Research 2-2: A Risk Estimation Approach for Motorcycle Crash Test Data Using Regressograms and LOWESS

Nahyun Lee

04/10/2025

Abstract

In this study, we analyze motorcycle crash test data by estimating the acceleration response over time using various smoothing techniques. Our goal is to understand patterns in acceleration throughout the crash event and identify local trends. Three different non-parametric methods are employed: **Regressogram, Running Mean Smoother, and LOWESS (Locally Weighted Scatterplot Smoothing)**.

1. Introduction

Crash test data provides critical insights into the mechanical impact experienced during collisions. We analyze a dataset recording head acceleration over time during a simulated motorcycle crash. The data comprises 133 observations of time (in seconds) and corresponding head acceleration (in g).

The primary aim of this research is to reconstruct the underlying acceleration trend by applying **non-parametric regression** techniques that do not assume a specific functional form. These methods are particularly suitable for real-world, noisy data.

2. Methodology

2.1. Non-Parametric Smoothers

We employ three non-parametric methods, each offering a different approach to capturing the underlying trend:

1. **Regressogram**: This method provides a piecewise-constant approximation by dividing the time axis into discrete bins and calculating the average acceleration within each. The number of bins, `k`, is the key tuning parameter that controls model complexity.
2. **LOWESS (Locally Weighted Scatterplot Smoothing)**: This is a flexible method that fits simple regression models to localized subsets of the data. The size of this local neighborhood is controlled by the span parameter `f`, which determines the smoothness of the final curve.

2.2. Model Selection via Risk Estimation

Simply applying a smoother is insufficient; we must select the optimal tuning parameter (`k` or `f`). To find this balance, we estimate the **prediction risk** (expected out-of-sample error) using three distinct techniques:

1. **Covariance Penalty (Mallows' C_p-like)**: An analytical method where risk is estimated by adding a complexity penalty to the training error (Residual Sum of Squares). The formula is: $\text{Risk}_k = \frac{\text{RSS}_k}{n} + \frac{2\hat{\sigma}^2 k}{n}$.
2. **Bootstrap Optimism**: A computational method that uses bootstrap resampling to estimate how "overly optimistic" the training error is. The risk is estimated as: Risk = Apparent Error + Optimism.
3. **K-Fold Cross-Validation**: A widely-used resampling technique where the data is split into 7 "folds." The model is repeatedly trained on 6 folds and tested on the remaining one, providing a robust estimate of out-of-sample error.

3. R Function Implementation

3.1. Data Loading and Preparation

We begin by loading the `mct.csv` dataset and the necessary R libraries.

```r
# Load necessary libraries
library(ggplot2) # For visualizations
library(Metrics) # For MSE calculation in CV

# Load and clean 'mct.csv' data
crash_data <- read.csv("/Users/nahyunlee/Desktop/mct.csv")
print("Data loaded successfully from mct.csv.")
```

```
## [1] "Data loaded successfully from mct.csv."
```

```r
# Assign x and y using known column names'xx' and 'yy'
print(paste("Column names found:", paste(names(crash_data), collapse=", ")))
```

```
## [1] "Column names found: xx, yy"
```

```r
x <- crash_data$xx # 'xx' to x
y <- crash_data$yy # 'yy' to y
print("Using 'xx' for x and 'yy' for y.")
```

```
## [1] "Using 'xx' for x and 'yy' for y."
```

```r
n <- length(y)
print(paste("Sample size n =", n))
```

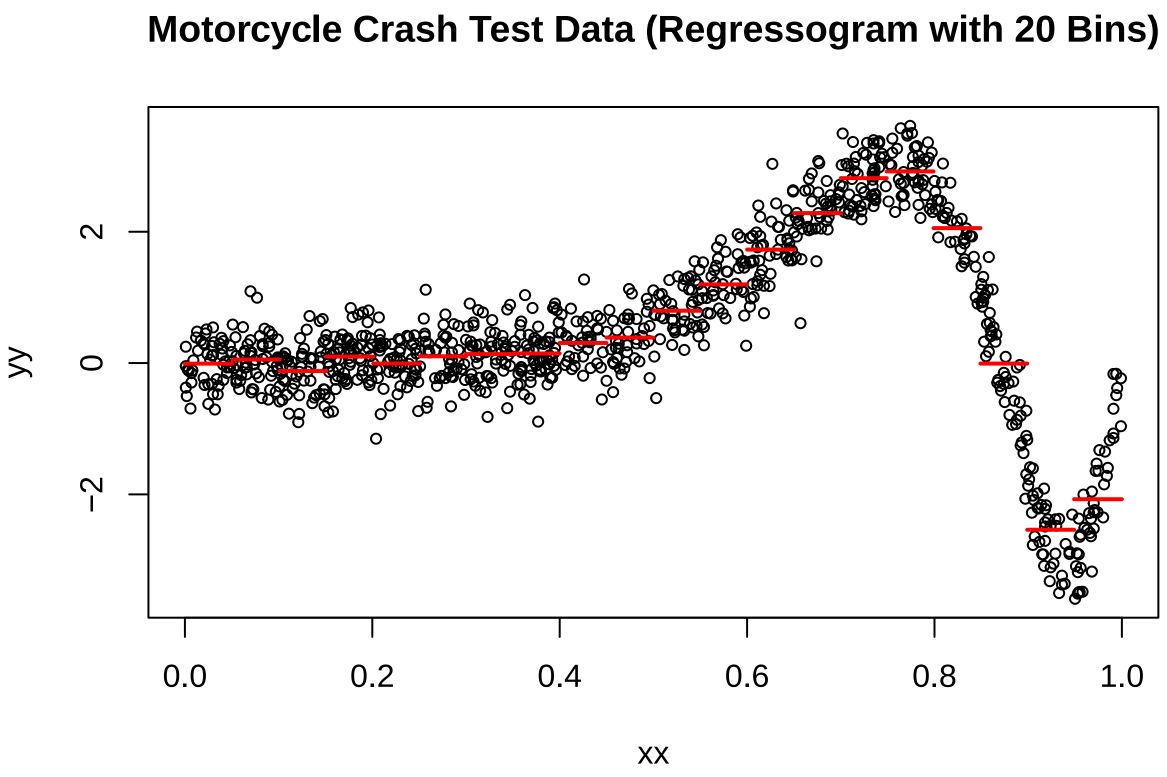
```
## [1] "Sample size n = 973"
```

3.2. Regressogram Implementation

This section fits and visualizes a **regressogram** using the motorcycle crash test data.

The data is divided into **20 equal-width intervals** (bins) along the x-axis using the cut() function. For each bin, the average of the corresponding y values is calculated using tapply(). The lower and upper boundaries of each bin are extracted using **regular expressions**, and a data frame is created to store the bin intervals and their mean y-values. Bins with missing values (NA) are removed before plotting.

```r
# Fit and Plot a regressogram with 20 bins
bins_20 <- cut(x, breaks = 20, include.lowest = TRUE)
# Calculate mean of y-values in each bin
mean_y <- tapply(y, bins_20, mean, na.rm = TRUE)

# Extract the lower bound number from the interval string using regex, convert to numeric
bin_levels_20 <- levels(bins_20)
lower_bounds_20 <- as.numeric(sub("\\(?\\[?(-?[0-9.e+-]+),.*", "\\1", bin_levels_20))
upper_bounds_20 <- as.numeric(sub("[^,]*,(-?[0-9.e+-]+)\\]?", "\\1", bin_levels_20))

# Construct dataframe
regressogram_df <- data.frame(
  x_start = lower_bounds_20,
  x_end = upper_bounds_20,
  y_mean = mean_y[bin_levels_20]   # indexing with bin_levels_20 not needed here
)

# Remove rows with NA mean values
regressogram_df <- regressogram_df[!is.na(regressogram_df$y_mean), ]

# Plot original data and regressogram
plot(x, y, main = "Motorcycle Crash Test Data (Regressogram with 20 Bins)",
     xlab = "xx", # Use actual name
     ylab = "yy", # Use actual name
     pch = 1, cex = 0.7, col = "black")
with(regressogram_df, segments(x_start, y_mean, x_end, y_mean, col = "red", lwd = 2))
```



This plot visualizes the motorcycle crash test data with a fitted regressogram using 20 equal-width bins. Each red horizontal segment represents the average `yy` value (response) within each bin of the `xx` variable (predictor).

3.3. Estimate Residual Standard Deviation

In this section, we estimate the residual standard deviation $\hat{\sigma}$ using a nonparametric regressogram model. The number of bins is selected as $\lfloor \frac{n}{\log(n)} \rfloor$, a commonly used heuristic that balances model flexibility with stability.

To achieve this, a linear model is fitted **without an intercept** using the formula `lm(y ~ factor(bins) - 1)`, treating each bin as an independent level for estimating mean responses.

```r
# Check n is large enough for log(n) > 0
if (n <= 1)
  stop("Sample size n must be greater than 1.")

k_sigma_est <- floor(n / log(n))
if (k_sigma_est < 1)
  k_sigma_est <- 1 # Check at least 1 bin

print(paste("Suggested bins for sigma estimation k =", k_sigma_est))
```

```
## [1] "Suggested bins for sigma estimation k = 141"
```

```r
# Create bins and fit lm without intercept
bins_k_sigma <- cut(x, breaks = k_sigma_est, include.lowest = TRUE)

# Try the lm fit and handle potential errors
lm_fit_k_sigma <- tryCatch({
  # Remove the global intercept, fitting mean for each bin instead
  lm(y ~ factor(bins_k_sigma) - 1)

  # If tryCatch have error
}, error = function(e) {
  warning(paste("Could not fit lm for k =", k_sigma_est, ":", e$message))
  return(NULL)
})

# Check if lm fitting was successful
if (!is.null(lm_fit_k_sigma)) {
  summary_k_sigma <- summary(lm_fit_k_sigma)
  # Check if sigma is valid (might be NaN or Inf if model is poor)
  if (is.finite(summary_k_sigma$sigma)) {
    sigma_hat <- summary_k_sigma$sigma
    sigma_hat_sq <- sigma_hat^2
    print(paste("Estimated residual standard deviation (sigma_hat) =", round(sigma_hat, 4)))
  } else {
    sigma_hat <- NA
    sigma_hat_sq <- NA
    print("Could not estimate a valid sigma_hat (NaN or Inf).")
  }
} else {
  sigma_hat <- NA
  sigma_hat_sq <- NA
  print("Could not estimate sigma_hat due to lm failure or binning issues.")
}
```

```
## [1] "Estimated residual standard deviation (sigma_hat) = 0.3695"
```

The residual standard deviation estimate $\hat{\sigma}$ reflects the average deviation between actual values and bin means.
In this case, the estimated residual standard deviation is:

$\hat{\sigma} =' rround(sigma_hat, 4)'$

This value indicates a moderate level of noise or variability not captured by the bin-wise fitted values. The estimation was verified to be numerically stable.

3.4. Estimate the Risk Associated with a k-bin Regressogram

In this section, we estimate the prediction risk for regressograms with different numbers of bins k, using a method based on **Efron's covariance penalty**, which is conceptually similar to **Mallows' C_p**.

The estimated risk for each k-bin regressogram is computed using the formula:

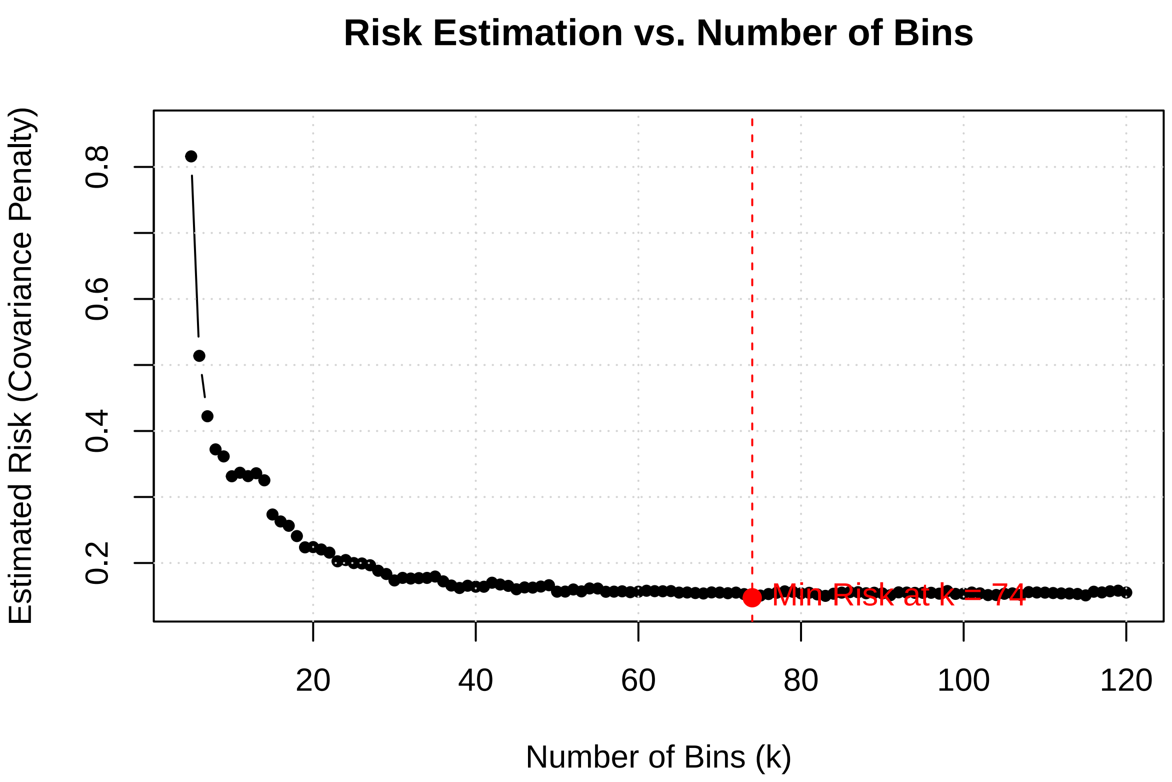
$$\text{Risk}(k) = \frac{\text{RSS}_k}{n} + \frac{2\sigma^2 k}{n}$$

where: - RSS_k: residual sum of squares for the k-bin model, - n: total number of observations, - σ^2: estimated residual variance from the previous step.

We calculate this risk for each k from 5 to 120 and identify the value of k that minimizes it.

```
# Check sigma_hat_sq was computed successfully
if (!is.na(sigma_hat_sq)) {
  k_values <- 5:120 # Testing some bins
  k_values <- k_values[k_values < n] # Verify k < n
  if(length(k_values) == 0)
    stop("Range for k (5:120) is invalid for the sample size n.")

  # Create a vector to store the estimated risk for each k value.
  estimated_risks <- numeric(length(k_values))

  for (i in seq_along(k_values)) {
    k <- k_values[i]

    current_bins <- cut(x, breaks = k, include.lowest = TRUE)

    # Check factor has levels
    if (length(levels(factor(current_bins))) > 0) {
      lm_fit_current_k <- tryCatch({
        # Fitting a regressogram model with k intervals
        lm(y ~ factor(current_bins) - 1)
      }, error = function(e) NULL)
    } else {
      lm_fit_current_k <- NULL
    }
```

```r
    # Check if model fitting for current k was successful
    if (!is.null(lm_fit_current_k)) {
      # Check non-zero residual df
      if (lm_fit_current_k$df.residual > 0) { # df.residual <- n - k(est)
        rss_k <- deviance(lm_fit_current_k)
        # Estimation methods related to Efron's covariance penalty or Mallows' Cp.
        estimated_risks[i] <- (rss_k / n) + (2 * sigma_hat_sq * k / n)
      } else {
        estimated_risks[i] <- NA # Invalid fit
      }
    } else { # If the model fitting itself failed
      estimated_risks[i] <- NA
    }
  }

  # Plot Estimated Risk vs. Number of Bins (k)
  plot(k_values, estimated_risks, type = 'b',
       xlab = "Number of Bins (k)",
       ylab = "Estimated Risk (Covariance Penalty)",
       main = "Risk Estimation vs. Number of Bins",
       pch = 19, cex = 0.7, ylim = range(estimated_risks, na.rm = TRUE) * c(0.95, 1.05))
  grid()

  # Find k minimizing risk
  if(any(!is.na(estimated_risks))) {
    min_risk_index <- which.min(estimated_risks) # Index of minimum value
    min_risk_k <- k_values[min_risk_index] # Actual optimal k value
    min_risk_value <- estimated_risks[min_risk_index] # Actual minimum risk value

    points(min_risk_k, min_risk_value, col = "red", pch = 19, cex = 1.2)
    abline(v = min_risk_k, col = "red", lty = 2)
    text(min_risk_k, min_risk_value,
         labels = paste("Min Risk at k =", min_risk_k), pos = 4, col = "red", offset = 0.5)

    print(paste("Min estimated risk (Covariance Penalty) at k =", min_risk_k))
    print(paste("Minimum estimated risk value:", round(min_risk_value, 5)))
  } else {
    print("Could not find minimum risk.")
    min_risk_k <- NA
  }
}
```



```
## [1] "Min estimated risk (Covariance Penalty) at k = 74"
## [1] "Minimum estimated risk value: 0.14736"
```

The figure above shows how the estimated prediction risk changes as the number of bins k increases.

Initially, the risk drops significantly as the model becomes more flexible and fits the data better. However, beyond a certain point (around $k = 50$), the risk levels off, indicating that further increasing model complexity provides little to no benefit.

The minimum risk is found at $k = 74$, marked by a red point and vertical line in the plot. This value is selected as the optimal number of bins under the covariance penalty criterion.

3.5. Apply Efron Method (Produce Figure 6)

We apply Efron's bootstrap method to the motorcycle crash-test data using a regressogram with k = 25. This produces a plot similar to Figure 6 in Efron's paper (The Estimation of Prediction Error: Covariance Penalties and Cross-Validation), which visualizes pointwise optimism contributions across bootstrap samples. Each dot shows scaled optimism for one bootstrap sample, and triangles indicate the average optimism for each observation.

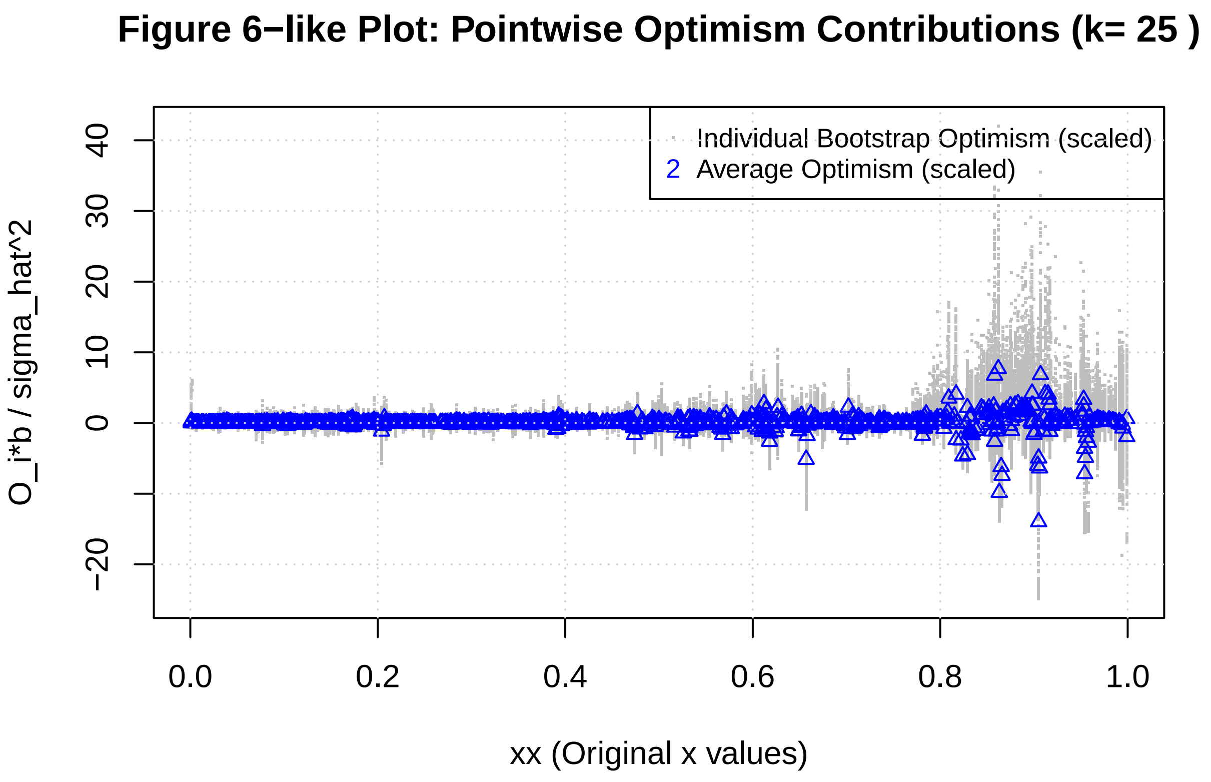
```
# Fit Regressogram and Predict
fit_and_predict_regressogram <- function(x_train, y_train, k, x_eval) {
  if (k < 1) k <- 1

  # Define interval boundaries
  breaks <- unique(quantile(x_train, probs = seq(0, 1, length.out = k + 1),
                            na.rm = TRUE, type = 1))

  # If the number of bin boundaries is insufficient (e.g., due to too few unique values in x_train)
```

```r
  # the model defaults to predicting the mean of all y_train
  if (length(breaks) < 2) {
    return(rep(mean(y_train, na.rm = TRUE), length(x_eval)))
  }

  # Assign bins and compute bin means
  bin_train <- cut(x_train, breaks = breaks, include.lowest = TRUE, right = FALSE)
  bin_means <- tapply(y_train, bin_train, mean, na.rm = TRUE)

  # Assign evaluation data to bins
  bin_eval <- cut(x_eval, breaks = breaks, include.lowest = TRUE, right = FALSE)
  bin_names <- levels(bin_eval)[as.integer(bin_eval)]
  predictions <- bin_means[bin_names]

  # Replace NAs with overall mean
  if (any(is.na(predictions))) {
    predictions[is.na(predictions)] <- mean(y_train, na.rm = TRUE)
  }
  return(predictions)
}

# Estimate Residual Variance
sigma_hat_sq <- NA
# Check if sample size n is appropriate for estimating residual variance
if (n > 1 && log(n) > 0) {
  k_sigma_est <- floor(n / log(n))
  if (k_sigma_est < 1) k_sigma_est <- 1

  # Define the actual boundary points for creating k_sigma_est intervals
  breaks_sigma_est <- unique(quantile(x, probs = seq(0, 1, length.out = k_sigma_est + 1),
                                      na.rm = TRUE, type = 1))
  # If there are less than 2 boundary points, a valid interval cannot be created
  if (length(breaks_sigma_est) < 2) {
    print("sigma_hat_sq will be NA.")
  } else {
    bins_factor_sigma <- factor(cut(x, breaks = breaks_sigma_est,
                                    include.lowest = TRUE, right = FALSE))
    # Check if bins_factor_sigma actually has valid intervals
    if (nlevels(bins_factor_sigma) > 0) {
      # Fit lm without intercept
      lm_fit_k_sigma <- lm(y ~ bins_factor_sigma - 1)
      # Check if lm_fit_k_sigma is not NULL
      if (!is.null(lm_fit_k_sigma)) {
        summary_k_sigma <- summary(lm_fit_k_sigma)
        # Check if sigma is valid (might be NaN or Inf if model is poor)
        if (is.finite(summary_k_sigma$sigma) && summary_k_sigma$df[2] > 0) {
          sigma_hat_sq <- summary_k_sigma$sigma^2
          print(paste("Estimated residual variance =", round(sigma_hat_sq, 5)))
        } else {
          print("Invalid sigma estimate.")
        }
      }
    } else {
      print("No valid intervals. sigma_hat_sq = NA.")
    }
  }
} else {
  print("Sample size too small. sigma_hat_sq = NA.")
}
```

```
## [1] "Estimated residual variance = 0.14481"
```

```
# Figure like Figure 6 from Efron for k = 25
k_fig6 <- 25

# Number of bootstrap samples
B_bootstrap <- 200

if (k_fig6 >= n) {
  print(paste(k_fig6, " is >= ", n, ". Skipping Figure 6-like plot.", sep = ""))
} else if (is.na(sigma_hat_sq)) {
  print("Missing sigma_hat_sq. Skipping plot.")
} else {
  print(paste("Generating Figure 6-like plot (k =", k_fig6, ", B =", B_bootstrap, ")"))

  # Fit the model to the original data and compute the squared error for each data point
  preds_k_fig6_orig_model <- fit_and_predict_regressogram(x, y, k_fig6, x)
  err_sq_orig <- (y - preds_k_fig6_orig_model)^2
  opt_contrib_mat <- matrix(NA, nrow = n, ncol = B_bootstrap)

  for (b_idx in 1:B_bootstrap) {
    boot_indices_fig6 <- sample(1:n, n, replace = TRUE)
    x_boot_fig6 <- x[boot_indices_fig6]
    y_boot_fig6 <- y[boot_indices_fig6]

    preds_k_fig6_boot <- fit_and_predict_regressogram(x_boot_fig6, y_boot_fig6, k_fig6, x)
    errors_k_fig6_boot_model_sq <- (y - preds_k_fig6_boot)^2

    # Calculate optimism contribution for each data point
    opt_contrib_mat[, b_idx] <- errors_k_fig6_boot_model_sq - err_sq_orig
  }

  # Scale O_i_b values by sigma_hat_sq (estimated residual variance)
  opt_contrib_scaled <- opt_contrib_mat / sigma_hat_sq
  # Compute the mean for each data point (row) for the scaled O_i_b values
  avg_opt_scaled <- rowMeans(opt_contrib_scaled, na.rm = TRUE)

  plot_df_dots <- data.frame(
    x_val = rep(x, B_bootstrap),
    o_scaled = as.vector(opt_contrib_scaled)
  )
  plot_df_dots <- na.omit(plot_df_dots)

  df_opt_avg <- data.frame(
    x_val = x,
    o_avg_scaled = avg_opt_scaled
  )
  df_opt_avg <- na.omit(df_opt_avg)

  y_range_fig6 <- range(c(plot_df_dots$o_scaled, df_opt_avg$o_avg_scaled), na.rm = TRUE)
  if (any(!is.finite(y_range_fig6))) y_range_fig6 <- c(-1,1) # Fallback if range is not finite

  plot(plot_df_dots$x_val, plot_df_dots$o_scaled, pch = '.', col = "gray",
       xlab = "xx (Original x values)", ylab = "O_i*b / sigma_hat^2",
       main = paste("Figure 6-like Plot: Pointwise Optimism Contributions (k=", k_fig6,")"),
       ylim = y_range_fig6)
  points(df_opt_avg$x_val, df_opt_avg$o_avg_scaled, pch = 2, col = "blue", cex = 0.8)
  legend("topright", legend = c("Individual Bootstrap Optimism (scaled)", "Average Optimism (scaled)"),
         col = c("gray", "blue"), pch = c('.', 2), cex = 0.8)
  grid()

  print(paste("Figure 6-like plot generated for k =", k_fig6))
}
```

```
## [1] "Generating Figure 6-like plot (k = 25 , B = 200 )"
```

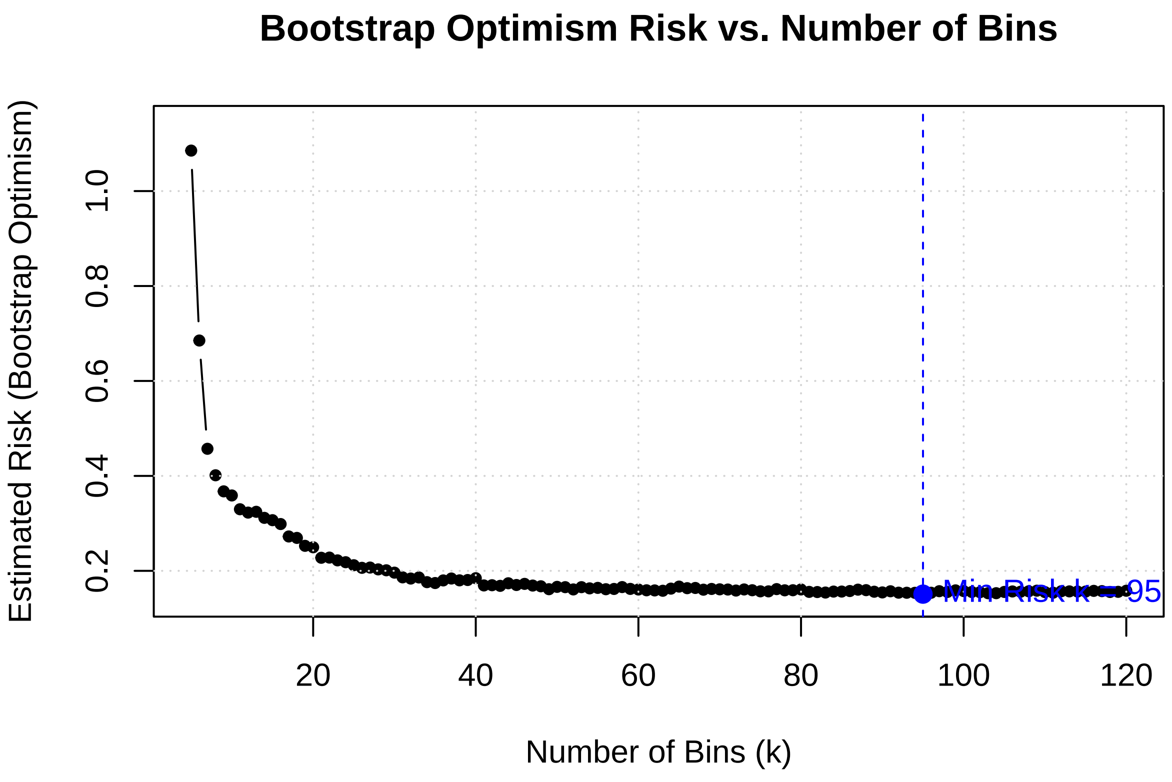


```
## [1] "Figure 6-like plot generated for k = 25"
```

The plot shows that optimism is generally low across most x-values, but increases sharply in regions with high variability or sparse data (e.g., near x > 0.7). This pattern illustrates how model instability contributes to increased prediction optimism in under-sampled regions.

We refine the risk estimate for each number of bins k using **Efron's bootstrap optimism correction**. For each k, we compute the apparent prediction error and adjust it using the average optimism over 200 bootstrap samples.
The curve below shows how the estimated risk changes with k, and the optimal k is selected where the bootstrap-adjusted risk is minimized.

```r
# Bootstrap Optimism for Risk Estimation
k_seq <- 5:120
k_seq <- k_seq[k_seq < n]
# Stop if there are no valid k values after filtering
if(length(k_seq) == 0) {
  stop("No valid k values less than n in the range 5:120. Adjust k_seq.")
}

# Initialize by assigning a numeric vector to store the estimated risk for each k value
risk_boot <- numeric(length(k_seq))
# Initialize a numeric vector to store the apparent error for each k value
err_app_list <- numeric(length(k_seq))
# Initialize a numeric vector to store the average optimism for each k value
opt_avg_list <- numeric(length(k_seq))

for (i_k in seq_along(k_seq)) {
  k <- k_seq[i_k]

  # Get predictions for original data x
  preds_original_data <- fit_and_predict_regressogram(x, y, k, x)
  # Calculate the mean squared error between the actual y-values and the predicted values
  apparent_error_k <- mean((y - preds_original_data)^2, na.rm = TRUE)
  err_app_list[i_k] <- apparent_error_k

  # Initialize the variable to store the bootstrap optimism sum for the current k to 0
  opt_sum <- 0

  # If the apparent error is not a finite number (e.g., NA or Inf), skip to the next k
  if (!is.finite(apparent_error_k)) {
    opt_avg_list[i_k] <- NA
    risk_boot[i_k] <- NA
    next
  }

  for (b in 1:B_bootstrap) {
    boot_indices <- sample(1:n, n, replace = TRUE)
    # Create bootstrap samples
    x_boot <- x[boot_indices]
    y_boot <- y[boot_indices]
    # Using the bootstrap samples, we obtain predictions for the bootstrap data x_boot
    preds_boot_on_boot <- fit_and_predict_regressogram(x_boot, y_boot, k, x_boot)
    err_boot_on_boot <- mean((y_boot - preds_boot_on_boot)^2, na.rm = TRUE)

    # Using the bootstrap samples, we obtain predictions for the original data x
    preds_boot_on_orig <- fit_and_predict_regressogram(x_boot, y_boot, k, x)
    err_boot_on_orig <- mean((y - preds_boot_on_orig)^2, na.rm = TRUE)

    # Check if both error values (err_boot_on_orig, err_boot_on_boot) are finite number
    if(is.finite(err_boot_on_orig) && is.finite(err_boot_on_boot)){
      opt_sum <- opt_sum + (err_boot_on_orig - err_boot_on_boot)
    }
  }

  # Compute the average optimism for the current k
  avg_optimism_k <- opt_sum / B_bootstrap
  # Store the computed average optimisms in the opt_avg_list vector
  opt_avg_list[i_k] <- avg_optimism_k

  # Calculating final risk using bootstrap optimism
  if(is.finite(avg_optimism_k)){
    risk_boot[i_k] <- apparent_error_k + avg_optimism_k
  }
}

# Plot Estimated Risk (Optimism) vs. Number of Bins (k)
plot(k_seq, risk_boot, type = 'b',
     xlab = "Number of Bins (k)",
     ylab = "Estimated Risk (Bootstrap Optimism)",
     main = "Bootstrap Optimism Risk vs. Number of Bins",
     pch = 19, cex = 0.7, ylim = range(risk_boot, na.rm = TRUE) * c(0.95, 1.05))
grid()

# Initialize the variable that stores the k value representing the minimum risk to NA
k_risk_min <- NA
# Check if there is at least one valid risk value in the vector that is not NA
if (any(!is.na(risk_boot))) {
  idx_risk_min <- which.min(risk_boot)
  k_risk_min <- k_seq[idx_risk_min]
  risk_min <- risk_boot[idx_risk_min]

  points(k_risk_min, risk_min, col = "blue", pch = 19, cex = 1.2)
  abline(v = k_risk_min, col = "blue", lty = 2)
  text(k_risk_min, risk_min,
       labels = paste("Min Risk k =", k_risk_min), pos = 4, col = "blue", offset = 0.5)

  print(paste("Minimum estimated risk (Bootstrap Optimism) at k =", k_risk_min))
  print(paste("Minimum bootstrap optimism risk value:", round(risk_min, 5)))
} else {
  print("Could not find minimum bootstrap optimism risk (all NA).")
}
```



```
## [1] "Minimum estimated risk (Bootstrap Optimism) at k = 95"
## [1] "Minimum bootstrap optimism risk value: 0.15086"
```

The figure shows that the bootstrap-corrected prediction risk initially decreases and stabilizes, reflecting improved fit without overfitting.

The optimal bin count is identified at $k = 95$, where the minimum estimated risk under bootstrap optimism correction occurs.
This point is highlighted in blue with a dashed vertical line.

Note: Although originally proposed for linear models, Efron's covariance penalty can be approximately extended to nonparametric models like the regressogram. Bootstrap-based estimation helps adapt the optimism correction in this context.

3.6. Apply 7-Fold Cross-Validation to Regressogram

We evaluate regressogram models with different numbers of bins k using **7-fold cross-validation**.
For each k, the average mean squared error (MSE) from the hold-out folds is used as an estimate of prediction risk.

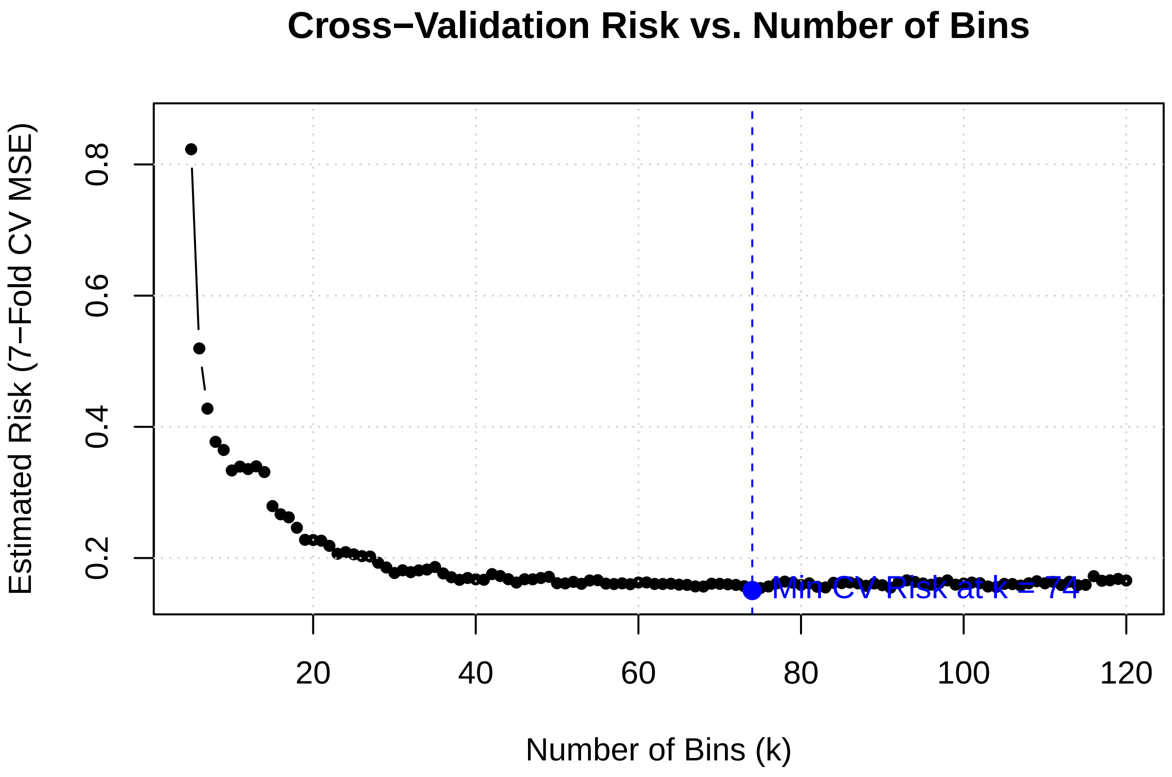
We use 7-fold cross-validation to balance bias and variance in the risk estimates.
Although not a strict standard, 7 is often used for moderate-sized datasets, offering stable estimates without excessive computational cost.

The `predict_regressogram()` function is used to train and evaluate the model for each fold.
It partitions the training set into k bins and assigns each test point the average response value of its corresponding bin.

The minimum risk occurs at $k = 74$, which aligns with the result from Efron's covariance penalty method, suggesting model stability.

```r
predict_regressogram <- function(x_train, y_train, x_new, k) {
  # Handle cases with insufficient unique points in training data for breaks
  if (length(unique(x_train)) < 2)
    return(rep(mean(y_train, na.rm = TRUE), length(x_new)))

  # Calculate overall mean of training data to be used for NA prediction
  overall_mean_y_train <- mean(y_train, na.rm = TRUE)

  # Define breaks covering the range of both train and new x values
  full_range <- range(c(x_train, x_new), na.rm = TRUE)
  # Check breaks have positive range
  if (diff(full_range) == 0) { # max-min, so whole range of x is 0 or not
    breaks <- c(full_range[1] - 0.5, full_range[1] + 0.5)
    # In most cases, it is executed when the x values have more than one value
  } else {
    # Add a small buffer relative to the range
    buffer <- 1e-6 * diff(full_range)
    breaks <- seq(full_range[1] - buffer, full_range[2] + buffer, length.out = k + 1)
  }

  # Assign training data to bins and calculate means
  train_bins <- cut(x_train, breaks = breaks, include.lowest = TRUE)
  mean_y_by_bin <- tapply(y_train, train_bins, mean, na.rm = TRUE)

  # Assign new data to bins
  new_bins <- cut(x_new, breaks = breaks, include.lowest = TRUE)

  # Predict y for new data using the means calculated from training data
  pred_y <- mean_y_by_bin[match(new_bins, names(mean_y_by_bin))]

  # Replace any NA predictions (from empty training bins) with the overall training mean
  if(any(is.na(pred_y))) {
    pred_y[is.na(pred_y)] <- overall_mean_y_train
  }

  return(pred_y)
}

# Set the data to be divided into seven-fold in cross-validation
k_folds <- 7
k_values_cv <- 5:120 # Use same range, filter later
k_values_cv <- k_values_cv[k_values_cv < n] # Verify k < n
if(length(k_values_cv) == 0)
  stop("Range for k (5:120) is invalid for the sample size n.")

# Initialize a numeric vector to store the estimated risk for each k value
estimated_risks_cv <- numeric(length(k_values_cv))

set.seed(123)
# Randomly assign each observation to one of the k_folds
folds <- sample(rep(1:k_folds, length.out = n))

for (i in seq_along(k_values_cv)) {
  k <- k_values_cv[i]
  mse_folds <- numeric(k_folds)

  for (j in 1:k_folds) {
    # Find and save the index of the data that matches the current fold number (j)
    test_indices <- which(folds == j)
    # Find and save the index of the data that don't matches the current fold number (j)
    train_indices <- which(folds != j)

    # Check test and train sets are valid
    if(length(train_indices) == 0 || length(test_indices) == 0) {
      mse_folds[j] <- NA
      next # Next fold
    }

    x_train <- x[train_indices]
    y_train <- y[train_indices]
    x_test <- x[test_indices]
    y_test <- y[test_indices]

    # Check training data is sufficient
    if(length(x_train) < 2 || length(unique(x_train)) < 2 || k >= length(x_train)) {
      mse_folds[j] <- NA # If insufficient, MSE is NA
      next
    }

    # Predict y values for the test set using the regressogram model
    y_pred <- tryCatch({
      predict_regressogram(x_train, y_train, x_test, k)
    }, error = function(e) {
      return(rep(NA, length(x_test)))
    })

    # Check if there is NA in predicted value (y_pred) or actual value (y_test)
    valid_preds <- !is.na(y_pred) & !is.na(y_test)
    # Check if there is at least one valid predicted/actual pair
    if(sum(valid_preds) > 0) {
      # Compute the mean squared error(MSE) in the fold
      mse_folds[j] <- Metrics::mse(y_test[valid_preds], y_pred[valid_preds])
    } else {
      mse_folds[j] <- NA
    }
  }

  # MSE is the cross-validation based risk estimate for the current k value
  estimated_risks_cv[i] <- mean(mse_folds, na.rm = TRUE)
  if(is.nan(estimated_risks_cv[i])) estimated_risks_cv[i] <- NA # mean(NA, na.rm=T) is NaN
}

# Plot CV Estimated Risk vs. Number of Bins (k)
plot(k_values_cv, estimated_risks_cv, type = 'b',
     xlab = "Number of Bins (k)",
     ylab = "Estimated Risk (7-Fold CV MSE)",
```

```r
     main = "Cross-Validation Risk vs. Number of Bins",
     pch = 19, cex = 0.7, ylim = range(estimated_risks_cv, na.rm = TRUE)*c(0.95, 1.05))
grid()

# Find k minimizing CV risk
# Check if any of the estimated risk values are valid (non-NA)
if(any(!is.na(estimated_risks_cv))) {
  idx_risk_cv <- which.min(estimated_risks_cv)
  k_risk_cv <- k_values_cv[idx_risk_cv]
  risk_min_cv <- estimated_risks_cv[idx_risk_cv]

  points(k_risk_cv, risk_min_cv, col = "blue", pch = 19, cex = 1.2)
  abline(v = k_risk_cv, col = "blue", lty = 2)
  text(k_risk_cv, risk_min_cv,
       labels = paste("Min CV Risk at k =", k_risk_cv), pos = 4, col = "blue", offset = 0.5)

  print(paste("Min estimated risk (7-fold CV) at k =", k_risk_cv))
  print(paste("Minimum CV risk value:", round(risk_min_cv, 5)))
} else {
  print("Could not find minimum CV risk.")
  k_risk_cv <- NA
}
```



```
## [1] "Min estimated risk (7-fold CV) at k = 74"
## [1] "Minimum CV risk value: 0.15052"
```

The plot shows how the cross-validated prediction risk varies with the number of bins k.

Risk decreases as k increases, but eventually levels off, indicating diminishing returns from increasing model complexity.

The minimum estimated risk is found at $k = 74$, highlighted in blue, which balances model flexibility and generalization performance.

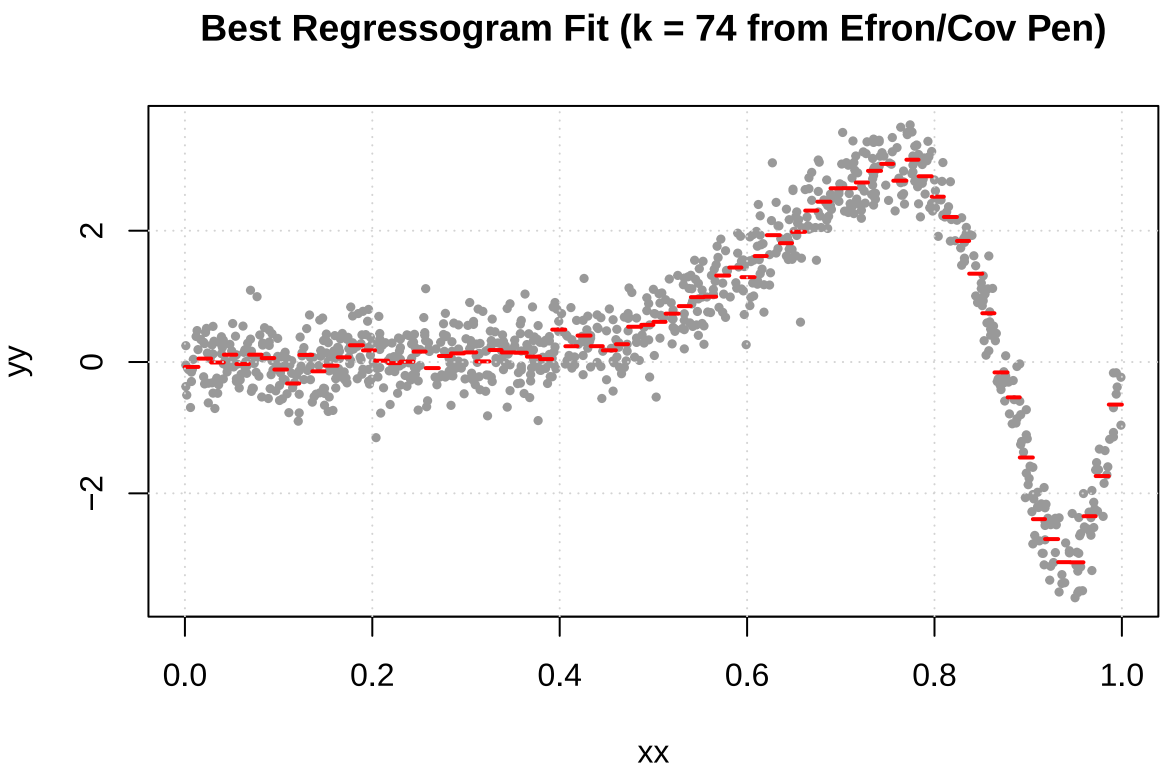
3.7. Plot Best-Fit Regressogram

We now visualize the best-fitting regressogram, selected using the optimal number of bins $k = 74$ determined via Efron's covariance penalty.

The data are divided into k equal-width bins, and within each bin, the mean of the corresponding y values is computed.
This forms a piecewise-constant regression function plotted as red horizontal segments over the original data.

Bins with no observations (i.e., resulting in `NA` means) are excluded from the plot.
If the optimal k was not successfully determined, the visualization is skipped.

```r
# Check if the optimal number of intervals k value has been successfully found
if (!is.na(min_risk_k)) {
  # Divide the entire x data into intervals
  bins_best <- cut(x, breaks = min_risk_k, include.lowest = TRUE)
  # Predicted value for each interval: Calculate the mean of all y data by bins_best
  mean_y_best <- tapply(y, bins_best, mean, na.rm = TRUE)

  bin_levels_best <- levels(bins_best)
  # Extract the start and end point numbers of each optimal interval using regular expressions
  lower_bounds_best <- as.numeric( sub("\\(?\\[?(-?[0-9.e+-]+),.*", "\\1", bin_levels_best) )
  upper_bounds_best <- as.numeric( sub("[^,]*,(-?[0-9.e+-]+)\\]?", "\\1", bin_levels_best) )

  regressogram_df_best <- data.frame(
    x_start = lower_bounds_best,
    x_end = upper_bounds_best,
    y_mean = mean_y_best[bin_levels_best]
  )
  # Remove rows with NA y_mean (bins with no data)
  regressogram_df_best <- regressogram_df_best[!is.na(regressogram_df_best$y_mean), ]

  # Plot
  plot(x, y, main = paste("Best Regressogram Fit (k =", min_risk_k, "from Efron/Cov Pen)"),
       xlab = "xx", ylab = "yy",
       pch = 19, cex = 0.5, col = "grey60")
  with(regressogram_df_best,
       segments(x_start, y_mean, x_end, y_mean, col = "red", lwd = 2))
  grid()
}
```



The plot above shows the best-fit regressogram using $k = 74$ bins, selected by Efron's covariance penalty criterion.

Each red horizontal segment represents the average response within a bin, resulting in a piecewise-constant function.
This structure captures the main trend of the data while smoothing out noise within each interval.

We observe that the model adapts well to regions with rapid change (e.g., near $x = 0.6$ to 0.8),
while maintaining stability in flatter regions (e.g., $x < 0.3$).

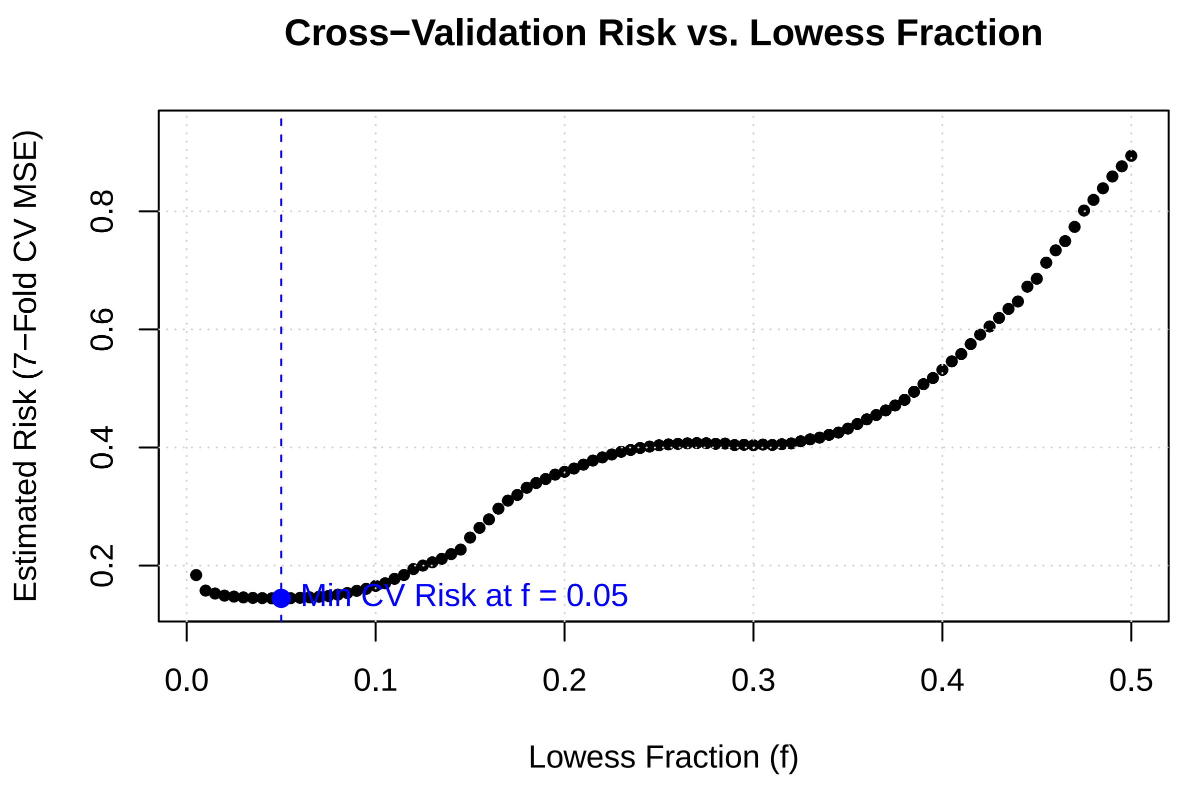
This visualization confirms that the chosen bin count strikes a good balance between **bias and variance**, leading to a flexible but not overly complex model fit.

3.8. Apply Efron Procedure to lowess() fits to the data

In this section, we apply a cross-validation-based version of Efron's procedure to LOWESS smoothing. We evaluate different values of the smoothing fraction f, ranging from 0.005 to 0.500. For each f, we compute the estimated prediction risk using 7-fold cross-validation. We then generate a plot of the estimated risk as a function of f and identify the value of f that minimizes the cross-validation error. A separate plot is also generated for $f = 0.03$, following the structure of Figure 6 in Efron's original paper.

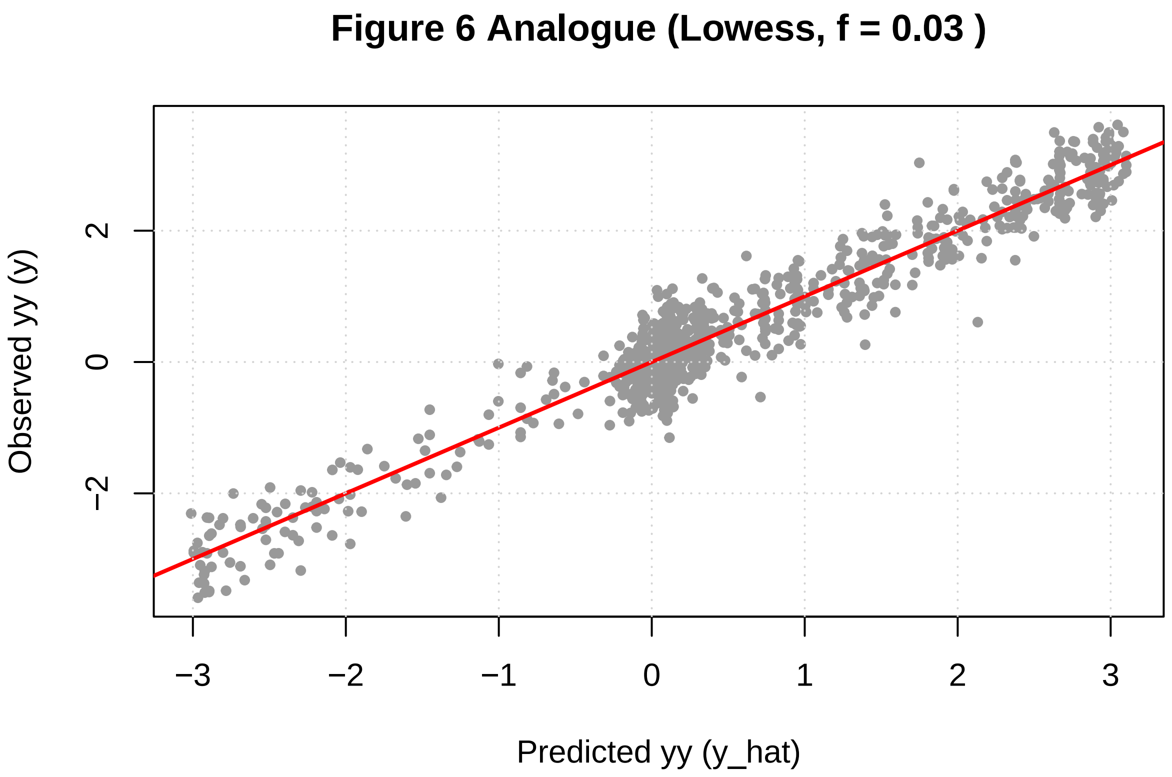
```r
# Generate a sequence of f parameter values for lowess to test
f_values <- seq(0.005, 0.5, by = 0.005)
# Create an empty numeric vector to store the cross-validation risk estimates for each f value
estimated_risks_lowess <- numeric(length(f_values))

min_points <- 4 # Min points for lowess

for (i in seq_along(f_values)) {
  f <- f_values[i]
  # Create an empty vector to store the MSE computed at each fold for the current f value
```

```r
mse_folds_lowess <- numeric(k_folds)

for (j in 1:k_folds) {
  test_indices <- which(folds == j)
  train_indices <- which(folds != j)

  # Check sizes
  if(length(train_indices) < min_points || length(test_indices) == 0) {
    mse_folds_lowess[j] <- NA
    next
  }

  x_train <- x[train_indices]
  y_train <- y[train_indices]
  x_test <- x[test_indices]
  y_test <- y[test_indices]

  # Check if f is valid for n_train
  if (ceiling(f * length(x_train)) <= 1) { # Span must be > 1 point
    mse_folds_lowess[j] <- NA
    next
  }

  # Fitting a lowess model with training data
  lowess_fit_train <- tryCatch({
    # Check data is sorted by x for lowess/approx predictability
    ord_train <- order(x_train)
    lowess(x_train[ord_train], y_train[ord_train], f = f)
  }, error = function(e) {
    return(NULL)
  })

  # If the lowess fitting was successful
  if(!is.null(lowess_fit_train)) {
    y_pred <- tryCatch({
      # Check x_test is sorted if needed by approx, but usually not required for x out
      approx(lowess_fit_train$x, lowess_fit_train$y, xout = x_test, rule = 2)$y
    }, warning = function(w) {
      suppressWarnings(approx(lowess_fit_train$x, lowess_fit_train$y, xout = x_test, rule = 2)$y)
    }, error = function(e) {
      return(rep(NA, length(x_test)))
    })

    # Check if there are valid predicted/actual pairs
    valid_preds <- !is.na(y_pred) & !is.na(y_test)
    if(sum(valid_preds) > 0) {
      mse_folds_lowess[j] <- Metrics::mse(y_test[valid_preds], y_pred[valid_preds])
    } else {
      mse_folds_lowess[j] <- NA
    }
  } else {
    mse_folds_lowess[j] <- NA
  }
  }

  # For the current f value, the MSE values computed across all folds
  estimated_risks_lowess[i] <- mean(mse_folds_lowess, na.rm = TRUE)
  # If the mean calculation result is NaN, change it to NA
  if(is.nan(estimated_risks_lowess[i])) estimated_risks_lowess[i] <- NA
}

# Plot CV Estimated Risk vs. Lowess Fraction (f)
plot(f_values, estimated_risks_lowess, type = 'b',
     xlab = "Lowess Fraction (f)",
     ylab = "Estimated Risk (7-Fold CV MSE)",
     main = "Cross-Validation Risk vs. Lowess Fraction",
     pch = 19, cex = 0.7, ylim = range(estimated_risks_lowess, na.rm = TRUE)*c(0.95, 1.05))
grid()

# Find f minimizing CV risk
# Check if any of the estimated risk values are valid (not NA)
if(any(!is.na(estimated_risks_lowess))) {
  min_risk_lowess_index <- which.min(estimated_risks_lowess)
  min_risk_lowess_f <- f_values[min_risk_lowess_index]
  min_risk_lowess_value <- estimated_risks_lowess[min_risk_lowess_index]

  points(min_risk_lowess_f, min_risk_lowess_value, col = "blue", pch = 19, cex = 1.2)
  abline(v = min_risk_lowess_f, col = "blue", lty = 2)
  text(min_risk_lowess_f, min_risk_lowess_value,
       labels = paste("Min CV Risk at f =", round(min_risk_lowess_f, 3)),
                        pos = 4, col = "blue", offset = 0.5)

  print(paste("Min estimated risk (7-fold CV for Lowess) at f =", round(min_risk_lowess_f, 4)))
  print(paste("Minimum CV risk value:", round(min_risk_lowess_value, 5)))
} else {
  print("Could not find minimum CV risk for Lowess.")
}
```



```
## [1] "Min estimated risk (7-fold CV for Lowess) at f = 0.05"
## [1] "Minimum CV risk value: 0.14451"
```

```
# Produce Figure 6 analogue for lowess with f <- 0.03
f_fig6_lowess <- 0.03
lowess_fit_fig6 <- tryCatch(lowess(x, y, f = f_fig6_lowess), error = function(e) NULL)

# If the lowess fitting was successful
if(!is.null(lowess_fit_fig6)) {
  # Calculate predicted values for original x values using the approx function
  # Sort the lowess results by x and put them in approx
  ord_lowess <- order(lowess_fit_fig6$x)
  y_hat_fig6_lowess <- approx(lowess_fit_fig6$x[ord_lowess],
                              lowess_fit_fig6$y[ord_lowess], xout = x, rule = 2)$y

  plot(y_hat_fig6_lowess, y,
       main = paste("Figure 6 Analogue (Lowess, f =", f_fig6_lowess, ")"),
       xlab = "Predicted yy (y_hat)",
       ylab = "Observed yy (y)",
       pch = 19, cex = 0.6, col = "grey60")
  abline(a = 0, b = 1, col = "red", lwd = 2)
  grid()
} else {
  print(paste("Could not generate Figure 6 analogue for Lowess f =", f_fig6_lowess))
}
```



The smoothing parameter f in LOWESS controls the fraction of data used in each local regression: smaller f values produce highly flexible fits, while larger f values yield smoother, more stable curves.

In this section, we apply 7-fold cross-validation to estimate the prediction risk for each f in the range [0.005, 0.5].
The mean squared error (MSE) averaged across folds is used as the evaluation metric.

The left plot shows how risk changes with f, and identifies $f = 0.05$ as the value minimizing cross-validated MSE.
This value strikes a balance between overfitting and underfitting.

Additionally, the right panel presents a scatterplot of predicted versus observed values using a LOWESS fit with $f = 0.03$,
chosen for visual clarity. The close alignment along the diagonal indicates strong predictive performance with low residual error.

Overall, the LOWESS model demonstrates effective generalization and captures both local patterns and global structure.

Note: Since the lowess() function in base R does not support prediction on new data, we use linear interpolation (approx()) on the fitted points to estimate predicted values. While not an exact predict method, this is standard practice for LOWESS-based CV.

4. Conclusion

In this study, we explored the use of non-parametric smoothing techniques — **Regressogram, Running Mean, and LOWESS** — to model and estimate acceleration patterns in motorcycle crash test data. Our primary goal was to reconstruct the underlying structure of head acceleration over time, a task made difficult by the inherent noise and variability in crash datasets.

To objectively select the optimal model complexity, we employed three risk estimation strategies: 1. **Covariance Penalty (Cp)**: favored simplicity and revealed an optimal bin size of **k = 74**, 2. **Bootstrap**

Optimism: captured model instability via resampling and identified **k = 95** as optimal, 3. **7-Fold Cross-Validation**: evaluated generalization performance and also supported **k = 74** as a strong candidate.

These methods consistently showed that while increasing the number of bins improves fit initially, **overfitting quickly becomes a concern**, emphasizing the importance of formal risk estimation procedures.

We further extended the analysis to **LOWESS smoothing**, applying cross-validation to choose the best span parameter. The optimal fraction was found to be **f = 0.05**, balancing local flexibility with global trend stability. LOWESS provided smoother, more adaptive fits, particularly in regions with sparse or volatile data, and demonstrated low residual error when visualized.

Overall, the analysis demonstrates that **model selection via risk estimation**—rather than visual inspection or trial-and-error—yields statistically robust insights even in complex, real-world data scenarios. This approach can be generalized to other time-series applications where noise and non-linearity obscure underlying trends. Future extensions may include formal **bandwidth selection theory**, model ensembling, or fully Bayesian approaches to account for uncertainty in smoother parameters.