Analytic and Bootstrap Approaches to Local Linear Regression

Mini-project Epiphany term

Raul Unnithan, Supervisor: Dr John Einbeck

2025-04-20

Introduction

Motivation

In non-parametric regression, the goal is to estimate an unknown regression function $m(x) = \mathbb{E}[Y \mid X = x]$ without pre-specifying a functional form. Among non-parametric methods, there are local polynomial estimators which approximate the regression function by fitting low-degree polynomials to subsets of the data, weighted by a kernel. This report looked at the local linear (LL) estimator, which balances bias and variance well and remains reliable near boundary regions.

The data examined in this report recorded the association between planting density and yield in the production of white Spanish onions at two locations in South Australia. The aim was to estimate the regression function linking onion density to expected yield and to quantify the uncertainty of this estimate using pointwise confidence intervals (CIs).

Two methods were implemented. The first was an analytical approach based on asymptotic approximations, with plug-in estimates for bias and variance derived from a quartic global polynomial fit and kernel-based density and variance estimates. The second was a simulation-based method, where CIs were constructed empirically by repeatedly resampling the observed data and re-estimating the regression function using the non-parametric paired bootstrap.

These two approaches were compared in terms of visual behaviour, smoothness, and average interval width. The results illustrated the trade-off between the efficiency and assumptions of the analytical method and the flexibility of the bootstrap alternative.

Exploratory Data Analysis

First, the necessary packages that were needed throughout the project were installed and loaded up .

```
library(KernSmooth)
```

```
## Warning: package 'KernSmooth' was built under R version 4.4.3
```

```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

```r
library(locpol)
library(SemiPar)
```

```
## Warning: package 'SemiPar' was built under R version 4.4.3
```

Next, the onions dataset was introduced and some initial exploratory data analysis was performed.

```r
data(onions)
str(onions)
```

```
## 'data.frame':    84 obs. of  3 variables:
##  $ dens    : num  23.5 26.2 27.8 32.9 33.3 ...
##  $ yield   : num  223 234 222 222 197 ...
##  $ location: int  0 0 0 0 0 0 0 0 0 0 ...
```

The three variables of the onions dataset were: density, yield and location. These refer to the areal density of plants (plants per square metre), the onion yield (in grams per plant) and the indicator of location: 0=Purnong Landing, 1=Virginia. However, the location variable was removed because it is not a continuous predictor. Also, we were only concerned with looking at the relationship between onion density and onion yield.

```r
onions.data <- onions[, -3] # remove location
```

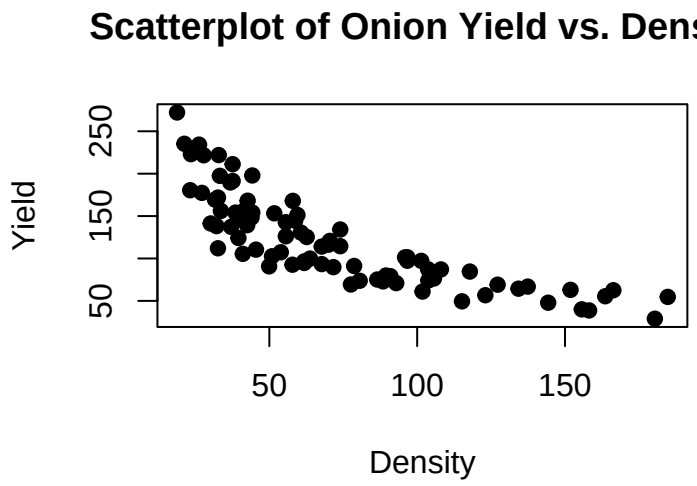
Missing data was then checked to confirm if any data needed to be imputed or those relevant rows removed. However, the `any()` function confirmed there was no missing data.

```r
any(is.na(onions.data))
```

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

Next, a plot of `onions.data` was used to examine the relationship between onion density and yield.

```r
plot(onions.data$dens, onions.data$yield, pch = 19,xlab = "Density",
     ylab = "Yield",main = "Scatterplot of Onion Yield vs. Density")
```



The above plot can be explained as follows: as planting density increases, each onion plant has access to less space, sunlight, and soil nutrients. This increased competition reduces the resources available to each plant, resulting in a lower yield per plant. This biological trade-off explains the observed negative correlation between onion yield and density.

Methodology and Code Decisions

Methodology

The formula for the LL estimator is given by:

$$\hat{m}_{LL}(x) = \frac{\sum_{i=1}^{n} v_i(x) y_i}{\sum_{i=1}^{n} v_i(x)}, \tag{1}$$

where

$$v_i(x) = w_i(x)\left(S_{n,2} - (x_i - x)S_{n,1}\right), \qquad S_{n,j} = \sum_{i=1}^{n} w_i(x)(x_i - x)^j, \quad j = 0, 1, 2, \ldots. \tag{2}$$

Here, $w_i(x) = K_h(x_i - x)$ denotes the kernel weight applied to each observation, $S_{n,j}$ are the local moment sums, and $v_i(x)$ are the weights used to estimate the intercept of a locally weighted linear fit at point x.

The asymptotic bias and variance of the LL estimator $\hat{m}_{LL}(x)$ are given by:

$$\text{Bias}[\hat{m}_{LL}(x)] = \frac{1}{2}h^2 m''(x)\mu_2 + o(h^2), \quad \text{Var}[\hat{m}_{LL}(x)] = \frac{\sigma^2}{nh} \cdot \frac{\nu_0}{f(x)} + o\left(\frac{1}{nh}\right), \tag{3}$$

where $m(x) = \mathbb{E}[Y \mid X = x]$ is the true regression function that we aim to estimate, and $m''(x)$ is its second derivative, which captures the local curvature of the function at point x. The bandwidth is denoted by h, and $f(x)$ is the probability density function of the predictor variable X evaluated at point x.

The constants $\mu_2 = \int u^2 K(u)\, du$ and $\nu_0 = \int K^2(u)\, du$ are determined by the choice of kernel K, and represent the second moment of the kernel and the zeroth moment of the squared kernel, respectively. $\sigma^2 = \text{Var}[\varepsilon \mid X = x]$ denotes the conditional variance of the error term, and n is the sample size.

We had now set up a good platform to starting building our CIs. However, some of the parameters could not be derived using exact values and instead they needed plug-in estimates.

There exists a method to calculate these plug-in estimates using the "Rule-of-thumb" bandwidth selector for the LL case. Under this approach $\check{m}(x)$ is estimated globally using a polynomial of degree 4, that is:

$$\check{m}(x) = \check{\alpha}_0 + \check{\alpha}_1 x + \check{\alpha}_2 x^2 + \check{\alpha}_3 x^3 + \check{\alpha}_4 x^4. \tag{4}$$

This meant we could estimate $m''(x)$ using the second derivative of the fitted polynomial as follows:

$$\check{m}''(x) = 2\check{\alpha}_2 + 6\check{\alpha}_3 x + 12\check{\alpha}_4 x^2. \tag{5}$$

This method also estimates the constant error standard deviation σ from the fitted model's residuals using:

$$\check{\sigma}^2 = \frac{1}{n-5}\sum_{i=1}^{n}\left(y_i - \check{m}(x_i)\right)^2 \tag{6}$$

Finally, all the above can be combined to give an estimate of the "rule-of-thumb" bandwidth, h_{ROT}:

$$h_{\text{ROT}} = \left[\frac{\nu_0\, \check{\sigma}^2}{\mu_2^2 \sum_{i=1}^{n} \check{m}''(x_i)^2}\right]^{1/5} n^{-1/5} \tag{7}$$

The equations in this subsection were used to derive the analytical and simulated CIs using the exact values and plug-in estimates for parameters in the above equations.

Helper Functions

To implement the LL regression interval approaches, we first defined several helper functions.

```
my.kernel<-function(u){return(dnorm(u))}

Sn <- function(xdat, x, h, j){sum(my.kernel((xdat - x)/h)*(xdat - x)^j)}

vix <- function(xdat, x, h){
  my.kernel((xdat - x)/h)*(Sn(xdat, x, h, 2) - (xdat - x)*Sn(xdat, x, h, 1))}

my.ll.smoother <- function(xdat, ydat, xgrid = xdat, h){
  G <- length(xgrid)
  est <- rep(0, G)
  for (j in 1:G){
    est[j] <- sum(ydat*vix(xdat, xgrid[j], h))/sum(vix(xdat, xgrid[j], h))}
  return(est)}
```

The function `my.kernel` returns the kernel weights used for local smoothing. Here a Gaussian kernel was specified. This did not matter because it was overridden in the functions for the analytical and simulated approaches. The `my.kernel()` function implements $K(u)$ in the formula for μ_2. See Equation 3.

The function `Sn()` calculates the required moments for local polynomial regression and function `vix()` computes the weights used in the LL estimation process. These implement the Equations labelled 2.

The function `my.ll.smoother()` then brings `my.kernel()`, `Sn()` and `vix()` together to perform the LL estimates over the specified grid of evaluation points. This implements Equation 1.

```
m <- function(coefs, x) {coefs[1] + coefs[2]*x + coefs[3]*x^2 + coefs[4]*x^3 + coefs[5]*x^4}

m2 <- function(coefs, x) {2 * coefs[3] + 6 * coefs[4]*x + 12 * coefs[5]*x^2}

bias <- function(h, m2, mu2) {(1/2) * (h^2) * m2 * mu2}

sd <- function(n, h, sigma, nu0, f) {
  variance <- ((1 / (n * h)) * (sigma^2) * (nu0 / f))
  sqrt(variance)}
```

The functions `m()` and `m2()` evaluate the fitted polynomial and its second derivative, respectively, and the latter was used in the estimation of bias. These functions implement Equations 4 and 5 respectively.

The functions `bias()` and `sd()` compute the analytical bias and standard deviation required for constructing the bias-corrected CIs. These derive the bias and variance in Equation 3. The square root was taken because these intervals are constructed using the standard deviations rather than the variance.

```
est_density <- function(data, xgrid, h, kernel = "gauss") {
  my.kernel <- switch(kernel,
                      "gauss" = function(x) dnorm(x),
                      "epa"   = function(x) { 3/4 * (1 - x^2) *
                          ifelse(abs(x) <= 1, 1, 0) },
                      "uni"   = function(x) { 0.5 * ifelse(abs(x) <= 1, 1, 0) })
```

```
  data <- na.omit(data)
  n <- length(data)
  est <- numeric(length(xgrid))
  denom <- n * h
  for (j in seq_along(xgrid)) {
    est[j] <- sum(my.kernel((data - xgrid[j]) / h)) / denom}
  return(est)}
```

Lastly, the function `est_density()` provides an estimate of the density of the predictor variable, $f(x)$, using the necessary kernel method. This was necessary in the calculation of variance of the LL estimator. See Equation 3.

Analytical Approach

Here is the function for the analytical approach to the LL estimator.

```
ana.ll <- function(data, xgrid, h = 1, kernel = "gauss", alpha = 0.05) {
  # Kernel constants
  nu0 <- switch(kernel,"gauss" = 1 / (2 * sqrt(pi)),"epa" = 3 / 5,"uni" = 1 / 2)
  mu2 <- switch(kernel,"gauss" = 1,"epa" = 1 / 5,"uni" = 1 / 3)

  n <- nrow(data)
  x <- data[, 1]
  y <- data[, 2]

  model <- lm(y ~ poly(x, 4, raw = TRUE))   # Fit degree 4 polynomial for derivatives
  coefs <- coef(model)
  mhat <- my.ll.smoother(xdat = x, ydat = y, xgrid = xgrid, h = h)

  mddash <- m2(coefs, xgrid)
  sigma2 <- sum((y - m(coefs, x))^2) / (n - 5)
  f <- est_density(x, xgrid, h, kernel)

  bias_val <- bias(h, mddash, mu2)
  sd_val <- sd(n, h, sqrt(sigma2), nu0, f)

  est <- mhat - bias_val # Bias-corrected estimate
  z <- qnorm(1-alpha/2)
  lower_ci <- est - z * sd_val # Compute CI: (estimate - bias) - 2 * sd
  upper_ci <- est + z * sd_val # Compute CI: (estimate - bias) + 2 * sd

  # Sort by xgrid to ensure lines() function plots properly
  ord <- order(xgrid)
  xgrid_sorted <- xgrid[ord]
  est_sorted <- est[ord]
  lower_ci_sorted <- lower_ci[ord]
  upper_ci_sorted <- upper_ci[ord]
```

```
  return(list(xgrid = xgrid_sorted, est = est_sorted,
              lower = lower_ci_sorted, upper = upper_ci_sorted))}
```

The `ana.ll()` function was constructed to implement the analytical pointwise CI for the LL estimator.

The function began by assigning the constants ν_0 and μ_2 based on the designated choice of kernel. These values are kernel-dependent and were used in the variance and bias calculations, respectively. The `switch()` statement allowed for switching between the Gaussian, Epanechnikov, and Uniform kernels.

The LL estimate $\hat{m}_{LL}(x)$ itself was then computed using a separate smoothing function `my.ll.smoother()`, as defined earlier.

In line with the plug-in methodology described above, a global polynomial of degree four was fitted to the data. The coefficients from this fit were used to compute plug-in estimates for the second derivative $m''(x)$, as described in Equation 5, and for the residual variance σ^2, as in Equation 6. These estimates were required to plug into the bias and variance expressions in Equation 3.

A kernel-based estimate of the density $f(x)$ was then obtained using a custom function `est_density`, which was evaluated at each point in the grid. This allowed the pointwise variance estimates to incorporate local variation in the density of the predictor variable.

The final bias-corrected CIs were then constructed using the form:

$$\hat{m}_{LL}(x) - \text{Bias} \pm 2 \cdot \text{SD},$$

where the constant 2 served as an approximate 95% normal quantile. The function concluded by sorting the evaluation grid and outputting the smoothed estimate along with the upper and lower confidence bounds, formatted for direct plotting using the `lines()` function.

Simulated Approach

A similar approach was used to build the simulated LL estimator but now this integrated bootstrap.

```
sim.ll <- function(data, xgrid, h, kernel = "gauss", alpha = 0.05, B = 300) {
  nu0 <- switch(kernel,"gauss" = 1 / (2 * sqrt(pi)),"epa" = 3 / 5,"uni" = 1 / 2)
  mu2 <- switch(kernel,"gauss" = 1,"epa"   = 1 / 5,"uni" = 1 / 3)

  n <- nrow(data)
  x <- data[, 1]
  y <- data[, 2]

  # Initialise matrices for bootstrap estimates, bias, and sd over xgrid
  bootMat <- matrix(NA, nrow = B, ncol = length(xgrid))
  biasMat <- matrix(NA, nrow = B, ncol = length(xgrid))
  sdMat <- matrix(NA, nrow = B, ncol = length(xgrid))

  # Non-parametric paired bootstrap
  for (b in seq_len(B)) {
    idx <- sample(seq_len(n), size = n, replace = TRUE)
    x_boot <- x[idx]
    y_boot <- y[idx]
    model_boot <- lm(y_boot ~ poly(x_boot, 4, raw = TRUE))
```

```
    coefs_boot <- coef(model_boot)
    mhat_boot <- my.ll.smoother(xdat = x_boot, ydat = y_boot, xgrid = xgrid, h = h)
    mddash_boot <- m2(coefs_boot, xgrid)
    sigma2_boot <- sum((y_boot - m(coefs_boot, x_boot))^2) / (n - 5)
    f_boot <- est_density(x_boot, xgrid, h, kernel)

    # Compute bias and sd for this bootstrap sample
    bias_val <- bias(h, mddash_boot, mu2)
    sd_val <- sd(n, h, sqrt(sigma2_boot), nu0, f_boot)

    # Store the bias-corrected estimate and sd
    bootMat[b, ] <- mhat_boot - bias_val
    sdMat[b, ] <- sd_val}

  # Compute means from bootstrap
  est_mean <- apply(bootMat, 2, mean)
  sd_mean <- apply(sdMat, 2, mean)

  z <- qnorm(1-alpha/2)
  lower_ci <- est_mean - z * sd_mean # Compute CI: (estimate - bias) +/- 2 * sd
  upper_ci <- est_mean + z * sd_mean # Compute CI: (estimate - bias) +/- 2 * sd

  ord <- order(xgrid)
  xgrid_sorted <- xgrid[ord]
  est_sorted <- est_mean[ord]
  lower_ci_sorted <- lower_ci[ord]
  upper_ci_sorted <- upper_ci[ord]

  return(list(xgrid = xgrid_sorted, est = est_sorted,
              lower = lower_ci_sorted, upper = upper_ci_sorted))}
```

The `sim.ll()` function was designed to construct pointwise CIs for the LL estimator using a non-parametric paired bootstrap approach.

As in the analytical version, the kernel-specific constants ν_0 and μ_2 were defined based on the chosen kernel. These were used to estimate the plug-in variance and bias within each bootstrap iteration.

The data were first prepared by extracting the predictor and response variables from the two-column matrix. Three matrices were then initialised to store the bias-corrected estimates, bias values, and standard deviation estimates from each of the B bootstrap replicates.

The function then used a non-parametric paired bootstrap, whereby entire (x_i, y_i) pairs were resampled with replacement. For each bootstrap sample, a global polynomial of degree four was fitted to estimate the second derivative $m''(x)$ and residual variance σ^2, as in Equations 5 and 6. The smoothed estimate $\hat{m}_{LL}(x)$ was computed using `my.ll.smoother`, and the kernel density $f(x)$ was re-estimated on the resampled predictor values.

For each replicate, the bias and variance were computed using the plug-in formulas from Equation 3. The resulting bias-corrected estimates and standard deviations were stored in their respective matrices.

Once all bootstrap samples were completed, the function computed the average of the bias-corrected estimates and the average standard deviation across replicates using R's in-built `apply()` function. These

were used to construct the final CIs in the form:

$$\hat{m}_{LL}(x) \pm 2 \cdot \text{SD},$$

where the constant 2 again served as an approximate 95% normal quantile. Note that the bias was implicitly handled via the plug-in correction inside each bootstrap replicate, rather than being explicitly applied to the final mean curve.

Finally, the function sorted the evaluation grid and returned the smoothed estimates along with the corresponding lower and upper confidence limits, enabling straightforward plotting and comparison with the analytical approach.

Analysis

Graphical Comparison

LL regression was implemented here using a Gaussian kernel due to its versatility. Next, the analytical and simulated CIs were coded using these helper functions.

R has an in-built function,`thumbBw()`, from the R package `locpol` that implements Equation 7 and here is how it applies on the `onions.data`:

```
xgrid <- seq(15, 185, by = 0.5)

# Define the bandwidth using the thumb rule
h_rot <- thumbBw(onions.data$dens, onions.data$yield, deg = 1, kernel = gaussK)
h_rot
```

```
## [1] 3.293061
```

This procedure can also be implemented as follows manually and this is what will be used for the plots.

```
thumbBw_manual <- function(x, y, kernel = "gauss") {
  n <- length(x)
  nu0 <- switch(kernel, "gauss" = 1 / (2 * sqrt(pi)), "epa" = 3 / 5, "uni" = 1 / 2)
  mu2 <- switch(kernel, "gauss" = 1, "epa"   = 1 / 5, "uni" = 1 / 3)

  pilot_fit <- lm(y ~ poly(x, degree = 4, raw = TRUE))
  coefs <- coef(pilot_fit)

  sigma2 <- sum((y - m(coefs, x))^2) / (n - 5)
  sum_m_double_sq <- sum(m2(coefs, x)^2)

  numerator <- nu0 * sigma2
  denominator <- mu2^2 * sum_m_double_sq
  h_rot <- (numerator / denominator)^(1/5) * n^(-1/5)
  return(h_rot)}

h_rot_manual <- thumbBw_manual(onions.data$dens, onions.data$yield, kernel = "gauss")
h_rot_manual
```

```
## [1] 1.374277
```

These values were slightly different because

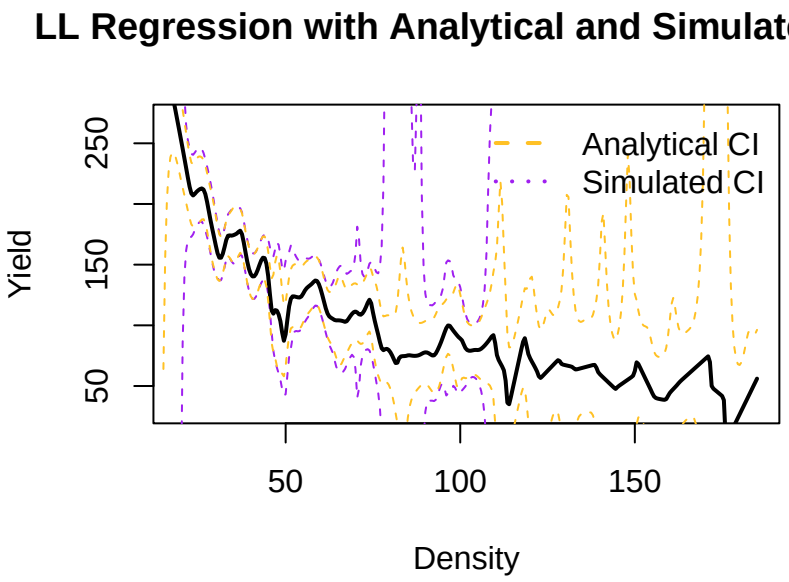
Before applying this, a seed was set to ensure reproducible results.

```
summary(onions.data)
```

```
##       dens            yield
##  Min.   : 18.78   Min.   : 28.96
##  1st Qu.: 39.54   1st Qu.: 78.46
##  Median : 61.78   Median :108.90
##  Mean   : 73.33   Mean   :119.70
##  3rd Qu.: 97.86   3rd Qu.:153.47
##  Max.   :184.75   Max.   :272.15
```

The summary was then extracted because it was useful in selecting the appropriate `x_grid` values when we derived the CIs.

```
# set.seed(123)
res_analytic <- ana.ll(onions.data, xgrid = xgrid, h = h_rot_manual,
                               kernel = "gauss", alpha = 0.05)
res_bootstrap <- sim.ll(onions.data, xgrid = xgrid, h = h_rot_manual,
                        kernel = "gauss", alpha = 0.05, B = 300)

plot(onions.data$dens, onions.data$yield, type = "n", pch = 19,xlab = "Density",
     ylab = "Yield", main = "LL Regression with Analytical and Simulated CIs")

# Simulated Approach
# lines(res_bootstrap$xgrid, res_bootstrap$est, col = "purple", lwd = 2)
lines(res_bootstrap$xgrid, res_bootstrap$lower, col = "purple", lty = 2)
lines(res_bootstrap$xgrid, res_bootstrap$upper, col = "purple", lty = 2)

# Analytical Approach
lines(res_analytic$xgrid, res_analytic$est, col = "black", lwd = 2)
lines(res_analytic$xgrid, res_analytic$lower, col = "goldenrod1", lty = 2)
lines(res_analytic$xgrid, res_analytic$upper, col = "goldenrod1", lty = 2)

legend("topright", legend = c("Analytical CI", "Simulated CI"),
       col = c("goldenrod1", "purple"), lwd = 2, lty = c(2, 3), bty = "n")
```



From the above plot, the analytical intervals appear narrower and more stable, while the simulated intervals fluctuate more and widen substantially in certain regions. These patterns reflect the trade-off between theoretical precision and empirical flexibility inherent in the two methods.

Interval Width Comparison

Here is a summary table comparing the average widths of CIs:

```
width_analytic <- mean(res_analytic$upper - res_analytic$lower)
width_bootstrap <- mean(res_bootstrap$upper - res_bootstrap$lower)

interval_widths <- data.frame(Method = c("Analytical Approach", "Simulated Approach"),
                              Average_Width = c(width_analytic, width_bootstrap))
interval_widths
```

```
##                Method Average_Width
## 1 Analytical Approach  4.325100e+02
## 2  Simulated Approach  2.306892e+46
```

This table furthers the insights from the above plot. It indicates a higher degree of uncertainty captured by the simulation-based method, potentially due to variability in resampled estimates of bias and variance. The analytical intervals, by contrast, were narrower and more stable, benefiting from smooth plug-in estimates and asymptotic approximations.

Conclusion

Both approaches gave intervals centered on similar regression estimates but varied in width and smoothness. The analytical intervals were narrower, less variable, and quicker to calculate but based on strong

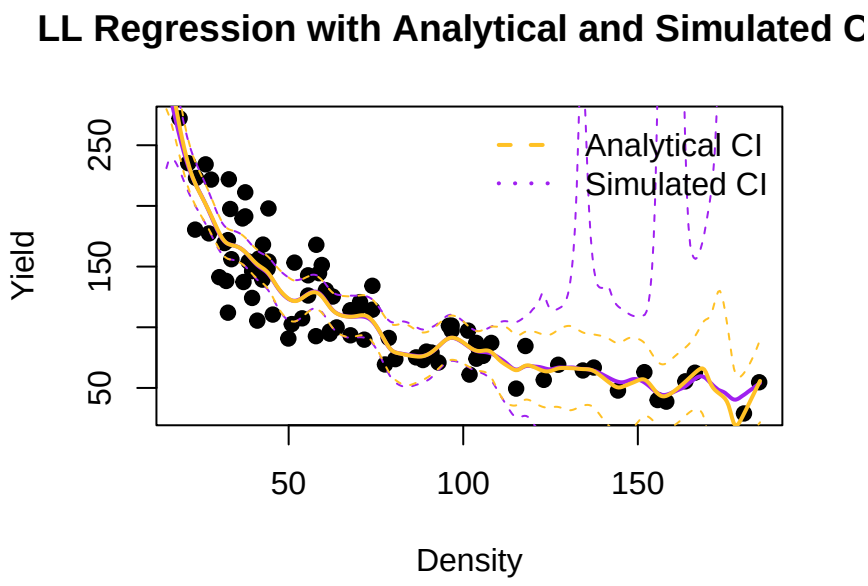
smoothness assumptions. Simulated intervals were more adaptable and light on assumptions but wigglier and broader corresponding to the resampling estimate instability.

Ultimately, analytical approach was more interpretable under optimal scenarios and bootstrap approach demonstrated uncertainty in finite samples. They are differentiated based on context: analytical approaches are suitable for well-specified modelling and bootstrap intervals are robust with complexity or under misspecification.

Appendix

Automatic Implementation of the "Rule-of-Thumb" Bandwidth

```
# Store the outputs of ana.ll and sim.ll
res_analytic <- ana.ll(onions.data, xgrid = xgrid, h = h_rot,
                               kernel = "gauss", alpha = 0.05)
res_bootstrap <- sim.ll(onions.data, xgrid = xgrid, h = h_rot,
                         kernel = "gauss", alpha = 0.05, B = 300)

plot(onions.data$dens, onions.data$yield, pch = 19,xlab = "Density",
     ylab = "Yield", main = "LL Regression with Analytical and Simulated CIs")

# Simulated Approach
lines(res_bootstrap$xgrid, res_bootstrap$est, col = "purple", lwd = 2)
lines(res_bootstrap$xgrid, res_bootstrap$lower, col = "purple", lty = 2)
lines(res_bootstrap$xgrid, res_bootstrap$upper, col = "purple", lty = 2)

# Analytical Approach
lines(res_analytic$xgrid, res_analytic$est, col = "goldenrod1", lwd = 2)
lines(res_analytic$xgrid, res_analytic$lower, col = "goldenrod1", lty = 2)
lines(res_analytic$xgrid, res_analytic$upper, col = "goldenrod1", lty = 2)

legend("topright", legend = c("Analytical CI", "Simulated CI"),
       col = c("goldenrod1", "purple"), lwd = 2, lty = c(2, 3), bty = "n")
```



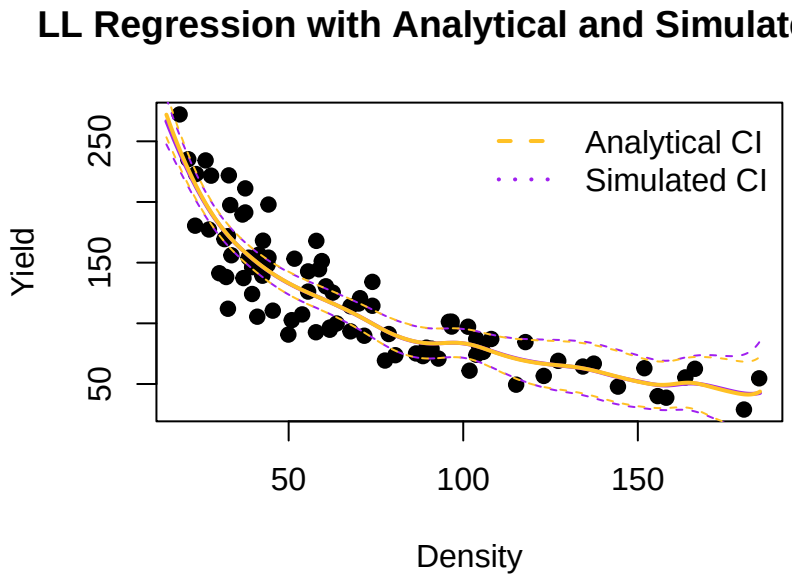

```
width_analytic <- mean(res_analytic$upper - res_analytic$lower)
width_bootstrap <- mean(res_bootstrap$upper - res_bootstrap$lower)
```

```r
interval_widths <- data.frame(Method = c("Analytical Approach", "Simulated Approach"),
                              Average_Width = c(width_analytic, width_bootstrap))
interval_widths
```

```
##                Method Average_Width
## 1 Analytical Approach      50.79783
## 2  Simulated Approach  675145.80941
```

Dpill Bandwidth Implementation

```r
h_dpill <- dpill(onions.data$dens, onions.data$yield)

# Store the outputs of ana.ll and sim.ll
res_analytic <- ana.ll(onions.data, xgrid = xgrid, h = h_dpill,
                              kernel = "gauss", alpha = 0.05)
res_bootstrap <- sim.ll(onions.data, xgrid = xgrid, h = h_dpill,
                        kernel = "gauss", alpha = 0.05, B = 300)

plot(onions.data$dens, onions.data$yield, pch = 19,xlab = "Density",
     ylab = "Yield",main = "LL Regression with Analytical and Simulated CIs")

# Simulated Approach
lines(res_bootstrap$xgrid, res_bootstrap$est, col = "purple", lwd = 2)
lines(res_bootstrap$xgrid, res_bootstrap$lower, col = "purple", lty = 2)
lines(res_bootstrap$xgrid, res_bootstrap$upper, col = "purple", lty = 2)

# Analytical Approach
lines(res_analytic$xgrid, res_analytic$est, col = "goldenrod1", lwd = 2)
lines(res_analytic$xgrid, res_analytic$lower, col = "goldenrod1", lty = 2)
lines(res_analytic$xgrid, res_analytic$upper, col = "goldenrod1", lty = 2)

legend("topright", legend = c("Analytical CI", "Simulated CI"),
       col = c("goldenrod1", "purple"), lwd = 2, lty = c(2, 3), bty = "n")
```




```
width_analytic <- mean(res_analytic$upper - res_analytic$lower)
width_bootstrap <- mean(res_bootstrap$upper - res_bootstrap$lower)

interval_widths <- data.frame(Method = c("Analytical Approach", "Simulated Approach"),
                              Average_Width = c(width_analytic, width_bootstrap))
interval_widths
```

```
##                Method Average_Width
## 1 Analytical Approach      31.00706
## 2  Simulated Approach      32.43705
```