densEstBayes: density estimation via Bayesian inference engines

Matt P. Wand

University of Technology Sydney

15th September, 2020

1 Introduction

Probability density function estimation, or *density estimation* for short, based on a univariate sample is a problem of fundamental importance in statistics and data analysis. Approaches to the density estimation problem number in the hundreds and stretch over several decades, but shortcomings such as reliable choice of smoothing parameter are typical. Moreover, the vast majority of density estimation procedures are frequentist with no allowance made for the variability due to smoothing parameter choice. The R package `densEstBayes` provides high-quality Bayesian density estimates based the relatively recent Bayesian inference engine paradigm. It makes use of the Bayesian inference engines provided by the Stan platform (Carpenter *et al.*, 2017). Other options include slice sampling (e.g. Neal, 2003) and semiparametric mean field variational Bayes (e.g. Rohde & Wand, 2016). The infrastructure of `densEstBayes` is such that new and improved Bayesian inference engines can be incorporated when they are developed.

All density estimates provided by `densEstBayes` are accompanied with pointwise credible intervals that allow for variability in smoothing parameter selection. The methodology, based on binning, scales well to very large sample sizes. Full details of the Bayesian inference engine approach to density estimation are given in Wand & Yu (2020).

2 Ilustrations for the 2011 Old Faithful Geyser Data

The `OldFaithful` dataset consists of the time intervals, in minutes, for all 3,507 adjacent pairs of eruptions of the Old Faithful Geyser in Yellowstone National Park, U.S.A., during 2011. The following R commands:

```
> library(densEstBayes)
> hist(OldFaithful2011,col = "gold",main = "",probability = TRUE,
+      xlab = "time interval between geyser eruptions (minutes)")
```

lead to the probability histogram shown in Figure 1:

A quick-to-compute Bayesian density estimate, based on the Bayesian inference engine paradigm described in Wand & Yu (2020), is obtained via the command:

```
> destSMFVB <- densEstBayes(OldFaithful2011,method = "SMFVB")
```

The method = "SMFVB" specification leads to *semiparametric mean field variational Bayes* (e.g. Rohde & Wand, 2016) being used for approximate Bayesian inference. A plot of the estimate is obtained using:

```
> plot(destSMFVB,xlab = "time interval between geyser eruptions (minutes)",
+  main = "method = \"SMFVB\" (semiparametric mean field variational Bayes)")
> rug(jitter(OldFaithful2011,amount = 0.2),col = "dodgerblue")
```

and is shown in Figure 2. The shaded region corresponds to approximate 95% pointwise credible intervals. A bimodal distribution is apparent, with the main mode at around 90 minutes and a secondary mode at around 65 minutes.

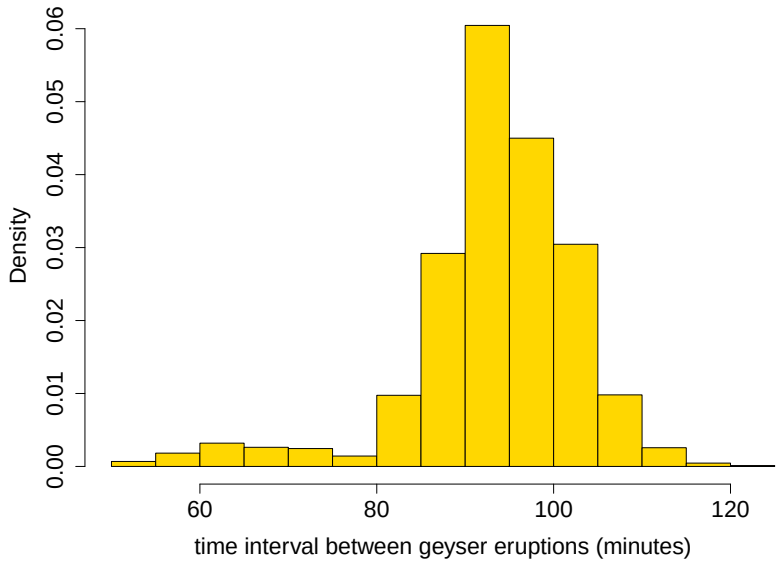

Figure 1: *A probability histogram for the time intervals (minutes) between all 3,507 adjacent pairs of eruptions of the Old Faithful Geyser during the year 2011. The histogram bins correspond to the default call to the* R *function* `hist()`.

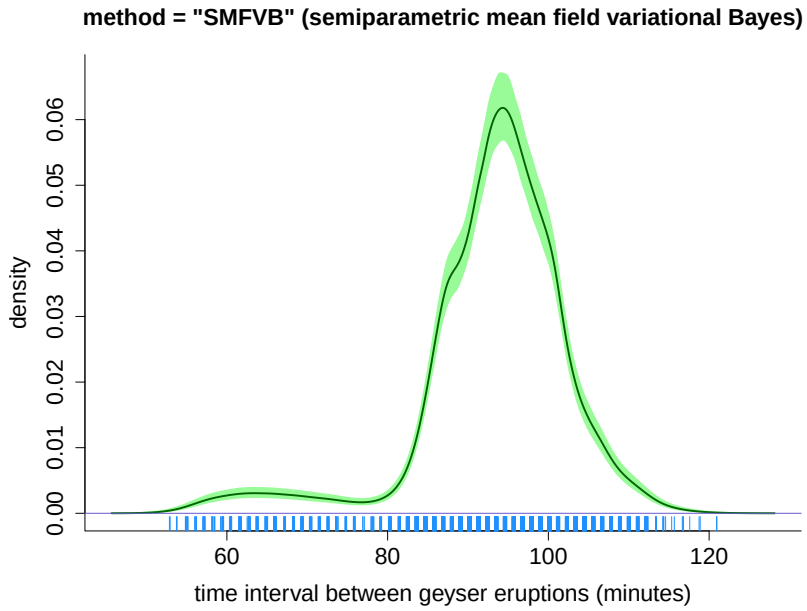

Figure 2: *Bayesian density estimate for the 2011 Old Faithful geyser data introduced in Figure 1. The pale green shaded region corresponds to approximate 95% pointwise credible intervals. The data are shown as tick marks, with some jittering, at the base of the plot.*

A density estimate based on the no-U-turn sampler (Hoffman & Gelman, 2014) as implemented in Stan (Carpenter *et al.*, 2017). Bayesian inference engine is obtained and plotted via the commands:

```
> destNUTS <- densEstBayes(OldFaithful2011,method = "NUTS")
> plot(destNUTS,xlab = "time interval between geyser eruptions (minutes)",
+      main = "method = \"NUTS\" (no-U-turn sampler)")
```

```
> rug(jitter(OldFaithful2011,amount = 0.2),col = "dodgerblue")
```

with the result displayed in Figure 3. The estimate and variability band are very similar to those obtained via semiparametric mean field variational Bayes with method = "SMFVB". Even though method = "SMFVB" can provide a fast Bayesian density estimate, it involves solving a high-dimensional optimisation problem for which convergence is difficult to guarantee for general input data sets. Further details on this issue are given in Section 4.1. The Figure 3

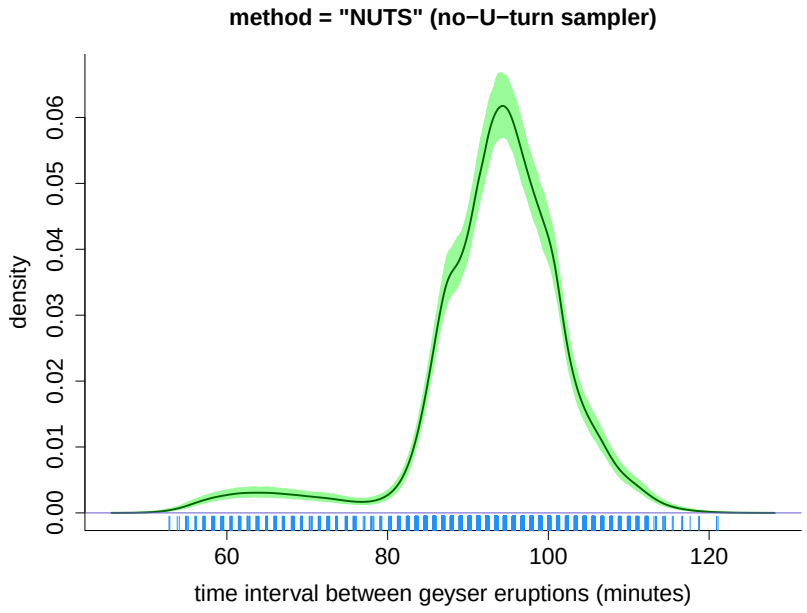

Figure 3: *Bayesian density estimate for the data from Figure 1 using the no-U-turn sampler. The pale green shaded region corresponds to approximate 95% pointwise credible intervals. The data are shown as tick marks, with some jittering, at the base of the plot.*

density estimate is based on a version of Markov chain Monte Carlo sampling and at least a cursory check of the relevant chains is well-advised. The function `checkChains()` facilitates graphical checks of the chains corresponding to vertical slices of the density estimate at 5 equally-spaced abscissae. The vertical lines in Figure 4 indicate the locations of the default vertical slices. The command:

```
> checkChains(destNUTS)
```

produces the graphic given in Figure 5. In this case, all chains seem to be well-behaved and the method = "NUTS" Bayesian density estimate shown in Figures 3 and 4 is trustworthy.

The commands:

```
> destDefault <- densEstBayes(OldFaithful2011) ; plot(destDefault)
```

produce and plot the default `densEstBayes()` estimate, and is based on slice sampling (method = "slice"). Usually the slice sampling-based estimate is computed more quickly than the Stan-dependent estimates. This is because the underlying C++ code for method = "slice" focusses on the specific model used for Bayesian density estimation whereas the method = "HMC" and method = "NUTS" use a Bayesian inference engine that is designed to handle a very general class of Bayesian models.

3 Controlling the Estimation Procedure

Various settings for Bayesian density estimation via the `densEstBayes()` can be controlled using the control argument in calls to the function `densEstBayes()`. For example, the following

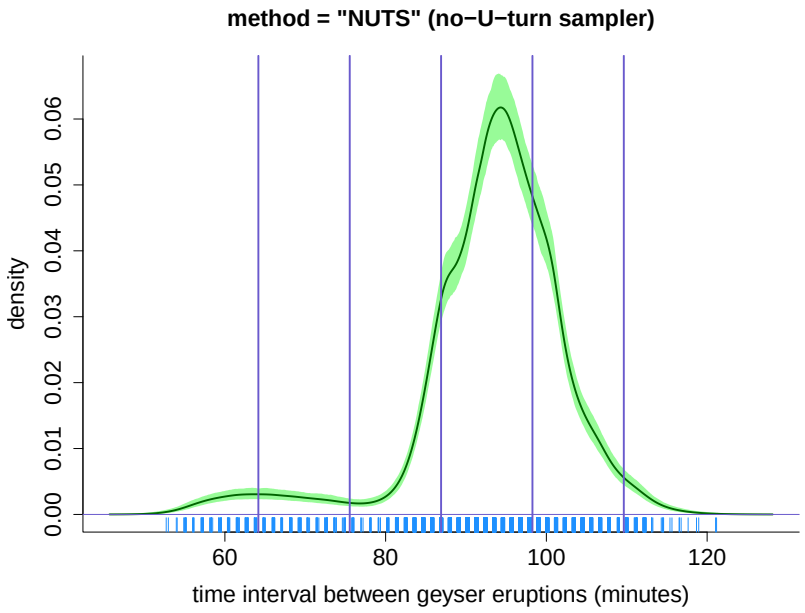

Figure 4: *The same as for Figure 3 but with the addition of lines that indicate the positions of the vertical slices for which chains for the log-density estimate are monitored by the function* `checkChains()`.

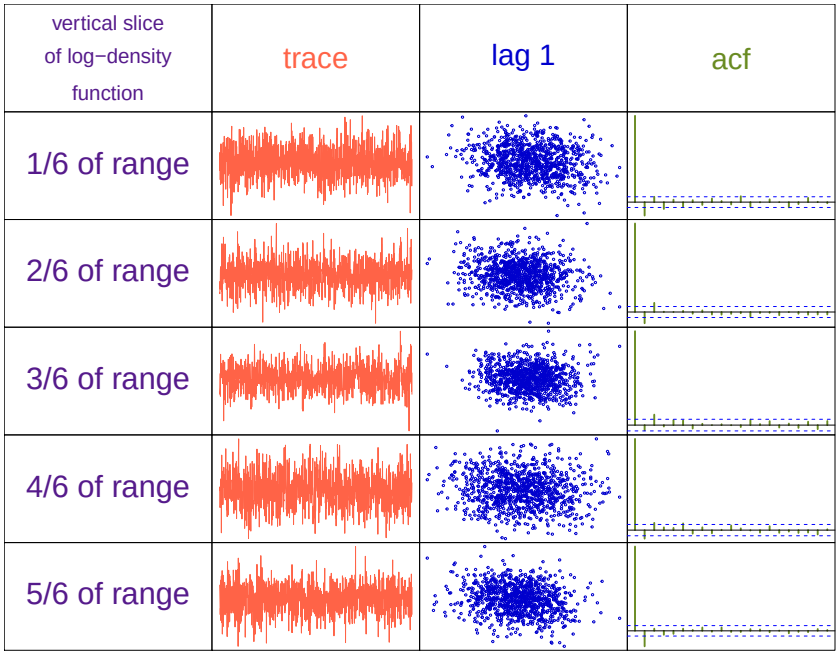

Figure 5: *The graphic produced by the command* checkChains(destNUTS) *where* densNUTS *is the* `densEstBayes()` *object corresponding to the estimate shown in Figure 5. The chains correspond to the estimates of the log-density function at the vertical slices shown in Figure 4. The second column is a trace (time series) plot of the chain. The third column is a scatterplot of the chain values against their previous (lag 1) values. The fourth column is the sample autocorrelation function, based on the* R *function* `acf()`.

command increases the warm-up size to $2,000$, the lengths of the kept chains to $3,500$ and then thins the kept chains by a factor of 5 when method = "HMC":

```
> destLongerChains <- densEstBayes(OldFaithful2011,method = "HMC",
+                                  control = densEstBayes.control(nWarm = 2000,
+                                  nKept = 3500,nThin = 5))

> checkChains(destLongerChains)
```

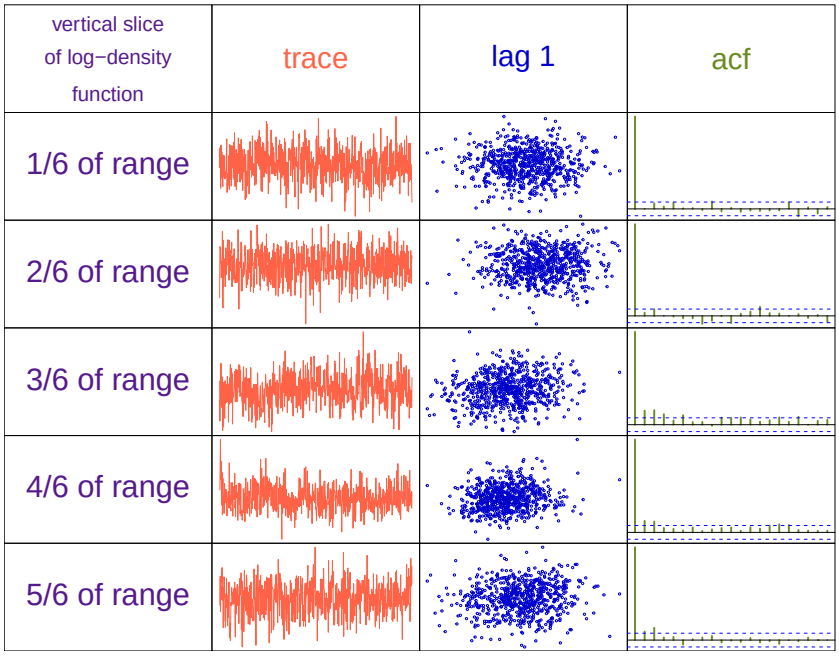

Figure 6: *The chains corresponding to the density estimate object* densLongerChains *with a warm-up of size* 2,000*, kept samples of size* 3,500 *and a thinning factor of* 5*.*

The default number of basis functions for the O'Sullivan penalized splines (e.g. Wand & Ormerod, 2008) is 50. For the 2011 Old Faithful geyser data it is likely that a much smaller basis is sufficient to resolve the structure in the data's density function and that 50 functions is an overkill for this particular dataset. To reduce the number of basis functions to 20 type:

```
> destSmallerBasis <- densEstBayes(OldFaithful2011,control =
+                                  densEstBayes.control(numBasis = 20))
```

The following code plots destSmallerBasis and compares it with the default density estimate:

```
> plot(destSmallerBasis,shade = FALSE,estCol = "blue",
+           xlab = "time interval between geyser eruptions (minutes)")
> rug(jitter(OldFaithful2011,amount = 0.2),col = "dodgerblue")
> destDefault <- densEstBayes(OldFaithful2011)
> plot(destDefault,shade = FALSE,estCol = "darkorange",add = TRUE)
> legend("topleft",legend = c("20 spline basis functions",
+        "50 spline basis functions"),lty = rep(1,2),
+        col = c("darkorange","blue"))
```

The full list of control parameters, and the purpose of each one, can be obtained by issuing the command:

```
> help(densEstBayes.control)
```

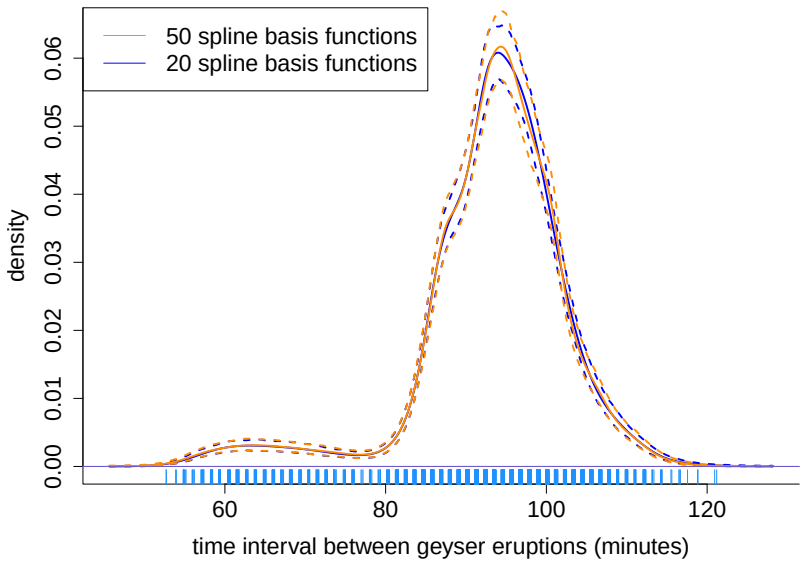

Figure 7: *Bayesian density estimates for the 2011 Old Faithful geyser data with the* densEstBayes() *default of 50 basis functions and a simpler one with 20 basis functions specified using the* control *argument of* densEstBayes().

4 Convergence Issues

Each of the methods used by densEstBayes() for achieving Bayesian density estimation rely on either an iterative algorithm (method = "SMFVB") or Markov chain Monte Carlo sampling (method = "HMC", method = "NUTS" and method = "slice") and, hence, are prone to convergence issues. We now provide some illustrations of such issues.

4.1 Fixed point iteration for semiparametric mean field variational Bayes

When method = "SMFVB" is specified in the call to densEstBayes() then a fixed point iterative scheme is used to obtain the semiparametric mean field variational Bayes approximate posterior distributions of spline basis coefficients. For the default number of spline basis functions, the approximation involves a 52-dimensional Multivariate Normal distribution with a full covariance matrix. The mean and covariance matrices are determined using the fixed point iterative scheme listed at the bottom of page 29 of Rohde and Wand (2016). Often convergence is quite rapid, and a fast and high quality Bayesian density estimate results. However, noting that the number of free parameters is $52 + \frac{1}{2} \times 52 \times 53 = 1,430$, guaranteeing convergence for general input data is a tall order. The following code illustrates this problem, starting with simulation of 100 observations from the 5th density function in Table 1 of Marron and Wand (1992), which is such that outliers tend to present:

```
> set.seed(2) ; x <- rMarronWand(100,5)
> destSMFVB <- densEstBayes(x,method = "SMFVB")
```

The following error message is returned:

```
Error:

 Call to densEstBayes() with method="SMFVB" has failed due
 to a singular matrix and/or non-convergence problem. Use
 method="slice", method="NUTS" or method="HMC" instead.
```

If, instead, the default of method = "slice" is used then the following code:

```
> destDefault <- densEstBayes(x) ; plot(destDefault)
> rug(x,col = "dodgerblue")
> xg <- seq(destDefault$range.x[1],destDefault$range.x[2],length = 1001)
> trueDensg <- dMarronWand(xg,5)
> lines(xg,trueDensg,col = "indianred3")
> legend("topleft",legend = c("estimate","truth"),lty = rep(1,2),
+        col = c("darkgreen","indianred3"))
```

produces the estimate shown in Figure 8. The density function from which the data were simulated is shown for comparison.

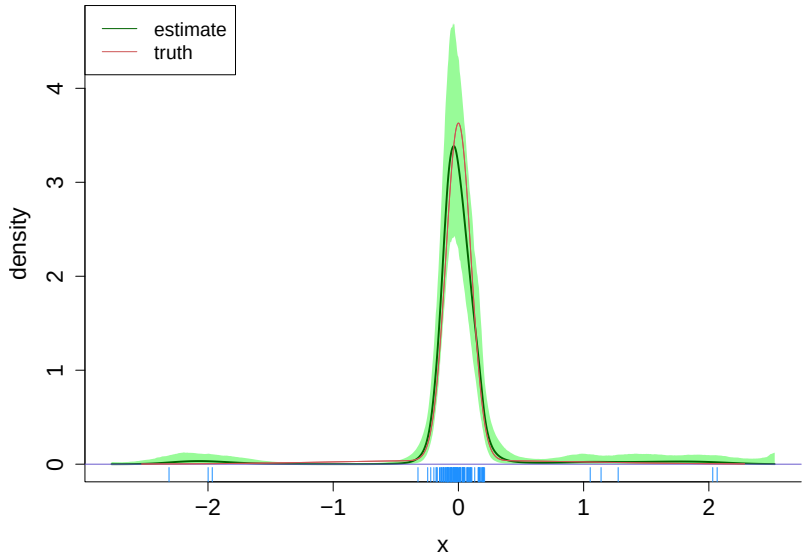

Figure 8: *The default* densEstBayes() *estimate for data simulated from the 5th density function in Table 1 of Marron and Wand (1992) via the function* rMarronWand(). *The true density function is also shown. The shaded region corresponds to pointwise 95% credible intervals.*

4.2 Markov chain Monte Carlo convergence issues

Consider density estimation for a dataset consisting of the widths of 93 makes of cars. The data are part of the R package MASS (Ripley, 2020) and the following code obtains and prints the sorted data:

```
> library(MASS) ; print(sort(Cars93$Width))
```

which results in:

```
 [1] 60 63 63 63 63 63 64 65 65 65 66 66 66 66 66 66 66 66 66 66 67 67 67 67 67
[26] 67 67 67 67 67 67 67 67 67 67 68 68 68 68 68 68 68 69 69 69 69 69 69 69 69
[51] 69 69 69 70 70 70 70 70 70 70 71 71 71 71 71 72 72 72 72 72 72 72 73 73 73
[76] 73 74 74 74 74 74 74 74 74 74 74 74 75 77 77 78 78 78
```

A Bayesian density estimate based on a default call to densEstBayes() and plot involves the code:

```
> destCarWidths <- densEstBayes(Cars93$Width)
> plot(destCarWidths,xlab = "car width (inches)")
> rug(Cars93$Width,col = "darkred")
> rug(jitter(Cars93$Width),col = "dodgerblue")
> legend("topleft",legend = c("actual data","jittered data"),
+        lty=rep(1,2),col=c("darkred","dodgerblue"))
```

and leads to the estimate shown in Figure 9 (note that the estimate depends on random draws within a Markov chain Monte Carlo scheme and is subject to change for different calls to `densEstBayes()`). The density estimate is not very visually appealing and is close to a linear

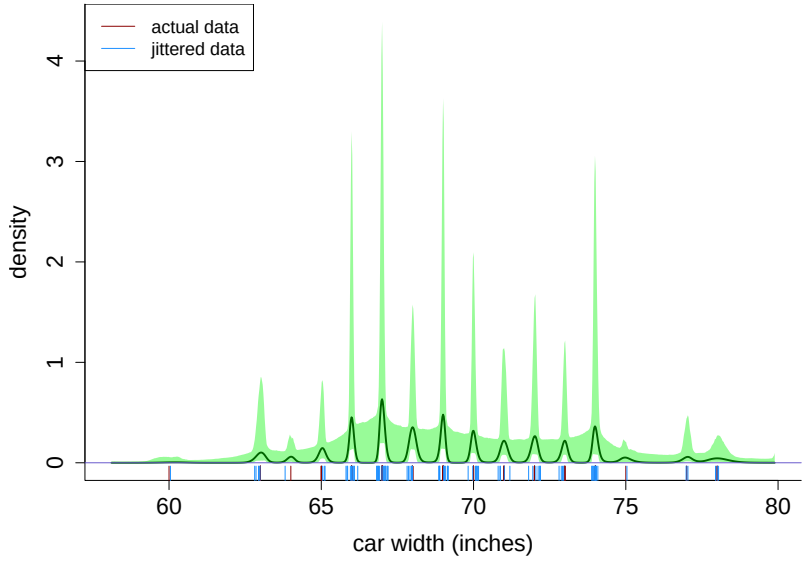

Figure 9: *Bayesian density estimate for data on widths (inches) of* 93 *cars from the data frame* `Cars93` *in the* R *package* `MASS`. *This estimate is produced from a call to the default version of the function* `densEstBayes()`. *The rug plots at the base of the plot show both the actual data, which has several clusters of identical observations, and a jittering of the data.*

combination of point masses at each of the unique data values. To check the chains on which this estimate is based we issue:

```
> checkChains(destCarWidths)
```

and obtain the graphic shown in Figure 10, which indicates that the Bayesian density estimation procedure is suffering from identifiability problems.

The culprit seems to be the degree to which these data have been discretised. The command:

```
> table(Cars93$Width)
```

leads to the output:

```
60 63 64 65 66 67 68 69 70 71 72 73 74 75 77 78
 1  5  1  3 10 15  7 11  7  5  7  4 11  1  2  3
```

and shows that there are only 16 unique values in a dataset of size 93.

The next set of commands allow for the slice sampling warmup and kept sample sizes to be much longer. A thinning factor of 10 is applied to the kept chains.

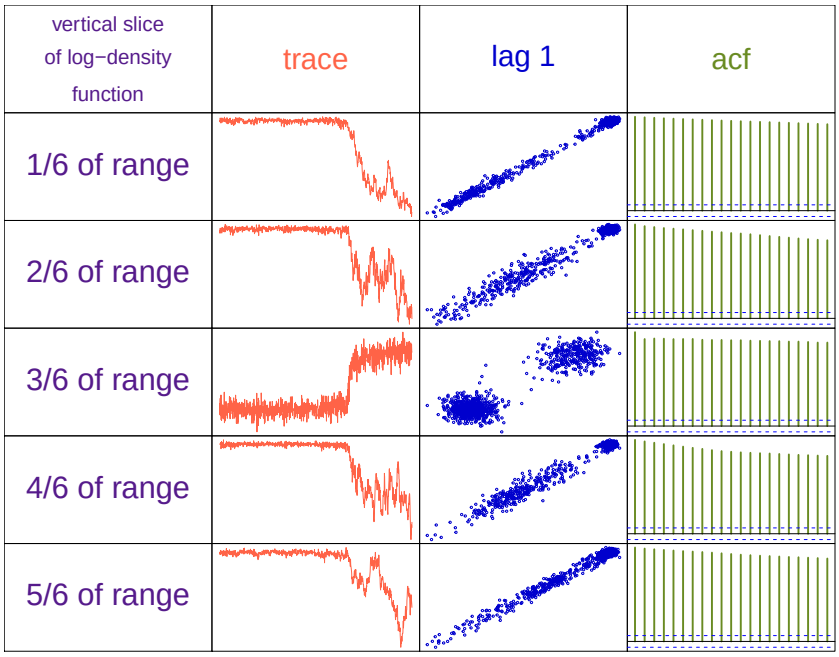

Figure 10: *The chains of the log-density estimates at five equally-spaced vertical slices for the slice sampling-based Bayesian density displayed in Figure 9.*

```
> destLongerChains <- densEstBayes(Cars93$Width,
+                      control = densEstBayes.control(nWarm = 10000,
+                                nKept = 10000,nThin = 10))
> plot(destLongerChains,xlab = "car width (inches)")
> rug(Cars93$Width,col = "darkred")
> rug(jitter(Cars93$Width),col = "dodgerblue")
> legend("topleft",legend = c("actual data","jittered data"),
+        lty = rep(1,2),col = c("darkred","dodgerblue"))
```

The resulting density estimate is shown in Figure 11 and indicates the slice sampling is converging to the linear combination of point masses density estimate. The command:

```
> checkChains(destLongerChains)
```

produces the Figure 12 graphic. The trace plots are flatter, but still not very satisfactory. The issue seems to be a type of degeneracy that arises in this Bayesian density estimation procedure when data on a continuous variable are discretised.

If we break the ties in the data by adding a small amount of random noise to each of the observations via code such as:

```
> carWidthsJittered <- jitter(Cars93$Width,amount = 0.2)
> destJitteredData <- densEstBayes(carWidthsJittered)
> plot(destJitteredData,xlab = "car width (inches)")
> rug(Cars93$Width,col = "darkred")
> rug(carWidthsJittered,col = "dodgerblue")
> legend("topleft",legend = c("actual data","jittered data"),
+        lty=rep(1,2),col=c("darkred","dodgerblue"))
```

then a density estimate such as shown in Figure 13 is shown. This estimate is in keeping with the continuous nature of the car widths variable. A check of the Figure 13 chains via the command:

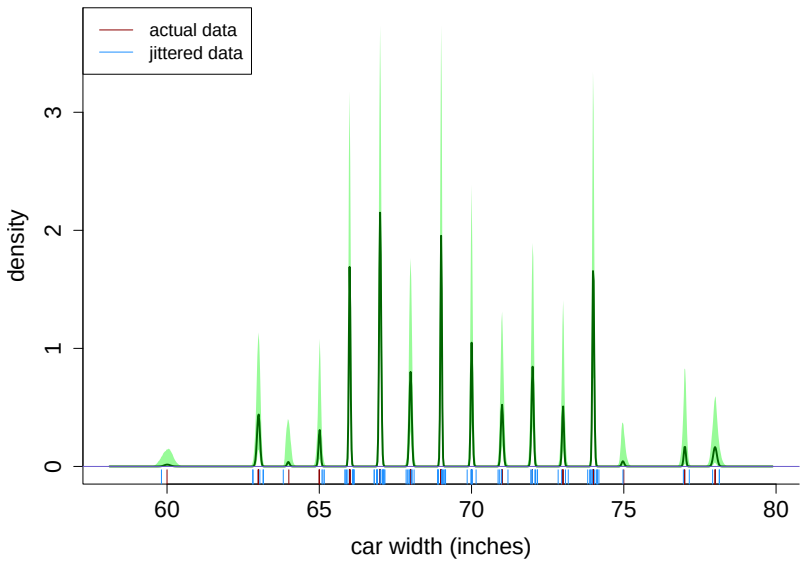

Figure 11: *The same as for Figure 11 except that the chains for the slice sampling are specified to have a warmup of length* $10,000$*, kept chains of length* $10,000$ *and a thinning factor of* 10*.*

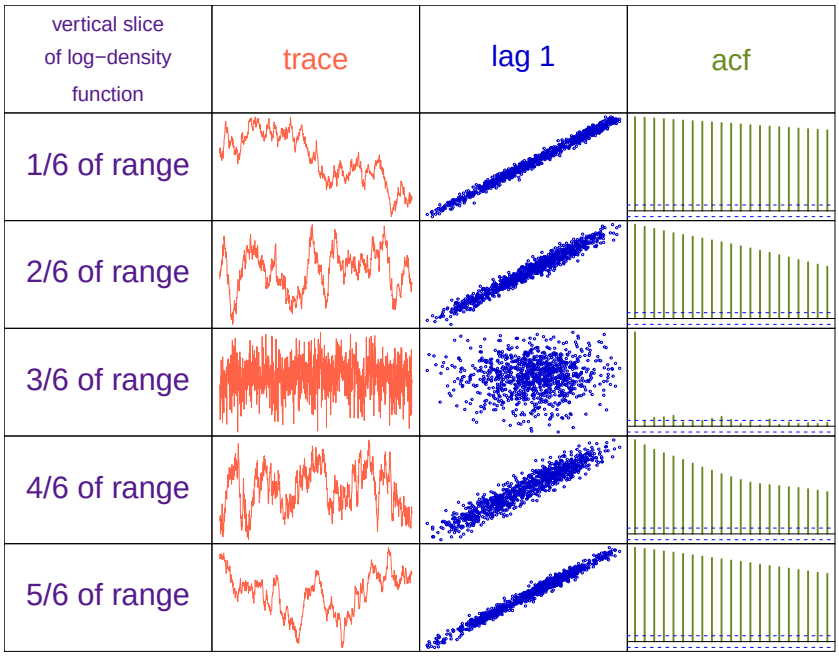

Figure 12: *The chains of the log-density estimates at five equally-spaced vertical slices for the slice sampling-based Bayesian density displayed in Figure 12.*

```
> checkChains(destJitteredData)
```

is shown in Figure 14 and shows that good Markov chain Monte Carlo behaviour is achieved.

To summarise, density estimation via Monte Carlo-based Bayesian inference engines can be affected by issues such as data discretisation and convergence checks via the `checkChains()` function are worthwhile.

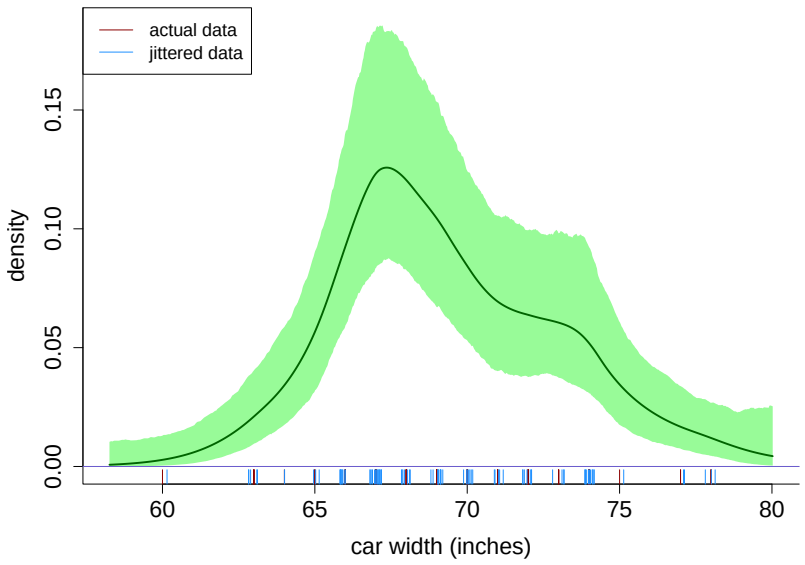

Figure 13: *The same as for Figure 11 but with the default call to* `densEstBayes()` *applied to the jittered data rather than the actual data.*

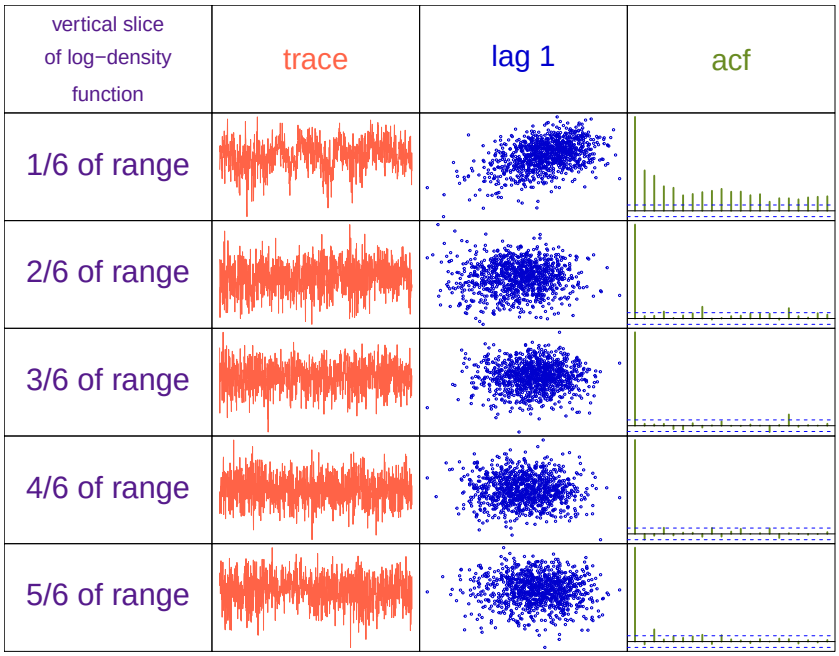

Figure 14: *The chains of the log-density estimates at five equally-spaced vertical slices for the slice sampling-based Bayesian density displayed in Figure 13.*

5 Display Options

After obtaining a Bayesian density estimation object using the `densEstBayes()` function there are various options for displaying the estimate. There are tweak factors such as the colour of the estimate curve and whether to use shading or curves for the variability band. Another display aspect is restriction to a sub-region. Some of these display options are explained and illustrated here.

The following commands obtain a density estimate of the 2011 Old Faithful geyser data

and plot the estimate with user-specified colours. The level of the pointwise credible intervals is set to 99%.

```
> dest <- densEstBayes(OldFaithful2011,method = "SMFVB")
> plot(dest,estCol = "darkmagenta",varBandCol = "pink",credLev = 0.99,
+      axisCol = "navy",
+      xlab = "time interval between geyser eruptions (minutes)")
> rug(jitter(OldFaithful2011,amount = 0.2),col = "forestgreen")
```

The result is shown in Figure 15.

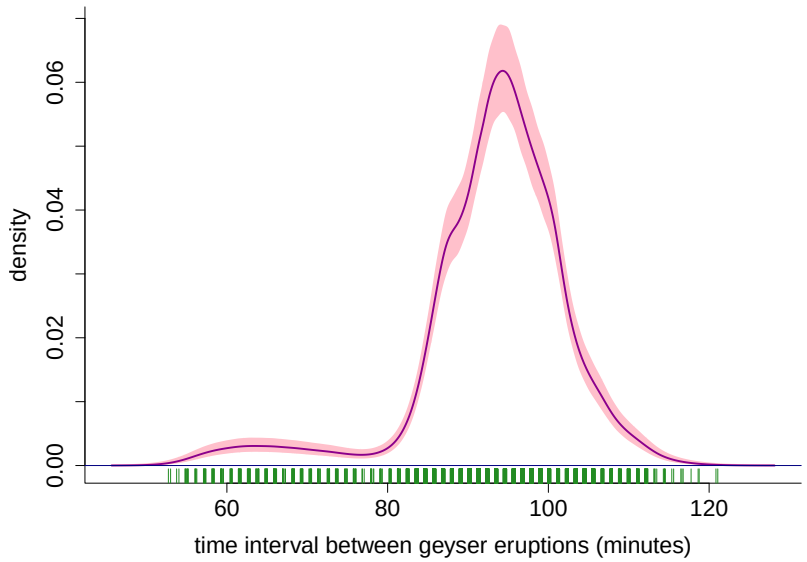

Figure 15: *Bayesian density estimate for the 2011 Old Faithful geyser data. The pink shaded region corresponds to approximate 99% pointwise credible intervals. The data are shown as tick marks, with some jittering, at the base of the plot.*

Now suppose that we want to zoom in on the region where the time interval between geyser eruptions is between 70 minutes and 100 minutes. Due to the normalization requirement, it is necessary to compute the estimate over the range of the data and then use the `predict()` function to obtain the plotting vectors:

```
> myXgrid <- seq(70,100,length = 501)
> predObj <- predict(dest,newdata = myXgrid,cred.fit = TRUE,credLev = 0.99)
> myEstGrid <- predObj$fit
> myLowGrid <- predObj$credLow.fit
> myUppGrid <- predObj$credUpp.fit
```

The abscissae plotting vector myXgrid and the ordinate plotting vectors myEstGrid, myLowGrid and myUppGrid can now be used to assemble the required graphic — with dashed curves used for the variability band:

```
> plot(0,type = "n",bty = "l",xlim = range(myXgrid),ylim = c(0,max(myUppGrid)),
+      xlab = "time interval between geyser eruptions (minutes)",
+      ylab = "density")
> abline(h = 0,col = "red")
> myColour <- "lightseagreen"
> lines(myXgrid,myEstGrid,col = myColour,lwd = 3)
```

```
> lines(myXgrid,myLowGrid,col = myColour,lwd = 3,lty = 2)
> lines(myXgrid,myUppGrid,col = myColour,lwd = 3,lty = 2)
> rug(jitter(OldFaithful2011[(OldFaithful2011>=70)&(OldFaithful2011<=100)],
+     amount = 0.2),col = "maroon")
```

The result is shown in Figure 16.

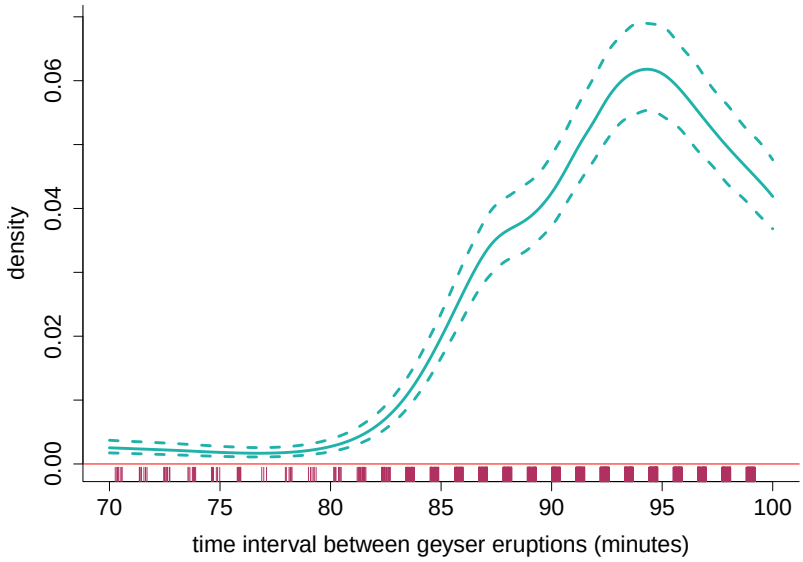

Figure 16: *Bayesian density estimate for the 2011 Old Faithful geyser data over the restricted interval of* 70 *to* 100 *minutes. The dashed curves correspond to approximate 99% pointwise credible intervals. The data are shown as tick marks, with some jittering, at the base of the plot.*

Finally, we provide an example involving two Bayesian density estimates plotted on the same axes. The data are part of the data frame carAuction within the R package HRW (Harezlak, Ruppert and Wand, 2019), which contains observations for 51 variables on 72,983 cars purchased at automobile auctions by dealerships. Here we study the variable carAuction$odomRead, which is the odometer reading, in miles, of the car at the time of purchase and divide the data according to whether or not the car is a considered a *bad buy* or *good buy* by the dealership. The following code creates the required data vectors:

```
> library(HRW)
> odomReadGoodBuy <- carAuction$odomRead[carAuction$IsBadBuy == 0]
> odomReadBadBuy <- carAuction$odomRead[carAuction$IsBadBuy == 1]
```

Bayesian density estimates for each data are obtain via:

```
> destGoodBuy <- densEstBayes(odomReadGoodBuy,method = "SMFVB")
> destBadBuy <- densEstBayes(odomReadBadBuy,method = "SMFVB")
```

Plotting vectors to display the two density estimates on the same set of axes are computed using:

```
> myOdomReadGrid <- seq(20000,120000,length = 1001)
> predObjGoodBuy <- predict(destGoodBuy,newdata = myOdomReadGrid,
+                           cred.fit = TRUE)
> estGridGoodBuy <- predObjGoodBuy$fit
```

```
> lowGridGoodBuy <-  predObjGoodBuy$credLow.fit
> uppGridGoodBuy <-  predObjGoodBuy$credUpp.fit
> predObjBadBuy <- predict(destBadBuy,newdata = myOdomReadGrid,
+                                     cred.fit = TRUE)
> estGridBadBuy <- predObjBadBuy$fit
> lowGridBadBuy <-  predObjBadBuy$credLow.fit
> uppGridBadBuy <-  predObjBadBuy$credUpp.fit
```

We are now ready to make the desired plot:

```
> ylimVal <- range(c(lowGridGoodBuy,uppGridGoodBuy,
+                    lowGridBadBuy,uppGridBadBuy))
> par(mfrow = c(1,1))
> myGoodColour <- "blue" ; myBadColour <- "darkorange"
> plot(0,type = "n",bty = "l",xlim = range(myOdomReadGrid),ylim = ylimVal,
+      xlab = "odometer reading (miles)",ylab = "density")
> abline(h = 0,col = "slateblue")
> lines(myOdomReadGrid,estGridGoodBuy,col = myGoodColour,lwd = 3)
> lines(myOdomReadGrid,lowGridGoodBuy,col = myGoodColour,lwd = 3,lty = 2)
> lines(myOdomReadGrid,uppGridGoodBuy,col = myGoodColour,lwd = 3,lty = 2)
> lines(myOdomReadGrid,estGridBadBuy,col = myBadColour,lwd = 3)
> lines(myOdomReadGrid,lowGridBadBuy,col = myBadColour,lwd = 3,lty = 2)
> lines(myOdomReadGrid,uppGridBadBuy,col = myBadColour,lwd = 3,lty = 2)
> legend("topleft",legend = c("good buy at auction","bad buy at auction"),
+        lty = rep(1,2),lwd = rep(3,2),col = c(myGoodColour,myBadColour))
```

The result is shown in Figure 17 and indicates that the density function of odometer reading values for bad buy cars is shifted to the right compared to that for good buy cars.

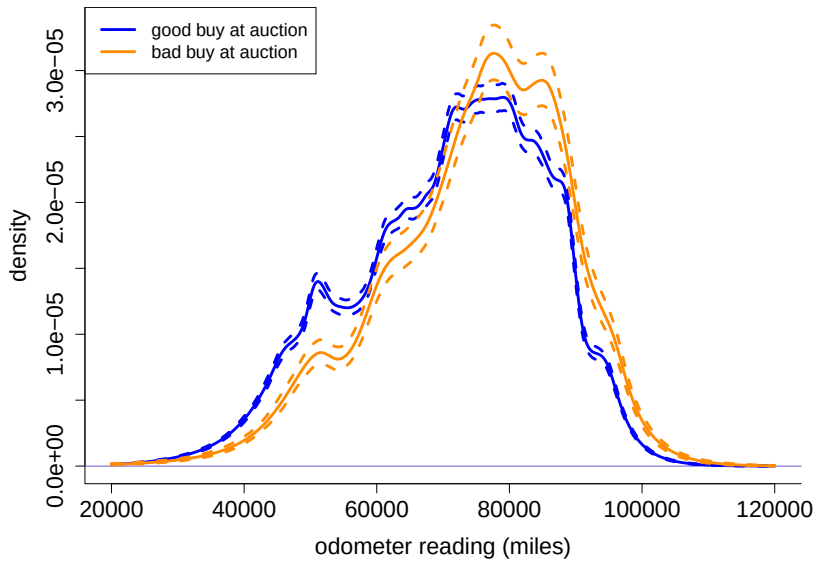

Figure 17: *Bayesian estimates of the density functions of odometer readings of 72,983 cars sold at auctions by automobile dealerships in U.S.A. The data have been divided according to whether or not the car is classified as a good buy (64,007 cars) or a bad buy (8,976 cars). The dashed curves correspond to pointwise 95% credible intervals.*

Acknowledgements

The author gratefully acknowledges assistance from James Yu. This research was supported by Australian Research Council grant DP140100441.

References

Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P. and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, **76**, Issue 1, 1–32.

Harezlak, J., Ruppert, D. and Wand, M.P. (2019). HRW: Datasets, functions and scripts for semiparametric regression supporting Harezlak, Ruppert and Wand (2018). R package version 1.0. https://CRAN.R-project.org/package=HRW

Hoffman, M.D. & Gelman, A. (2014). The no-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, **15**, 1593–1623.

Marron, J. S. and Wand, M.P. (1992). Exact mean integrated squared error. *The Annals of Statistics*, **20**, 712–736.

Neal, R. (2003). Slice sampling (with discussion). *The Annals of Statistics*, **31**, 705–767.

Rohde, D. and Wand, M.P. (2016). Semiparametric mean field variational Bayes: General principles and numerical issues. *Journal of Machine Learning Research*, **17(172)**, 1–47.

Ripley, B. (2020). MASS: Functions and datasets to support Venables and Ripley *Modern Applied Statistics with* S (4th Edition). R package version 7.3. http://www.stats.ox.ac.uk/pub/MASS4

Wand, M.P. and Ormerod, J.T. (2008). On semiparametric regression with O'Sullivan penalized splines. *Australian and New Zealand Journal of Statistics*, **50**, 179–198.

Wand, M.P. and Yu, J.F.C. (2020). Density estimation via Bayesian inference engines. http://arxiv.org/abs/2009.06182