Nonlinear Regression

Table of contents

To illustrate nonlinear regression we use a model that combines a Gaussian and a logistic function.

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.

- **nls.multstart**: To identify reasonable starting values for the nonlinear regression.
- **nlme**: To fit the nonlinear regression model.
- **boot**: To calculate the bootstrapped 95% CI.
- **caret**: To compute model performance indices.
- **plot_base**: A pre-configured ggplot object for visualisation.
- **Training and Test Data sets**: Required for cross-validation.

```
# Load necessary packages
library(tidyverse)
library(nls.multstart)
library(nlme)
library(boot)
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 Nonlinear Function

Create a custom nonlinear function by combining the formula for a Gaussian function and the formula for a logistic function using a plus sign.

```
# Define the nonlinear function
nonlinfunc <- function(mnths, bas, amp, wid, cen, newequi) {
  return ((bas + amp * exp(-(mnths - cen)^2 / (2 * wid^2))) + (newequi*(1 /
   ↪  (1 + exp(-(mnths - cen) / wid)))
  ))
}
```

Analysis

Identify Starting Values

We fit the nonlinear regression model including random effects in `nlme`. Because this requires starting values for the parameters, we estimate the model without random effects, using `nls_multstart`. This function repeatedly fits the model, each time using different starting values. The results of the model without random effects will inform the specification of starting values for the model with random effects.

```r
# Fit the model without random effects, specify reasonable limits for the
↪  starting values of the parameters
nonlin_norandomeffects <- nls_multstart(lifesatisfaction ~ nonlinfunc(mnths,
↪  bas, amp, wid, cen, newequi),
                           data = wido,
                           lower=c(bas=-7, amp=-7, wid=-200, cen=-200,
                            ↪  newequi=-7),
                           upper=c(bas=7, amp=7, wid=200, cen=200, newequi=7),
                           start_lower = c(bas=-7, amp=-7, wid=-200, cen=-200,
                            ↪  newequi=-7),
                           start_upper = c(bas=7, amp=7, wid=200, cen=200,
                            ↪  newequi=7),
                           iter = 500,
                           supp_errors = "Y")

summary(nonlin_norandomeffects)
```

```
Formula: lifesatisfaction ~ nonlinfunc(mnths, bas, amp, wid, cen, newequi)

Parameters:
         Estimate Std. Error t value Pr(>|t|)
bas       4.91869    0.04360 112.815  < 2e-16 ***
amp      -0.67021    0.08010  -8.367  < 2e-16 ***
wid     -10.85987    1.62611  -6.678 3.01e-11 ***
cen       5.01145    1.61313   3.107  0.00192 **
newequi   0.28449    0.05627   5.056 4.62e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.081 on 2317 degrees of freedom
```

```
Number of iterations to convergence: 44
Achieved convergence tolerance: 1.49e-08
```

Fitting the Model

We use these results to specify starting values for the model with random effects. We add random effects for the "baseline"- and "new equilibrium"-parameters.

```
# Fit the model with the identified starting values and random effects for
↪  baseline and new equilibrium
nonlin_randomeffects_bas_newequi <- nlme(lifesatisfaction ~ nonlinfunc(mnths,
↪  bas, amp, wid, cen, newequi),
                                      data = wido,
                                      fixed= bas + amp + wid + cen + newequi
                                       ↪  ~ 1,
                                      random = bas + newequi ~ 1,
                                      groups = ~ id,
                                      start = c(bas=5.2, amp=-0.7, wid=10.9,
                                       ↪  cen=5.0, newequi=-0.3))

summary(nonlin_randomeffects_bas_newequi)
```

```
Nonlinear mixed-effects model fit by maximum likelihood
  Model: lifesatisfaction ~ nonlinfunc(mnths, bas, amp, wid, cen, newequi)
  Data: wido
       AIC     BIC    logLik
  5307.388 5359.14 -2644.694

Random effects:
 Formula: list(bas ~ 1, newequi ~ 1)
 Level: id
 Structure: General positive-definite, Log-Cholesky parametrization
         StdDev    Corr
bas      0.8598767 bas
newequi  0.9276880 -0.384
Residual 0.6248258

Fixed effects:  bas + amp + wid + cen + newequi ~ 1
            Value Std.Error   DF   t-value p-value
bas      5.167437 0.0657563 2110  78.58468   0e+00
amp     -0.716649 0.0591928 2110 -12.10703   0e+00
```

```
wid      6.539394 0.5826737 2110  11.22308   0e+00
cen      2.513732 0.6178153 2110   4.06874   0e+00
newequi -0.315205 0.0777650 2110  -4.05330   1e-04
 Correlation: 
        bas    amp    wid    cen   
amp     -0.078                     
wid      0.103  0.336              
cen     -0.068 -0.048  0.241       
newequi -0.437 -0.082 -0.011  0.157

Standardized Within-Group Residuals:
        Min          Q1         Med          Q3         Max 
-4.89041434 -0.48124295  0.06419164  0.54053713  3.50118182 

Number of Observations: 2322
Number of Groups: 208 
```

```
intervals(nonlin_randomeffects_bas_newequi)
```

```
Approximate 95% confidence intervals

 Fixed effects:
             lower       est.      upper
bas      5.0386217  5.1674368  5.2962518
amp     -0.8326063 -0.7166491 -0.6006918
wid      5.3979505  6.5393944  7.6808384
cen      1.3034460  2.5137316  3.7240172
newequi -0.4675448 -0.3152051 -0.1628653

 Random Effects:
  Level: id 
                      lower       est.      upper
sd(bas)           0.7682378  0.8598767  0.9624467
sd(newequi)       0.8057144  0.9276880  1.0681267
cor(bas,newequi) -0.5197132 -0.3835289 -0.2283371

 Within-group standard error:
    lower      est.     upper 
0.6052182 0.6248258 0.6450687 
```

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.

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

# To avoid convergence issues, we adapt the control values for the nlme fit
 ↪  (increase iterations, etc.)
control_options <- nlmeControl(
  maxIter = 2000,    # Increase max number of iterations
  pnlsMaxIter = 500, # Increase the max number of iterations for the PNLS
   ↪  step
  msMaxIter = 2000,  # Increase the maximum iterations for the optimization
   ↪  step
    pnlsTol = 0.1,   # Relax tolerance for PNLS step
)

# Define a function to refit the nonlinear mixed-effects model and generate
 ↪  predictions
predict_fun <- function(data, indices) {
  # Resample the data using the bootstrap indices
  boot_data <- data[indices, ]

  # Refit the nonlinear mixed-effects model on the resampled data
  boot_model <- nlme(
    lifesatisfaction ~ nonlinfunc(mnths, bas, amp, wid, cen, newequi),
    data = boot_data,
    fixed = bas + amp + wid + cen + newequi ~ 1,
    random = bas + newequi ~ 1,
    groups = ~ id,
    start = c(bas=5.2, amp=-0.7, wid=10.9, cen=5.0, newequi=-0.3),
    control = control_options
  )

  # Predict on the original dataset
  # Predict using only fixed effects (set level = 0)
  return(predict(boot_model, newdata = data, level = 0))
}
```

```
# Perform the bootstrap
boot_results <- boot(data = wido, statistic = predict_fun, R = 1000, sim =
 ↪  "ordinary")

# Calculate 95% confidence intervals from bootstrapped results
ci <- apply(boot_results$t, 2, quantile, probs = c(0.025, 0.975), na.rm = T)

# Assign the results to the original data frame
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_nl_fix <- predict(nonlin_randomeffects_bas_newequi,
 ↪  wido, level = 0)

# Predict individual-level trajectories based on random effects
wido$lifesatisfaction_nl_rand <- predict(nonlin_randomeffects_bas_newequi,
 ↪  newdata = wido)
```

Selecting a Random Sample for Plotting

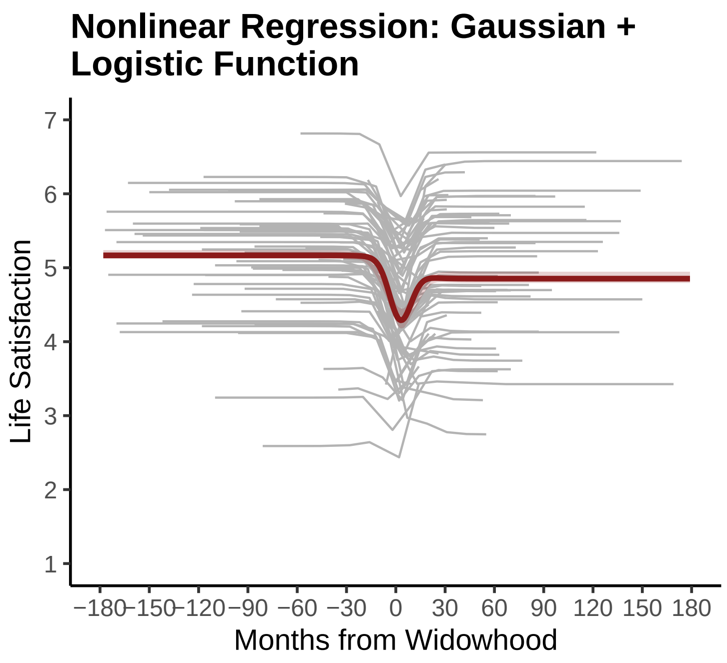
For better visualisation, select a random sample of individuals to display their individual trajectories.

```
# 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(x = mnths, y = lifesatisfaction_nl_rand, group = id),
    color = "grey70", linewidth = 0.4
  ) +
  geom_ribbon(
    data = wido,
    aes(x = mnths, ymin = lower_bound, ymax = upper_bound),
    fill = "firebrick4", alpha = 0.2
  ) +
  geom_line(
    data = wido,
    aes(x = mnths, y = lifesatisfaction_nl_fix),
    color = "firebrick4", linewidth = 1
  ) +
  ggtitle("Nonlinear Regression: Gaussian + \nLogistic Function") +
  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).

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

```
[1] 5359.14
```

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

```
  R2_FE R2_FE.1 R2_FE.2
1  0.28    0.29    0.39
```

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

```
  R2_RE R2_RE.1 R2_RE.2
1  0.73    0.43    0.58
```

Cross-Validation

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

```r
# Initialize vectors to store the performance metrics
R2_values <- c()
RMSE_values <- c()
MAE_values <- c()

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

  # Fit the nonlinear model using nlme
  nlme_model <- nlme(
    lifesatisfaction ~ nonlinfunc(mnths, bas, amp, wid, cen, newequi),
    data = training_data,
    fixed = bas + amp + wid + cen + newequi ~ 1,
    random = bas + newequi ~ 1 | id,
    start = c(bas=5.2, amp=-0.7, wid=10.9, cen=5.0, newequi=-0.3),
  )

  # Predict fixed effects
  predictions <- predict(nlme_model, newdata = test_data, level = 0)

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

  # Compute performance metrics
  R2_value <- R2(predictions, test_data$mean_ls)
  RMSE_value <- RMSE(predictions, test_data$mean_ls)
  MAE_value <- MAE(predictions, test_data$mean_ls)

  # Store the metrics
  R2_values <- c(R2_values, R2_value)
  RMSE_values <- c(RMSE_value, RMSE_value)
  MAE_values <- c(MAE_values, MAE_value)
}

# 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(
  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 0.02
2    MAE 0.59 0.07
3   RMSE 0.74 0.00
```