Piecewise Regression

Table of contents

To illustrate piecewise regression, we fit a two-piece linear-linear model.

To reproduce the results, it is necessary to prepare the data set, plot base, and training and test data sets, as outlined in the "Data Preparation" section.

Preparation

Loading Required Packages and Data

Load the necessary packages, data sets, and other supporting files. Each element serves a specific purpose:

- **`tidyverse`**: For data manipulation and visualisation.
- **`lme4` and `lmerTest`**: To fit and analyse mixed-effects models.
- **`caret`**: To compute model performance indices.

- `plot_base`: A pre-configured ggplot object for visualisation.
- **Training and Test Data sets**: Required for cross-validation.

```r
# Load necessary packages
library(tidyverse)
library(lme4)
library(lmerTest)
library(caret)

# Load the data set
load("data/wido.rdata")

# Load the pre-configured plot base
plot_base <- readRDS("objects/plot_base.rds")

# Load training and test datasets for cross-validation
training_datasets <- readRDS("objects/training_datasets.rds")
test_datasets <- readRDS("objects/test_datasets.rds")
```

Create time variables

Create time variables for the parameters of the segments:

- postD is a dummy variable with 0 for all measurements before the transition and 1 for all measurements after. This quantifies the shift in life satisfaction level post-transition.
- preLin has negative values indicating the time before the transition, and is 0 after the transition. This captures the rate of change in life satisfaction pre-transition.
- postLin, is 0 before the transition and has positive values afterward indicating the time after the transition. This captures the rate of change in life satisfaction post-transition.

The intercept captures the life satisfaction level before the transition.

```r
# Create time variables
wido <- wido %>%
  mutate(postD = if_else(mnths <= 0, 0, 1),
         preLin = if_else(mnths <= 0, mnths, 0),
         postLin = if_else(mnths <= 0, 0, mnths))
```

To avoid multicollinearity because we use multiple (correlated) time variables in analysis, standardise the preLin and postLin variables.

```r
# Standardise preLin and postLin
wido$preLin_s <- scale(wido$preLin)
wido$postLin_s <- scale(wido$postLin)
```

Analysis

Fitting the Model

Fit the piecewise model using the newly created (standardised) time variables. This model includes both fixed and random effects for the time terms to account for person-specific trajectories.

```r
# Fit the piecewise model
pw <- lmer(
  lifesatisfaction ~ postD + preLin_s + postLin_s +
    (postD + preLin_s + postLin_s | id),
  data = wido)

# Display the summary of the model
summary(pw)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: lifesatisfaction ~ postD + preLin_s + postLin_s + (postD + preLin_s +
    postLin_s | id)
   Data: wido

REML criterion at convergence: 5239.8

Scaled residuals:
    Min      1Q  Median      3Q     Max
-4.9090 -0.4860  0.0641  0.5612  3.9181

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 id       (Intercept) 0.76547  0.8749
          postD       0.61449  0.7839   -0.44
          preLin_s    0.05851  0.2419    0.34 -0.09
```

```
          postLin_s   0.04506  0.2123    0.06 -0.19 -0.40
 Residual             0.35199  0.5933
Number of obs: 2322, groups:  id, 208

Fixed effects:
             Estimate Std. Error        df t value Pr(>|t|)
(Intercept)   5.17060    0.06676 206.50957  77.451  < 2e-16 ***
postD        -0.42436    0.07103 211.09018  -5.974 9.70e-09 ***
preLin_s     -0.16439    0.03013  90.22569  -5.456 4.22e-07 ***
postLin_s     0.23386    0.03165  63.12876   7.389 4.15e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
          (Intr) postD  prLn_s
postD     -0.509
preLin_s   0.223 -0.301
postLin_s  0.247 -0.366 -0.061
```

```r
# Compute confidence intervals for the model parameters
round(confint(pw), 2)
```

```
Computing profile confidence intervals ...

            2.5 % 97.5 %
.sig01       0.78   0.98
.sig02      -0.58  -0.27
.sig03       0.08   0.56
.sig04      -0.27   0.33
.sig05       0.66   0.91
.sig06      -0.39   0.26
.sig07      -0.47   0.21
.sig08       0.18   0.31
.sig09      -1.00   0.24
.sig10       0.13   0.29
.sigma       0.57   0.61
(Intercept)  5.04   5.30
postD       -0.56  -0.28
preLin_s    -0.23  -0.10
postLin_s    0.17   0.30
```

Visualisation

Bootstrapping Confidence Intervals

Use bootstrapping to estimate the confidence intervals for the predicted values of the model. This provides a robust measure of uncertainty.

```
# For reproducibility
set.seed(123)

# Bootstrapping for confidence intervals of the predictions
boot_results <- bootMer(pw, FUN = function(x) predict(x, newdata = wido,
↪  re.form = NA),
                        nsim = 1000)

# Extract the 95% confidence intervals from the bootstrapped results
ci <- apply(boot_results$t, 2, quantile, probs = c(0.025, 0.975))

# Assign the lower and upper bounds to the data
wido$lower_bound <- ci[1, ]
wido$upper_bound <- ci[2, ]
```

Predicting Average and Individual Trajectories

Predict both the population-level (fixed effects) and individual-level (random effects) trajectories of life satisfaction.

```
# Predict population-level trajectories based on fixed effects
wido$lifesatisfaction_pw_f <- predict(pw, newdata = wido, re.form = NA)

# Predict individual-level trajectories based on random effects
wido$lifesatisfaction_pw_r <- predict(pw, newdata = wido, re.form = NULL)
```

Selecting a Random Sample for Plotting

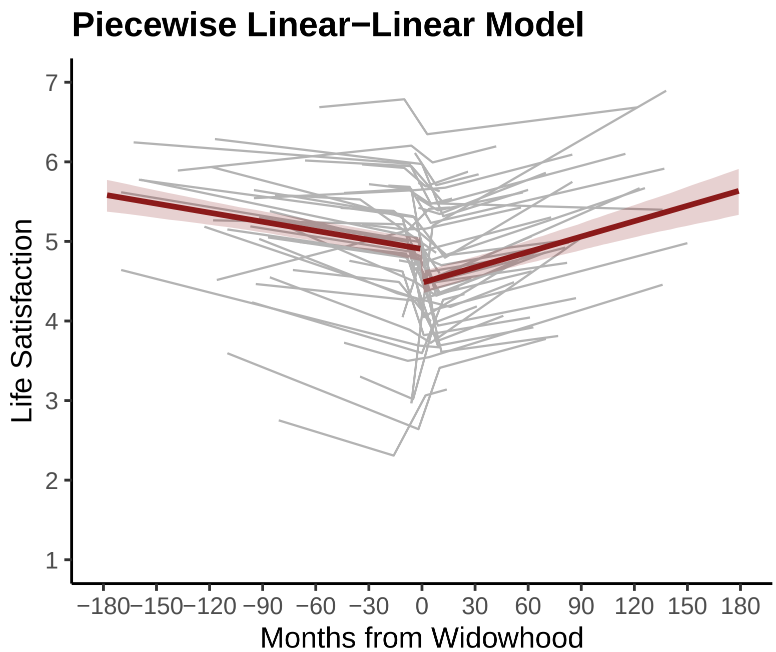
For better visualisation, select a random sample of individuals to display their individual trajectories.

```r
# For reproducibility
set.seed(123)

# Randomly sample 50 participants
rsample_ids <- sample(unique(wido$id), 50)

# Filter the data to include only the randomly selected participants
wido_rsample <- wido %>%
  filter(id %in% rsample_ids)
```

Creating the Plot

Combine all elements to create the plot, which includes individual trajectories, the population trajectory, and the confidence interval of the population trajectory.

```r
# Create the plot using the pre-configured plot base
plot_base +
  geom_line(
    data = wido_rsample,
    aes(mnths, lifesatisfaction_pw_r, group = id),
    color = "grey70",
    linewidth = 0.4
  ) +
  geom_line(
    data = wido,
    aes(
      x = mnths,
      y = ifelse(mnths == 0, NA, lifesatisfaction_pw_f)
      ),
    color = "firebrick4",
    linewidth = 1
  ) +
  geom_ribbon(
    data = wido %>% filter(mnths != 0),
    aes(ymin = lower_bound, ymax = upper_bound, x = mnths),
    alpha = 0.2,
    fill = "firebrick4"
  ) +
  ggtitle("Piecewise Linear-Linear Model") +
  theme(plot.title = element_text(size = 13, face = "bold"))
```



Model Performance

Evaluating the Model

Assess the model's performance using the Bayesian Information Criterion (BIC), R-squared (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

```
# Compute BIC for the fitted model
round(BIC(pw), 2)
```

```
[1] 5356.08
```

```
# Calculate R², MAE, and RMSE for the fixed effects predictions
data.frame(
  R2_FE = round(R2(wido$lifesatisfaction_pw_f, wido$m_lifesat_per_mnth), 2),
  MAE_FE = round(MAE(wido$lifesatisfaction_pw_f, wido$m_lifesat_per_mnth),
   ↪  2),
  RMSE_FE = round(RMSE(wido$lifesatisfaction_pw_f, wido$m_lifesat_per_mnth),
   ↪  2)
)
```

```
  R2_FE MAE_FE RMSE_FE
1  0.27   0.29    0.39
```

```
# Calculate R², MAE, and RMSE for the random effects predictions
data.frame(
  R2_RE = round(R2(wido$lifesatisfaction_pw_r, wido$lifesatisfaction), 2),
  MAE_RE = round(MAE(wido$lifesatisfaction_pw_r, wido$lifesatisfaction), 2),
  RMSE_RE = round(RMSE(wido$lifesatisfaction_pw_r, wido$lifesatisfaction), 2)
)
```

```
  R2_RE MAE_RE RMSE_RE
1  0.77    0.4    0.53
```

Cross-Validation

To assess the replicability of the model, perform cross-validation using the training and test datasets. For each training dataset, fit the model and compute performance metrics for the associated test dataset R^2, MAE, and RMSE.

```
# Initialise vectors to store performance metrics
R2_values <- c()
MAE_values <- c()
RMSE_values <- c()

# Perform cross-validation
for (i in 1:length(training_datasets)) {
  # Get the current training and test dataset
  training_data <- training_datasets[[i]]
  test_data <- test_datasets[[i]]

  # Create time variables
  training_data <- training_data %>%
  mutate(postD = if_else(mnths <= 0, 0, 1),
         preLin = if_else(mnths <= 0, mnths, 0),
         postLin = if_else(mnths <= 0, 0, mnths))

  test_data <- test_data %>%
  mutate(postD = if_else(mnths <= 0, 0, 1),
         preLin = if_else(mnths <= 0, mnths, 0),
         postLin = if_else(mnths <= 0, 0, mnths))
```

```
  # Standardise preLin and postLin
  training_data$preLin_s <- scale(training_data$preLin)
  training_data$postLin_s <- scale(training_data$postLin)

  test_data$preLin_s <- scale(test_data$preLin)
  test_data$postLin_s <- scale(test_data$postLin)

  # Fit the model
  pw <- lmer(
    lifesatisfaction ~ postD + preLin_s + postLin_s +
      (postD + preLin_s + postLin_s | id),
    data = training_data)

  # Predict fixed effects
  test_predictions <- predict(pw, test_data, re.form = NA)

  # Compute average test trajectory
  test_data <- test_data %>%
    group_by(mnths) %>%
    mutate(m_lifesat_per_mnth = mean(lifesatisfaction, na.rm = TRUE))

  # Calculate performance metrics
  R2_values <- c(R2_values, R2(test_predictions,
 ↪  test_data$m_lifesat_per_mnth))
  MAE_values <- c(MAE_values, MAE(test_predictions,
 ↪  test_data$m_lifesat_per_mnth))
  RMSE_values <- c(RMSE_values, RMSE(test_predictions,
 ↪  test_data$m_lifesat_per_mnth))
}

# Compute average performance metrics (mean)
  average_R2 <- mean(R2_values)
  average_MAE <- mean(MAE_values)
  average_RMSE <- mean(RMSE_values)

# Compute average performance metrics (SD)
  sd_R2 <- sd(R2_values)
  sd_MAE <- sd(MAE_values)
  sd_RMSE <- sd(RMSE_values)

# Combine the mean and standard deviation into one data.frame
combined_metrics <- data.frame(
```

```r
  Metric = c("R²", "MAE", "RMSE"),
  Mean = round(c(average_R2, average_MAE, average_RMSE), 2),
  SD = round(c(sd_R2, sd_MAE, sd_RMSE), 2)
)

# Print the combined metrics
print(combined_metrics)
```

```
  Metric Mean   SD
1     R² 0.10 0.07
2    MAE 0.58 0.08
3   RMSE 0.76 0.13
```