks = c(-6, 4, 14, 24, 34, 44, 54, 64, 74, 84))
hist(x, main = "", breaks = c(-4, 6, 16, 26, 36, 46, 56, 66, 76, 86))
```



```r
par(mfrow = c(1,1))
```

Start of lecture 2 material

Bin Width Choice

- Typically, intervals in histograms are chosen to be equal in length, but they do not have to be
- We have seen that the appearance of a histogram is affected by both the number of bins and also by the location of the break points.
- We can think of the number and locations of the bins (or breakpoints) as 'parameters', where we want to find the optimal values of these parameters for a given dataset or sample so that we can 'see' the structure in the data
- Of these two 'parameters', the most important is choosing the number of bins, or equivalently how wide each bin should be

Algorithms in `hist` for choosing the number of bins

- There are three algorithms available in `hist` for calculating the number of bins for a histogram
- The default algorithm is Sturges rule (`Sturges`)
- The other two algorithms were developed by Scott (`Scott`) and by Freedman and Diaconis (or `FD`)
- Have a look at the help page using `?hist`

- The number of bins determines coarseness of histogram
- The more observations you have, the more classes you can have and still get a "reasonable looking" histogram
- So the algorithms will need to depend on the sample size, n.

Sturges Rule

- Recall that the bin width is called h
- A rule designed for symmetric unimodal distributions (e.g. normal distribution):

$$h = \frac{x_{max} - x_{min}}{\lceil 1 + \log_2 n \rceil}$$

- or, equivalently, that the best number of bins is given by $\lceil 1 + \log_2 n \rceil$
- Here, the sample size is n, and $\lceil y \rceil$ (the ceiling of y) rounds y up to the next integer:

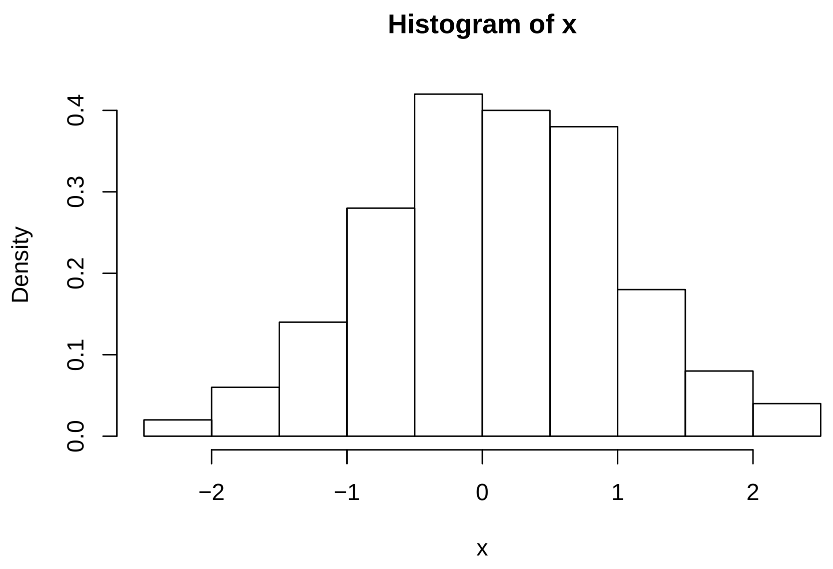
$$\lceil 2.01 \rceil = \lceil 2.9 \rceil = \lceil 3 \rceil = 3$$

Sturges Rule in `hist`

- Sturges rule is a guideline (one of many such 'Rules of Thumb')
- R uses Sturges rule as a default in the `hist` function
- However R also modifies the break points given by Sturges rule to make the histogram look "pretty" (the precise definition of this modification is beyond the scope of this course, but you can have a look at `?pretty`)

Lets go back to this example:

```r
set.seed(1)
x = rnorm(100)
hist(x, freq=F)
```



```r
min(x)
```

```
## [1] -2.2147
```

```r
max(x)
```

```
## [1] 2.401618
```

The minimum and maximum were -2.214700 and 2.401618, so Sturges rule would give a bin width h of

$$h = \frac{2.401618 - (-2.214700)}{\lceil 1 + \log_2 100 \rceil} = 0.5770397$$

This can be calculated in R using

```r
(max(x)-min(x))/ceiling(1+log2(100))
```

```
## [1] 0.5770397
```

However, R actually modified this to a bin width of 0.5 instead, which makes the x-axis easier to read as it gives rounder numbers.

Breakpoints Specification in R

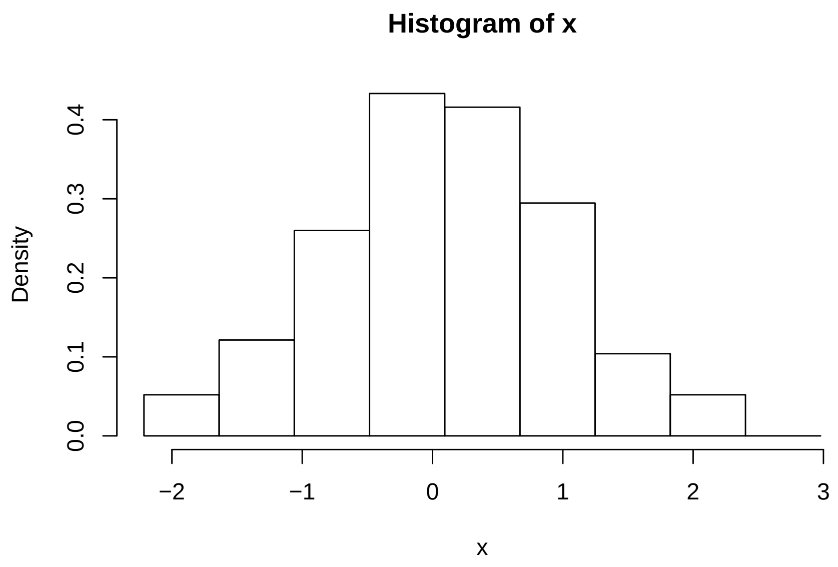
Suppose you want to overrule R and specify the bins in the histogram? For example, suppose you want to apply Sturges Rule exactly . . .

- We have already seen that the `hist` function allows the user to specify either the number of classes/bins, or to specify the *breakpoints* between each bin
- If the break points are $t_1, t_2, \ldots, t_{b+1}$, then an observation x belongs to bin i if $t_i \leq x < t_{i+1}$
- Note that these intervals are left closed and right open
- So if an observation falls exactly on breakpoint t_i then it will go into the upper bin
- However, this convention can be switched to $t_i < x \leq t_{i+1}$ using the `right=TRUE` argument

Forcing Sturges Rule in R

```r
    set.seed(1)
    x = rnorm(100)

    h = (max(x) - min(x)) / ceiling(1 + log2(100)) # Sturges Rule
```

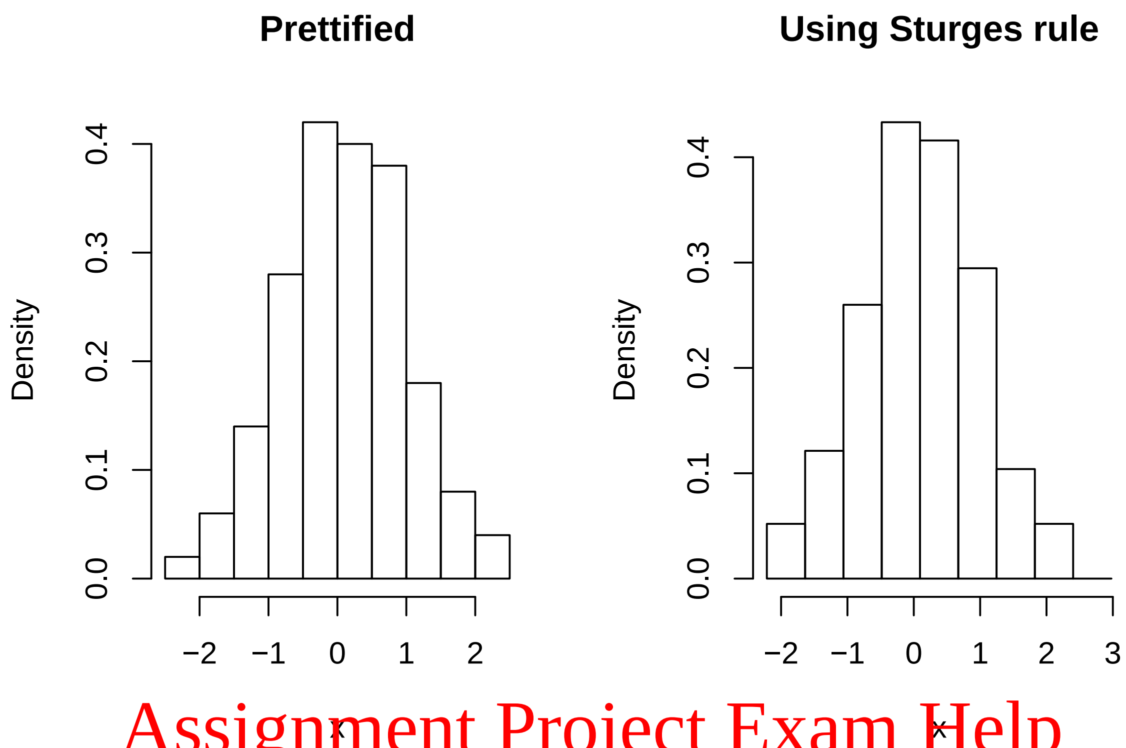
```
breakpoints = seq(min(x), max(x) + h, h)

hist(x, breaks=breakpoints, freq=F)
```



- Notice the trick of the upper limit of the sequence being set to `max(x) + h` to ensure it is beyond the maximum of the data
- If `seq(min(x), max(x), h)` then the last breakpoint will (usually) be below the maximum of the data

We can compare the two plots side-by-side:

```
set.seed(1)
x = rnorm(100)

h = (max(x) - min(x)) / ceiling(1 + log2(100)) # Sturges Rule
breakpoints = seq(min(x), max(x) + h, h)

par(mfrow=c(1,2))
hist(x, freq=F, main="Prettified")
hist(x, breaks=breakpoints, freq=F, main="Using Sturges rule")
```



```r
par(mfrow=c(1,1))
```

Area Under Density Histogram

The area under a density histogram should be equal to 1. This area is the sum of all rectangles and is:

$$\sum_{i=1} f(t_i)h$$

Recall, that we can use the data stored in the object `H` once it has been created by R:

```r
# so we were able to use H$breaks
  H$breaks
```

```
##  [1] -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5
```

```r
  # and H$density
  H$density
```

```
##  [1] 0.02 0.04 0.24 0.22 0.44 0.36 0.26 0.26 0.08 0.08
```

```r
  # to do this calculation.
```

So we can show that this holds empirically using R:

```r
sum(H$density * (H$breaks[2] - H$breaks[1]))
```

```
## [1] 1
```

where `(H$break[2]-H$break[1])` is the bin width h.

Using Histograms to Estimate Densities

- Histograms can be interpreted as an estimate of the density function for a continuous random variable

- Height of the bin at a given value on the x-axis is the density estimate
- Therefore if there are f_i observations in the interval $[t_i, t_{i+1})$, then the estimated density is

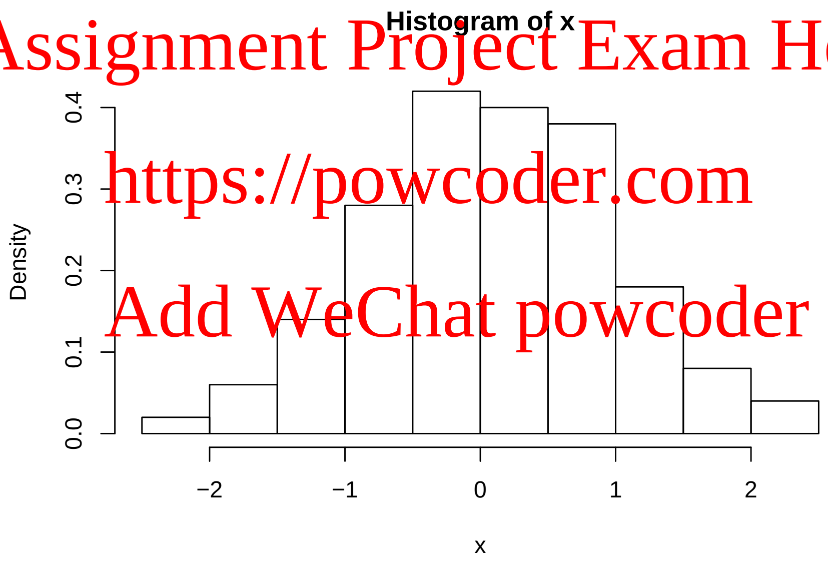
$$\widehat{f}(x) = \frac{f_i I(t_i \le x < t_{i+1})}{nh}$$

  where h is the bin width and I is an indicator function equal to 1 if the condition in parentheses holds and otherwise 0 (we also stated this on page 3 of these notes).

An example

- Lets say you want to estimate the density when $x = 0.6$.
- How do you do this from the histogram output in R?
- First, find the bin in which $x = 0.6$ falls, then obtain the height of the bin
- This can be read off the histogram graph by eye
- But to be more precise, use the histogram object itself:

```r
set.seed(1)
x = rnorm(100)
H = hist(x, freq=F)
```



```r
x.to.find = 0.6

min(which(H$breaks >= x.to.find))
```

```
## [1] 8
```

```r
H$breaks[7:8]
```

```
## [1] 0.5 1.0
```

```r
H$density[7]
```

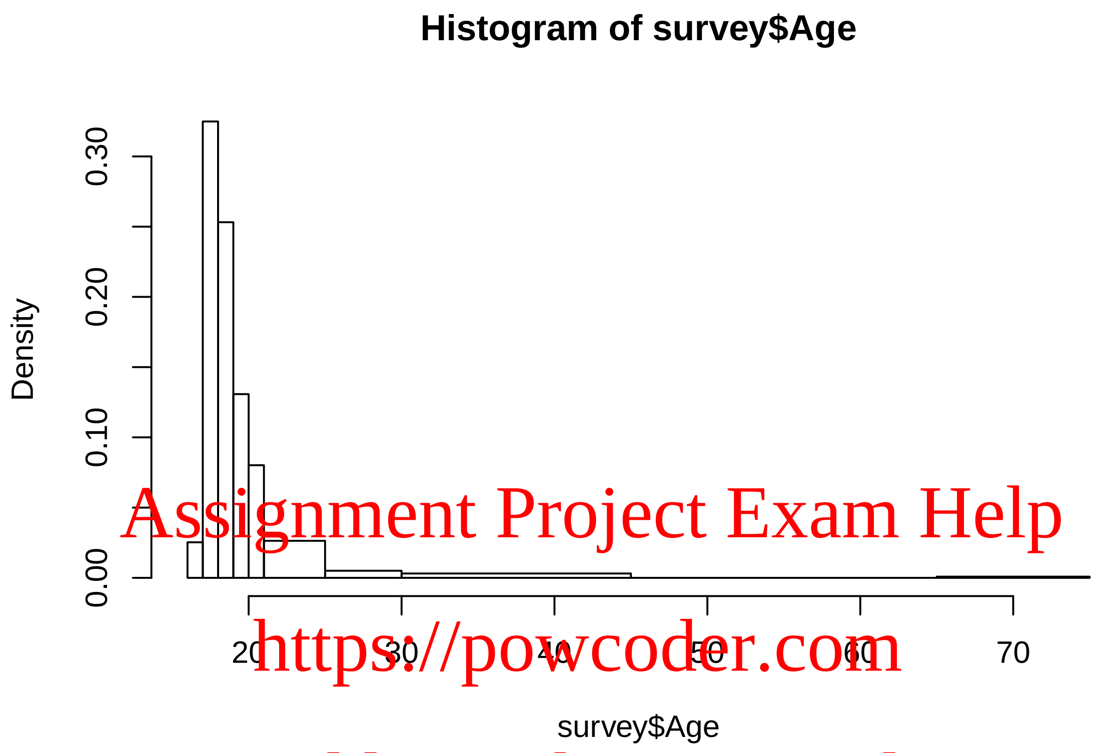
```
## [1] 0.38
```

In this case, $x = 0.6$ is between the 7th and 8th breakpoints which define the interval $[0.5, 1)$, which has density given by :

$$\widehat{f}(0.6) = \texttt{H\$density[7]} = 0.38$$

Example with real data: Ages With Unequal Bin Widths

```
library(MASS)
data(survey)
breakpoints=c(16,17,18,19,20,21,25,30,45,65,75)
H=hist(survey$Age,breaks=breakpoints) # defaults to density for unequal bins
```



The area is the sum of all rectangles, but now the bin width varies in each bin h_i, given by

$$\sum_{i=1}^{b} \widehat{f}(t_i)h_i$$

The bin widths aren't given directly by the `hist` function, but you can use the `diff` function to find them:

```
sum(H$density * diff(H$breaks))
```

```
## [1] 1
```

Other (Equal) Bin Width Rules of Thumb

- If we know the true density, we can compare the histogram estimate of the density with the true density
- One idea is to measure how close Sturges Rule gets to the true density "**on average**" across lots of samples (simulations) from a particular population distribution
- If we have different such Rules of Thumb we could determine which one performs best "on average" (see Wikipedia for examples)
- How do we measure the performance?

Mean Integrated Squared Error

- Let's define the error at any particular x value to be $\widehat{f}(x) - f(x)$ where $f(x)$ is the true density and $\widehat{f}(x)$ is the density estimate
- A common measure of performance is the mean integrated squared error (MISE) evaluated over all values of x:

$$MISE = E\left[\int_{-\infty}^{\infty} \left(\widehat{f}(x) - f(x)\right)^2 \, dx\right] \tag{1}$$

- This measure evaluates the fit of the estimated density to the true density, but averaged (hence the expectation) over all possible samples from the population distribution

Discussion of the MISE

- You've already seen the mean squared error (MSE). The MSE is useful for measuring performance for single parameters, but is not directly suitable for functions
- Hence the need for the integration in the MISE
- Note also that the true density $f(x)$ is not random, but $\widehat{f}(x)$ is random as it depends on the randomly selected sample even for fixed x
- For example, for $x = 0.6$, $\widehat{f}(0.6)$ is random as it depends on the randomly selected sample data that went to make up the density histogram

Start of lecture 3 material

Other (Equal) Bin Width Rules of Thumb

- Minimizing the MISE for the normal density (by choosing optimal bin widths) resulted in Scott's **Normal Reference Rule** for choosing bin width:

$$h = \frac{3.5 \times \widehat{\sigma}}{n^{1/3}}$$

- This normal reference rule is sensitive to outliers, due to the use of the sample standard deviation $\widehat{\sigma}$
- The Freedman-Diaconis Rule replaces the $3.5\widehat{\sigma}$ with twice the interquartile range $(2 * IQR)$ which is more robust to outliers:

$$h = \frac{2 \times IQR}{n^{1/3}}$$

- There are many others! See Wikipedia page on "histogram" for more examples

Which Rule of Thumb to Use?

- There is no master rule telling you which Rule of Thumb to use
- A rule of thumb is that the Sturges rule produces fewer bins than Scott's, and Scott's produces fewer than Freedman-Diaconis, at least when the underlying distribution is normal or exponential
- But this is just a rule of thumb and for particular dataset/population distribution there will be exceptions
- In practice, statisticians often plot the histogram a few times using different numbers of bins until it 'looks right'
- What do we mean by 'looks right'?
  - need bin width large enough so the general shape (bell shaped, skewed, heavy tailed, one or more modes, etc.) of the underlying density can be visualised; and
  - sufficiently small to visualise the sample variation, so that the detail of the sample uncertainty is not lost

A few extra good points, raised by students

- Just like with the density histogram, the area under a relative frequency plot is equal to one. However the height of the bins in a density histogram gives the density, but the height of the bins in a relative frequency plot is only equal to the density when the width of the bins is equal to one.
- In R we can use either = or <- for 'assigning' things, but they mean exactly ythe same thing and you can use either. The modern, up to date thing to write is =, but people who have been using R for a long time often use <-. This is because the software people used before R (called S+, and before that S) used <- so this was also carried over to R. <- is a bit more flexible since you can also write assignments in the opposite direction, using ->.

```
v = 10
v
```

```
## [1] 10
```

```
v <- 11
v
```

```
## [1] 11
```

```
12 -> v
v
```

```
## [1] 12
```

- I mentioned 'generic' functions in lecture 2. These are frequently used functions such as `plot`, `summary`, and `print`, that work on many different kinds of 'objects'. These generic functions work by looking for the 'class' of the object, and then doing the appropriate thing for that class. You can see all of the available methods for a generic function by using the R function `methods`, for example:

```
methods(summary)
```

```
##  [1] summary.aov                    summary.aovlist*
##  [3] summary.aspell*                summary.check_packages_in_dir*
##  [5] summary.connection             summary.data.frame
##  [7] summary.Date                   summary.default
##  [9] summary.ecdf*                  summary.factor
## [11] summary.glm                    summary.infl*
## [13] summary.lm                     summary.loess*
## [15] summary.loglm*                 summary.manova
## [17] summary.matrix                 summary.mlm*
## [19] summary.negbin*                summary.nls*
## [21] summary.packageStatus*         summary.polr*
## [23] summary.POSIXct                summary.POSIXlt
## [25] summary.ppr*                   summary.prcomp*
## [27] summary.princomp*              summary.proc_time
## [29] summary.rlang_error*           summary.rlang_trace*
## [31] summary.rlm*                   summary.srcfile
## [33] summary.srcref                 summary.stepfun
## [35] summary.stl*                   summary.table
## [37] summary.tukeysmooth*           summary.warnings
## see '?methods' for accessing help and source code
```

Variations of the histogram

There are several variations aimed at improving the basic histogram for density estimation, and we will look briefly at a few of these.

The most common is the *frequency polygon*, covered next. The aim here is to 'smooth' the histogram by connecting points on it by a continuous curve. The simplest approach being the frequency polygon.

Another method is the *average shifted histogram*, or ASH, which is also briefly covered. This is intended to overcome the problem of the appearnace of a histogram being sensitive to the location of the bins.

A third is the *histospline*. This is constructed by interpolating knots of the empirical CDF with a cubic spline, and then differentiating this spline (and isn't discussed any further in this course).

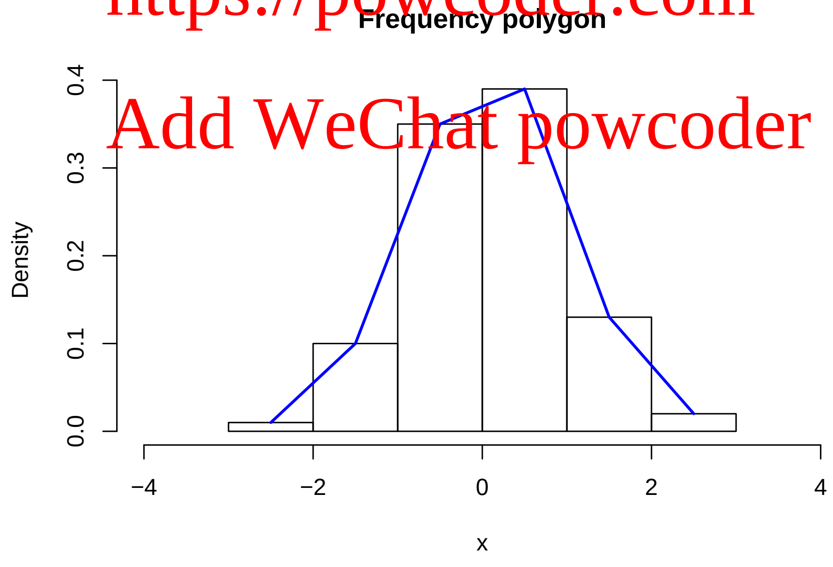
Finally, kernel estimators, where the bins (or kernels) are centred on each data point, and have different shapes. Here the problem of the choice of the location of the bins doesn't arise, although there is still the problem of having to decide which shape of kernel to use).

Frequency polygons

- Histograms are not very good density estimators as they are discontinuous
- An obvious alternative, therefore, is to use frequency polygons which are continuous.

For example

```
set.seed(1)
x = rnorm(100)
H = hist(x, breaks=5, freq=F, main="Frequency polygon", xlim=c(-4,4))

lines(H$mid, H$density, lwd=2, col="blue")
```

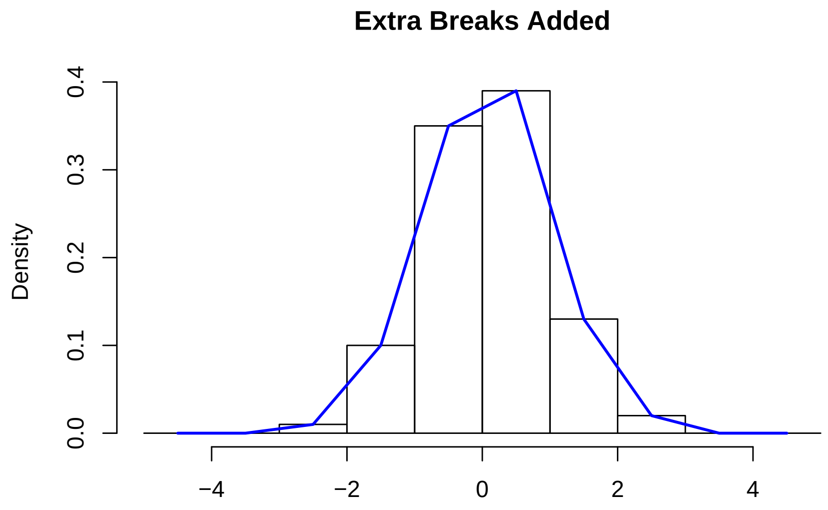


- For a frequency polygon, you can evaluate $\widehat{f}(x_i)$, for each x_i that is at the midpoint of an interval $[t_i, t_{i+1})$, and connect this points with straight lines (i.e., linear interpolation)
- The vertices of the polygon are at the values `(H$mids, H$density)` from the `histogram` object

This is very basic, but is 'smoother' than the histogram itself, so possibly an improvement.

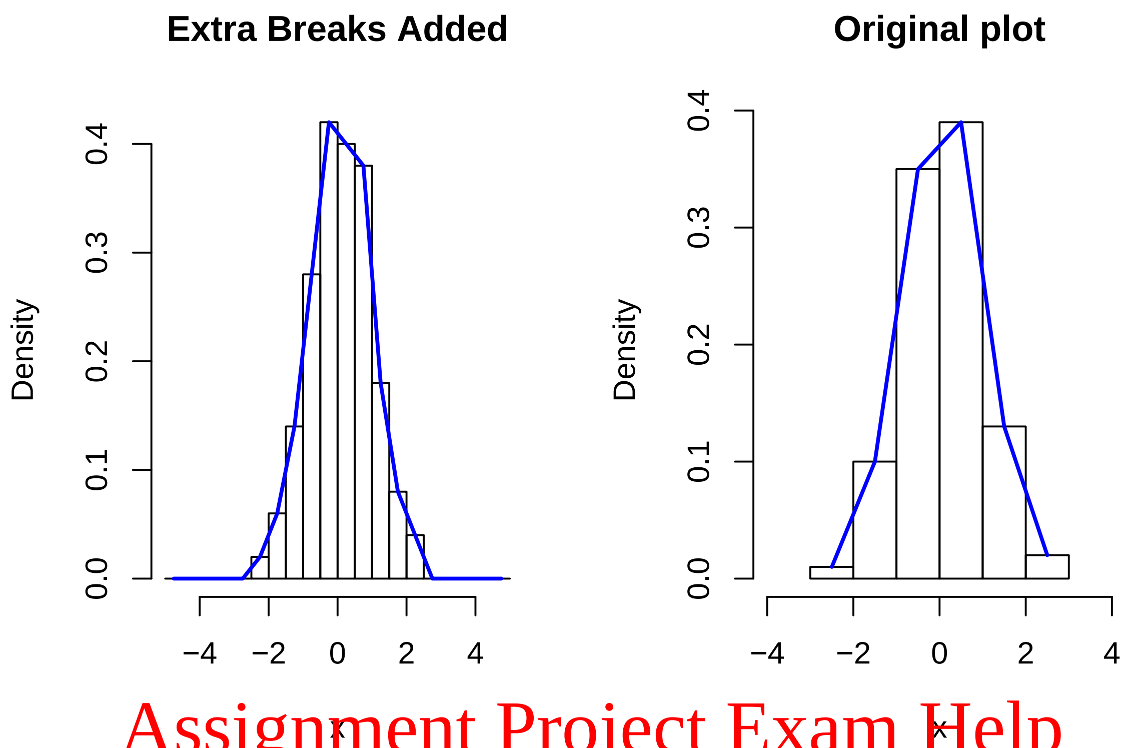
The result from the line plot at the ends is not quite right, however, as the frequency polygon should come down to zero. Probably the easiest way to resolve this problem is to add more bins, so that the first and last bins have a count of zero.

```r
set.seed(1)
x = rnorm(100)
H = hist(x, breaks=seq(-5,5), freq=F, main="Extra Breaks Added")

lines(H$mid, H$density, lwd=2, col="blue")
```



We could also include a greater number of bins . . .

```r
par(mfrow = c(1,2))

set.seed(1)
x = rnorm(100)

H = hist(x, breaks=seq(-5,5, 0.5), freq=F, main="Extra Breaks Added")
lines(H$mid, H$density, lwd=2, col="blue")

H = hist(x,breaks=5,freq=F,main="Original plot",xlim=c(-4,4))
lines(H$mid, H$density, lwd=2, col="blue")
```



```r
par(mfrow = c(1,1))
```

There are also methods to optimize bin widths for frequency polygons rather than histograms, which we don't cover but which you can investigate yourself.

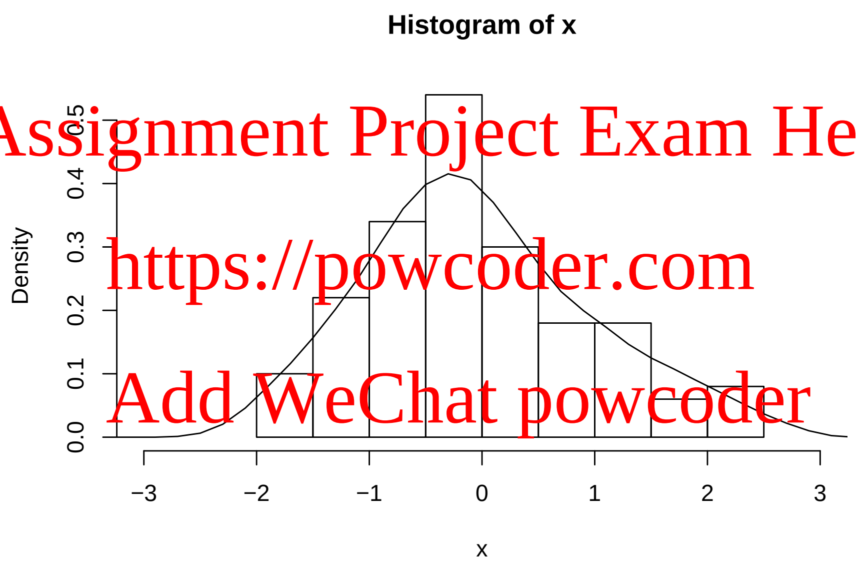
Average shifted histogram

This is implemented in the package `ash`, and is a method developed by Scott.

```r
library(ash)
x <- rnorm(100)          # data
ab <- c(-5,5)            # bin interval
bins <- bin1(x, ab, 50)  # bin x into 50 bins over ab
f <- ash1(bins, m=5)     # where m is a smoothing parameter
f
```

```
## $x
##  [1] -4.90000000 -4.70000000 -4.49999999 -4.29999999 -4.09999999 -3.89999998
##  [7] -3.69999998 -3.49999998 -3.29999997 -3.09999997 -2.89999997 -2.69999997
## [13] -2.49999996 -2.29999996 -2.09999996 -1.89999995 -1.69999995 -1.49999995
## [19] -1.29999994 -1.09999994 -0.89999994 -0.69999994 -0.49999993 -0.29999993
## [25] -0.09999993  0.10000008  0.30000008  0.50000008  0.70000008  0.90000009
## [31]  1.10000009  1.30000009  1.50000010  1.70000010  1.90000010  2.10000011
## [37]  2.30000011  2.50000011  2.70000011  2.90000012  3.10000012  3.30000012
## [43]  3.50000013  3.70000013  3.90000013  4.10000014  4.30000014  4.50000014
## [49]  4.70000014  4.90000015
##
## $y
##  [1] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##  [7] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.001215121
## [13] 0.006270626 0.020372035 0.045934591 0.079957992 0.116066602 0.156690662
## [19] 0.201725164 0.250360022 0.306195604 0.360351018 0.398514833 0.415361517
## [25] 0.405820566 0.370237003 0.322667250 0.273492335 0.229852972 0.199534942
## [31] 0.173522343 0.146444635 0.124887482 0.107680762 0.089558950 0.071857182
## [37] 0.053975395 0.036783676 0.022127211 0.010111010 0.002430242 0.000000000
## [43] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [49] 0.000000000 0.000000000
##
## $m
## [1] 5
##
## $ab
## [1] -5  5
##
## $kopt
## [1] 2 2
##
## $ier
## [1] 0
```

```r
hist(x, freq=F, xlim=c(-3,3))
lines(f$x, f$y)
```



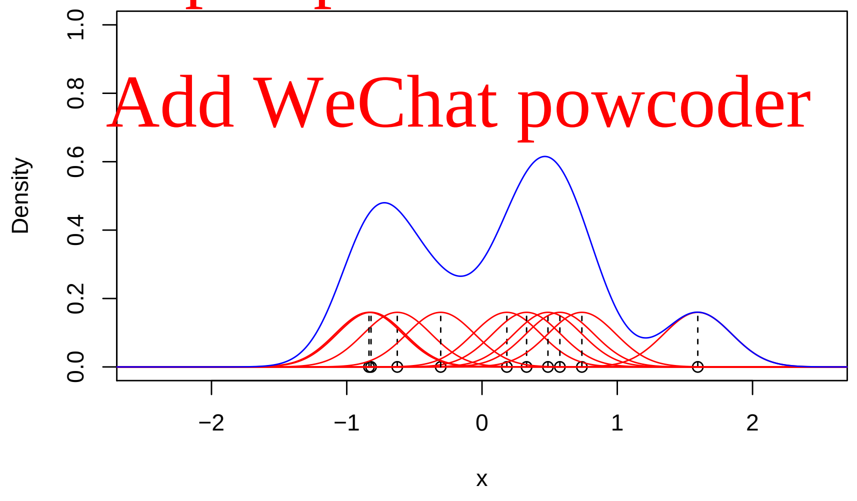
- This method aims to overcome the problem of the appearance of a histogram being sensitive to the location of the bins (rather than the width (or ‘shape’) of the bins))
- The estimate of the density at a given point is taken to be the average of several histograms with equal bin widths but different bin locations.
- This method has good statistical properties and is computationally efficient in the multivariate case.

Kernel density estimation

Key concepts

- The frequency polygons are not very smooth, whereas in real life applications we expect the underlying probability density function to be smoothed.
- The idea with a kernel density estimator is to spread the mass of each sample data point out smoothly using a small "kernel" centred on each datapoint
- The contribution from each small kernel is then summed up to give the density estimate

```
set.seed(1)

n=10
x=rnorm(n)
h=0.25

plot(x, rep(0,n), ylim=c(0,1), xlim=c(-2.5, 2.5),xlab="x", ylab="Density")

x.to.plot=seq(-5, 5, 0.01)
for (i in 1:n) {
  lines(x.to.plot, dnorm(x.to.plot, x[i],h)/n, col="red")
  lines(rep(x[i], 2), c(0, dnorm(x[i], x[i], h))/n, lty=2)
}

temp <- function(x.to.plot){mean(dnorm(x.to.plot, x, h))}
lines(x.to.plot, sapply(x.to.plot, temp), col="blue")
```



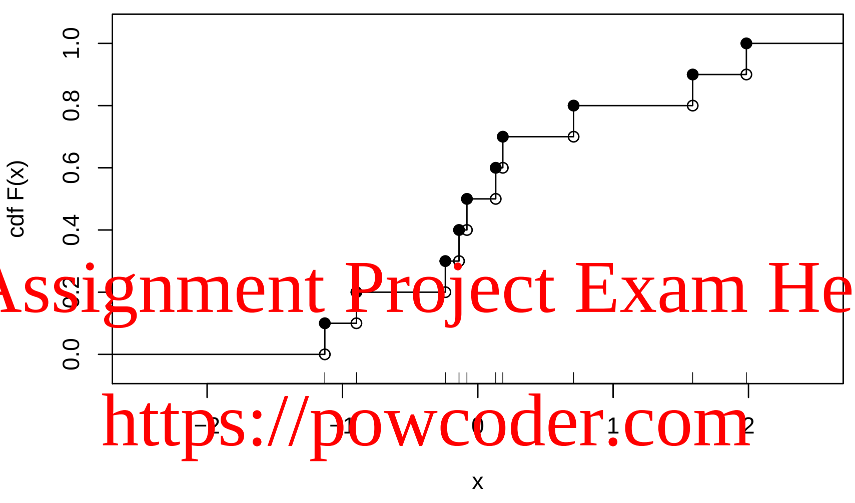
KDE Derivation (NOT EXAMINABLE)

- To derive the theory for the kernel density estimator, we have to go back some first year definitions
- Let X be a continuous random variable with cdf given by $F(x)$ then the density is defined as $f(x)$
- Given a sample of data $X_1, X_2, \ldots, X_n$ then the cdf is naturally estimated using

$$\widehat{F}(x) = \frac{1}{n} \sum I(X_i \leq x)$$

We start by thinking about the ECDF:

```
set.seed(2)

n=10
x=sort(rnorm(n))
allx=c(-100,x,100)
y=(1:n)/n
ally=c(0,y,1)
plot(allx,ally,xlim=c(-2.5,2.5),ylim=c(-0.05,1.05),xlab="x",ylab="cdf F(x)",type="s")
points(x,y,pch=19)
points(x,c(0,y[-n]))
rug(x)
```



The derivative of this estimator is useless since it is just mass points at each datapoint, and it is not a continuous density function as needed.

Instead, let's consider a local linear approximation to the derivative of $\widehat{F}(x)$, for some small $h > 0$:

$$\widehat{f}(x) = \frac{\widehat{F}(x+h) - \widehat{F}(x-h)}{2h}$$

The numerator is a proportion of the datapoints in the interval $(x-h, x+h]$:

$$\begin{aligned}
\widehat{F}(x+h) - \widehat{F}(x-h) &= \frac{\text{number of points in } (x-h, x+h]}{n} \\
&= \frac{\sum I(x-h < X_i \le x+h)}{n}
\end{aligned}$$

and so the density derivative becomes

$$\begin{aligned}
\widehat{f}(x) &= \frac{\widehat{F}(x+h) - \widehat{F}(x-h)}{2h} \\
&= \frac{\sum I(x-h < X_i \le x+h)}{2nh} \\
&= \frac{1}{nh}\sum \frac{1}{2} I\left(\frac{|X_i - x|}{h} \le 1\right) \\
&= \frac{1}{nh}\sum k\left(\frac{X_i - x}{h}\right) \qquad (2)
\end{aligned}$$

where the kernel function $k(\cdot)$ is given by:

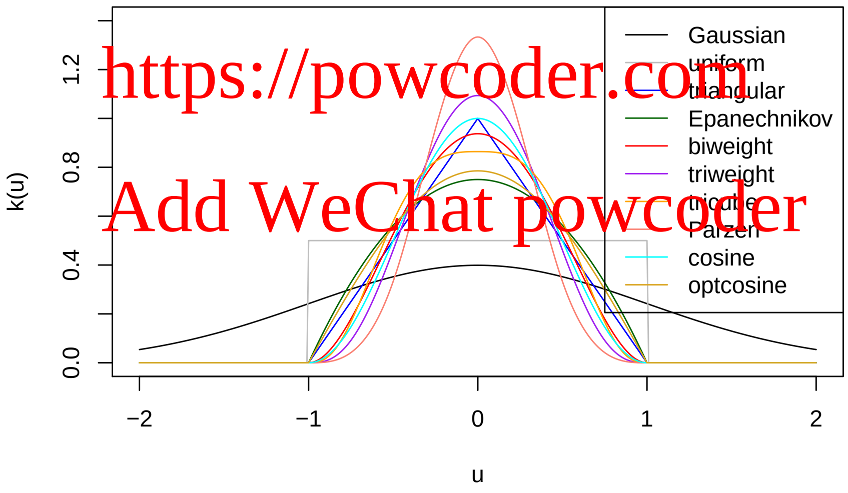
$$k(u) = \begin{cases} 1/2 & \text{for } |u| \le 1 \\ 0 & \text{for } |u| > 1. \end{cases}$$

Equation (2) is called a kernel density estimator.

Examples of kernels

Commonly the Gaussian density function is used as the kernel, but there are a huge number of kernels people use. A selection of the most commonly used kernels are plotted below.

```r
library(evmix)
xx = seq(-2, 2, 0.01)
plot(xx, kdgaussian(xx), type = "l", col = "black",ylim = c(0, 1.4),xlab="u",ylab="k(u)")
lines(xx, kduniform(xx), col = "grey")
lines(xx, kdtriangular(xx), col = "blue")
lines(xx, kdepanechnikov(xx), col = "darkgreen")
lines(xx, kdbiweight(xx), col = "red")
lines(xx, kdtriweight(xx), col = "purple")
lines(xx, kdtricube(xx), col = "orange")
lines(xx, kdparzen(xx), col = "salmon")
lines(xx, kdcosine(xx), col = "cyan")
lines(xx, kdoptcosine(xx), col = "goldenrod")
legend("topright", c("Gaussian", "uniform", "triangular", "Epanechnikov",
                     "biweight", "triweight", "tricube", "Parzen", "cosine", "optcosine"), lty = 1,
       col = c("black", "grey", "blue", "darkgreen", "red", "purple", "orange",
               "salmon", "cyan", "goldenrod"))
```



We will use a package called `evmix` in R to implement kernel density estimation, since this was written by a staff member at UC who has very recently left, Carl Scarrott.

Kernel Density Estimator

The kernel density estimator (given by equation (2) in the derivation earlier) is

$$\widehat{f}(x) \quad = \quad \frac{1}{nh}\sum k\left(\frac{X_i - x}{h}\right)$$

where the kernel function $k(\cdot)$ usually has the properties:

1. non-negative $k(u) \geq 0$
2. integrates to one $\displaystyle\int_{-\infty}^{\infty} k(u)du = 1$
3. symmetry $k(-u) = k(u)$

Usually the kernel function is a valid probability density function.

h is the 'bandwidth', and acts as a smoothing parameter.

Some definitions of kernels

All the kernels shown in the plot are defined on u where u is $[-1, 1]$, and are zero outside of this range. The only exception is the Gaussian kernel which is defined on the whole real line.

Common functions for $k(u)$ are

Gaussian	$\frac{1}{\sqrt{2\pi}}\exp(-u^2/2)$
rectangular	$\frac{1}{2}$
triangular	$1-\lvert u\rvert$
biweight	$\frac{15}{16}(1-u^2)^2$
Epanechnikov	$\frac{3}{4}(1-u^2)$
cosine	$\frac{\pi}{4}\cos\left(\frac{\pi}{2}u\right)$

An example using the rectangular kernel

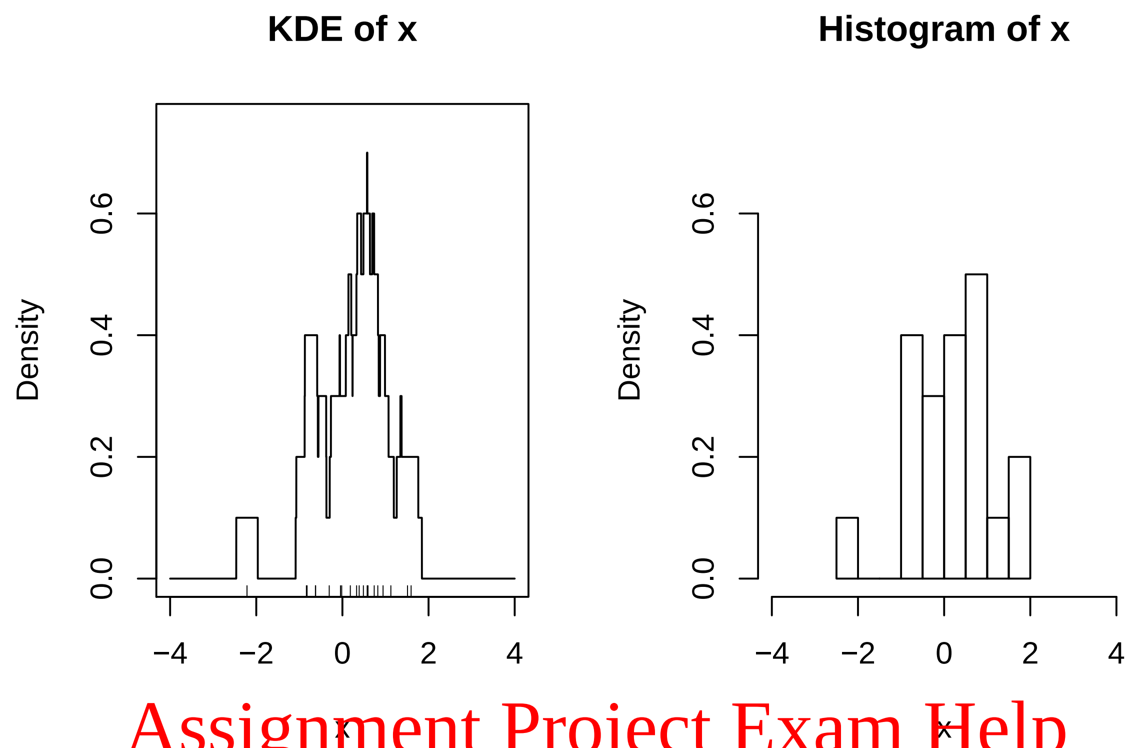
- The rectangular kernel function $k(\cdot)$ is the density function for the Uniform distribution on $[-1, 1]$, and is given by:

$$k(u) = \begin{cases} 1/2 & \text{for } |u| \le 1 \\ 0 & \text{for } |u| > 1 \end{cases}$$

- The area under each kernel between $(x-h, x+h]$ is given by h, so the sum in Equation (2) is 1, so it is a density.

The following example simulates 20 datapoints from a standard Normal and calculates the kernel density estimate. A rectangular kernel is used with the bandwidth h set to $h = 0.25$. Also plotted is the corresponding density histogram with default settings.

```r
library(evmix)

set.seed(1)
n = 20
x = rnorm(n)

h = 0.25

x.to.plot = seq(-4,4,0.001)
y.to.plot = dkden(x.to.plot, x, h, kernel="rectangular")

par(mfrow=c(1,2))
# kernel density estimator
plot(x.to.plot, y.to.plot, type="l",xlab="x",ylab="Density", ylim=c(0,0.75),
     main = "KDE of x")
rug(x) # adds a tickmark on x axis where each datapoint is located

# density histogram
hist(x, freq=F, xlim=c(-4,4), ylim=c(0,0.75))
```



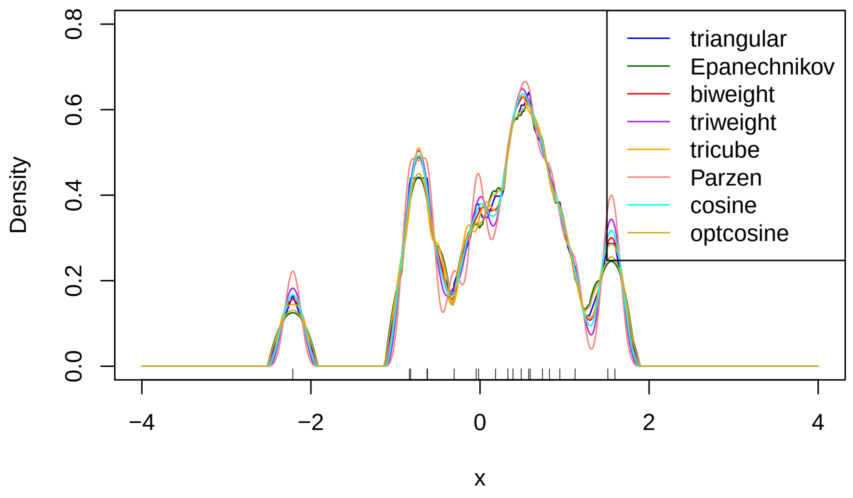
```
par(mfrow=c(1,1))
```

The rectangular kernel gives quite a rough appearance to the kernel density estimator.

An example comparing different kernels

Here we use a range of different kernels with a small simulated dataset. The bandwidth $h = 0.3$ is the same in all cases.

```
library(evmix)

set.seed(1)
n = 20
x = rnorm(n)

h = 0.3

x.to.plot = seq(-4, 4, 0.001)

plot(x.to.plot, dkden(x.to.plot,x,h, kernel="triangular"), col="blue",type="l",
     ylim=c(0,0.8),xlab="x",ylab="Density")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="epanechnikov"), col="darkgreen")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="biweight"), col="red")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="triweight"), col="purple")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="tricube"), col="orange")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="parzen"), col="salmon")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="cosine"), col="cyan")
lines(x.to.plot, dkden(x.to.plot,x,h, kernel="optcosine"), col="goldenrod")
rug(x)
legend("topright", c("triangular", "Epanechnikov",
                     "biweight", "triweight", "tricube", "Parzen", "cosine", "optcosine"),
       lty = 1, col = c("blue", "darkgreen", "red", "purple", "orange",
                        "salmon", "cyan", "goldenrod"))
```



We can see that the KDE is essentially the same for all the kernels.

Start of lecture 4 material

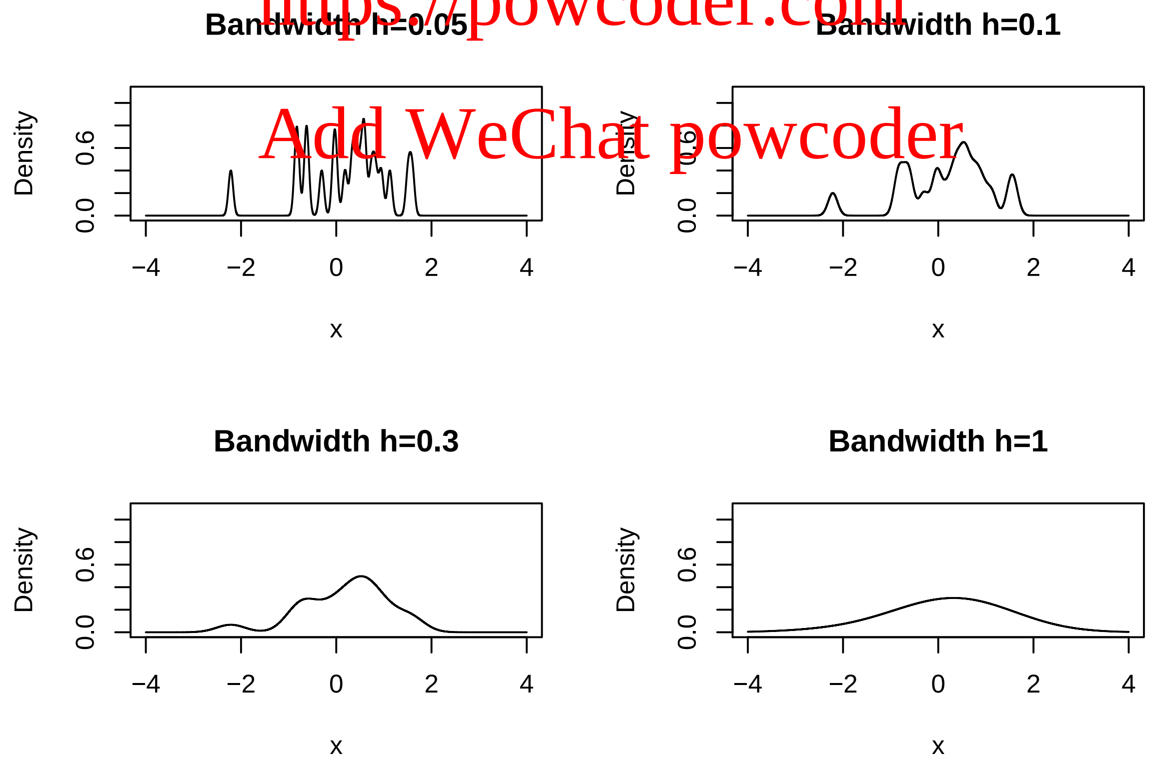
Choice of bandwidth h is critical

As in histogram estimation, it turns out that the bandwidth (like the bin width) is the parameter that has the most effect of the appearance of the kernel density estimator.

Optimal bandwidths have been derived for different situations using the MISE as a criterion.

We can see the effect of changing the bandwidth in the following plot, each using the same default kernel and the same dataset.

```r
library(evmix)

set.seed(1)
n=20
x=rnorm(n)

x.to.plot=seq(-4, 4, 0.0001)

par(mfrow=c(2, 2))
plot(x.to.plot, dkden(x.to.plot,x,0.05), main="Bandwidth h=0.05",
     type="l", ylim=c(0,1.1),xlab="x",ylab="Density")
plot(x.to.plot, dkden(x.to.plot,x,0.1), main="Bandwidth h=0.1",
     type="l", ylim=c(0,1.1),xlab="x",ylab="Density")
plot(x.to.plot, dkden(x.to.plot,x,0.3), main="Bandwidth h=0.3",
     type="l", ylim=c(0,1.1),xlab="x",ylab="Density")
plot(x.to.plot, dkden(x.to.plot,x,1), main="Bandwidth h=1",
     type="l", ylim=c(0,1.1),xlab="x",ylab="Density")
```



```r
par(mfrow=c(1, 1))
```

We can see that:

- The bandwidth of $h = 0.05$ is undersmoothing as there is too much variation

- The bandwidth of $h = 1$ is oversmoothing as there is too little variation
- The bandwidth of $h = 0.3$ seems about right.

Bandwidths in different R packages

There are lots and lots of packages in R that do kernel density estimation.

Minimising the MISE is difficult to do, so that many methods have been developed to this task, leading to these packages.

The different packages are optimised for different tasks. Some are very computationally efficient - they may be very fast for low dimensional problems, or capable of handing high dimensions. Some may be slower but more accurate.

Popular packages/functions include:

- `density`, which is part of the base installation of R, and does univariate density estimation
- `ks`
- `KernSmooth`
- `np`
- `sm`
- `GenKern`
- `kerdiest`

plus many more.

However these packages differ primarily in terms of how they specify and select the bandwidth, so care needs to be taken when specifying the bandwidth, because of these different definitions.

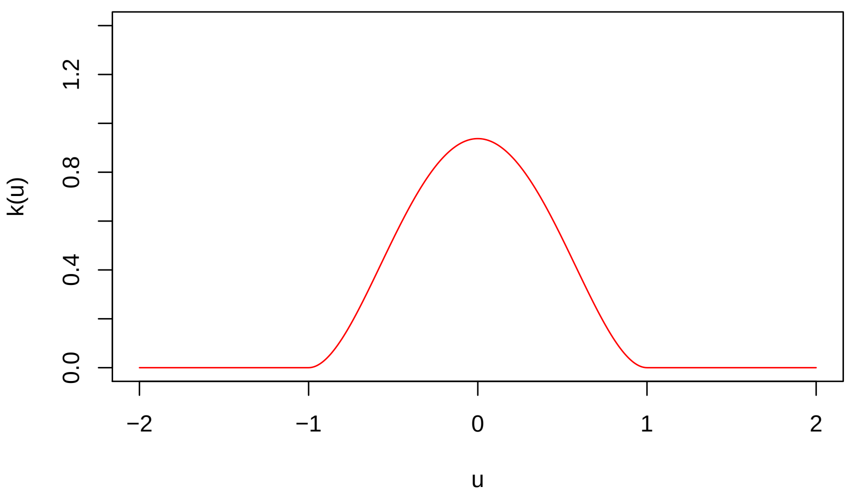
For example, if We look at the differences between the definitions used in the kernel density estimates from the:

- inbuilt `density` function and
- `dkden` function from the `evmix` library

then they are different (for all kernels except the Gaussian kernel), despite the user specifying the same bandwidth value.

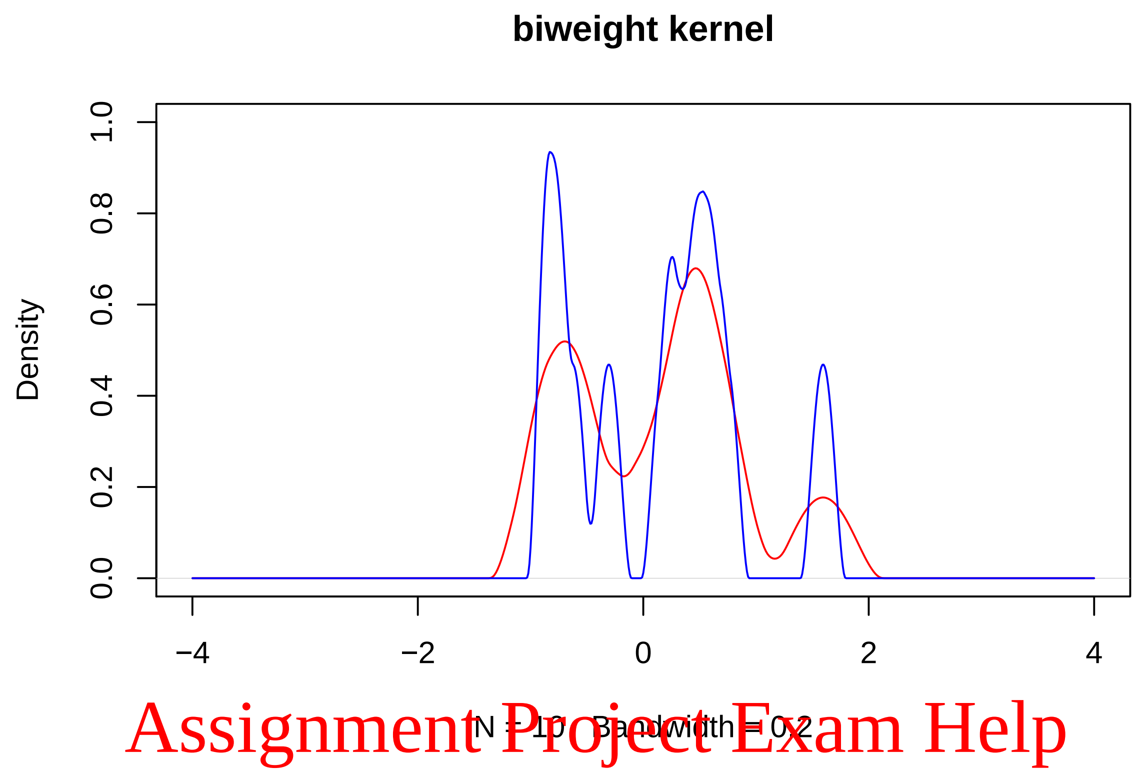
For example, we can compare the bi-weight kernel, which is

```
library(evmix)
xx = seq(-2, 2, 0.01)
plot(xx, kdbiweight(xx), type = "l", col = "red", ylim = c(0, 1.4),xlab="u",ylab="k(u)")
```



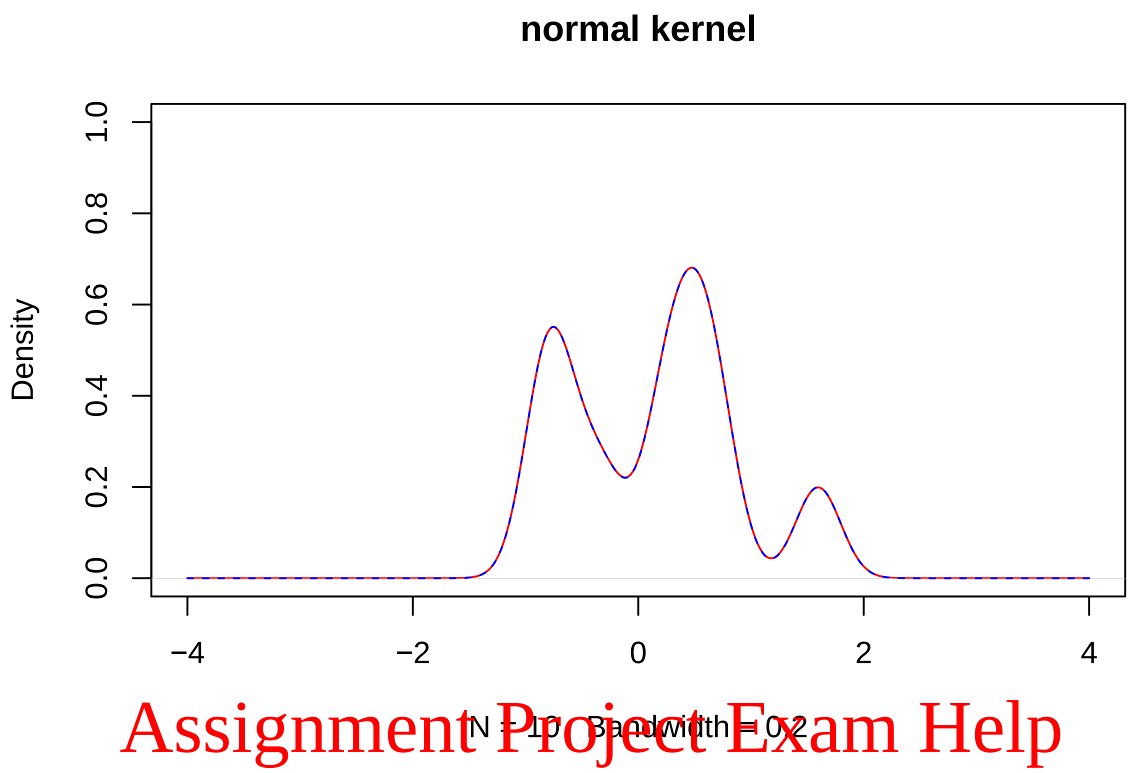
If we specify the bandwidth as $h = 0.2$ then we get:

```r
library(evmix)
set.seed(1)
x=rnorm(10)
h=0.2

# Use inbuilt density function
y = density(x, bw = h, kernel="biweight", from=-4, to=4)
plot(y, col="red", main="biweight kernel", ylim=c(0,1))
# use evmix
x.to.plot = seq(-4, 4, 0.01)
y.to.plot = dkden(x.to.plot, x, h, kernel="biweight")
lines(x.to.plot, y.to.plot, col="blue")
```



However, comparing the Gaussian kernel density plots, we can see that they are the same:

```
library(evmix)
set.seed(1)
x=rnorm(10)
h=0.2

# Use inbuilt density function
y = density(x, h, kernel="gaussian", from=-4, to=4)
plot(y, col="red", main="normal kernel", ylim=c(0,1))
# use evmix
y.to.plot = dkden(x.to.plot, x, h, kernel="gaussian")
lines(x.to.plot, y.to.plot, col="blue", lty=2)
```



We have just compared two packages/functions here, but these differences are true in general.

The most common bBandwidth specification

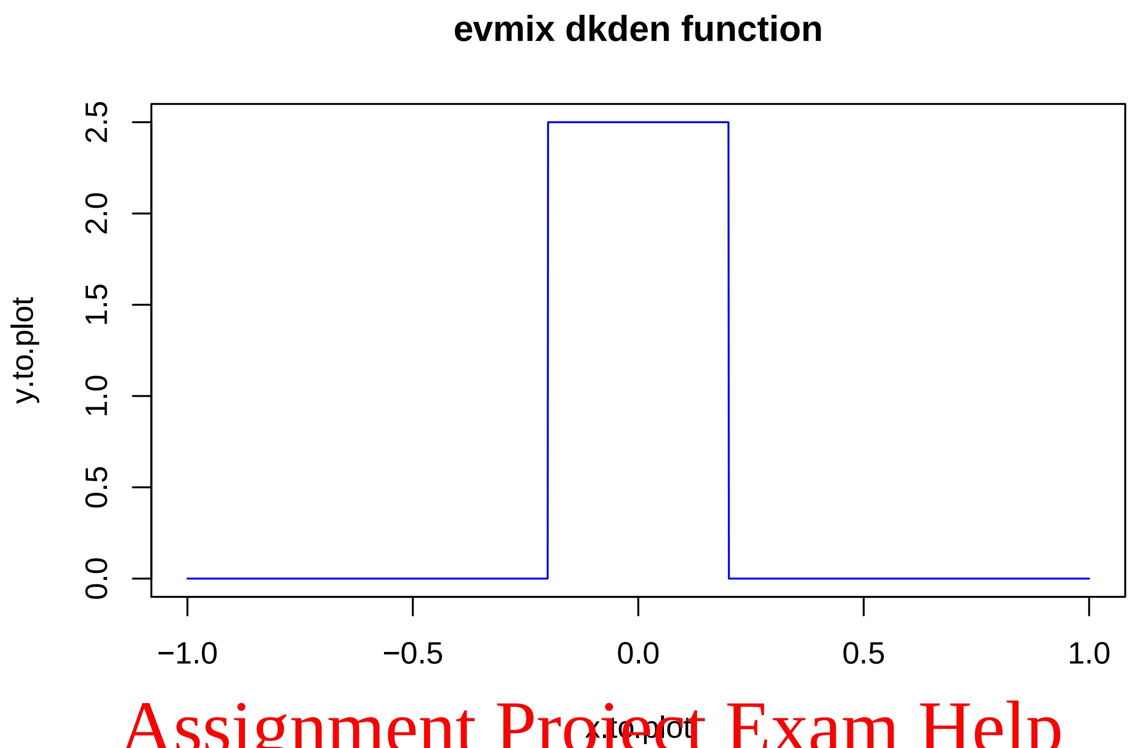
The most common definition of the bandwidth is the value h in equation (2) above:

$$\hat{f}(x) = \frac{1}{nh}\sum k\left(\frac{x_i - x}{h}\right)$$

Given all the kernels (except the Gaussian) listed on page 25 are only defined on $|u| \leq 1$ then on the original x-scale they are bounded between $[x_i - h, x_i + h]$ around each datapoint x_i

The following graph is a KDE for a single datapoint at $x = 0$ using a rectangular kernel and bandwidth 0.2, using the package `evmix`:

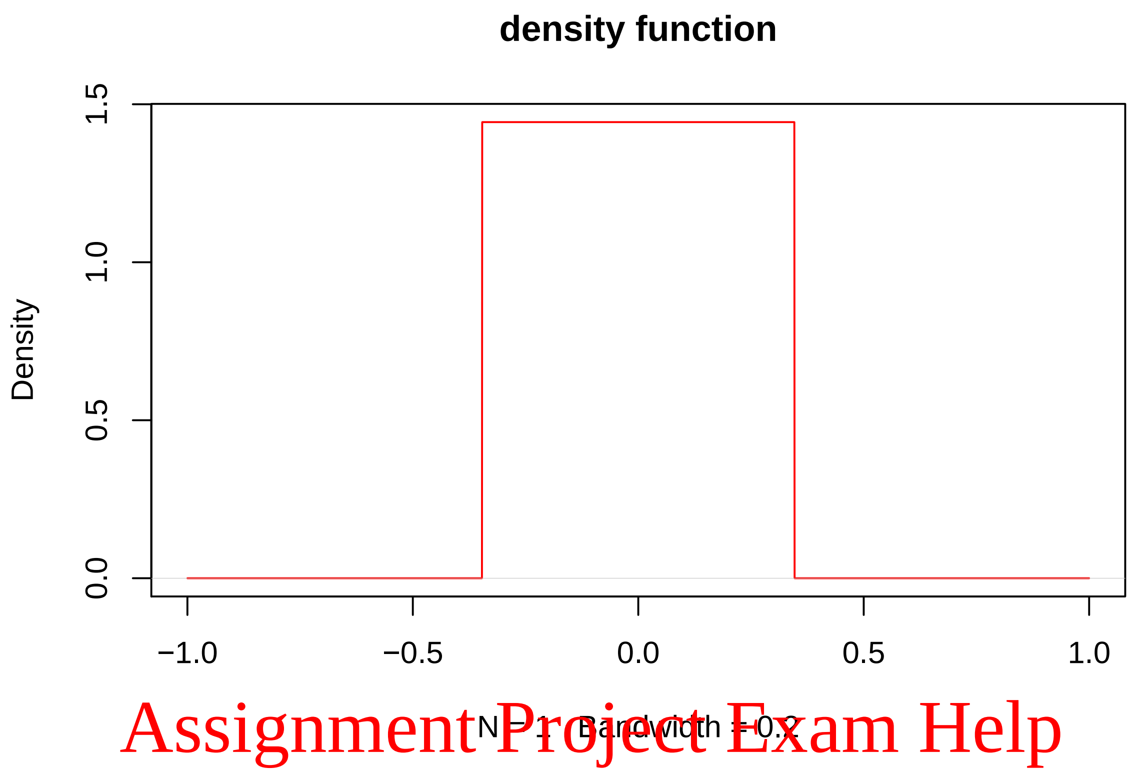
```r
library(evmix)
x = 0
h = 0.2

x.to.plot = seq(-1, 1, 0.001)
y.to.plot = dkden(x.to.plot, x, h, kernel="rectangular")
plot(x.to.plot, y.to.plot, type="l", col="blue", main="evmix dkden function")
```



The `evmix` package by default uses the definition of the bandwidth given above (with the bandwidth argument called `lambda`), so that the density has a height of $2.5 = \frac{1}{2h}$ which is the reciprocal of twice the bandwidth (i.e. the range over which the kernel mass is spread).

We can compare this to the plot when we use the `density` function, but with the bandwidth also set to be $0.2):

```
y = density(x, h, kernel="rectangular", from=-1, to=1, n=10000)
plot(y, col="red", main="density function")
```



An alternative bandwidth specification

The problem with the usual definition is that each kernel has different “spread”.

A common alternative is to specify bandwidth as number of “standard deviations” of the kernel function itself.

Table of kernel variances below given on Wikipedia: http://en.wikipedia.org/wiki/Kernel_(statistics)

Kernel	Variance
Gaussian	1
rectangular	1/3
triangular	1/6
biweight	1/7
Epanechnikov	1/5
cosine	$1 - 8/\pi^2$

This is the bandwidth definition in the `density` function.

Consequences of this alternative bandwidth

The Gaussian kernel has standard deviation 1, and the two bandwidth definitions are the same in this case.

This is why the lower graph comparing the Gaussian kernels is same for `evmix` library and `density` function.

However, the rectangular kernel has standard deviation $1/\sqrt{3}$ (sd $= \sqrt{\text{var}}$). As a consequence, a bandwidth of 0.2 given to the `density` function equates to the usual bandwidth being:

$$h = \frac{0.2}{1/\sqrt{3}} = 0.346$$

This is why the KDE is spread over $[-0.346, 0, 346]$ with a height of $1/2h = 1.443$.

Bandwidth Specification in R Functions

The bandwidth argument for the `density` function is `bw`.

The functions in the `evmix` library can take both definitions:

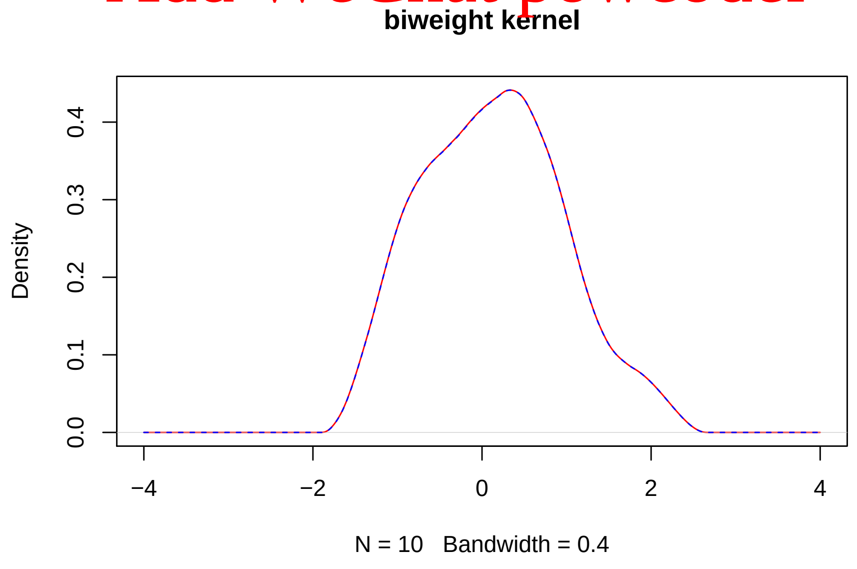
- `lambda` - usual bandwidth h from eq. (2)
- `bw` - kernel standard deviations

The `evmix` library has functions to convert between them:

- `klambda` - convert `bw` to `lambda`
- `kbw` - convert `lambda` to `bw`

Fow example, the following demonstrates how the bandwidth value can be converted between these different definitions, using `evmix`:

```
library(evmix)
set.seed(1)
n = 10
x = rnorm(n)
bw = 0.4

# Use inbuilt density function
y = density(x, bw, kernel="biweight", from=-4, to=4)
plot(y, col="red", main="biweight kernel")

# Convert bw to lambda equivalent
h = klambda(bw, kernel="biweight")

# Use evmix function with usual bandwidth
x.to.plot = seq(-4, 4, 0.01)
y.to.plot = dkden(x.to.plot, x, lambda = h, kernel="biweight")
lines(x.to.plot, y.to.plot, col="blue", lty=2)
```



The plots now overlay each other because the bandwidths are equivalent.