Gaussian Process Regression

Table of contents

To illustrate Gaussian Process regression, we use an exponentiated quadratic kernel with input warping.

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.

Note that we ran these analyses on a Computation Server for more computational power.

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.
- **`lgpr`**: To fit the Gaussian Process regression.
- **`rstan`**: Required by `lgpr`.
- **`mvtnorm`**: For experimenting with prior distributions.
- **`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(lgpr)
library(rstan)
library(caret)
library(mvtnorm)

# 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")
```

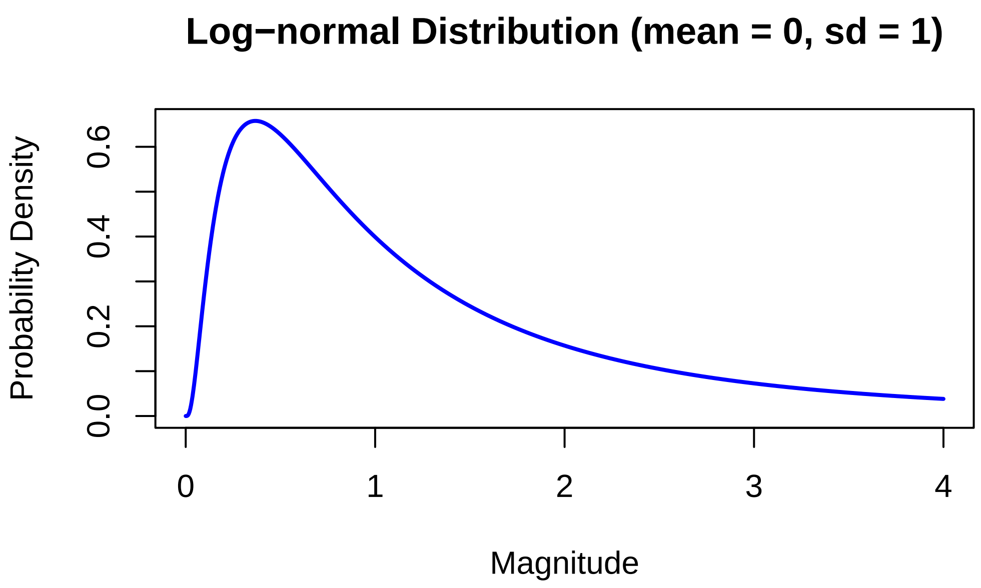
Prior specification

The `lgpr` package adopts a fully Bayesian approach, necessitating prior definitions for parameters. A prior is a probability distribution that reflects initial assumptions about a parameter. As the parameters in our kernel cannot be negative, we use log-normal distributions to define our priors, as these are non-negative distributions.

Magnitude Parameter

In the lgpr package, the response variable is automatically standardised to have a mean of 0 and an SD of 1 to speed up estimation. We therefore specify our magnitude prior so that it reflects expected changes on this standardised scale. We expect the change to not be much larger than 1 point on this scale. However, we want to specify an uninformative prior, so also allow for smaller and larger changes, by using a log-normal distribution with `mean = 0` and `sd = 1`. By modifying the `mean` and `sd` in the code below, alternative prior distributions can be explored.

```r
# Define mean and SD of the log-normal distribution
mean_priordistribution_magnitude <- 0
sd_priordistribution_magnitude <- 1

# Create a sequence of x values (range of values to plot)
x <- seq(0, 4, length.out = 1000)
# Calculate the log-normal density for each x value
y <- dlnorm(x, meanlog = mean_priordistribution_magnitude, sdlog =
 ↪  sd_priordistribution_magnitude)

# Plot the log-normal distribution to see what magnitude values are expected
 ↪  based on this prior
plot(x, y, type = "l", col = "blue", lwd = 2,
     main = "Log-normal Distribution (mean = 0, sd = 1)",
     xlab = "Magnitude", ylab = "Probability Density")
```

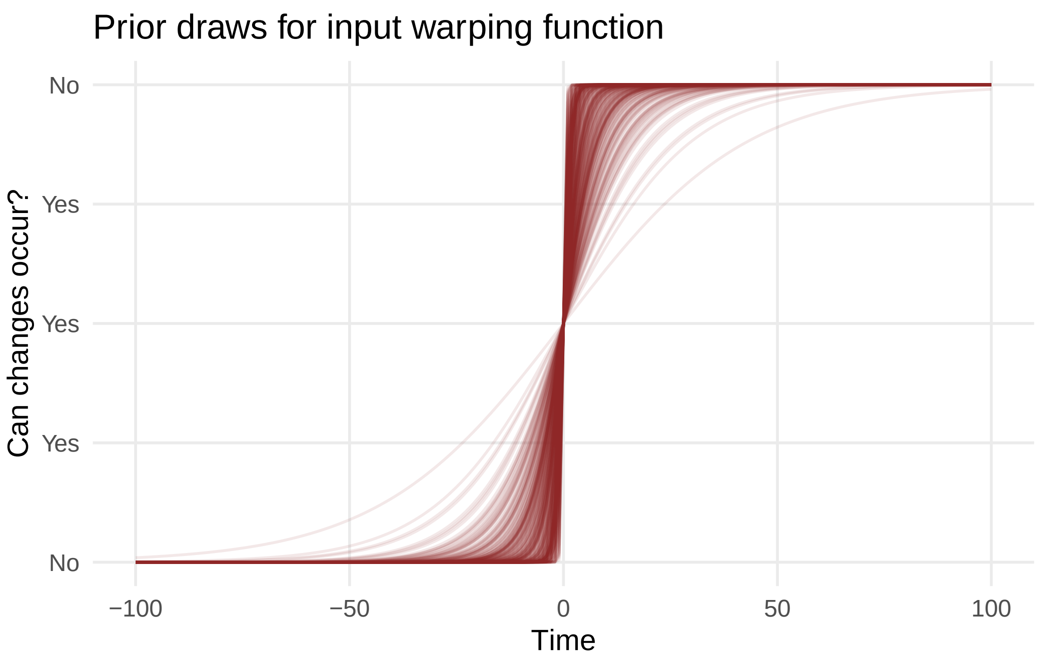


Warping Parameter

For the warping parameter, we expect most of the change to occur within 3 years before and after widowhood. Yet, again, we want to specify an uninformative prior and therefore also allow for wider or more narrow windows of change. We again use a log-normal distribution. To see how the warping function, and thereby the window of change, changes based on different priors for the warping parameter, the values of the `mean` and `sd` can be changed in the code below. The red lines in the plot indicate the time window in which this warping function allows changes to occur.

```r
# Define mean and SD of the log-normal distribution
mean_priordistribution_warping <- -0.7
sd_priordistribution_warping <- 0.9

# Take draws from this distribution
wrp_draws <- stats::rlnorm(300, meanlog = mean_priordistribution_warping,
 ↪  sdlog = sd_priordistribution_warping)

# Plot the resulting input warping function draws
lgpr:::plot_inputwarp(wrp_draws, seq(-100, 100, by = 1), alpha = 0.1) +
  theme_minimal() +
  theme(panel.grid.minor = element_blank()) +
  labs(x = "Time", y = "Can changes occur?", title = "Prior draws for input
   ↪  warping function") +
  scale_y_continuous(breaks = c(-1, -0.5, 0, 0.5, 1), labels = c("No", "Yes",
   ↪  "Yes", "Yes", "No"))
```

```
Scale for y is already present.
Adding another scale for y, which will replace the existing scale.
```

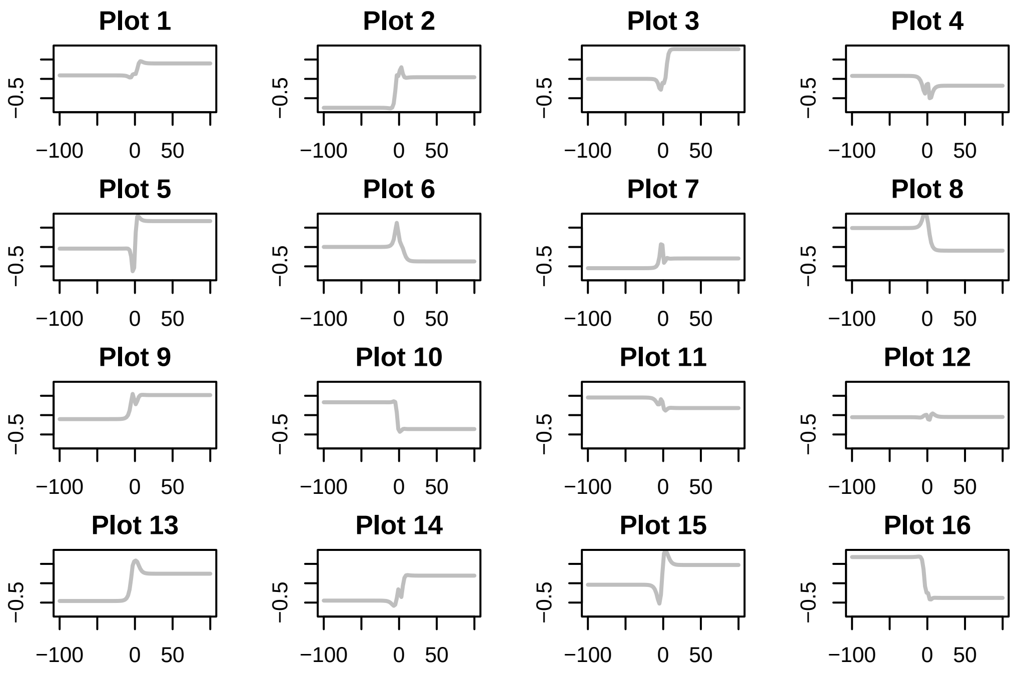


The prior for the warping function depicted above, a log-normal distribution with `mean = -0.7` and `sd = 0.9`, indicates that we expect most changes within [-36, 36] months surrounding widowhood, but also allows for changes in more narrow or broader time windows (e.g., [-100, 100] months surrounding widowhood).

Lengthscale Parameter

As the effect of the lengthscale interacts with that of the warping parameter, its prior is best identified by considering all priors specified so far in concert. We again use a log-normal distribution. In the code below, the values of the `mean` and `sd` can be modified to inspect how different prior distributions for the lengthscale result in different possible trajectories. Note that the code below can also be used to experiment with different priors for the other parameters, by keeping the lengthscale prior distribution fixed.

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

# Define mean and SD of the log-normal distribution
mean_priordistribution_lengthscale <- 0
sd_priordistribution_lengthscale <- 1

# Define warping function
warpfun <- function(time,warpingparameter){
  2 / (1 + exp( - warpingparameter * time)) - 1
}

# Define kernel with warping function
kernelGP <- function(X1, X2, lengthscale, alpha, warpingparameter,
 ↪  fun=function(x,w){x}){

  X1mat <- matrix(rep(fun(X1,warpingparameter),length(X2)),nrow=length(X1))
  X2mat <-
 ↪  matrix(rep(fun(X2,warpingparameter),length(X1)),nrow=length(X1),byrow=TRUE)

  alpha**2 * exp(- .5 * (X1mat - X2mat) * (X1mat - X2mat) / lengthscale**2 )

}

n <- 100
X <- seq(-100, 100, length=n)
mu = rep(0,length(X))

# Set up a 4x4 plotting panel
par(mfrow = c(4, 4), mar = c(2, 2, 2, 2))

# Loop to generate 16 plots
for (i in 1:16) {

# Lengthscale parameter
lengthscale1 <- dlnorm(1, meanlog = mean_priordistribution_lengthscale, sdlog
 ↪  = sd_priordistribution_lengthscale)

# Magnitude parameter (potentially modify prior distribution here!)
alpha1 <- dlnorm(1, meanlog = 0, sdlog = 1)

# Warping parameter (potentially modify prior distribution here!)
warpingparameter <- dlnorm(1, meanlog=-0.7,sdlog=0.9)
covm = kernelGP(X, X, lengthscale = lengthscale1, alpha = alpha1,
 ↪  warpingparameter = warpingparameter, fun = warpfun)

# Intercept
beta0 <- 0

  # Draw a line from the GP
  draw_f <- mvtnorm::rmvnorm(1, mean = mu, sigma = covm)
  # Plot the line
  plot(X, beta0 + draw_f, type = "l", col = "grey", lwd = 2,
       ylim = c(beta0 - 2 * alpha1, beta0 + 2 * alpha1),
       ylab = "y", xlab = "x", main = paste("Plot", i))
}
```



A log-normal prior distribution with `mean = 0` and `sd = 1`for the lengthscale, together with the magnitude and warping parameter priors identified earlier, can result in pretty different, but within this context realistic trajectories. These priors thus are relatively uninformative, while ensuring realistic bounds for the parameters. Therefore, this prior is specified. Note that in `lgpr` the magnitude parameter is called alpha.

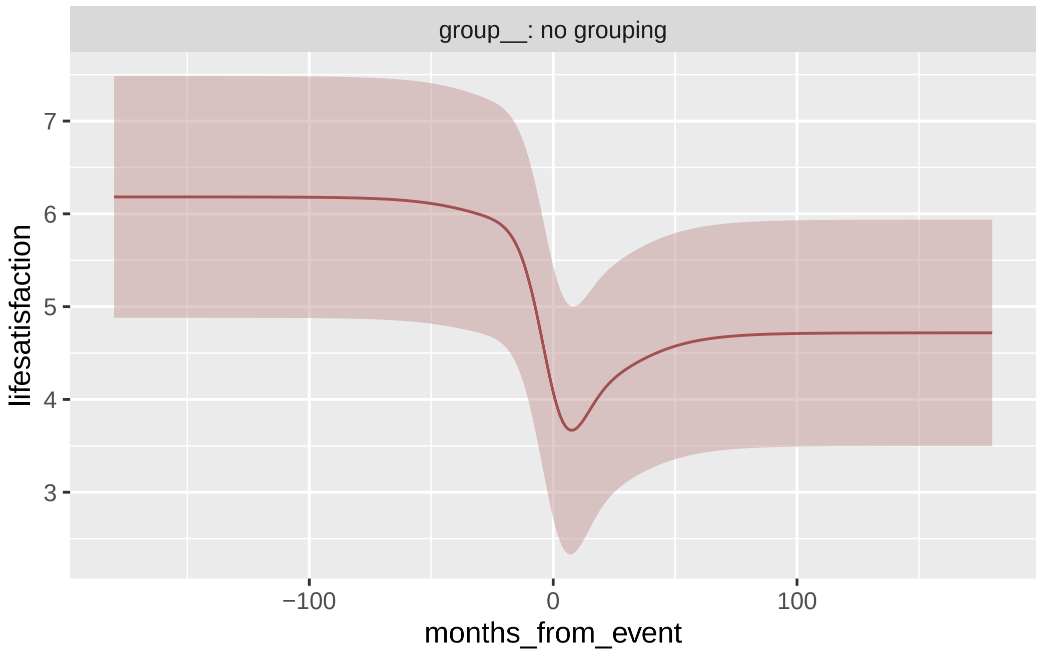
```
prior <- list(
  alpha = log_normal(0, 1),
  ell = log_normal(0, 1),
  wrp = log_normal(-0.7, 0.9)
)
```

Analysis

Fitting the Model

We first attempt to estimate person-specific deviations for all parameters (magnitude, length-scale, and warping) in addition to the overall ("fixed") effect of time. This approach is comparable to including random intercepts and slopes in a linear model.

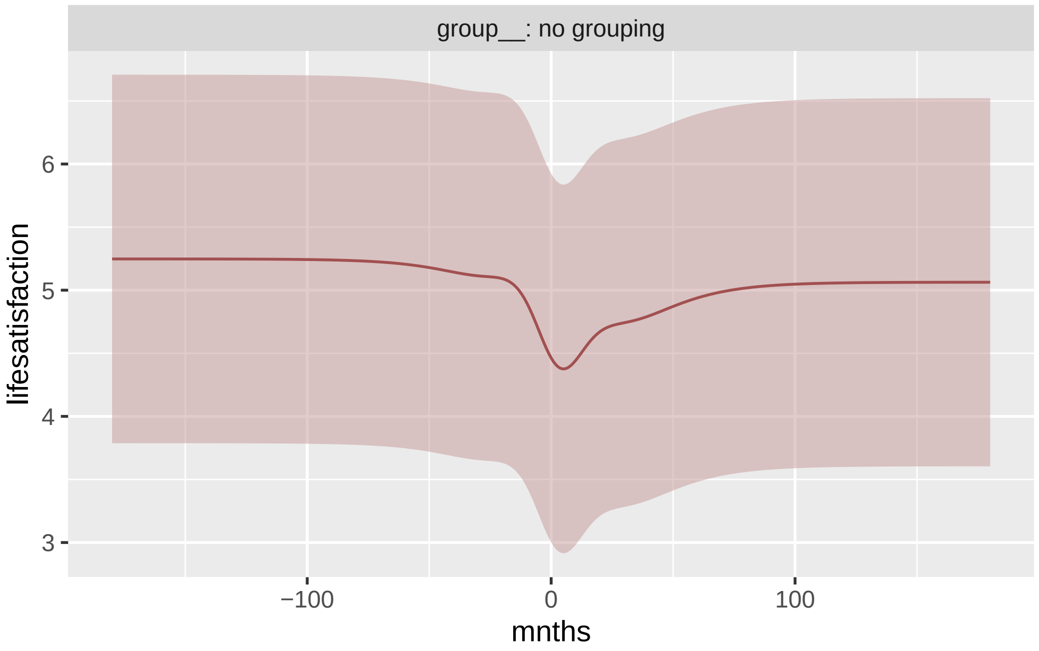
```
# For reproducibility
set.seed(123)

# Fit model
gp <- lgp(lifesatisfaction ~ gp_ns(mnths) + gp_ns(mnths) * zs(id), data =
 ↪  wido, prior = prior)
```



A quick glance at this model indicates that it acts a bit strange: it estimates more than 2 points change in life satisfaction on a scale of 1 to 7, which is twice as large as in all other models considered. We suspect the model is somewhat too complicated to be estimated reliably with this number of observations per individual.

We therefore fit a simpler model estimating stable person-specific deviations from the average trajectory, analogous to random intercepts.

```r
gp <- lgp(lifesatisfaction ~ gp_ns(mnths) + zs(id), data = wido, prior =
 ↪  prior)
```



This adjustment results in change estimates that are more in line with the results of the other models. The function below prints the posterior distributions of the parameters. The first alpha is the magnitude of the shared time effect, the second alpha is the person-specific time effect.

```
An object of class lgpfit. See ?lgpfit for more info.
Inference for Stan model: lgp.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
alpha[1] 0.43    0.01 0.23 0.20 0.29 0.37 0.50  1.04  1404 1.00
alpha[2] 0.73    0.00 0.04 0.66 0.70 0.73 0.76  0.81  2619 1.00
ell[1]   0.37    0.01 0.20 0.18 0.26 0.31 0.39  0.98   485 1.01
wrp[1]   0.05    0.00 0.02 0.03 0.04 0.05 0.06  0.12   516 1.01
sigma[1] 0.64    0.00 0.01 0.62 0.63 0.64 0.64  0.66  2597 1.00

Samples were drawn using NUTS(diag_e) at Tue Dec 17 01:12:35 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

The lgpr package standardises the response variable to a mean of zero and standard deviation of one. To interpret magnitude estimates on the original scale, they are multiplied by the original variance.

The magnitude of the shared time-effect is:

```
round(0.43*var(wido$lifesatisfaction), 2)
```

```
[1] 0.53
```

The magnitude of the person-specific time-effect is:

```
round(0.73*var(wido$lifesatisfaction), 2)
```

```
[1] 0.9
```

Visualisation

Standard Deviation

To visualise the results, we depict the shared and person-specific trajectories as predicted by the posterior means of the parameters. The posterior mean is the mean of the posterior distribution of the parameter. It is the expected value of the parameter, given the data and prior. To show the uncertainty of the estimated shared trajectory, we plot the interval between two standard deviations from the posterior mean. Within this interval we expect the true trajectory to lie with approximately 95% probability.

Predicting Average and Individual Trajectories

Predict both the shared (fixed effects) and person-specific (random effects) trajectories of life satisfaction. We create an extra function below to wrangle the predictions in the format we need for our plot.

```r
# Define time sequence
t <- seq(-178, 179, by = 1)

# Generate prediction dataset
x_pred <- new_x(wido, t, group_by = NA, x = "mnths")
```

```r
# Create function to wrangle the predictions to the correct format
process_predictions <- function(predicted) {
  # Extract and reshape predictions (mean and standard deviation)
  x <- as.data.frame(predicted@x)
  y <- as.data.frame(predicted@y_mean) %>%
    pivot_longer(cols = everything()) %>%
    dplyr::select(-name) %>%
    rename(y = value)

  std <- as.data.frame(predicted@y_std) %>%
    pivot_longer(cols = everything()) %>%
    dplyr::select(-name) %>%
    rename(std = value)

  # Combine and calculate bounds
  data <- cbind(x, y, std) %>%
    mutate(lower = y - std * 1.96,
           upper = pmin(y + std * 1.96, 7))  # Cap upper bound at 7
  return(data)
}

# Apply function for predictions for shared effect (f = fixed effect) and
↪  person-specific effect (r = random effects)
gp_pred_f <- process_predictions(pred(gp, x_pred, reduce = mean, verbose =
↪  FALSE))
gp_pred_r <- process_predictions(pred(gp, reduce = mean, verbose = FALSE))
```

Selecting a Random Sample for Plotting

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

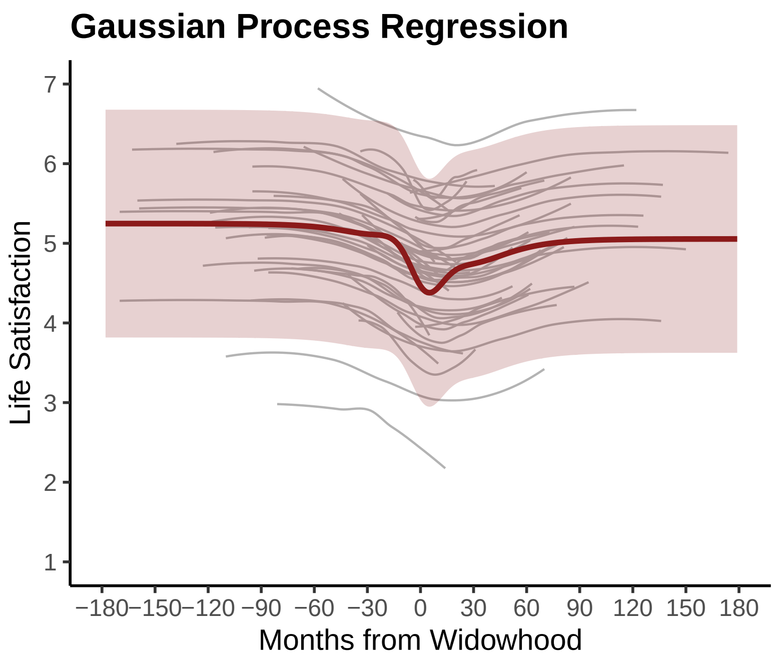
```r
# For reproducibility
set.seed(123)
```

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

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

Creating the Plot

Combine all elements to create the plot, which includes individual trajectories, the population trajectory, and its standard deviation.

```r
plot_base +
  geom_smooth(data = gp_pred_r_rsample, aes(mnths, y, group = id), se = F,
   ↪  span = 0.90, color = "grey70", linewidth = 0.4) +
  geom_ribbon(data = gp_pred_f, aes(ymin = lower, ymax = upper, x = mnths),
              alpha = 0.2, fill = "firebrick4") +
  geom_line(data = gp_pred_f, aes(x = mnths, y = y),  color = "firebrick4",
   ↪  linewidth = 1) +
  ggtitle("Gaussian Process Regression") +
  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
# Because we want to compare our predicted average trajectory, with the
↪  actual average trajectory, we need to merge the predicted dataframe with
↪  the original data
gp_pred_f <- gp_pred_f %>%
  dplyr::select(mnths, y)
gp_pred_f <- merge(wido, gp_pred_f, key = "mnths")

# Now we can calculate R², MAE, and RMSE for the fixed effects predictions
data.frame(
  R2_FE = round(R2(gp_pred_f$y, gp_pred_f$m_lifesat_per_mnth), 2),
  MAE_FE = round(MAE(gp_pred_f$y, gp_pred_f$m_lifesat_per_mnth), 2),
  RMSE_FE = round(RMSE(gp_pred_f$y, gp_pred_f$m_lifesat_per_mnth), 2)
)
```

```
  R2_FE MAE_FE RMSE_FE
1  0.32   0.28    0.38
```

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

```
  R2_RE MAE_RE RMSE_RE
1  0.63   0.51    0.67
```

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.

```r
# Initialize vectors to store the performance metrics
R2_values_gp <- c()
RMSE_values_gp <- c()
MAE_values_gp <- c()

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

  # Fit the model
  gp <- lgp(lifesatisfaction ~ gp_ns(mnths) + zs(id),
            data     = training_data,
            iter     = 500,
            chains   = 4,
            refresh  = 10,
            prior = prior)

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

  # Predict fixed effects
  x_pred <- new_x(test_data, seq((min(test_data$mnths)),
 ↪  (max(test_data$mnths)), by = 1), group_by = NA, x = "mnths")
  pred_gp_f <- pred(gp_model, x_pred, reduce = mean, verbose = FALSE)

  gp_pred_x_f <- as.data.frame(pred_gp_f@x)
  gp_pred_y_f <- as.data.frame(pred_gp_f@y_mean)
  gp_pred_y_f <- pivot_longer(gp_pred_y_f, cols = everything())
  gp_pred_y_f <- dplyr::select(gp_pred_y_f, -name)
  gp_pred_y_f <- rename(gp_pred_y_f, gp_pred_ls_f = value)
  gp_pred_all_f <- cbind(gp_pred_x_f, gp_pred_y_f)
  pred_gp_f <- gp_pred_all_f %>%
    dplyr::select(mnths, gp_pred_ls_f)

  pred_gp_f <- merge(test_data, pred_gp_f, key = "mnths")

  # Compute performance metrics
  R2_value <- R2(pred_gp_f$gp_pred_ls_f, pred_gp_f$m_lifesat_per_mnth)
  RMSE_value <- RMSE(pred_gp_f$gp_pred_ls_f, pred_gp_f$m_lifesat_per_mnth)
  MAE_value <- MAE(pred_gp_f$gp_pred_ls_f, pred_gp_f$m_lifesat_per_mnth)

  # Store the metrics
  R2_values_gp <- c(R2_values_gp, R2_value)
  RMSE_values_gp <- c(RMSE_values_gp, RMSE_value)
  MAE_values_gp <- c(MAE_values_gp, MAE_value)
}

# Compute average performance metrics (mean)
average_R2_gp <- mean(R2_values_gp)
average_RMSE_gp <- mean(RMSE_values_gp)
average_MAE_gp <- mean(MAE_values_gp)

# Compute standard deviation of performance metrics (SD)
sd_R2_gp <- sd(R2_values_gp)
sd_RMSE_gp <- sd(RMSE_values_gp)
sd_MAE_gp <- sd(MAE_values_gp)

# Combine the mean and standard deviation into one data.frame
combined_metrics_gp <- data.frame(
  Metric = c("R²", "MAE", "RMSE"),
  Mean = round(c(average_R2_gp, average_MAE_gp, average_RMSE_gp), 2),
  SD = round(c(sd_R2_gp, sd_MAE_gp, sd_RMSE_gp), 2)
)

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

```
  Metric Mean   SD
1     R² 0.12 0.04
2    MAE 0.72 0.16
3   RMSE 0.90 0.20
```