Statistics Surveys
ISSN: 1935-7516

Supplement to "A Review of Uncertainty Quantification for Density Estimation"

Shaun McDonald

Department of Statistics & Actuarial Science
Simon Fraser University
Room SC K10545
8888 University Drive
Burnaby, B.C.
Canada V5A 1S6
e-mail: shaun_mcdonald@sfu.ca

and

David Campbell

School of Mathematics and Statistics
4302 Herzberg Laboratories
Carleton University
1125 Colonel By Drive
Ottawa, ON, K1S 5B6
e-mail: davecampbell@math.carleton.ca

Introduction

This supplement contains the source code for the simulation study described in Section 9 of "A Review of Uncertainty Quantification for Density Estimation". It also contains additional computational and technical details omitted from the main manuscript, as well as some code for further exploration of UQ methods. All analysis was performed with R [12], using a combination of pre-existing packages and our own source code. The latter can be found at https://github.com/ShaunMcDonald1021/density_inference.

As described in the main paper, the data $\boldsymbol{X}$ is a sample of size 1000 from a Gaussian mixture $f_0 = 0.5\mathcal{N}\left(\frac{1}{2}, \frac{1}{49}\right) + 0.5\mathcal{N}\left(\frac{5}{7}, \frac{1}{490}\right)$.

```
set.seed(24601)

mu = c(0.5, 5/7) # Means
sigma = c(1, sqrt(0.1))/7 # Standard deviations
f0 = function(x) 0.5*(dnorm(x, mu[1], sigma[1]) +
                        dnorm(x, mu[2], sigma[2])) # The true density
eval_grid = seq(0, 1, by = 0.002) # Fine grid of evaluation points
n = 1000

mix_comp = sample(1:2, n, replace = TRUE) # Vector of mixture components for samples
X = rnorm(n, mu[mix_comp], sigma[mix_comp])
```

The following sections describe and implement the different UQ methods used for the simulation study, all with a nominal level of $1 - \alpha = 0.95$.

*Supported by an NSERC Alexander Graham Bell Canada Graduate Scholarship.

KDE methods

This section explores frequentist UQ methods based on a standard kernel density estimator. First, we implement the pointwise bias-corrected confidence intervals of Calonico, Cattaneo and Farrell [1], for which we require the `nprobust` R package [2].

```
install.packages('nprobust', repos = "https://cloud.r-project.org")
library(nprobust)
```

In keeping with the theory of Calonico, Cattaneo and Farrell [1], we use the compactly-supported Epanechnikov kernel. Their theory also assumes that the KDE bandwidth is selected to minimize pointwise MSE, *separately at each point*. Here, we instead use a global bandwidth to minimize *integrated* MSE, which ensures smooth estimates. We conjecture that the same "big-O" asymptotics underpinning the theory for pointwise-optimal bandwidths should translate when using the globally optimal one.

```
ccf_est = kdrobust(X, eval = eval_grid, bwselect = 'imse-dpi', bwcheck = 0)
```

Next, we implement the simultaneous bootstrap confidence bands of Cheng and Chen [3]. Note that their theory assumes the true density has compact support, and it and its gradient are equal to zero at the boundaries. Our choice of f_0 does not satisfy these assumptions, but we argue that it is sufficiently flat and low at the endpoints of the interval to be considered "close enough". For consistency with the pointwise inference described above, we continue to use the Epanechnikov kernel. Unfortunately, this means that Cheng and Chen's variable-width bands cannot be implemented, as they involve quantities of the form

$$\left| \frac{f^*(x) - f_0(x)}{\sigma^*(x)} \right|,$$

where f^* is a KDE based on a bootstrap sample and σ^* is the usual sample estimate of its standard deviation. The problem arises from the use of a compactly-supported kernel: $\sigma^*(x)$ can easily be zero for x near the boundaries if there are no points around there in a given bootstrap sample. Thus, we must use Cheng and Chen's fixed-width bands instead.

```
h = ccf_est$Estimate[1,'h'] # Use the same bandwidth from the KDE obtained above
B = 1000 # Number of bootstrap iterations
piv_quants = numeric(B)
for(b in 1:B){
  X_star = sample(X, n, replace = TRUE)
  ccf_est_boot = kdrobust(X_star, eval = eval_grid, h = h, bwcheck = 0)
  piv_quants[b] = max(abs(ccf_est_boot$Estimate[,'tau.bc'] - ccf_est$Estimate[,"tau.bc"]))
  # The pointwise quantiles of these pivotal quantities are used to construct the band
}
```

Bernstein polynomial methods

Next, we try Bayesian UQ methods based on the Bernstein polynomial model of Petrone [9, 10]. All UQ here is based on a run of the MCMC algorithm proposed by Petrone [10], ran for 5000 iterations and discarding the first 1000 as burn-in. Our implementation of the algorithm can be obtained from our Github repository.

```
source(paste("https://raw.githubusercontent.com/ShaunMcDonald1021/",
             "density_inference/main/code/unknown_dim_basis_dens_functions.R", sep = ''))
```

Recall from Section 4.2 of the main paper that a truncated discrete prior must be placed on the dimensionality K; here we use a uniform prior over the integers $\{1, \ldots, 150\}$. We chose $M = 10$ for the concentration parameter of the Dirichlet process prior, which is used to induce a prior on the coefficients of the basis expansion.

```
petrone_est = run_mcmc(list(X), K0 = 4, rate = 1, K_log_prior = numeric(150),
                       basis_type = 'Bernstein', M = 10, S = 5000, x_min = 0, x_max = 1)
```

Pointwise 95% credible intervals are obtained in a straightforward way: by using the pointwise (0.025, 0.975) quantiles of the density draws from the MCMC run. We also implement variable-width simultaneous credible bands based on median absolute deviations (MADs) [4]. Edwards, Meyer and Christensen applied such bands to the spectral density of a time series, but the machinery easily translates to the (probability) density UQ context.

```
post_med = apply(petrone_est$mcmc_chains$f[1001:5000,,], 2, median) # Posterior median
abs_devs = abs(sweep(petrone_est$mcmc_chains$f[1001:5000,6:496,],
                     2, post_med[6:496])) # Absolute deviations
mad = apply(abs_devs, 2, median)
width_quant = quantile(apply(sweep(abs_devs, 2, mad, "/"), 1, max), 0.95)
```

Unfortunately, the simultaneous bands are quite wide relative to the other UQ methods (see below). In fact, here we have only taken the bands over the interval [0.01, 0.99], as taking them over the entire interval [0, 1] renders them too wide to be visually useful. Recall from Section 4.3 of the main paper that these bands (obtained entirely from MCMC output) are of the form

$$\hat{f}(x) \pm \xi_\alpha \text{MAD}[f(x)],$$

where $\hat{f}$ is the posterior median and ξ_α is the $1-\alpha$-quantile of $\sup_x \left(\left| f(x) - \hat{f}(x) \right| / \text{MAD}[f(x)] \right)$. Most density draws f have absolute deviations $\left| f(x) - \hat{f}(x) \right|$ which are quite small near the boundaries, so that the MAD is also quite small there. However, a moderate proportion of the draws have higher tail values, and therefore ξ_α turns out to be fairly large.

Logspline methods

Here we look at a logspline density estimate with the pointwise bootstrapped confidence intervals described by Kooperberg and Stone [6, 7]. For this, we need the `logspline` package [5].

```
install.packages('logspline', repos = "https://cloud.r-project.org")
library(logspline)
```

This package implements a stepwise knot addition/deletion algorithm to fit a logspline density estimate. We use it with most of its default settings except for `maxknots = 20`. This allows for consistency with the bootstrap estimates below, for which the default `maxknots` setting did not always result in convergence.

```
logspline_est = logspline(X, lbound = 0, ubound = 1, maxknots = 20)
```

As described in Kooperberg and Stone [7], confidence intervals for f_0 can be constructed using a Gaussian approximation. First, we obtain an estimate of the standard error of $\log \hat{f}$ using a small number of bootstrap samples (25, in this case).

```
logspline_est_boot_mat = matrix(0, 25, length(eval_grid))

for(b in 1:25){
  X_star = sample(X, n, replace = TRUE)
  logspline_est_boot = logspline(X_star, lbound = 0, ubound = 1, maxknots = 20)
  logspline_est_boot_mat[b,] = log(dlogspline(eval_grid, logspline_est_boot))
}

st_err = apply(logspline_est_boot_mat, 2, sd)
```

With this standard error estimate $\hat{\sigma}$, the pointwise confidence intervals for f_0 are of the form $\exp\left[\log \hat{f}(x) \pm z_{\alpha/2}\hat{\sigma}(x)\right]$, where $z_{\alpha/2}$ is the $\alpha/2$-quantile of the standard Normal distribution.

Dirichlet process mixture methods

The final type of UQ method considered here is based on the Dirichlet process mixture (DPM) model, as implemented in the `dirichletprocess` package [11]. This package uses a marginal sampling algorithm from Neal [8], but full UQ is possible due to the conjugacy of the Dirichlet process, as described in Section 7.1 of the main paper.

```
install.packages('dirichletprocess', repos = "https://cloud.r-project.org")
library(dirichletprocess)
```

The mixture kernel $\kappa(\cdot \mid \theta)$ is taken to be Gaussian with location-scale parameters $\theta = \left(\mu, \sigma^2\right) \sim G$, where G is a Dirichlet process prior with Normal Inverse-Gamma base measure. We use the default choices for the base measure hyperparameters, as well as those for the Gamma prior on the Dirichlet process concentration parameter.

Following the advice given in the `dirichletprocess` package documentation, we linearly transform the sample to have zero mean and unit variance, as this helps with MCMC convergence. We run the sampler for 5000 iterations, discarding the first 1000 as burn-in. Pointwise credible intervals are once again obtained from the pointwise (0.025, 0.975) quantiles of the MCMC density draws.

```
dp = DirichletProcessGaussian((X-mean(X))/sd(X))
dp = Fit(dp, its=5000)

# Obtain posterior density draws. Evaluation points must be linearly transformed
# as the sample was
posteriorFit <- sapply(1001:5000,
                       function(i) PosteriorFunction(dp, i)((eval_grid-mean(X))/sd(X)))
```

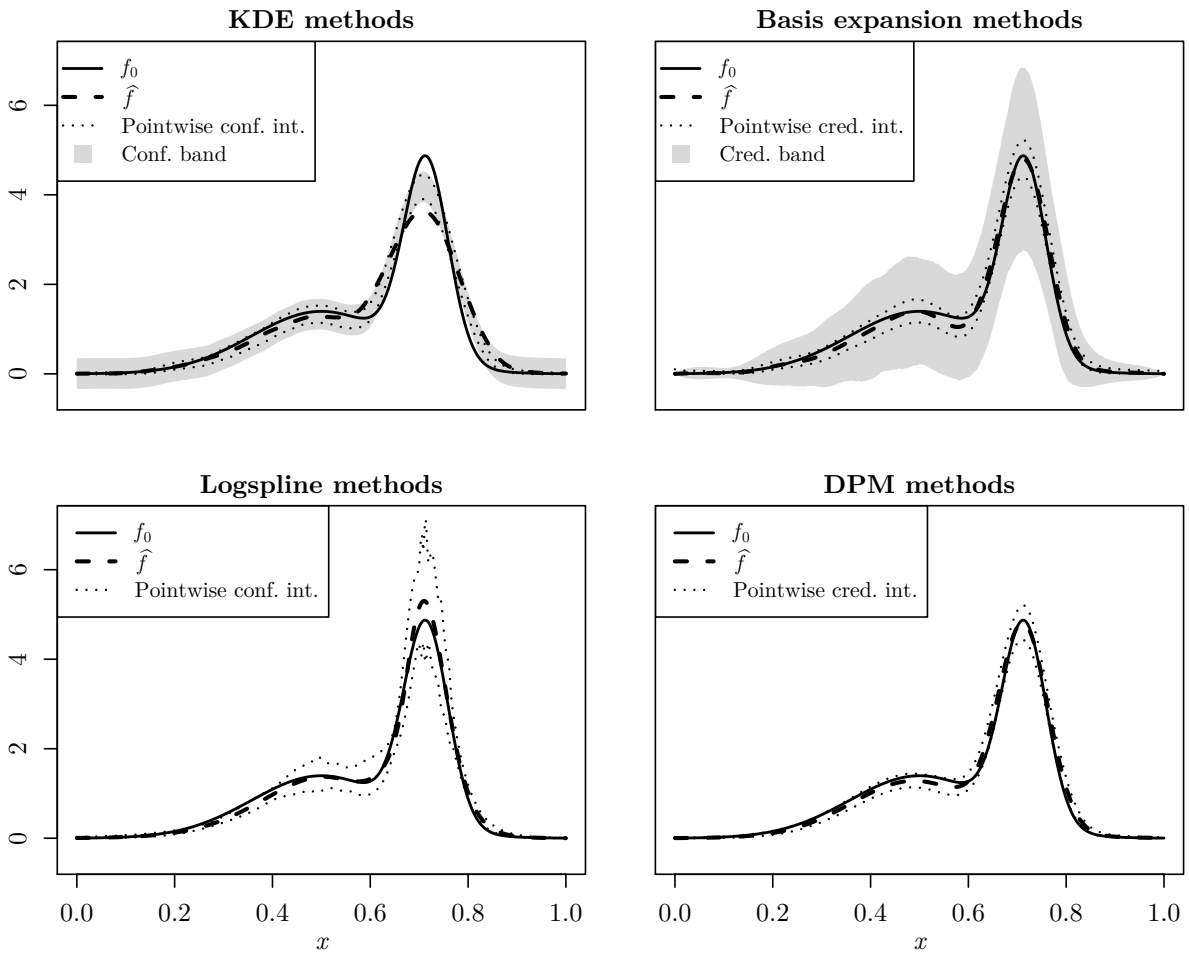
Plotting

Having run all of the models, we are now ready to recreate Figure 1 from the main paper.

```
# Use Latin Modern Roman font if you want it to look fancy like the one below
# Can install it from https://www.fontsquirrel.com/fonts/latin-modern-roman
# install.packages('extrafont', repos = 'https://cloud.r-project.org')
# library(extrafont)
# font_import(pattern = 'lmroman*')
# loadfonts(device = 'win') # Depending on your system/how you want to export the figure
# setEPS()
# postscript(file = 'fig.eps', width = 8, height = 8*11/14, family = 'LM Roman 10')
par(mfrow = c(2,2), mar = c(2.6, 3.1, 2.1, 0.6), cex.axis = 1.25, cex.lab = 1.25,
    cex.main = 1.25, mgp = c(2.1, 1, 0), family = 'LM Roman 10')

## KDE stuff
plot(eval_grid, ccf_est$Estimate[,"tau.us"], type = 'l', ylim = c(-0.5, 7.125), col = NA,
     ylab = '', xaxt = 'n', xlab = '')
title('KDE methods', line = 0.5)

# Plot bias-corrected simultaneous confidence band first so lines can be overlaid on top
polygon(c(eval_grid, rev(eval_grid)),
        c(ccf_est$Estimate[,'tau.bc'] + quantile(piv_quants, 0.95),
          rev(ccf_est$Estimate[,'tau.bc'] - quantile(piv_quants, 0.95))),
```

```
        border = NA, col = rgb(0.85, 0.85, 0.85))

# Plot the KDE itself
lines(eval_grid, ccf_est$Estimate[,"tau.us"], lty = 2, lwd = 2.5)

# Bias-corrected pointwise confidence intervals
lines(eval_grid, ccf_est$Estimate[,"tau.bc"] + qnorm(0.975)*ccf_est$Estimate[,"se.rb"],
      lty = 3, lwd = 1.5)
lines(eval_grid, ccf_est$Estimate[,"tau.bc"] - qnorm(0.975)*ccf_est$Estimate[,"se.rb"],
      lty = 3, lwd = 1.5)

# The true density
lines(eval_grid, f0(eval_grid), lwd = 1.75)

legend('topleft', legend = c(expression(italic(f[0])), expression(italic(hat(f))),
                             "Pointwise conf. int.", "Conf. band"),
       lty = c(1, 2, 3, 0), lwd = c(1.75, 2.5, 1.5, 0), pch = c(NA, NA, NA, 22),
       col = c('black', 'black', 'black', NA),
       pt.bg = c(NA, NA, NA, rgb(0.85, 0.85, 0.85)), pt.cex = 2, y.intersp = 1.1)

## Bernstein polynomial stuff
par(mar = c(2.6, 2.1, 2.1, 1.6))
plot(eval_grid, post_med, type = 'l', ylim = c(-0.5, 7.125), col = NA, ylab = '',
     yaxt = 'n', xaxt = 'n', xlab = '')
title('Basis expansion methods', line = 0.5)

# Simultaneous credible band based on MAD's, but only over the interval [0.01, 0.99]
# Again, plot this first to overlay lines
polygon(c(eval_grid[6:496], rev(eval_grid[6:496])),
        c(post_med[6:496] + width_quant*mad, rev(post_med[6:496] - width_quant*mad)),
        border = NA, col = rgb(0.85, 0.85, 0.85))

# Plot the posterior median
lines(eval_grid, post_med, lty = 2, lwd = 2.5)

# Pointwise credible intervals
lines(eval_grid, apply(petrone_est$mcmc_chains$f[1001:5000,,], 2, quantile, 0.025),
      lty = 3, lwd = 1.5)
lines(eval_grid, apply(petrone_est$mcmc_chains$f[1001:5000,,], 2, quantile, 0.975),
      lty = 3, lwd = 1.5)

# The true density
lines(eval_grid, f0(eval_grid), lwd = 1.75)

legend('topleft', legend = c(expression(italic(f[0])), expression(italic(hat(f))),
                             "Pointwise cred. int.", "Cred. band"),
       lty = c(1, 2, 3, 0), lwd = c(1.75, 2.5, 1.5, 0), pch = c(NA,NA,NA,22),
       col = c('black', 'black', 'black', NA),
       pt.bg = c(NA, NA, NA, rgb(0.85, 0.85, 0.85)), pt.cex = 2, y.intersp = 1.1)

## Logspline stuff
par(mar = c(3.6, 3.1, 1.1, 0.6))
# Plot the logspline estimate
```

```
plot(eval_grid, dlogspline(eval_grid, logspline_est), type = 'l', ylim = c(-0.5, 7.125),
     lty = 2, lwd = 2.5, xlab = expression(italic(x)), ylab = '')
title('Logspline methods', line = 0.5, xpd = NA)

# Exponentiated pointwise confidence intervals from bootstrap std. err. estimates
lines(eval_grid, exp(log(dlogspline(eval_grid, logspline_est)) + qnorm(0.975)*st_err),
      lty = 3, lwd = 1.5)
lines(eval_grid, exp(log(dlogspline(eval_grid, logspline_est)) - qnorm(0.975)*st_err),
      lty = 3, lwd = 1.5)

# The true density
lines(eval_grid, f0(eval_grid), lwd = 1.75)

legend('topleft', legend = c(expression(italic(f[0])), expression(italic(hat(f))),
                             "Pointwise conf. int."),
       lty = c(1, 2, 3), lwd = c(1.75, 2.5, 1.5), col = c('black','black','black'),
       pt.cex = 2, y.intersp = 1.1)

## Dirichlet process mixture stuff
par(mar = c(3.6, 2.1, 1.1, 1.6))
# Plot posterior mean. Note that it must be scaled since data was linearly transformed
plot(eval_grid, apply(posteriorFit, 1, mean)/sd(X), type = 'l', ylim = c(-0.5, 7.125),
     lty = 2, lwd = 2.5, xlab = expression(italic(x)), ylab = '', yaxt = 'n')
title('DPM methods', line = 0.5, xpd = NA)

# Pointwise credible intervals
lines(eval_grid, apply(posteriorFit, 1, quantile, 0.975)/sd(X), lty = 3, lwd = 1.5)
lines(eval_grid, apply(posteriorFit, 1, quantile, 0.025)/sd(X), lty = 3, lwd = 1.5)

# The true density
lines(eval_grid, f0(eval_grid), lwd = 1.75)

legend('topleft', legend = c(expression(italic(f[0])), expression(italic(hat(f))),
                             "Pointwise cred. int."),
       lty = c(1, 2, 3), lwd = c(1.75, 2.5, 1.5), col = c('black','black','black'),
       pt.cex = 2, y.intersp = 1.1)

# dev.off()
```



Further assessment of uncertainty sets

It may be of interest to compare the methods explored here in terms of their performance under repeated sampling: namely, in terms of coverage probability and interval/band diameter. To that end, one may also wish to see how their performance changes as the sample size n is varied. We alluded to some informal experiments to this effect in Section 9 of the main paper, but did not undertake a full analysis of these properties. We caution that such an analysis will take a long time in computational terms, since it requires each of the above UQ methods (in particular, the Bayesian ones based on lengthy MCMC chains) to be run repeatedly. The computational burden increases with larger sample sizes, as one may expect. Nevertheless, below we provide code for researchers who wish to investigate these properties. Further analysis and modification of this code can be done easily in R.

```
UQ_assess = function(n, R = 200, p = 0.95){
  # A function to assess coverage probability for the above UQ methods,
  # using samples from the same true density considered above.
  # n = sample size
  # R = repetitions (i.e. take r samples of size n and construct UQ sets for each one)
  # p = nominal confidence/credibility level

  # Outputs two arrays:
  # coverage_prob: length(eval_grid)x6 array,
  # containing pointwise coverage probabilities for all six UQ methods
```

```
# set_lengths: length(eval_grid)xRx6 array, containing pointwise diameters
# (upper bound - lower bound) for all UQ methods in each replication
# (so one can, e.g., look at mean & variance of UQ set diameters for each method)

# Setup
mu = c(0.5, 5/7)
sigma = c(1, sqrt(0.1))/7
f0 = function(x) 0.5*(dnorm(x, mu[1], sigma[1]) + dnorm(x, mu[2], sigma[2]))
eval_grid = seq(0, 1, by = 0.002)
true_vals = f0(eval_grid)


coverage_indicators = array(0, dim = c(length(eval_grid), R, 6))
set_lengths = array(0, dim = c(length(eval_grid), R, 6))

for(r in 1:R){
  # Get sample
  mix_comp = sample(1:2, n, replace = TRUE)
  X = rnorm(n, mu[mix_comp], sigma[mix_comp])

  ## KDE stuff
  ccf_est = kdrobust(X, eval = eval_grid, bwselect = 'imse-dpi', bwcheck = 0)
  # Pointwise intervals
  coverage_indicators[,r,1] =
    (true_vals < (ccf_est$Estimate[,"tau.bc"] +
                    qnorm((1+p)/2)*ccf_est$Estimate[,"se.rb"])) &
    (true_vals > (ccf_est$Estimate[,"tau.bc"] -
                    qnorm((1+p)/2)*ccf_est$Estimate[,"se.rb"]))
  set_lengths[,r,1] = 2*qnorm((1+p)/2)*ccf_est$Estimate[,"se.rb"]

  # Bootstrapped bands
  h = ccf_est$Estimate[1,'h']
  B = 1000
  piv_quants = numeric(B)
  for(b in 1:B){
    X_star = sample(X, n, replace = TRUE)
    ccf_est_boot = kdrobust(X_star, eval = eval_grid, h = h, bwcheck = 0)
    piv_quants[b] = max(abs(ccf_est_boot$Estimate[,'tau.bc'] -
                              ccf_est$Estimate[,"tau.bc"]))
  }
  coverage_indicators[,r,2] =
    (true_vals < ccf_est$Estimate[,'tau.bc'] +
                                quantile(piv_quants, p)) &
    (true_vals > ccf_est$Estimate[,'tau.bc'] -
       quantile(piv_quants, p))
  set_lengths[,r,2] = 2*quantile(piv_quants, p)

  ## Bernstein polynomial stuff
  petrone_est = run_mcmc(list(X), K0 = 4, rate = 1, K_log_prior = numeric(150),
                    basis_type = 'Bernstein', M = 10, S = 5000, x_min = 0, x_max = 1)

  # Pointwise intervals
  coverage_indicators[,r,3] =
```

```
      (true_vals > apply(petrone_est$mcmc_chains$f[1001:5000,,],
                                                  2, quantile, (1-p)/2)) &
      (true_vals < apply(petrone_est$mcmc_chains$f[1001:5000,,],
                         2, quantile, (1+p)/2))
    set_lengths[,r,3] =
      apply(petrone_est$mcmc_chains$f[1001:5000,,], 2, quantile, (1+p)/2) -
      apply(petrone_est$mcmc_chains$f[1001:5000,,],2, quantile, (1-p)/2)

    # Simultaneous bands
    post_med = apply(petrone_est$mcmc_chains$f[1001:5000,,], 2, median)
    abs_devs = abs(sweep(petrone_est$mcmc_chains$f[1001:5000,6:496,], 2, post_med[6:496]))
    mad = apply(abs_devs, 2, median)
    width_quant = quantile(apply(sweep(abs_devs, 2, mad, "/"), 1, max), p)
    coverage_indicators[6:496,r,4] =
      (true_vals[6:496] < post_med[6:496] + width_quant*mad) &
      (true_vals[6:496] > post_med[6:496] - width_quant*mad)
    set_lengths[6:496,r,4] = 2*width_quant*mad

    ## Logspline stuff
    logspline_est = logspline(X, lbound = 0, ubound = 1, maxknots = 20)

    logspline_est_boot_mat = matrix(0, 25, length(eval_grid))
    for(b in 1:25){
      X_star = sample(X, n, replace = TRUE)
      logspline_est_boot = logspline(X_star, lbound = 0, ubound = 1, maxknots = 20)
      logspline_est_boot_mat[b,] = log(dlogspline(eval_grid, logspline_est_boot))
    }
    st_err = apply(logspline_est_boot_mat, 2, sd)

    coverage_indicators[,r,5] =
      (true_vals < exp(log(dlogspline(eval_grid, logspline_est))
                       + qnorm((1+p)/2)*st_err)) &
      (true_vals > exp(log(dlogspline(eval_grid, logspline_est)) + qnorm((1-p)/2)*st_err))

    set_lengths[,r,5] =
      exp(log(dlogspline(eval_grid, logspline_est)) + qnorm((1+p)/2)*st_err) -
      exp(log(dlogspline(eval_grid, logspline_est)) + qnorm((1-p)/2)*st_err)

    ## DPM stuff
    dp = DirichletProcessGaussian((X-mean(X))/sd(X))
    dp = Fit(dp, its=5000)
    posteriorFit = sapply(1001:5000,
                          function(i) PosteriorFunction(dp, i)((eval_grid-mean(X))/sd(X)))

    coverage_indicators[,r,6] =
      (true_vals < apply(posteriorFit, 1, quantile, (1+p)/2)/sd(X)) &
      (true_vals > apply(posteriorFit, 1, quantile, (1-p)/2)/sd(X))
    set_lengths[,r,6] = apply(posteriorFit, 1, quantile,(1+p)/2)/sd(X) -
      apply(posteriorFit, 1, quantile, (1-p)/2)/sd(X)
  }
  coverage_probs = apply(coverage_indicators, c(1,3), mean)
  return(list(coverage_probs = coverage_probs, set_lengths = set_lengths))
}
```

References

[1] Calonico, S., Cattaneo, M. D. and Farrell, M. H. (2018). On the Effect of Bias Estimation on Coverage Accuracy in Nonparametric Inference. *Journal of the American Statistical Association* **113** 767–779.

[2] Calonico, S., Cattaneo, M. D. and Farrell, M. H. (2019). nprobust: Nonparametric Kernel-Based Estimation and Robust Bias-Corrected Inference. *Journal of Statistical Software* **91** 1–33.

[3] Cheng, G. and Chen, Y.-C. (2019). Nonparametric inference via bootstrapping the debiased estimator. *Electronic Journal of Statistics* **13** 2194–2256.

[4] Edwards, M. C., Meyer, R. and Christensen, N. (2019). Bayesian nonparametric spectral density estimation using B-spline priors. *Statistics and Computing* **29** 67–78.

[5] Kooperberg, C. (2020). logspline: Routines for Logspline Density Estimation R package version 2.1.16.

[6] Kooperberg, C. and Stone, C. J. (2003). Confidence Intervals for Logspline Density Estimation. 285–295. Springer, New York, NY.

[7] Kooperberg, C. and Stone, C. J. (2004). Comparison of Parametric and Bootstrap Approaches to Obtaining Confidence Intervals for Logspline Density Estimation. *Journal of Computational and Graphical Statistics* **13** 106–122.

[8] Neal, R. M. (2000). Markov Chain Sampling Methods for Dirichlet Process Mixture Models Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal of Computational and Graphical Statistics* **9** 249–265.

[9] Petrone, S. (1999a). Random Bernstein Polynomials. *Scandinavian Journal of Statistics* **26** 373–393.

[10] Petrone, S. (1999b). Bayesian density estimation using Bernstein polynomials. *Canadian Journal of Statistics* **27** 105–126.

[11] J. Ross, G. and Markwick, D. (2020). dirichletprocess: Build Dirichlet Process Objects for Bayesian Modelling R package version 0.4.0.

[12] R Core Team (2020). R: A Language and Environment for Statistical Computing R Foundation for Statistical Computing, Vienna, Austria.