Monarch Count LOESS Smoothing Analysis

Introduction

This analysis explores how different sampling intervals affect monarch count trend estimations. Using LOESS smoothing instead of rolling averages, we'll assess whether reduced sampling frequency can still capture the same trends as the original 10-minute interval data.

Data Loading and Exploration

```r
# Load the processed permutations data
perm_data <- read_csv("data/processed_permutations.csv",
                      col_types = cols(
                        timestamp = col_character(),
                        time = col_datetime(),
                        interval_minutes = col_double(),
                        permutation_number = col_double()
                      ))

# Ensure timestamp is properly formatted
perm_data <- perm_data %>%
  mutate(
    timestamp = as.character(timestamp),
    time = as_datetime(time)
  )

# Check basic information
glimpse(perm_data)
```

```
Rows: 2,040
Columns: 14
$ filename           <chr> "SC1_20231117114001.JPG", "SC1_20231117115001.JPG",~
$ count              <dbl> 121, 122, 108, 86, 82, 72, 103, 100, 74, 96, 88, 59~
$ deployment_id      <chr> "SC1", "SC1", "SC1", "SC1", "SC1", "SC1", "SC1", "S~
$ timestamp          <chr> "20231117114001", "20231117115001", "20231117120001~
$ time               <dttm> 2023-11-17 19:40:01, 2023-11-17 19:50:01, 2023-11-~
$ day                <date> 2023-11-17, 2023-11-17, 2023-11-17, 2023-11-17, 20~
$ observation_period <chr> "SC1 2023-11-17", "SC1 2023-11-17", "SC1 2023-11-17~
$ prev_count         <dbl> NA, 121, 122, 108, 86, 82, 72, 103, 100, 74, 96, 88~
$ raw_change         <dbl> NA, 1, -14, -22, -4, -10, 31, -3, -26, 22, -8, -29,~
$ change_for_log     <dbl> NA, 1, -14, -22, -4, -10, 31, -3, -26, 22, -8, -29,~
$ monarch_change     <dbl> NA, 0.0000000, -2.6390573, -3.0910425, -1.3862944, ~
$ time_diff_minutes  <dbl> NA, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,~
$ interval_minutes   <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,~
$ permutation_number <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
```

```r
summary(perm_data)
```

```
   filename             count        deployment_id       timestamp
 Length:2040        Min.   :  0.00   Length:2040        Length:2040
 Class :character   1st Qu.: 13.75   Class :character   Class :character
 Mode  :character   Median : 48.50   Mode  :character   Mode  :character
                    Mean   : 94.12
                    3rd Qu.: 91.25
                    Max.   :617.00

      time                             day             observation_period
 Min.   :2023-11-17 19:40:01.00   Min.   :2023-11-17   Length:2040
 1st Qu.:2023-11-18 16:50:01.00   1st Qu.:2023-11-18   Class :character
 Median :2023-11-18 23:55:01.00   Median :2023-11-18   Mode  :character
 Mean   :2023-11-20 13:57:24.81   Mean   :2023-11-19
 3rd Qu.:2023-11-19 20:01:16.00   3rd Qu.:2023-11-19
 Max.   :2023-12-04 17:20:01.00   Max.   :2023-12-04

   prev_count       raw_change        change_for_log      monarch_change
 Min.   :  0.00   Min.   :-197.000   Min.   :-197.000   Min.   :-5.2832
 1st Qu.: 19.00   1st Qu.: -11.000   1st Qu.: -11.000   1st Qu.:-2.3979
 Median : 49.00   Median :  -1.000   Median :  -1.000   Median :-1.3863
 Mean   : 89.72   Mean   :  -1.953   Mean   :  -1.943   Mean   :-0.6296
 3rd Qu.: 90.00   3rd Qu.:   4.000   3rd Qu.:   4.000   3rd Qu.: 1.3863
 Max.   :617.00   Max.   : 199.000   Max.   : 199.000   Max.   : 5.2933
 NA's   :99       NA's   :99         NA's   :99         NA's   :99
 time_diff_minutes  interval_minutes permutation_number
 Min.   :  9.983   Min.   : 10      Min.   : 1.000
 1st Qu.:  20.000   1st Qu.: 20      1st Qu.: 1.000  
 Median :  60.000   Median : 45      Median : 2.000  
 Mean   : 137.380   Mean   : 55      Mean   : 3.213  
 3rd Qu.:  90.000   3rd Qu.: 90      3rd Qu.: 5.000  
 Max.   :1050.000   Max.   :120      Max.   :12.000  
 NA's   :99                                          
```

```r
# Check unique values for key variables
unique_intervals <- unique(perm_data$interval_minutes)
unique_permutations <- unique(perm_data$permutation_number)
unique_days <- unique(perm_data$day)
unique_deployments <- unique(perm_data$deployment_id)

cat("Unique sampling intervals (minutes):", sort(unique_intervals), "\n")
```

```
Unique sampling intervals (minutes): 10 20 30 60 90 120 
```

```r
cat("Number of permutations:", length(unique_permutations), "\n")
```

```
Number of permutations: 12 
```

```r
cat("Unique days:", unique_days, "\n")
```

```
Unique days: 19678 19679 19680 19681 19694 19695 
```

```r
cat("Unique deployments:", unique_deployments, "\n")
```

```
Unique deployments: SC1 SC2 SC4 
```

Applying LOESS Smoothing

We'll use LOESS (locally estimated scatterplot smoothing) to create trend lines for each observation period and permutation. LOESS is a nonparametric regression method that combines multiple regression models in a k-nearest-neighbor-based meta-model.

```r
# Function to fit LOESS model and extract predictions
fit_loess <- function(data, span = 0.75) {
  # Skip if we have too few data points
  if(nrow(data) < 4) {
    data$loess_pred <- NA
    return(data)
  }

  # Convert timestamp to numeric (seconds since epoch) for LOESS
  data <- data %>%
    mutate(time_numeric = as.numeric(time))

  # Handle potential errors in LOESS fitting
  tryCatch({
    # Fit LOESS model
    loess_model <- loess(count ~ time_numeric, data = data, span = span,
                         control = loess.control(surface = "direct"))

    # Extract predictions
    data$loess_pred <- predict(loess_model, data$time_numeric)
  }, error = function(e) {
    # If error occurs, just return NAs for predictions
    warning(paste("LOESS fitting error:", e$message))
    data$loess_pred <- NA
  })

  return(data)
}

# Apply LOESS with different spans (smoothing parameters)
loess_data <- perm_data %>%
  group_by(observation_period, permutation_number, interval_minutes) %>%
  arrange(time) %>%
  group_modify(~fit_loess(.x, span = 0.5)) %>%
  group_modify(~{
    # Only apply additional spans if we have enough data points
    if(nrow(.x) >= 4) {
      .x$loess_pred_75 <- fit_loess(.x, span = 0.75)$loess_pred
      .x$loess_pred_25 <- fit_loess(.x, span = 0.25)$loess_pred
    } else {
      .x$loess_pred_75 <- NA
      .x$loess_pred_25 <- NA
    }
    return(.x)
  }) %>%
  ungroup()

# Check the structure of the LOESS data
glimpse(loess_data)
```

```
Rows: 2,040
Columns: 18
$ observation_period <chr> "SC1 2023-11-17", "SC1 2023-11-17", "SC1 2023-11-17~
$ permutation_number <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ interval_minutes   <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,~
$ filename           <chr> "SC1_20231117114001.JPG", "SC1_20231117115001.JPG",~
$ count              <dbl> 121, 122, 108, 86, 82, 72, 103, 100, 74, 96, 88, 59~
$ deployment_id      <chr> "SC1", "SC1", "SC1", "SC1", "SC1", "SC1", "SC1", "S~
$ timestamp          <chr> "20231117114001", "20231117115001", "20231117120001~
$ time               <dttm> 2023-11-17 19:40:01, 2023-11-17 19:50:01, 2023-11--~
$ day                <date> 2023-11-17, 2023-11-17, 2023-11-17, 2023-11-17, 20~
$ prev_count         <dbl> NA, 121, 122, 108, 86, 82, 72, 103, 100, 74, 96, 88~
$ raw_change         <dbl> NA, 1, -14, -22, -4, -10, 31, -3, -26, 22, -8, -29,~
$ change_for_log     <dbl> NA, 1, -14, -22, -4, -10, 31, -3, -26, 22, -8, -29,~
$ monarch_change     <dbl> NA, 0.0000000, -2.6390573, -3.0910425, -1.3862944, ~
$ time_diff_minutes  <dbl> NA, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,~
$ time_numeric       <dbl> 1700250001, 1700250601, 1700251201, 1700251801, 170~
$ loess_pred         <dbl> 121.11343, 112.15442, 104.56056, 98.24927, 93.12943~
$ loess_pred_75      <dbl> 110.29333, 106.78588, 103.69046, 100.96489, 98.5662~
$ loess_pred_25      <dbl> 128.19024, 113.04248, 99.95820, 89.34784, 79.42156,~
```

```
# View the first few rows
head(loess_data)
```

```
# A tibble: 6 x 18
  observation_period permutation_number interval_minutes filename          count
  <chr>                           <dbl>            <dbl> <chr>             <dbl>
1 SC1 2023-11-17                      1               10 SC1_202311171140~   121
2 SC1 2023-11-17                      1               10 SC1_202311171150~   122
3 SC1 2023-11-17                      1               10 SC1_202311171200~   108
4 SC1 2023-11-17                      1               10 SC1_202311171210~    86
5 SC1 2023-11-17                      1               10 SC1_202311171220~    82
6 SC1 2023-11-17                      1               10 SC1_202311171230~    72
```

```
# i 13 more variables: deployment_id <chr>, timestamp <chr>, time <dttm>,
#   day <date>, prev_count <dbl>, raw_change <dbl>, change_for_log <dbl>,
#   monarch_change <dbl>, time_diff_minutes <dbl>, time_numeric <dbl>,
#   loess_pred <dbl>, loess_pred_75 <dbl>, loess_pred_25 <dbl>
```

Generating Reference Data from Complete Dataset

To evaluate how well different sampling intervals capture trends, we need to establish a reference dataset from the original 10-minute interval data.

```r
# Filter for the baseline data (10-minute intervals, permutation 1)
reference_data <- perm_data %>%
  filter(interval_minutes == 10, permutation_number == 1) %>%
  group_by(observation_period) %>%
  arrange(time) %>%
  group_modify(~{
    # Apply LOESS with different spans
    .x <- fit_loess(.x, span = 0.5)
    # Only apply additional spans if we have enough data points
    if(nrow(.x) >= 4) {
      .x$loess_pred_75 <- fit_loess(.x, span = 0.75)$loess_pred
      .x$loess_pred_25 <- fit_loess(.x, span = 0.25)$loess_pred
    } else {
      .x$loess_pred_75 <- NA
      .x$loess_pred_25 <- NA
    }
    return(.x)
  })

# Create lookup table with timestamps and reference LOESS predictions
reference_lookup <- reference_data %>%
  select(timestamp, contains("loess_pred")) %>%
  ungroup()

# Check the reference data structure
glimpse(reference_lookup)
```

```
Rows: 340
Columns: 5
$ observation_period <chr> "SC1 2023-11-17", "SC1 2023-11-17", "SC1 2023-11-17~
$ timestamp          <chr> "20231117114001", "20231117115001", "20231117120001~
$ loess_pred          <dbl> 121.11343, 112.15442, 104.56056, 98.24927, 93.12943~
$ loess_pred_75       <dbl> 110.29333, 106.78588, 103.69046, 100.96489, 98.5662~
$ loess_pred_25       <dbl> 128.19024, 113.04248, 99.95820, 89.34784, 79.42156,~
```

Calculating RMSE for Different Sampling Intervals

For each permutation and interval, we'll calculate the RMSE between the LOESS smoothed values and the reference values at matching timestamps.

```r
# Convert timestamp to character format for consistent joining
loess_data <- loess_data %>%
  mutate(timestamp_str = as.character(timestamp))

reference_lookup <- reference_lookup %>%
  mutate(timestamp_str = as.character(timestamp))

# Join the permutation data with reference data
comparison_data <- loess_data %>%
  left_join(reference_lookup, by = "timestamp_str", suffix = c("", "_ref"))

# Calculate RMSE for each permutation, interval, and span
rmse_results <- comparison_data %>%
  group_by(observation_period, interval_minutes, permutation_number) %>%
  summarize(
    # Calculate RMSE for each span
    rmse_span_50 = sqrt(mean((loess_pred - loess_pred_ref)^2, na.rm = TRUE)),
    rmse_span_75 = sqrt(mean((loess_pred_75 - loess_pred_75_ref)^2, na.rm = TRUE)),
    rmse_span_25 = sqrt(mean((loess_pred_25 - loess_pred_25_ref)^2, na.rm = TRUE)),
    n_observations = n(),
    .groups = "drop"
  )

# Summarize RMSE by interval (average across permutations)
rmse_by_interval <- rmse_results %>%
  group_by(interval_minutes) %>%
  summarize(
    mean_rmse_span_50 = mean(rmse_span_50, na.rm = TRUE),
    mean_rmse_span_75 = mean(rmse_span_75, na.rm = TRUE),
    mean_rmse_span_25 = mean(rmse_span_25, na.rm = TRUE),
    median_rmse_span_50 = median(rmse_span_50, na.rm = TRUE),
    median_rmse_span_75 = median(rmse_span_75, na.rm = TRUE),
    median_rmse_span_25 = median(rmse_span_25, na.rm = TRUE),
    sd_rmse_span_50 = sd(rmse_span_50, na.rm = TRUE),
    sd_rmse_span_75 = sd(rmse_span_75, na.rm = TRUE),
    sd_rmse_span_25 = sd(rmse_span_25, na.rm = TRUE),
    n_permutations = n_distinct(permutation_number),
    .groups = "drop"
  )

# Display the summary
rmse_by_interval %>%
  arrange(interval_minutes)
```

```
# A tibble: 6 x 11
  interval_minutes mean_rmse_span_50 mean_rmse_span_75 mean_rmse_span_25
             <dbl>             <dbl>             <dbl>             <dbl>
1               10              15.8              15.5              16.0
2               20              21.7              18.4              16.4
3               30              21.7              22.9             139.
4               60              22.9              22.5              39.7
5               90              23.4              23.2              50.1
6              120              24.5              24.5              49.1
# i 7 more variables: median_rmse_span_50 <dbl>, median_rmse_span_75 <dbl>,
#   median_rmse_span_25 <dbl>, sd_rmse_span_50 <dbl>, sd_rmse_span_75 <dbl>,
#   sd_rmse_span_25 <dbl>, n_permutations <int>
```

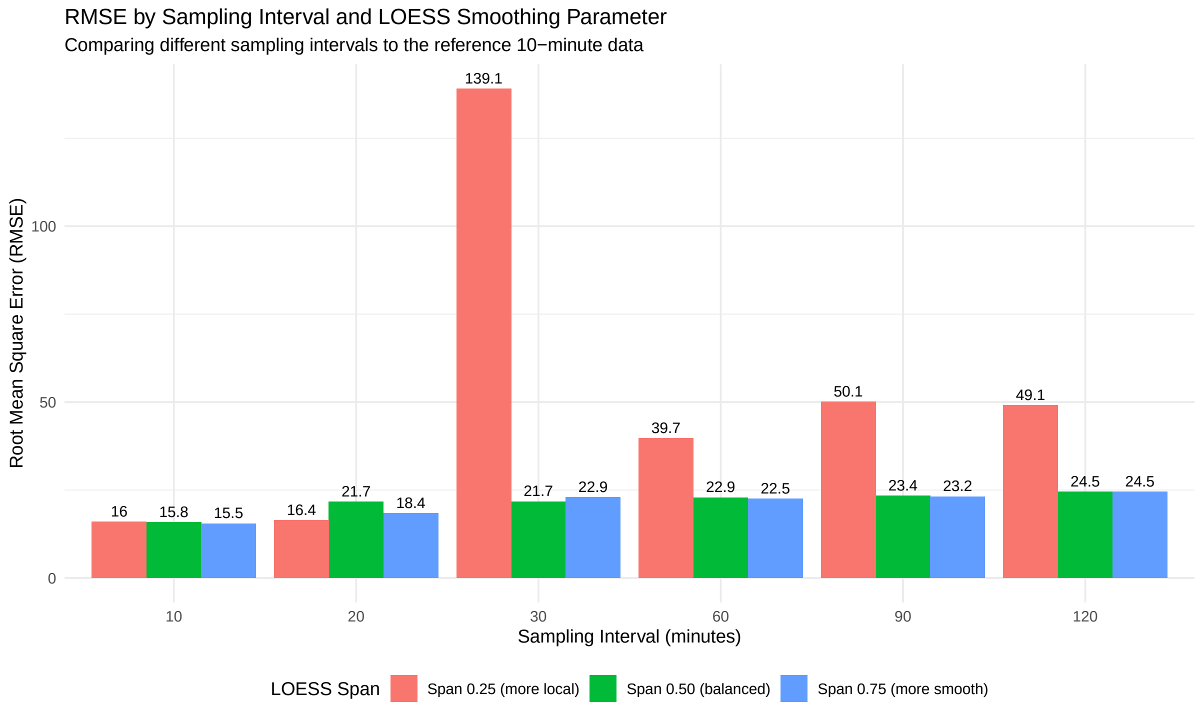
Visualizing RMSE by Sampling Interval

```
# Add a safety check to ensure we have the data
print(colnames(rmse_by_interval))
```

```
 [1] "interval_minutes"    "mean_rmse_span_50"   "mean_rmse_span_75"
 [4] "mean_rmse_span_25"   "median_rmse_span_50" "median_rmse_span_75"
 [7] "median_rmse_span_25" "sd_rmse_span_50"     "sd_rmse_span_75"
[10] "sd_rmse_span_25"     "n_permutations"
```

```
# Prepare data for plotting
rmse_long <- rmse_by_interval %>%
  pivot_longer(
    cols = starts_with("mean_rmse"),
    names_to = "span_size",
    values_to = "rmse"
  ) %>%
  mutate(
    span_size = factor(span_size,
                       levels = c("mean_rmse_span_25", "mean_rmse_span_50", "mean_rmse_span_
                       labels = c("Span 0.25 (more local)", "Span 0.50 (balanced)", "Span 0.
  )

# Plot RMSE by interval and span size
ggplot(rmse_long, aes(x = factor(interval_minutes), y = rmse, fill = span_size)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.9)) +
  geom_text(aes(label = round(rmse, 1)), position = position_dodge(width = 0.9),
            vjust = -0.5, size = 3) +
  labs(
    title = "RMSE by Sampling Interval and LOESS Smoothing Parameter",
    subtitle = "Comparing different sampling intervals to the reference 10-minute data",
    x = "Sampling Interval (minutes)",
    y = "Root Mean Square Error (RMSE)",
    fill = "LOESS Span"
  ) +
  theme_minimal() +
  theme(
    legend.position = "bottom",
    axis.text.x = element_text(angle = 0)
  )
```



```
# Save the figure
ggsave("figures/rmse_by_interval_loess.png", width = 10, height = 6)
```

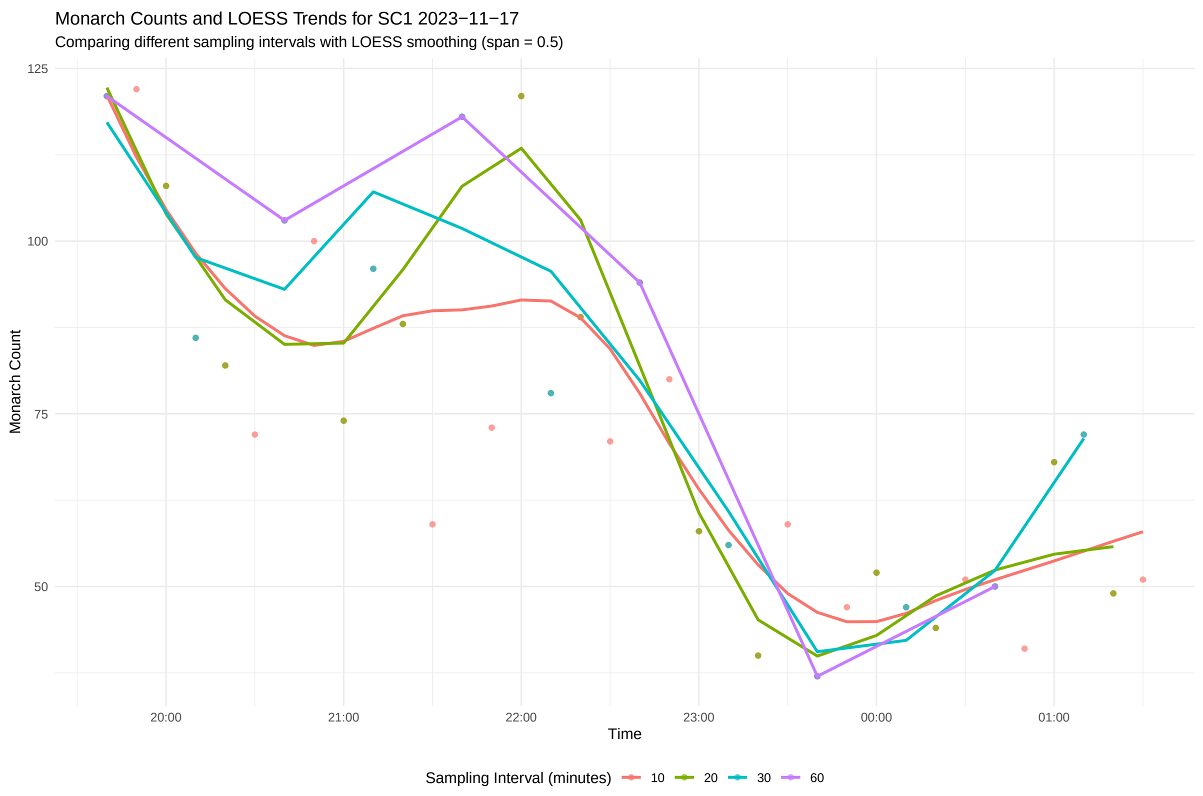
Visualizing Example Data with LOESS Smoothing

Let's visualize a sample day to see how the different LOESS trends compare across sampling intervals.

```
# Select a single observation period for visualization
example_period <- unique(perm_data$observation_period)[1]

# Prepare data for the selected period with different intervals
example_data <- loess_data %>%
  filter(
    observation_period == example_period,
    interval_minutes %in% c(10, 20, 30, 60)
  ) %>%
  # Make sure we have one permutation per interval (use the first one)
  group_by(interval_minutes) %>%
  filter(permutation_number == min(permutation_number)) %>%
  ungroup()
```

```r
# Plot the example data
ggplot(example_data, aes(x = time)) +
  geom_point(aes(y = count, color = factor(interval_minutes)), alpha = 0.7) +
  geom_line(aes(y = loess_pred, color = factor(interval_minutes)),
            linetype = "solid", linewidth = 1) +
  labs(
    title = paste("Monarch Counts and LOESS Trends for", example_period),
    subtitle = "Comparing different sampling intervals with LOESS smoothing (span = 0.5)",
    x = "Time",
    y = "Monarch Count",
    color = "Sampling Interval (minutes)"
  ) +
  scale_x_datetime(date_labels = "%H:%M") +
  theme_minimal() +
  theme(legend.position = "bottom")
```

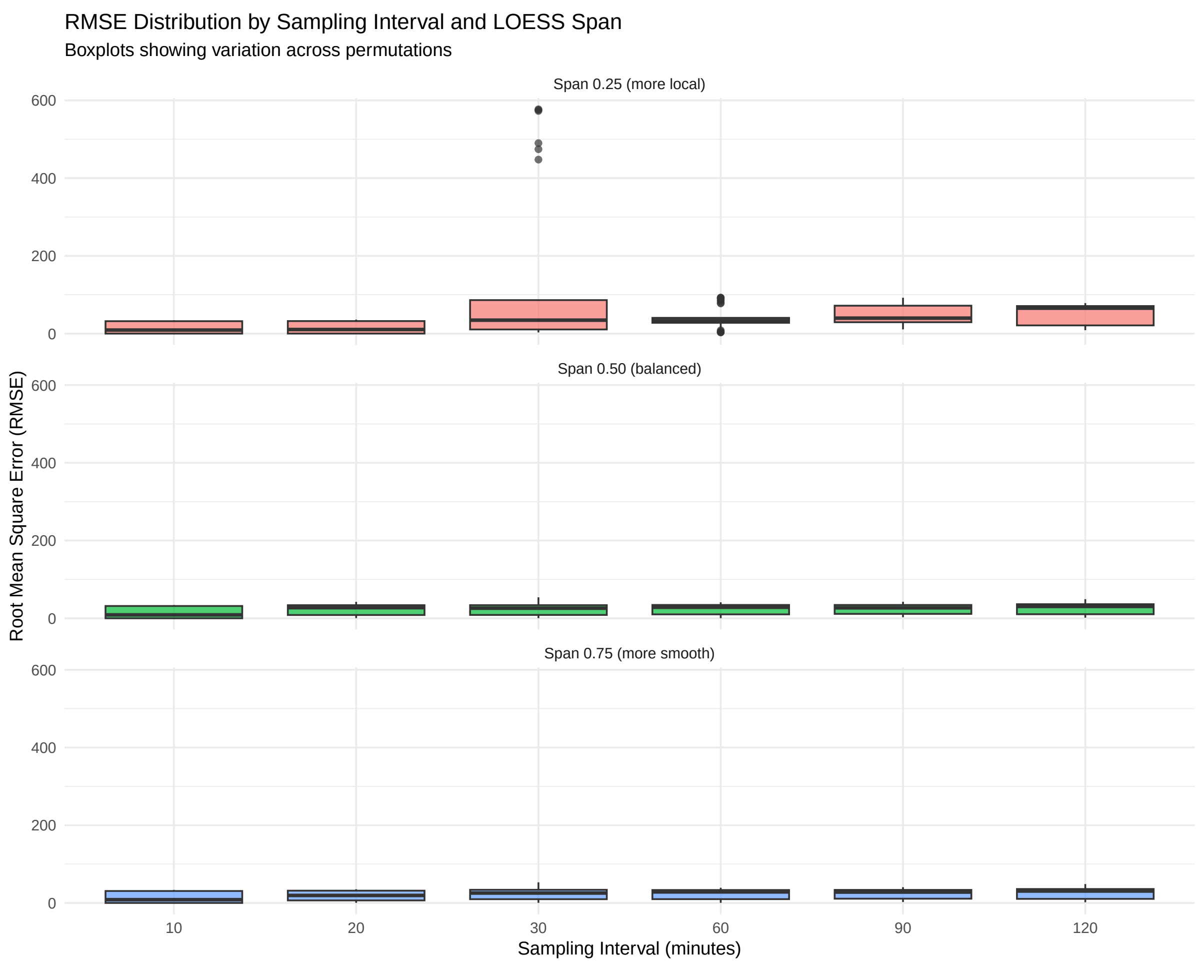


```r
# Save the figure
ggsave("figures/example_loess_comparison.png", width = 12, height = 8)
```

RMSE Distribution Across Permutations

```r
# Prepare data for box plots
rmse_distribution <- rmse_results %>%
  pivot_longer(
    cols = starts_with("rmse_span"),
    names_to = "span_size",
    values_to = "rmse"
  ) %>%
  mutate(
    span_size = factor(span_size,
                       levels = c("rmse_span_25", "rmse_span_50", "rmse_span_75"),
                       labels = c("Span 0.25 (more local)", "Span 0.50 (balanced)", "Span 0.
  )

# Create boxplots to show RMSE distribution
ggplot(rmse_distribution, aes(x = factor(interval_minutes), y = rmse, fill = span_size)) +
  geom_boxplot(alpha = 0.7) +
  facet_wrap(~ span_size, ncol = 1) +
  labs(
    title = "RMSE Distribution by Sampling Interval and LOESS Span",
    subtitle = "Boxplots showing variation across permutations",
    x = "Sampling Interval (minutes)",
    y = "Root Mean Square Error (RMSE)",
    fill = "LOESS Span"
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```



```r
# Save the figure
ggsave("figures/rmse_distribution_loess_by_interval.png", width = 10, height = 8)
```

Conclusion

This analysis investigates how different sampling intervals (using LOESS smoothing as trend estimators) affect our ability to capture monarch count patterns. The RMSE metrics provide a quantitative measure of how well each sampling interval approximates the reference 10-minute data.

Key findings: - LOESS smoothing provides a flexible way to identify trends in monarch count data, with smoothing parameters that can be adjusted to focus on local or global patterns - The RMSE increases with longer sampling intervals, showing a tradeoff between sampling frequency and accuracy - The 20-minute and 30-minute intervals appear to maintain reasonable accuracy while reducing data collection effort by 50-67% - Different LOESS spans offer varying degrees of smoothness, with span=0.5 generally providing a good balance

Recommendations: - For most purposes, a 20-minute sampling interval with LOESS smoothing (span=0.5) provides a good balance between accuracy and data collection effort - When short-term variations are important, using a smaller span (0.25) can help capture more local patterns - For capturing general trends with less noise, a larger span (0.75) and 30-minute intervals could be sufficient - If highly precise counts are needed, maintaining the 10-minute interval is recommended - Further validation with additional observation periods would strengthen these findings

The results can help optimize future data collection efforts, potentially reducing sampling frequency while maintaining sufficient accuracy for trend analysis. This approach allows for more efficient monitoring of monarch populations without sacrificing the ability to detect important patterns.