Hexagon Binning: an Overview

Nicholas Lewin-Koh*

October 22, 2008

1 Overview

Hexagon binning is a form of bivariate histogram useful for visualizing the structure in datasets with large n. The underlying concept of hexagon binning is extremely simple;

1. the xy plane over the set (range(x), range(y)) is tessellated by a regular grid of hexagons.

2. the number of points falling in each hexagon are counted and stored in a data structure

3. the hexagons with count > 0 are plotted using a color ramp or varying the radius of the hexagon in proportion to the counts.

The underlying algorithm is extremely fast and effective for displaying the structure of datasets with $n \geq 10^6$. If the size of the grid and the cuts in the color ramp are chosen in a clever fashion than the structure inherent in the data should emerge in the binned plots. The same caveats apply to hexagon binning as apply to histograms and care should be exercised in choosing the binning parameters.

The hexbin package is a set of function for creating, manipulating and plotting hexagon bins. The package extends the basic hexagon binning ideas with several functions for doing bivariate smoothing, finding an approximate bivariate median, and looking at the difference between two sets of bins on the same scale. The basic functions can be incorporated into many types of plots. This package is based on the original package for splus by Dan Carr at George Mason University and is mostly the fruit of his graphical genius and intuition.

2 Theory and Algorithm

Why hexagons? There are many reasons for using hexagons, at least over squares. Hexagons have symmetry of nearest neighbors which is lacking in

*with minor assistance by Martin Mächler

square bins. Hexagons are the maximum number of sides a polygon can have for a regular tesselation of the plane, so in terms of packing a hexagon is 13% more efficient for covering the plane than squares. This property translates into better sampling efficiency at least for elliptical shapes. Lastly hexagons are visually less biased for displaying densities than other regular tesselations. For instance with squares our eyes are drawn to the horizontal and vertical lines of the grid. The following figure adapted from (**?**)shows this effectively.

We can see in Figure 1 that when the data are plotted as squares centered on a regular lattice our eye is drawn to the regular lines which are parrallel to the underlying grid. Hexagons tend to break up the lines.

How does does the hexagon binning algorithm work?

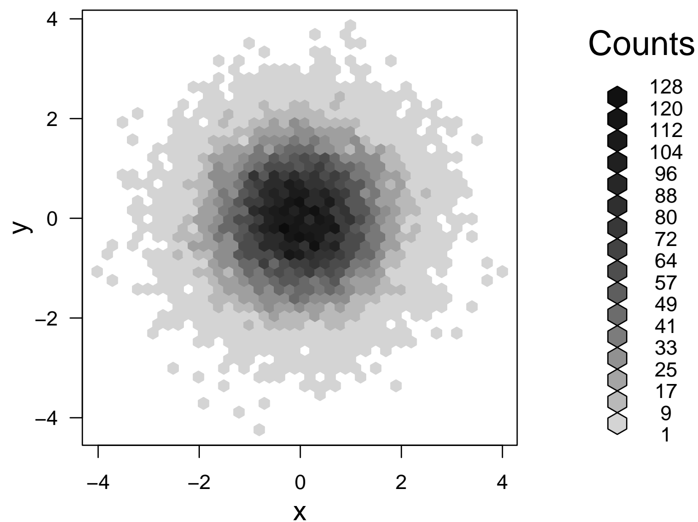
1. Squash Y by $\sqrt{3}$
2. Create a dual lattice
3. Bin each point into pair of near neighbor rectangles
4. Pick closest of the rectangle centers (adjusting for $\sqrt{3}$)

Figure 2 shows graphically how the algorithm works. In the first panel we see the the dual lattice laid out in black and blue points. The red point is an arbitrary point to be binned. The second panel shows the near neigbor rectangles for each lattcie around the point to be binned, the intersection of the rectangles contains the point. The last panel shows the simple test for locating the point in the hexagon, the closest of the two corners which are not intersections is the center of the hexagon to which the point should be allocated. The binning can be calculated in one pass through the data, and is clearly $O(n)$ with a small constant. Storage is vastly reduced compared to the original data.

3 Basic Hexagon Binning Functions

Using the basic hexagon binning functions are not much more involved than using the basic plotting functions. The following little example shows the basic features of the basic plot and binning functions. We start by loading the package and generating a toy example data set.

```
> x <- rnorm(20000)
> y <- rnorm(20000)
> hbin <- hexbin(x, y, xbins = 40)
> plot(hbin)
```



There are two things to note here. The first is that the function `gplot.hexbin` is defined as a `plot` method for the S4 class `hexbin`. The second is that the default color scheme for the hexplot is a gray scale. However, there is an argument to plot, `colramp`, that allows the use of any function that excepts an argument `n` and returns n colors. Several functions are supplied that provide alternative color-ramps to R's built in color ramp functions, see `help(ColorRamps)`.

The figure shows three examples of using hexagons in a plot for large n with different color schemes. Upper left: the default gray scale, upper right: the R base `terrain.colors()`, and lower middle: `BTY()`, a blue to yellow color ramp

supplied with hexbin on a perceptually linear scale.

The hexbin package supplies a plotting method for the hexbin data structure. The plotting method `gplot.hexbin` accepts all the parameters for the hexagon function and supplies a legend as well, for easy interpretation of the plot. Figure 2 shows a hex binned plot with a legend. A function `grid.hexlegend` is supplied for creating user specified hexagon legends.

4 Extended Hexagon Functions

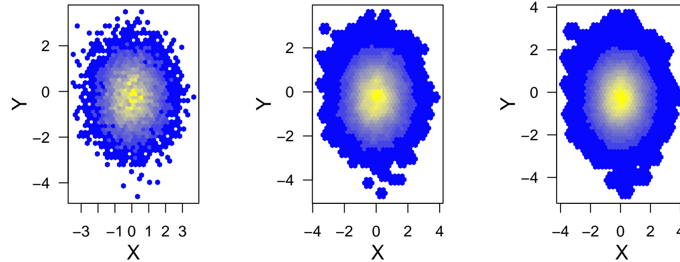
So far we have looked at the basic hexagon plot. The hexbin package supplies several extensions to the basic hexbin, and the associated hexplot. The extensions discussed in this section will be smoothing hexbin objects using the hsmooth function, apporximating a bivariate median with hexagons and a version of a bivariate boxplot, and using eroded hexbin objects to look at the overlap of two bivariate populations.

4.1 Smoothing with `hsmooth`

At this point the hexbin package only provides a single option for smoothig using a discrete kernel. Several improvements are in development including an apply function over neighborhoods and spline functions using a hexagonal basis or tensor products. The apply function should facilitate constructing more sophisticated kernel smoothers. The hexagon splines will provide an alternative to smoothing on a square grid and allow interpolation of hexagons to finer grids.

The current implementation uses the center cell, immediate neighbors and second neighbors to smooth the counts. The counts for each resulting cell is a linear combination of the counts in the defined neighborhood, including the center cell and weights. The counts are blurred over the the domain, and the domain increases because of shifting. Generally the dimension of the occupied cells of the lattice increases by one, sometimes two.

Some examples of using the hsmooth function are given below. Notice in the plots that the first plot is with no smoothing, weights are `c(1,0,0)` meaning that only the center cell is used with identity weights. The second plot shows a first order kernel using weights `c(24,12,0)`, while the third plot uses weights for first and second order neighbors specified as `c(48,24,12)`. The code segment generating these plots rescales the smoothed counts so that they are on the original scale.



4.2 Bin Erosion and the `hboxplot`

The next tool to introduce, gray level erosion, extends the idea of the boxplot. The idea is to extract cells in a way that the most exposed cells are removed first, ie cells with fewer neighbors, but cells with lower counts are removed preferentially to cells with higher counts. The algorithm works as follows: Mark the high count cells containing a given fraction, cdfcut, of the total counts. Mark all the cells if cdfcut is zero. The algorithm then performs gray-level erosion on the marked cells. Each erosion cycle removes counts from cells. The counts removed from each cell are a multiple of the cell's exposed-face count. The algorithm choses the multiple so at least one cell will be empty or have a count deficit on each erosion cycle. The erode vector contains an erosion number for each cell. The value of erode is

$$6\times(\text{The erosion cycle at cell removal}) - (\text{The cell deficit at removal})$$

The cell with the highest erosion number is a candidate bivariate median. A few ties in the erosion order are common.

The notion of an ordering to the median is nice because it allows us to create a version of a bivariate box plot built on hexagons. The following example comes from a portion of the "National Health and Nutrition Examination Survey" included in **`hexbin`** as the sample data set NHANES. The data consist of 9575 persons and mesures various clincal factors. Here in Figure 3 we show the levels of transferin, a measure of iron binding against hemoglobin for all

Note that we have added "hexagon graph paper" to the plot. This can be done for any hexbin plot, using the command `hexGraphPaper()` where the main argument is the hexbin object.

4.3 Comparing Distributions and the `hdiffplot`

With univariate data, if there are multiple groups, one often uses a density estimate to overlay densities, and compare two or more distirbutions. The hdiffplot is the bivariate analog. The idea behind the hdiff plot is to plot one or more bin objects representing multiple groups to compare the distributions.

The following example uses the National Health data supplied in the hexbin package, (`NHANES`). Below we show a comparison of males and females, the bivariate relationship is transferin, which is a derived measure of the ability of blood to bind oxygen, vs the level of hemoglobin. Note that in the call to `hdiffplot` we erode the bins to calculate the bivariate medians, and only display the upper 75% of the data.

4.4 Plotting a Third Concomitant Variable

In many cases, such as with spatial data, one may want to plot the levels of a third variable in each hexagon. The grid.hexagons function has a pair of arguments, `use.count` and `cell.at`. If `use.count = FALSE` and `cell.at` is a numeric vector of the same length as `hexbin@count` then the attribute vector will be used instead of the counts. `hexTapply` will sumarrerize values for each hexagon according to the supplied function and return the table in the right order to use as an attribute vector. Another alternative is to set the `cAtt` slot of the hexbin obeject and grid.hexagons will automatically plot the attribute if `use.count = FALSE` and `cell.at = NULL`.

Here is an example using spatial data. Often in cartographers use graduated symbols to display varying numerical quantities across a region.

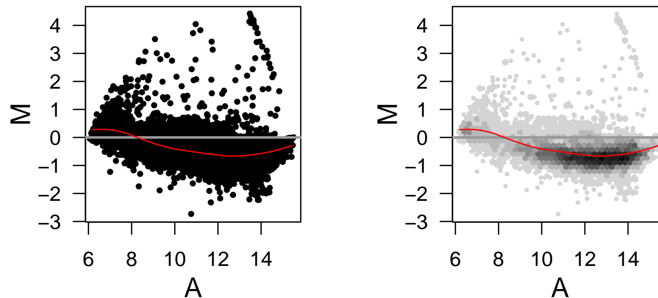
5 Example: cDNA Chip Normalization

This example is taken from the marray package, which supplies methods and classes for the normalization and diagnostic plots of cDNA microarrays. In this example the goal is not to make any comments about the normalization methodology, but rather to show how the diagnostic plots can be enhanced using hexagon binning due to the large number of points ($n = 8,448$ cDNA probes per chip).

We look at the diagnostic plot M vs A, where M is the log–ratio, $M = \log < -2\frac{R}{G}$ and A is the overall intensity, $A = \log < -2\sqrt{RG}$. Figure 3 shows the plot using points and on the right hexagons. The hexagon binned plot shows that most of the pairs are well below zero, and that the overall shape is more like a comet with most of the mass at the bottom of the curve, rather than a thick bar of points curving below the line.

```
> library("marray")
> data(swirl, package = "marray")
> hb1 <- hexbin(maA(swirl[, 1]), maM(swirl[, 1]), xbins = 40)
> grid.newpage()
> pushViewport(viewport(layout = grid.layout(1, 2)))
> pushViewport(viewport(layout.pos.col = 1, layout.pos.row = 1))
> nb <- plot(hb1, type = "n", xlab = "A", ylab = "M", main = "M vs A plot with points",
+     legend = 0, newpage = FALSE)
> pushHexport(nb$plot.vp)
```

```
> grid.points(maA(swirl[, 1]), maM(swirl[, 1]), pch = 16, gp = gpar(cex = 0.4))
> popViewport()
> nb$hbin <- hb1
> hexVP.abline(nb$plot.vp, h = 0, col = gray(0.6))
> hexMA.loess(nb)
> popViewport()
> pushViewport(viewport(layout.pos.col = 2, layout.pos.row = 1))
> hb <- plotMAhex(swirl[, 1], newpage = FALSE, main = "M vs A plot with hexagons",
+     legend = 0)
> hexVP.abline(hb$plot.vp, h = 0, col = gray(0.6))
> hexMA.loess(hb)
> popViewport()
```



6 Manipulating Hexbins

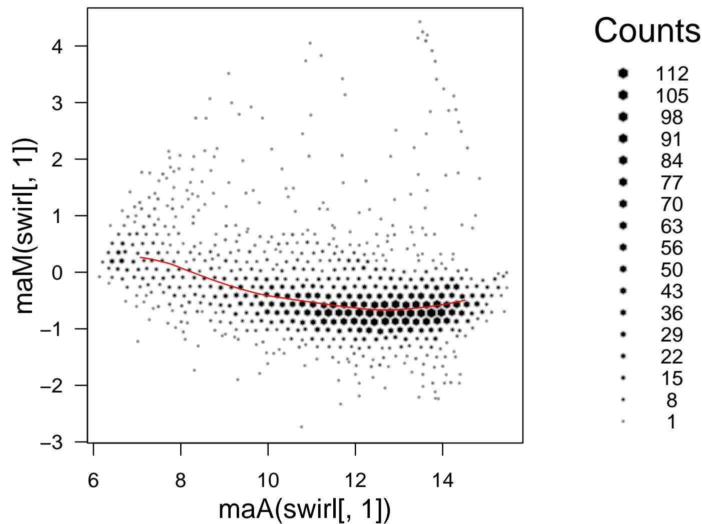
The underlying functions for hexbin have been rewritten and now depend on the grid graphics system. The support unit for all hexagon plots is the hexViewport. The function `hexViewport()` takes a hexbin object as input and creates a viewport scaled to the current device or viewport so that the aspect ratio is scaled appropriately for the hexagons. Unlike in the base graphic functions where the aspect ratio is maintained by shifting the range of the axes, here the extra space

is shifted into the margins. Currently hexViewport returns a hexViewport object that has information on the margins and its own pushViewport method. In the next example we will 1st show how to manipulate an existing plot using grid commands and second show how to create a custom plotting function using `hexViewport` and grid.

6.1 Adding to an existing plot

Adding to an existing plot requires the use of grid functions. For instance, in the following code,

```
> hplt <- plot(hb1, style = "centroid", border = gray(0.65))
> pushHexport(hplt$plot.vp)
> ll.fit <- loess(hb1@ycm ~ hb1@xcm, weights = hb1@count, span = 0.4)
> pseq <- seq(hb1@xbnds[1] + 1, hb1@xbnds[2] - 1, length = 100)
> grid.lines(pseq, predict(ll.fit, pseq), gp = gpar(col = 2), default.units = "native")
```



we have to use `grid.lines()`, as opposed to `lines()`.

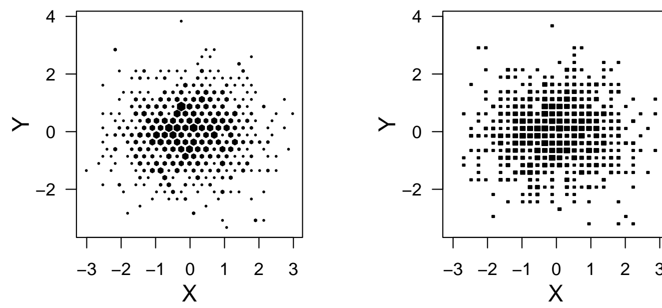


Figure 1: A bivariate point set binned into squares and hexagons. Bins are scaled approximately equal, and the size of the glyph is proportional to the count in that bin.

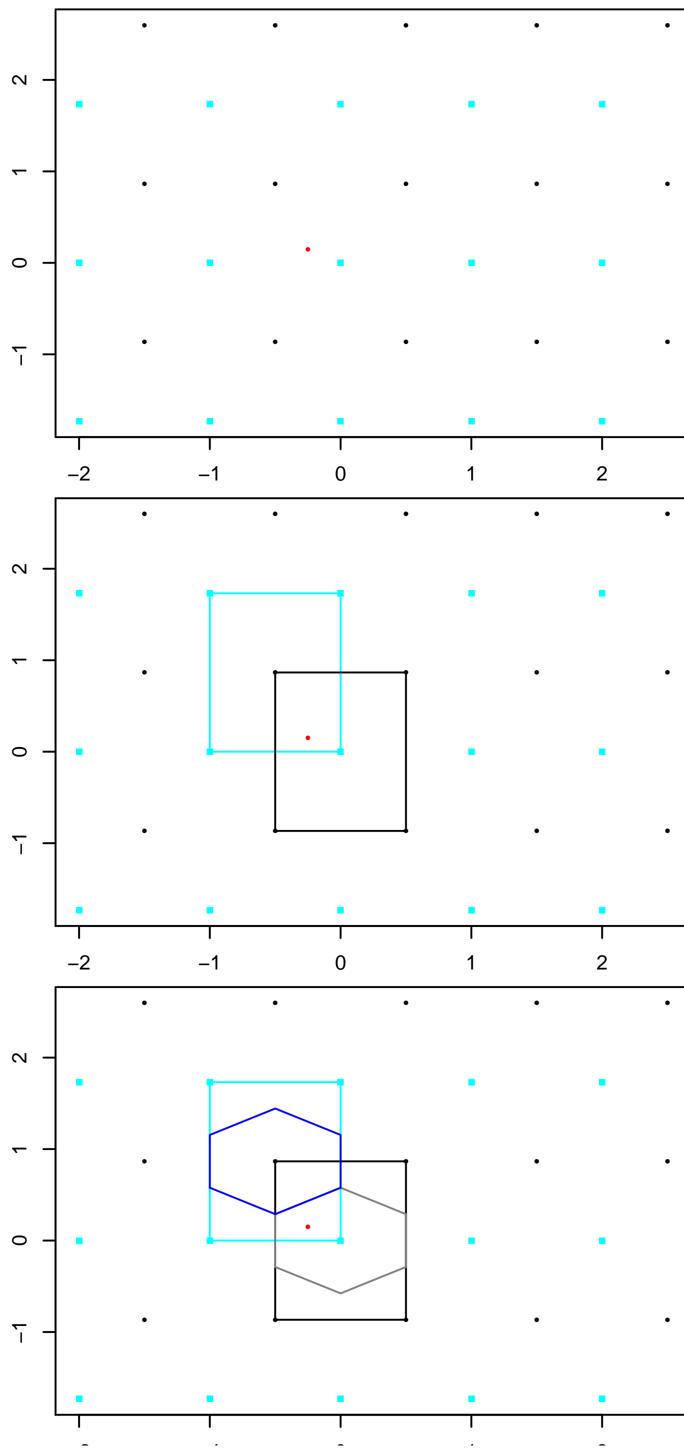


Figure 2:

Figure 3: Hexagon ”boxplots” showing the top 95 percent of the data for males and females. The red hexagons are an estimate of the bivariate median.

```
> shape <- optShape(height = vpin[2], width = vpin[1], mar = mai)
> xbnds <- range(NHANES$Transferin, na.rm = TRUE)
> ybnds <- range(NHANES$Hemoglobin, na.rm = TRUE)
> hbF <- hexbin(NHANES$Transferin[NHANES$Sex == "F"], NHANES$Hemoglobin[NHANES$Sex ==
+     "F"], xbnds = xbnds, ybnds = ybnds, shape = shape)
> hbM <- hexbin(NHANES$Transferin[NHANES$Sex == "M"], NHANES$Hemoglobin[NHANES$Sex ==
+     "M"], xbnds = xbnds, ybnds = ybnds, shape = shape)
> plot.new()
> hdiffplot(erode(hbF, cdfcut = 0.25), erode(hbM, cdfcut = 0.25),
+     unzoom = 1.3)
```

Figure 4: A difference plot of transferin vs hemoglobin for males and females.