Kernel density estimation

Suppose that you have a complicated function that you can't represent analytically and that you can't sample at a huge number of points. One way this might happen is that the function is itself very time-consuming to evaluate. Another way is that the "function" might relate directly to observables; for example, in a problem that Mike Boylan-Kolchin and I started but haven't finished, we might think about the location and velocity of satellite galaxies relative to the main galaxy. We can only see a certain number of satellite galaxies, so we can't just keep sampling.

In such a problem, we might want to have an idea of the underlying continuous probability distribution in our parameters (such as the position and velocity for the satellite galaxy example). Is there a way to do this using only a limited set of samples? In particular, can we do this if our samples are not weighted in any particular way? For example, if we are looking at satellite galaxies, each galaxy has the same weight.

A standard approach involves kernel density estimation (KDE). Each discrete point in our sample is replaced by an extended probability distribution, called a kernel, and the probability density at any given point in the space is then estimated to be the sum of the kernels at the chosen point, over all of the discrete samples (after proper normalization). Thus if the chosen point is close to many sample points its estimated probability density will be larger than if it is far away from any sample point. Once you compute the smoothed approximation to the probability distribution from your discrete data, you can then sample from it much more rapidly and can thus perform various statistical tasks with your approximation to the true distribution.

One might imagine that the best way to maximize information would be for the width of the kernel associated with each sample point to be very small, because this procedure would retain nearly full memory of the position of each sample point in the parameter space, whereas a broad kernel would smooth out and reduce information. Although it is true that the width of a kernel (usually called the bandwidth in this context) should not be too large (in the limit of infinite size it guarantees a uniform probability distribution), it is also bad to have a bandwidth that is too narrow. For example, in the limit that the bandwidth is much narrower than the distance between any two sample points, every point is effectively isolated and thus the "smoothed" probability distribution is just a set of spikes. Thus a major issue in the theory of kernel density estimation is how the bandwidth should be determined. It turns out that the functional form of the kernel is less important; we choose a multivariate Gaussian because of its many advantageous mathematical properties.

Before getting into details, we will take advantage of the Wikipedia page on kernel density estimation, https://en.wikipedia.org/wiki/Kernel_density_estimation, to show two figures that indicate how the procedure works. In Figure 1 we see how the KDE approach

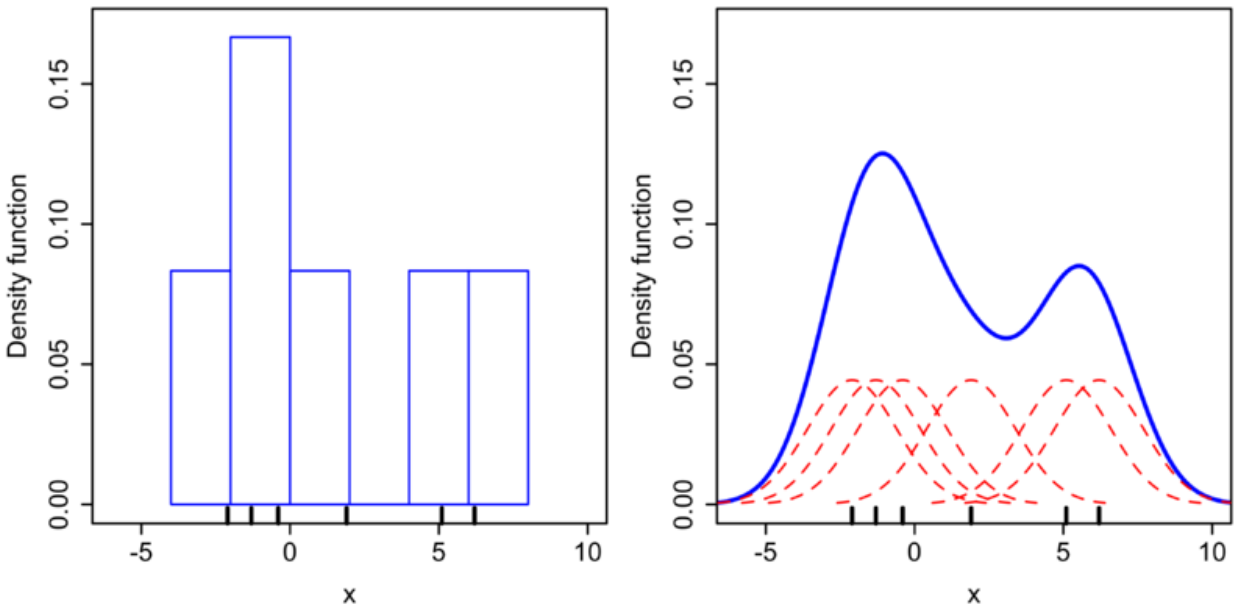

Fig. 1.— Comparison of a histogram and a kernel density estimate, from https://en.wikipedia.org/wiki/Kernel_density_estimation. The sample values are shown by the short vertical black lines near the x-axis. On the left we have a histogram of the values. On the right we have a kernel density estimate; to each point is associated a Gaussian with standard deviation 1.5 (red dashed lines), and note that the amplitude is the same for every Gaussian. The final kernel density estimate is the sum of the Gaussians, which makes the blue curve. We see that the kernel density estimate is smoother than the histogram, and indeed it converges faster to the true underlying distribution faster than a histogram.

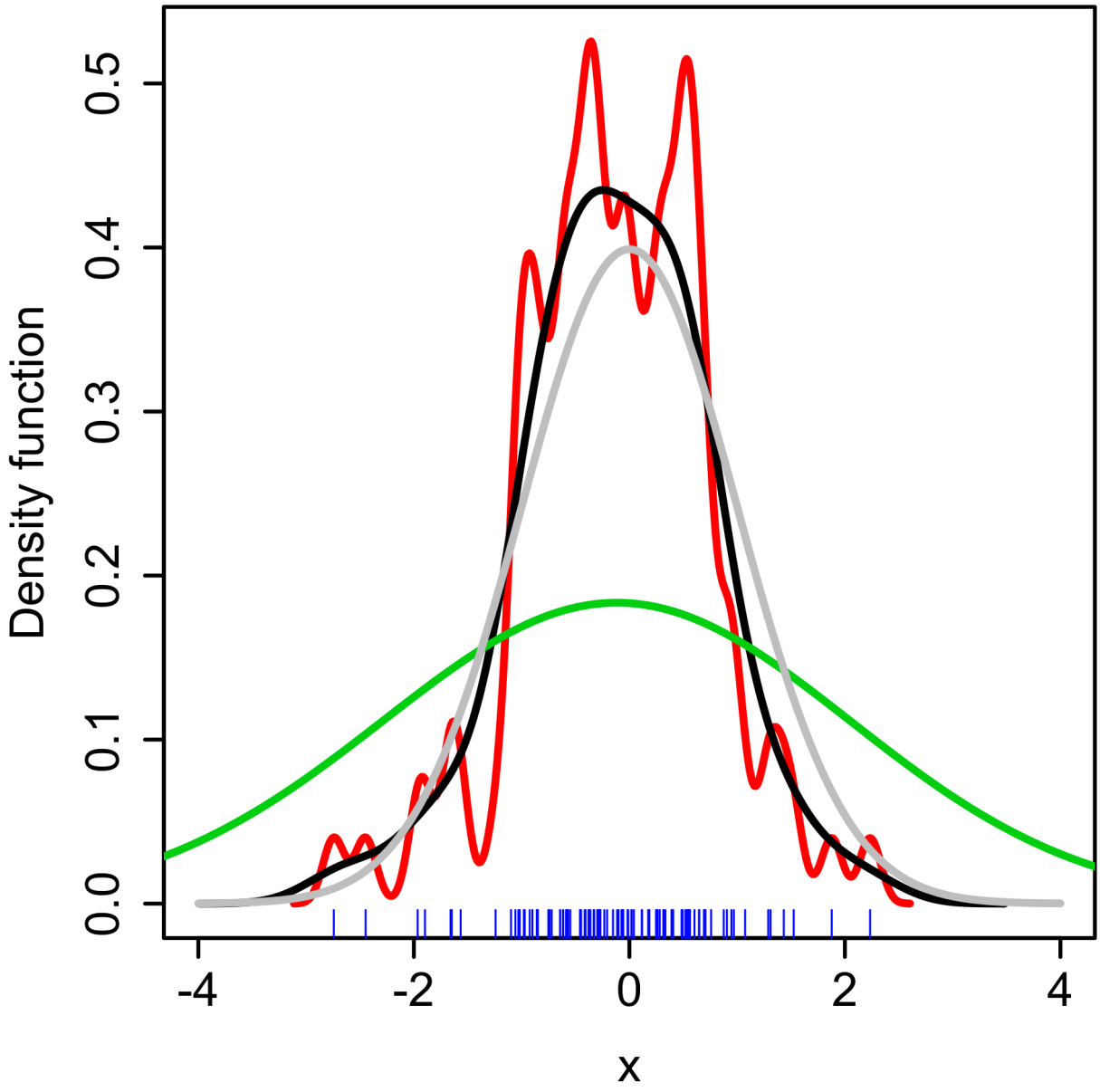

Fig. 2.— Consequences of different choices for the bandwidth, again from https://en.wikipedia.org/wiki/Kernel_density_estimation. The original 100 samples are shown by the short blue vertical ticks near the x-axis, and were drawn from a zero-centered unit-variance normal distribution (shown by the grey curve). Again, the kernel is a normal distribution. The red curve shows the result of kernel density estimation when the standard deviation of the kernel is 0.05; the black curve when it is 0.337; and the green curve when it is 2. We see that when the bandwidth is too narrow, the KDE curve is very choppy; when it is too wide, we have "oversmoothing" and important aspects of the distribution are missed; when we choose more optimally the summed curve is a fairly good representation of the original distribution, although of course not perfect.

works, where a Gaussian kernel has been used. In Figure 2 we see the result of various choices for the width of the kernel. Clearly, there are some better choices for the bandwidth than others, and indeed the choice of bandwidth is much more important than the functional shape of the kernel (for example, Gaussians are fine for the shape). But how do we choose the bandwidth?

To explore some of the details of kernel density estimation, we can use http://sfb649.wiwi.hu-berlin.de/fedc_homepage/xplore/ebooks/html/spm/spmhtmlnode18.html as a guide (see also Rosenblatt 1956, http://dx.doi.org/10.1214/aoms/1177728190; and Parzen 1962, http://dx.doi.org/10.1214/aoms/1177704472 for some of the original references). We will now jump to thinking about possibly multidimensional kernels. Then even if we have chosen to use a Gaussian we have to think about the axis lengths and orientation of the Gaussian kernel as well as its overall width. For example, one could imagine using a spherical kernel, which would mean that the bandwidth is the same in any direction. A generalization of this would be to use a bandwidth that can be represented by a diagonal matrix, with values that could be different on different parts of the diagonal. A further generalization of this is to use a full bandwidth matrix $\mathbf{H}$, where $\mathbf{H}$ is a real, symmetric matrix. Then, using equation (3.60) of the first reference above, we find that the estimate of the probability density at a given point $\mathbf{x}$ given $\mathbf{H}$ and samples at the points $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ is

$$\hat{f}_{\mathbf{H}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\det(\mathbf{H})} \mathcal{K}\left\{\mathbf{H}^{-1}(\mathbf{x} - \mathbf{X}_i)\right\} \ , \tag{1}$$

where n is the number of points in our sample. Here $\mathcal{K}(\mathbf{v})$, where $\mathbf{v}$ is a column vector, is

$$\mathcal{K}(\mathbf{v}) = \exp(-\mathbf{v}^T\mathbf{v}/2) \ . \tag{2}$$

For $\mathbf{H}$ we use the rule of thumb from Silverman and Green 1986, *Density Estimation for Statistics and Data Analysis*:

$$\mathbf{H} = n^{-1/(d+4)}\mathbf{\Sigma}^{1/2} \tag{3}$$

where d is the number of dimensions (of the data; for example, if we measure the 3-D position of a galaxy and nothing else, then $d = 3$) and $\mathbf{\Sigma}$ is the covariance matrix. To calculate the covariance matrix, we assume that each sample point is a d-dimensional vector $(Y_1, Y_2, \ldots, Y_d)$. Let μ_i be the mean of Y_i over the sample, and let $E[(Y_i - \mu_i)(Y_j - \mu_j)]$ be the expected value (i.e., the mean) of $(Y_i - \mu_i)(Y_j - \mu_j)$ over the sample. Then as we saw in Lecture 8 the ij element of Σ is

$$\Sigma_{ij} = E[(Y_i - \mu_i)(Y_j - \mu_j)] \ . \tag{4}$$

The covariance matrix is real and symmetric, which makes it straightforward to compute its square root (e.g., see the Wikipedia page on the square root of a matrix): let V be a matrix with the d eigenvectors as columns. Then we can write

$$\mathbf{\Sigma} = VDV^{-1} \tag{5}$$

where D is the diagonal matrix with the eigenvalues as elements. Taking the positive square root of D, $D^{1/2}$, we get

$$\mathbf{\Sigma}^{1/2} = V D^{1/2} V^{-1} \ . \tag{6}$$

Public codes exist to compute the orthonormal eigenvectors and eigenvalues of real symmetric matrices.

Sounds like we're all set! But there is a fly in the ointment. Look again at Equation 3. The bandwidth scales with the number of points n like $n^{1/(d+4)}$, where d is the dimensionality of the data. This is remarkably slow convergence! Even for $d = 1$, the scaling is $n^{1/5}$, which means that to halve your bandwith you need 32 times as many points. That's not by itself catastrophic, because with more points the overlap is of course greater and thus your approximation remains smooth. Nonetheless, because of the slow convergence and because it is useful to have more than one method for a given task, we will now talk about an alternative approach.

k-d trees

Another way to construct probability densities is based purely on the points that have been sampled, rather than attempting to smooth the data as in KDE methods. The method uses k-d trees, where d means "dimensions" and thus this is a construction of trees in k dimensions. The basic idea is represented in Figure 3, which we take from https://dcc.ligo.org/LIGO-P1400054/public/kdtrees.pdf. In brief, one dimension at a time, you divide the domain into halves, then quarters, then eighths, ..., each time figuring out the median of the region in question (so, for example, you might divide into halves for the x dimension, then divide each of those two into halves in the y dimension, then each of those four into eighths in the z dimension, then in the x direction again if you have three dimensions in your data).

As always there is ambiguity about how to report a given credible region, but one good approach is a "greedy algorithm". After you have divided your region into boxes, you compute the probability density by simply dividing the number of points in the box by the volume of the box (normally we won't have the fortuituous situation depicted in the figure, where each box has exactly the same number of points). Then you "greedily" take the box with the highest probability density, then the next highest density, and so on, until you have reached the desired fraction of the total number of points (e.g., 68.3% if you want a "1 sigma" credible region). Again, note that there is no smoothing in this procedure; it's a simple partition of the points you have already sampled. The intent is not to get a continuous probability density, but is instead to represent the credible regions. Note that you can do this for any pair (or triple, or quadruple, ...) of parameters from a converged MCMC sample: just list all of the combinations of your interesting parameters from the converged sample, and those provide the points that you partition.

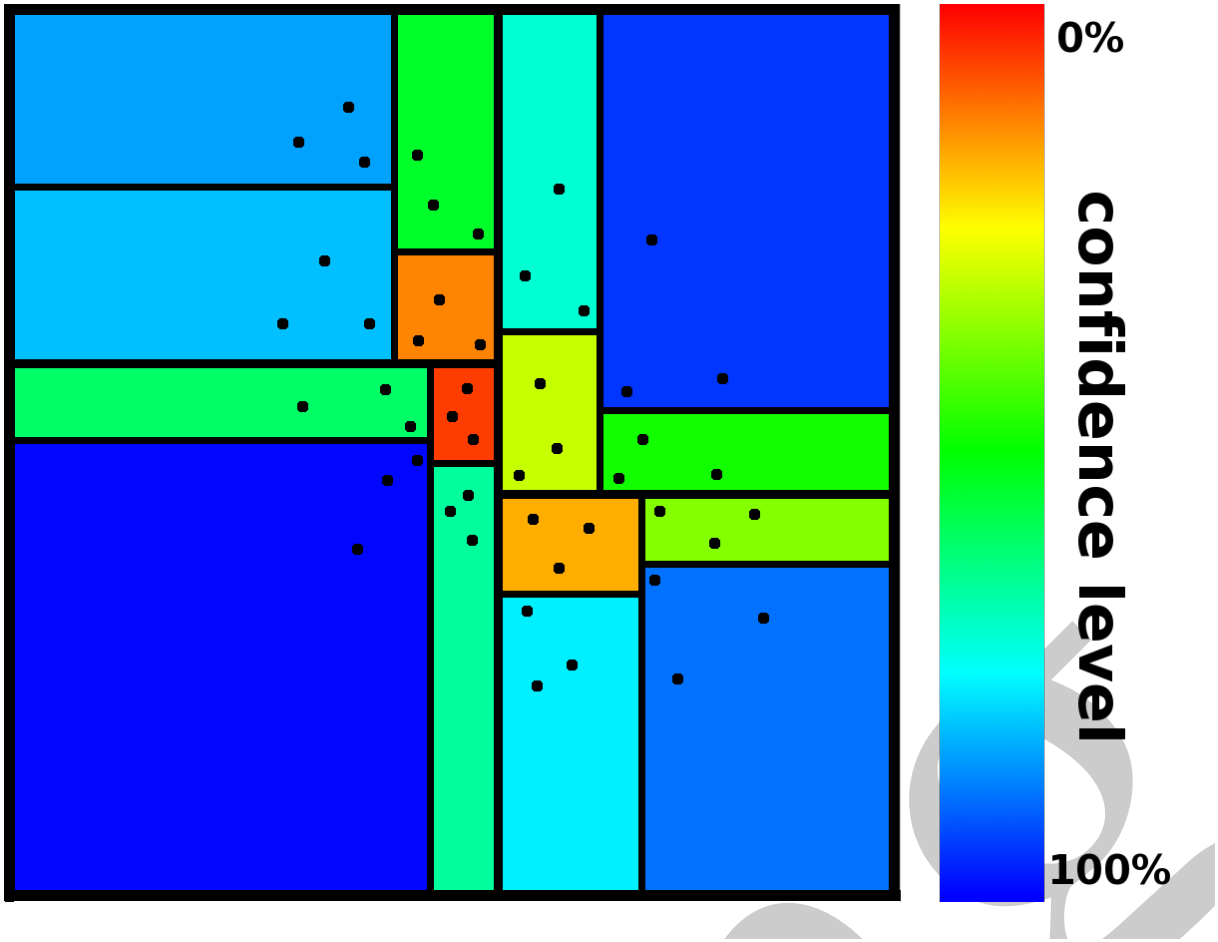

Fig. 3.— A good representation of how a k-d tree works for two dimensions ($d = 2$), from https://dcc.ligo.org/LIGO-P1400054/public/kdtrees.pdf. We start with 48 points sampled according to a posterior probability density. Let us call the horizontal axis "x" and the vertical axis "y". The first step is to place a vertical bar (at constant x) that divides the sample in half, so that there are 24 points on each side. The next step is to place bars at constant y in each of the halves so that there are 12 points above and 12 below; note that the bars have different y-values on each side. Then each of the four regions is divided in two by a bar at constant x, then finally each of the resulting eight regions is divided in two by a bar at constant y. As we have discussed, the construction of the credible regions is nonunique, but here a "greedy algorithm" is used: starting with the highest-density region (in this case, the smallest box, given that each box has by design three points) and progressing to lower-density regions until the desired fraction of the total probability has been reached.

This method is fast and easy to implement, but there is a subtle bias pointed out by the authors of https://dcc.ligo.org/LIGO-P1400054/public/kdtrees.pdf. Suppose we have selected some points from a one-dimensional distribution that is uniform in some range and zero outside that range. Due to fluctuations, the median of that distribution will not be in the exact middle of the range, and this will be true for each subsequent subdivision. Thus there will be a bias, which will therefore lead to a misrepresentation of the distribution.

The authors suggest that the remedy is to (1) begin by picking, at random, half of the sample points, (2) use that half to produce the *bounds* of the boxes that result from the partitioning of the data space, then (3) populate the boxes only with the *other* half of the points to get the final result and read out the densities. In the example of a uniform density, the original procedure would typically produce boxes of different sizes, but in step (3) the larger boxes will likely have more points put in them, which means that the final density estimate will not be biased. I have used this approach in my own work, and it seems to be reliable and fast.

In summary, there are different ways to take a finite set of samples and use them to represent an underlying smooth distribution. As always, I recommend that you try them and and determine which approach suits your problems best!