Joint model illustration and evaluation

Sean van der Merwe

2024-03-18

Table of contents

The purpose of this document is to generate a single sample that mimics what was observed in an actual study, then fit the proposed model(s) to that sample, then illustrate and evaluate the fit.

The code use here should then be transferable to the real data set or any other simulation of similar data.

Introduction

Vrijens et al. (2006) evaluated the influence of a pharmaceutical care program on the adherence and persistence of patients to a prescribed regimen of once-daily atorvastatin, a medication for managing cholesterol levels and mitigating the risk of cardiovascular diseases. The study divided participants into two cohorts: a control group that received standard care and an intervention group engaged in a pharmaceutical care program. The study required scheduled follow-up visits over 12 months based on the dosing frequency prescribed. Patients' adherence is a continuous proportion outcome, tracked monthly and reflecting compliance in the last month. This adherence data is derived from electronically compiled dosing histories. Persistence is a time-to-event outcome that measures the treatment regimen's duration from initiation to discontinuation. The initial day of the study was recorded as the day the patients first opened their pill containers, initiating the baseline period that lasted until the second visit to the pharmacy. The duration of this baseline period varied among individuals.

The Kumaraswamy distribution is written as

$$f\left(y;\alpha_1,\alpha_2\right)=\alpha_1\alpha_2 y^{\alpha_1-1}\left(1-y^{\alpha_1}\right)^{\alpha_2-1},$$

where $0<y<1$ and $\alpha_1,\alpha_2>0$ are shape parameters. An alternative parameterization of the Kumaraswamy distribution involves expressing α_1 and α_2 in terms of the q^{th} quantile location parameter κ_q, defined such that $F^{-1}\left(q\right)=\kappa_q$, where F^{-1} is the inverse of the CDF. In this parameterization, α_1 and α_2 are given by $\alpha_1=\frac{\log\left(1-(1-q)^{\frac{1}{\alpha_2}}\right)}{\log\left(\kappa_q\right)}$ and $\alpha_2=\frac{\log(1-q)}{\log(1-e^{-\psi})}$, with $\psi>0$ as the precision parameter.

Suppose that y_{ij} is the continuous proportions outcome for patient $i=1,\dots,I$ and time point $j=1,\dots,J_i$. The regression model of the q^{th} conditional quantile of y_{ij} is written as

$$\kappa_{q_{ij}}=g^{-1}(\eta_i\left(t_j\right)),$$

where

$$\eta_i\left(t_j\right)=\left(\beta_0+b_{0i}\right)+\left(\beta_{\text{time}}+b_{\text{time}_i}\right)t_j+\mathbf{x}_i\beta_{\text{cov}}$$

is the linear predictor corresponding to patient i at time point t_j, g is the logit link function defined via its inverse as $g^{-1}\left(a\right)=\frac{e^a}{1+e^a}$; β_0 is the fixed intercept; β_{time} is the fixed coefficient for time t_j; $\mathbf{x}_i$ is a vector of covariates for patient i (excluding time); β_{cov} is a vector of fixed coefficients corresponding to the covariates in $\{\mathbf{x}_i\}$; and b_{0i} and b_{time_i} are the random intercept and slope for patient i, respectively. Assuming that the continuous proportions data follow the Kumaraswamy distribution, the PDF for a given continuous proportions outcome $0<y_{ij}<1$ is

$$f\left(y_{ij};\alpha_{ij_1},\alpha_2\right)=\alpha_{ij_1}\alpha_2 y_{ij}^{\alpha_{ij_1}-1}\left(1-y_{ij}^{\alpha_{ij_1}}\right)^{\alpha_2-1},$$

where $\alpha_{ij_1}=\frac{\log\left(1-(1-q)^{\frac{1}{\alpha_2}}\right)}{\log\left(\kappa_{q_{ij}}\right)}$ and $\alpha_2=\frac{\log(1-q)}{\log(1-e^{-\psi})}$, and ψ is a precision parameter.

Define T_i as the time from the start of the follow-up period to the occurrence of the event of interest for patient i. Similarly, let C_i denote the censoring time for patient i, indicating the time point beyond which the patient's event status is unknown. Introduce T_i^* as the observed event time for patient i, defined as the minimum of T_i and C_i ($T_i^*=\min\left(T_i,C_i\right)$). Define ξ_i as the event indicator, where $\xi_i=1$ if the event of interest occurs ($T_i\leq C_i$) and $\xi_i=0$ if the observation is censored ($T_i>C_i$). The time-to-event submodel is formulated using the Cox proportional hazards model to analyse the time until an event of interest occurs, considering both the event time and censoring. The hazard function for patient i at time t in the Cox model is given by:

$$h_i\left(t\right)=h_0\left(t\right)\exp\left(\phi\eta_i\left(t\right)+\mathbf{w}_i\gamma\right),$$

where $h_i\left(t\right)$ denotes the hazard function for patient i at time t; $h_0\left(t\right)$ is the baseline hazard function; $\eta_i\left(t\right)$ represents the linear predictor derived from the longitudinal submodel for patient i at time t; and ϕ is the coefficient that quantifies the association between the longitudinal outcome and the hazard of the event, measuring the impact of the longitudinal process on the event risk; $\mathbf{w}_i$ is a design matrix for patient i containing additional covariates; and γ is a vector of coefficients corresponding to the additional covariates in $\mathbf{w}_i$.

In the INLA framework, the baseline hazard $h_0(t)$ is discretized into intervals to allow flexibility in modelling the hazard over time. Specifically, for each interval $k = 1, \dots, K$, the hazard within the interval is assumed to be constant and is defined in terms of the RW2 process:

$$h_0(t) = \exp(\delta_k) \quad \text{for } t \in (s_{k-1}, s_k],$$

where δ_k represents the log-baseline hazard at time t within the interval k, and $(s_{k-1}, s_k]$ denotes the time interval.

The RW2 model for the baseline hazard is formulated as follows:

$$\delta_k = 2\delta_{k-1} - \delta_{k-2} + \epsilon_k,$$

with ϵ_k being a Gaussian noise term with mean zero and variance σ^2, *i.e.*, $\epsilon_k \sim N(0, \sigma^2)$. This formulation ensures a smoothing effect, reflecting the change from δ_{k-1} to δ_k based on the previous change (from δ_{k-2} to δ_{k-1}).

The event time and censoring are explicitly considered, with the linear predictor $\eta_i(t)$ from the longitudinal submodel serving as a dynamic covariate.

The survival function for patient i, denoted as $S_i(t)$, is defined as the probability of survival beyond time t, expressed as the exponential of the negative cumulative hazard function:

$$S_i(t) = \exp(-H_i(t)),$$

where $H_i(t)$ is the cumulative hazard function for patient i. In the context of the RW2 model, the cumulative hazard is calculated as:

$$H_i(t) = \sum_{k=1}^{K(t)} \left[\int_{s_{k-1}}^{\min(t, s_k)} \exp(\delta_k + \phi\eta_i(u))\, du \right],$$

where $K(t)$ is the number of intervals up to time t.

Preliminaries and packages

Only critical packages are used for maximum portability, stability, and maintainability.

```r
options(scipen = 12)
library(tidyverse)
library(knitr)
library(INLA) # Bayesian inference with INLA
library(PermAlgo) # Permutation algorithm for survival times
library(mvtnorm)
library(survival)
```

```r
set.seed(111)
```

```r
kable <- function(...) {
  knitr::kable(..., format = "html", table.attr = "style = \"color: white;\"")
}
shortestinterval <- function(postsims, width=0.95) { # Coded by Sean van der Merwe, UFS
  postsims |> sort() -> sorted.postsims
  round(length(postsims)*width) -> gap
  sorted.postsims |> diff(gap) |> which.min() -> pos
  sorted.postsims[c(pos, pos + gap)] }

theme_set(theme_bw())
```

Single simulation

A single simulation is generated, but with a lot of subjects to replicate a large study.

```r
# Define simulation parameters
ParamList <- list(
  nsujet = 200, # Number of individuals
  q = 0.5, # Quantile of longitudinal measurements
  # Continuous proportion data
  beta_0 = 2.5, # Intercept
  beta_1 = -0.3, # Slope
  beta_2 = -0.5, # Binary covariate intercept
  beta_3 = 0.1, # Continuous covariate
  beta_4 = -0.2, # Binary covariate slope
  psi = 0.3, # Kumaraswamy scale parameter
  SD1 = 0.1, # SD of random intercept
  SD2 = 0.1, # SD of random slope
  rho = 0.2, # Random intercept/slope correlation coefficient
  # Survival data
  phi = -0.3, # Effect of Y's linear predictor on the risk of event
  gamma_1 = c(-0.5), # Binary covariate
  gamma_2 = c(0.1), # Continuous covariate
  gapLongi = 1/2, # Gap between longitudinal measurements
  gap = 1/40, # Used to generate many time points for permutation algorithm
  followup = 10, # Follow-up time
  endpoint = 15 # Prediction endpoint
)
```

Data generation and model fitting functions

```r
# Function to simulate from qkumar distribution
rkum <- function(n, Kappa, psi, q) {
  beta <- log(1 - q)/log(1 - exp(-psi)) # Solve beta of qkumar
  alpha <- log(1 - (1 - q)^(1/beta))/log(Kappa) # Solve alpha of qkumar
  (1 - (1 - runif(n))^(1/beta))^(1/alpha) # Transform using inverse CDF
}
```

```r
# This is a function that takes the parameter list as input and produces data sets
gendata <- function(p) {
  Covar <- p$rho*p$SD1*p$SD2
  Sigma <- matrix(c(p$SD1^2, Covar, Covar, p$SD2^2), 2) # Random effects covariance matrix
  mestime <- seq(0, p$followup, p$gap) # Measurement times
  timesLongi <- mestime[sapply(seq(0, p$followup, p$gapLongi), \(timepoint)
                               which.min(abs(mestime - timepoint)))] # Visit times
  time <- rep(mestime, p$nsujet) # Time column
  nmesindiv <- p$followup/p$gap + 1 # Max number of individual measurements
  nmesy <- nmesindiv*p$nsujet # Max total number of longitudinal measurements
  idY <- rep(1:p$nsujet, each = nmesindiv) # Individual ID
  MVnorm <- rmvnorm(p$nsujet, rep(0, 2), Sigma)
  b_int <- rep(MVnorm[, 1], each = nmesindiv) # Random intercept Y
  b_slo <- rep(MVnorm[, 2], each = nmesindiv) # Random slope Y
  binX <- rbinom(p$nsujet, 1, 0.5) # Binary covariate
  ctsX <- rnorm(p$nsujet, 1, 0.5) # Continuous covariate
  # Replicate binX and ctsX for each time point of each subject
  rep_binX <- rep(binX, each = nmesindiv)
  rep_ctsX <- rep(ctsX, each = nmesindiv)

  # Linear predictors
  linPredY <- (p$beta_0 + b_int) +
    (p$beta_1 + b_slo)*time +
    p$beta_2*binX[idY] +
    p$beta_3*ctsX[idY] +
    p$beta_4*binX[idY]*time

  # Continuous outcome Y
  Y <- rkum(nmesy, Kappa = plogis(linPredY), psi = p$psi, q = p$q) # Logit link usage

  # Permutation algorithm to generate survival times dependent on linear predictors
  DatTmp <- permalgorithm(p$nsujet, nmesindiv,
                          Xmat = matrix(c(linPredY, rep_binX, rep_ctsX), nrow = p$nsujet*nmesindiv)
                          eventRandom = round(rexp(p$nsujet, 0.003) + 1, 0), # ~40% events
                          censorRandom = runif(p$nsujet, 1, nmesindiv), # Uniform random censoring
                          XmatNames = c("linPredY", "binPred", "otherPred"), # Association
                          betas = c(p$phi, p$gamma_1, p$gamma_2) # Association and hazard regressio
                         )
  # Extract last line for each ID (= event/censoring time)
  survDat <- DatTmp |>
    group_by(Id) |>
    summarise(
      Id = last(Id),
      eventTimes = mestime[last(Stop) + 1],
      Event = last(Event)
      ) |> as.data.frame()
  survDat$binX <- binX[survDat$Id]  # Align binX with survival data
  survDat$ctsX <- ctsX[survDat$Id]  # Align ctsX with survival data
  DatTmp <- DatTmp |> mutate(
    time = mestime[Start + 1], # Measurement time of the longitudinal outcomes
    Uid = paste(Id, time) # Unique identifier to match covariates and longitudinal outcomes
  )
  longDat3 <- merge(DatTmp[, c("Uid", "Id", "time")],
                    data.frame(Uid = paste(idY, time),
                               binX = rep_binX,
                               ctsX = rep_ctsX,
                               Y = Y), by = "Uid")
  longDat <- sapply(longDat3[longDat3$time %in% timesLongi, -1], as.numeric)
  longDat <- as.data.frame(longDat[order(longDat[, "Id"], longDat[, "time"]), ])
  list(survDat = survDat, longDat = longDat)
}

fitINLA <- function(d, q = 0.5) {
  survDat <- d$survDat
  longDat <- d$longDat

  # Model fit
  NL <- nrow(longDat)
  NS <- nrow(survDat)

  # Prepare data for INLA
  # Cox model structure for survival (with Bayesian smooth splines for the baseline hazard, i.e., "r
  CoxExt <- inla.coxph(YS ~ -1 + Intercept + binX + ctsX,
                       control.hazard = list(model = "rw2", scale.model = TRUE,
                                             diagonal = 1e-2, constr = TRUE,
                                             hyper = list(prec = list(prior = "pc.prec", param = c(0
                       data = list(YS = inla.surv(time = c(survDat$eventTimes), event = c(survDat$Ev
                                   Intercept = rep(1, NS),
                                   eta = survDat$Id,
                                   binX = survDat$binX,
                                   ctsX = survDat$ctsX
                                  )
                       )

  # Time weight for time-dependent covariates in survival (i.e., fixed and random slope)
  t.weight <- CoxExt$data$baseline.hazard.time + 0.5*CoxExt$data$baseline.hazard.length
  nsCox <- nrow(CoxExt$data) # Number of intervals for survival
  NY <- NL + nsCox
  IDcox <- CoxExt$data$expand..coxph # Random effects ID for survival
  CoxExt$data$eta <- 1:nsCox # Unique ID for shared linear predictor Y

  # Replicate binX and ctsX according to structure of CoxExt$data
  Rep_binX <- survDat$binX[CoxExt$data$expand..coxph]
  Rep_ctsX <- survDat$ctsX[CoxExt$data$expand..coxph]
  Rep_intX <- survDat$binX[CoxExt$data$expand..coxph]*t.weight

  # Merge replicated covariates with CoxExt$data
  IDBC <- data.frame(CoxExt$data, binX = Rep_binX, intX = Rep_intX, ctsX = Rep_ctsX)
```

```r
covariates <- data.frame(
  InteY = rep(1, NY), # Intercept Y
  TIMEY = c(longDat$time, t.weight), # Time Y
  binXY = c(longDat$binX, IDBC$binX), # Binary covariate X for Y
  intX = c(longDat$time*longDat$binX, IDBC$intX), # Interaction (treatment-by-time)
  ctsXY = c(longDat$ctsX, IDBC$ctsX), # Continuous covariate X for Y
  IDY = c(longDat$Id, IDcox), # Random intercept Y
  IDY_s = c(NS + longDat$Id, NS + IDcox), # Random slope Y
  WY_s = c(longDat$time, t.weight), # Weight random slope Y
  u1 = c(rep(NA, NL), CoxExt$data$eta), # Y association with survival
  w1 = c(rep(NA, NL), rep(-1, nsCox)) # Y weight association with survival
)

Y.joint <- list(
  Y = c(longDat$Y, rep(NA, nsCox)), # Y
  Y.eta = c(rep(NA, NL), rep(0, nsCox)) # Y association with survival
)

jointdf <- data.frame(covariates, Y.joint)
joint.DataCox <- c(as.list(inla.rbind.data.frames(jointdf, CoxExt$data)), CoxExt$data.list)
Yjoint <- joint.DataCox[c("Y", "Y.eta", "y..coxph")] # Outcomes (longitudinal and survival)
joint.DataCox$Y <- Yjoint

# Update formula from the Cox model structure to add longitudinal part
formulaJ <- update(CoxExt$formula,
                   Yjoint ~ . -1 + InteY + TIMEY + intX + binXY + ctsXY +
                   f(IDY, model = "iid2d", n = NS*2, constr = FALSE,
                     hyper = list(theta = list(param = c(10, 1, 1, 0)))) +
                   f(IDY_s, WY_s, copy ="IDY") +
                   f(u1, w1, model = "iid",
                     hyper = list(prec = list(initial = -6, fixed = TRUE)),
                     constr = FALSE) +
                   f(eta, copy = "u1",
                     hyper = list(beta = list(fixed = FALSE, param = c(0, 0.16),
                                              initial = 1))))

# INLA function call
inla.setOption(inla.mode = "experimental")
JMinla <- inla(verbose = FALSE,
               formula = formulaJ,
               family = c("qkumar", "Gaussian", CoxExt$family),
               data = joint.DataCox,
               control.fixed = list(mean = 0, prec = 0.16, mean.intercept = 0, prec.intercept = 0.
               control.family = list(
                 list(control.link = list(quantile = q)),
                 list(hyper = list(prec = list(initial = 12, fixed = TRUE))),
                 list()),
               E = joint.DataCox$E..coxph,
               control.inla = list(int.strategy = "eb"),
                  control.compute=list(config = TRUE)
                  )
  JMinla
}
```

```
d <- gendata(ParamList)
JMinla <- fitINLA(d, q = ParamList$q)
```

```
Warning in .recacheSubclasses(def@className, def, env): undefined subclass
"ndiMatrix" of class "replValueSp"; definition not updated
```

The code above this point was written by Dr Burger, with minimal modification by Dr van der Merwe.

Model fit illustration

The code from here on was written by Dr van der Merwe.

First we draw posterior samples.

We would like to draw about 10000 samples, but for initial testing and experimentation we will draw a smaller sample, then increase it for publication.

```
n_post_sims <- 2000
psims <- inla.posterior.sample(n_post_sims, JMinla)
```

Fit statistics

We begin by establishing that our model output matches expectations by comparing predictions to observations.

```
observed <- d$longDat$Y
n_obs_long <- length(observed)
pred_raw_prec <- psims |> sapply(\(sim) {
  sim$hyperpar["precision for qkumar observations"]
})
```

Drawing the longitudinal predictions straight from INLA can be done as follows:

```
predicted_raw <- psims |> sapply(\(sim) {
  sim$latent[seq_len(n_obs_long)]
})
predicted_INLA <- seq_len(n_post_sims) |> sapply(\(sim) {
  rkum(n_obs_long, plogis(predicted_raw[,sim]), pred_raw_prec[sim], ParamList$q)
})
standardised_residual_INLA <- sapply(seq_len(n_obs_long), \(i) {
```

```r
  mean(predicted_INLA[i, ] <= observed[i])
})
yhat_INLA <- rowMeans(predicted_INLA)
```

Constructing the longitudinal predictions from the parameters themselves:

```r
NS <- ParamList$nsujet
n_latent <- nrow(psims[[1]]$latent)
latent_names <- psims[[1]]$latent |> rownames()
# latent_names |> tail(10)
latent_pos_InteY <- which(latent_names |> startsWith("InteY"))
latent_pos_Intercept <- which(latent_names |> startsWith("Intercept"))
# Number of predictors after intercept, time, and binary treatment interaction:
n_other_preditors <- n_latent - latent_pos_InteY - 2
```

```r
beta_effects <- psims |> sapply(\(sim) {
  sim$latent[seq(latent_pos_InteY, n_latent),]
})
beta_pred <- cbind(rep(1, n_obs_long),
                   d$longDat$time,
                   d$longDat$time*d$longDat$binX,
                   as.matrix(d$longDat[,3:(ncol(d$longDat)-1)])) %*% beta_effects
gamma_effects <- psims |> sapply(\(sim) {
  sim$latent[seq(latent_pos_Intercept, latent_pos_InteY - 1),]
})
random_effects_level <- psims |> sapply(\(sim) {
  sim$latent[which(latent_names |> startsWith("IDY"))[seq_len(NS)],]
})
random_effects_slope <- psims |> sapply(\(sim) {
  sim$latent[which(latent_names |> startsWith("IDY"))[seq_len(NS) + NS],]
})
eta_pred <- beta_pred + random_effects_level[d$longDat$Id,] + random_effects_slope[d$longDat$Id,]*d$
predicted_manual <- seq_len(n_post_sims) |> sapply(\(sim) {
  rkum(n_obs_long, plogis(eta_pred[,sim]), pred_raw_prec[sim], ParamList$q)
})
standardised_residual_manual <- sapply(seq_len(n_obs_long), \(i) {
  mean(predicted_manual[i, ] <= observed[i])
})
yhat_manual <- rowMeans(predicted_manual)
```

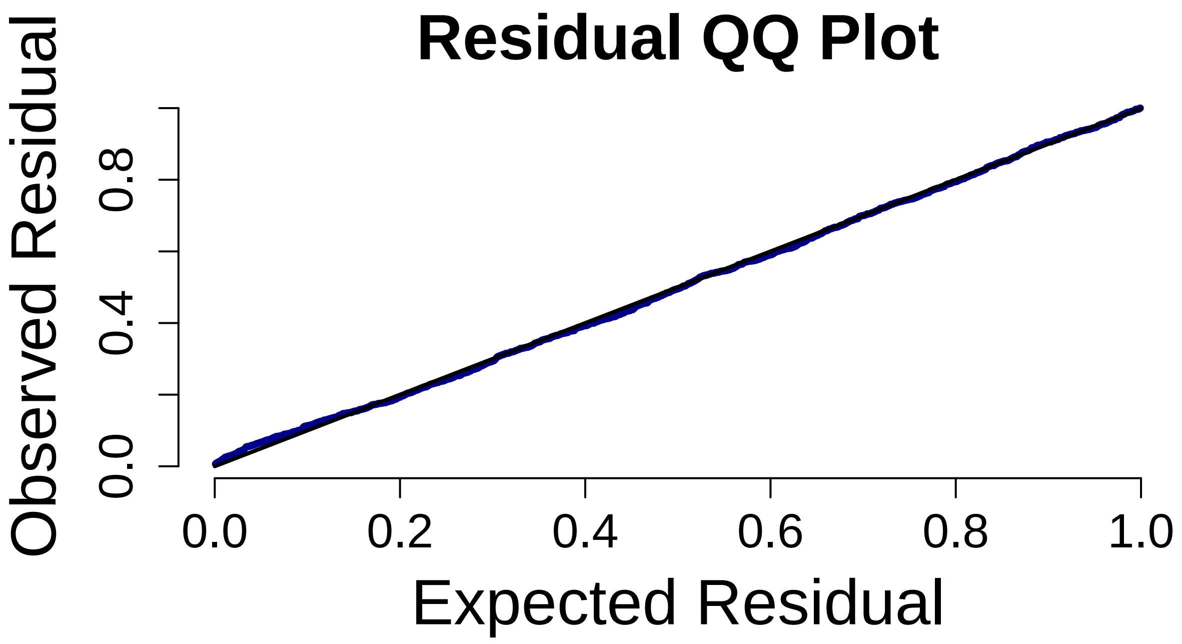
Functions for checking standardised residuals:

```r
QQPlot <- function(std_res, NObs = length(std_res)) {
  par(mar = c(4.5, 5.0, 3.0, 0.5))
  qqplot(seq_len(NObs)/(NObs + 1), std_res, main = "Residual QQ Plot", xlab = "Expected Residual", y
  lines(c(0, 1), c(0, 1), lwd = 2)
}
```

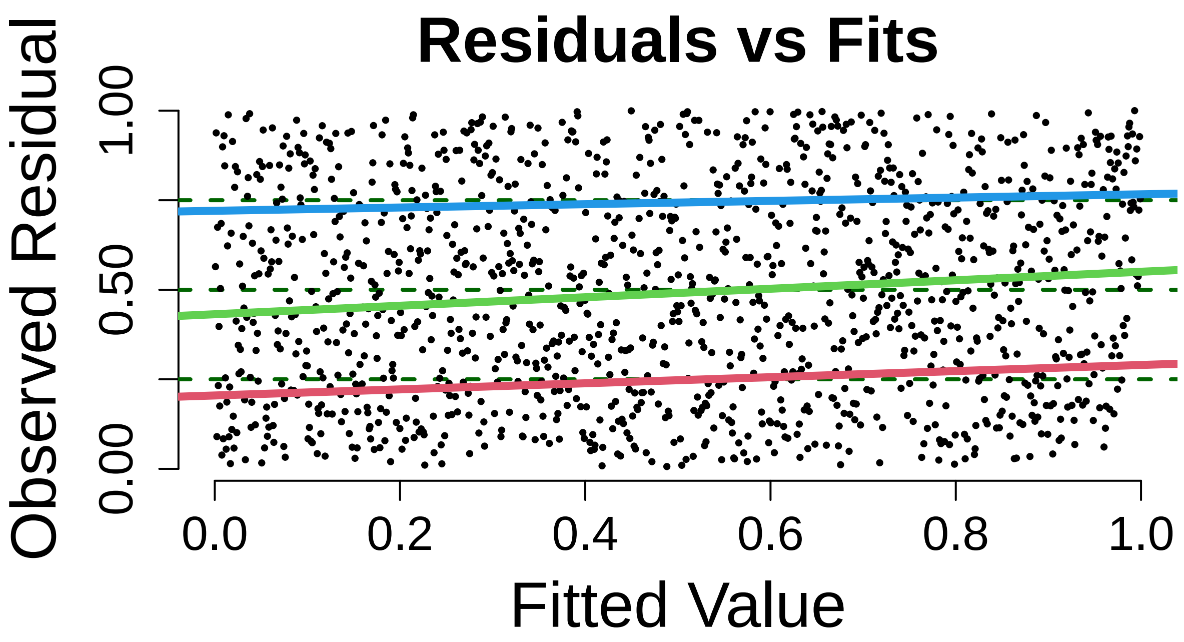
```
ResPlot <- function(std_res, yhat, NObs = length(std_res)) {
  x <- seq_len(NObs)/(NObs + 1)
  y <- std_res[order(yhat)]
  ylvls <- c(0.25, 0.5, 0.75)
  par(mar = c(4.5, 5.0, 3.0, 0.5))
  plot(x, y, main = "Residuals vs Fits", type = 'n', xlab = "Fitted Value", ylab = "Observed Residua
  axis(2, at = c(0, ylvls, 1), cex.axis = 1.5)
  abline(h = ylvls, lty = 2, lwd = 2, col = "darkgreen")
  points(x, y, cex = 0.5, pch = 16)
  for (j in seq_along(ylvls)) {
    abline(quantreg::rq(y ~ x, ylvls[j]), col = j + 1, lwd = 4)
  }
}
```

Standardised residuals using INLA fitted values

```
standardised_residual_INLA |> QQPlot()
```



```
standardised_residual_INLA |> ResPlot(yhat_INLA)
```



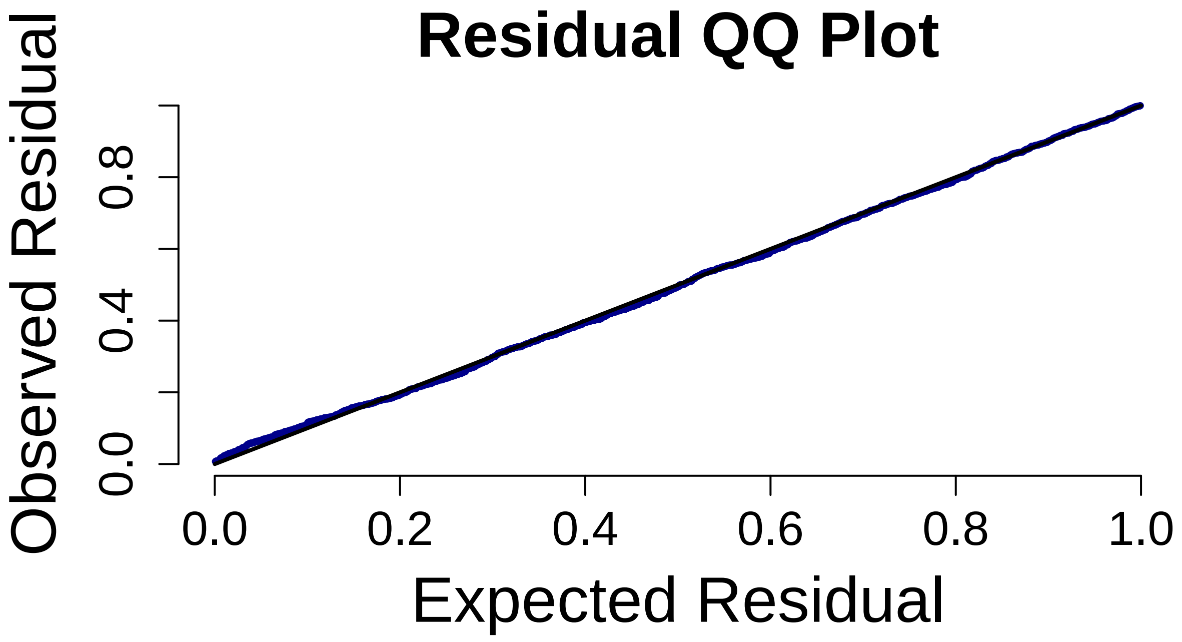
```r
standardised_residual_INLA |> ks.test('punif')
```

```
Warning in ks.test.default(standardised_residual_INLA, "punif"): ties should
not be present for the Kolmogorov-Smirnov test
```

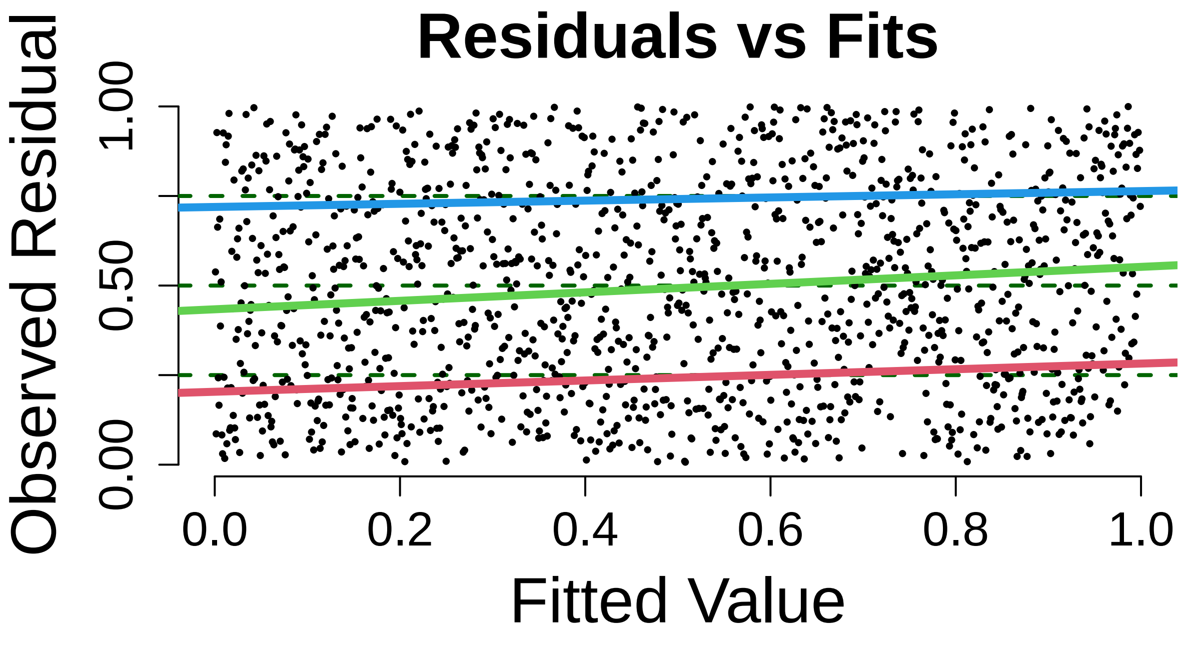
```

	Asymptotic one-sample Kolmogorov-Smirnov test

data:  standardised_residual_INLA
D = 0.020923, p-value = 0.6198
alternative hypothesis: two-sided
```

Standardised residuals using manually calculated values

```r
standardised_residual_manual |> QQPlot()
```



```
standardised_residual_manual |> ResPlot(yhat_manual)
```



```
standardised_residual_manual |> ks.test('punif')
```

```
Warning in ks.test.default(standardised_residual_manual, "punif"): ties should
not be present for the Kolmogorov-Smirnov test
```

```
	Asymptotic one-sample Kolmogorov-Smirnov test
```

```
data:  standardised_residual_manual
D = 0.020577, p-value = 0.6408
alternative hypothesis: two-sided
```

Plots for a single subject

```r
sbj <- d$longDat$Id |> factor(levels = seq_len(ParamList$nsujet)) |> table() |> which.max()
```

We will use subject 25 only for now.

We begin by isolating the subject's data and predictions made so far.

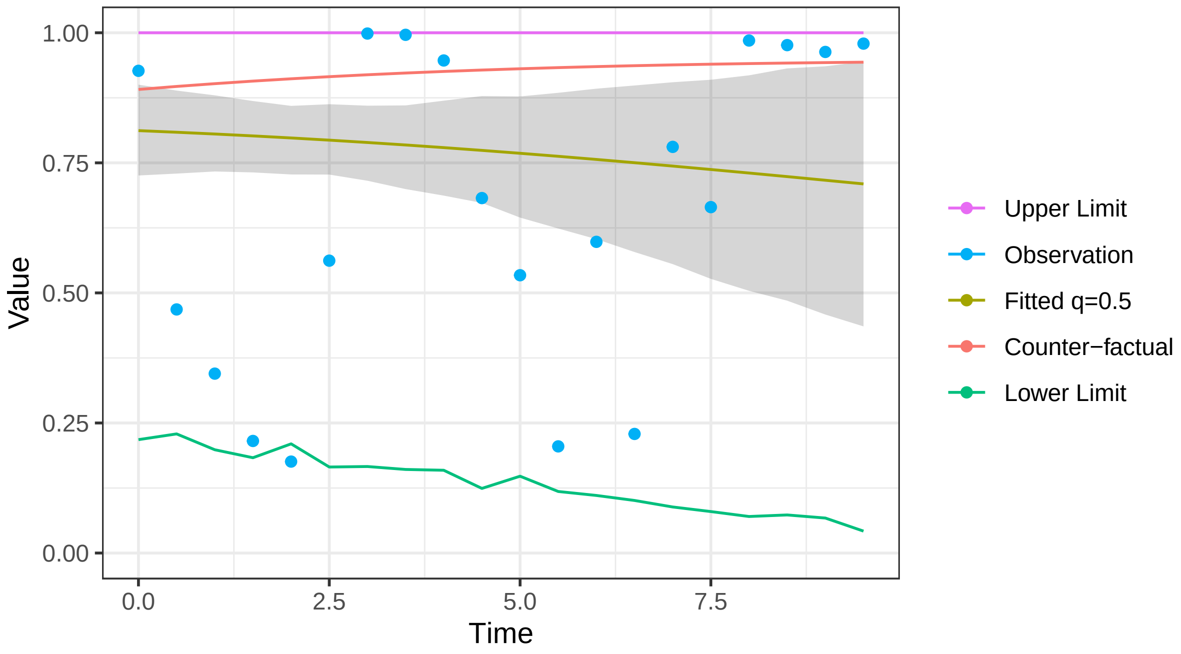
```r
surv_sbj <- d$survDat |> filter(Id == sbj)
rows_sbj <- which(d$longDat$Id == sbj)
nrows_sbj <- length(rows_sbj)
long_sbj <- d$longDat |> filter(Id == sbj)
predicted_sbj <- predicted_INLA[rows_sbj,]
fitted_sbj <- predicted_raw[rows_sbj,]
random_effects_level_sbj <- random_effects_level[sbj,]
random_effects_slope_sbj <- random_effects_slope[sbj,]
```

Then we construct a basic plot at only the observed time points:

```r
plot_line_names <- c('Observation', paste0('Fitted q=', ParamList$q), 'Lower Limit', 'Upper Limit',
```

```r
sbj_intervals <- fitted_sbj |> plogis() |> apply(1, shortestinterval)
sbj_pred_intervals <- predicted_sbj |> apply(1, shortestinterval)
sbj_shade_data <- data.frame(
  Time = long_sbj$time,
  Low = sbj_intervals[1,],
  High = sbj_intervals[2,]
  )
sbj_plot_data <- data.frame(
  Item = rep(plot_line_names, each = nrows_sbj),
  Time = rep(long_sbj$time, times = length(plot_line_names)),
  Value = c(long_sbj$Y,
            fitted_sbj |> plogis() |> rowMeans(),
            sbj_pred_intervals[1,],
            sbj_pred_intervals[2,],
            (fitted_sbj +
              ((1-2*long_sbj$binX)*long_sbj$time) %*% beta_effects[3,,drop=FALSE] +
              (1-2*long_sbj$binX) %*% beta_effects[4,,drop=FALSE]
            ) |> plogis() |> rowMeans()
            )
  )
```

```r
sbj_plot_data |> ggplot(aes(x = Time)) +
  geom_ribbon(aes(ymin = Low, ymax = High), data = sbj_shade_data, alpha = 0.2) +
  geom_line(aes(y = Value, group = Item, colour = Item), data = \(x) filter(x, Item != 'Observation'
  geom_point(aes(y = Value, group = Item, colour = Item), data = \(x) filter(x, Item == 'Observatio
  scale_colour_discrete(breaks = plot_line_names[c(4,1,2,5,3)]) +
  theme(legend.title = element_blank()) +
  ylim(0,1)
```

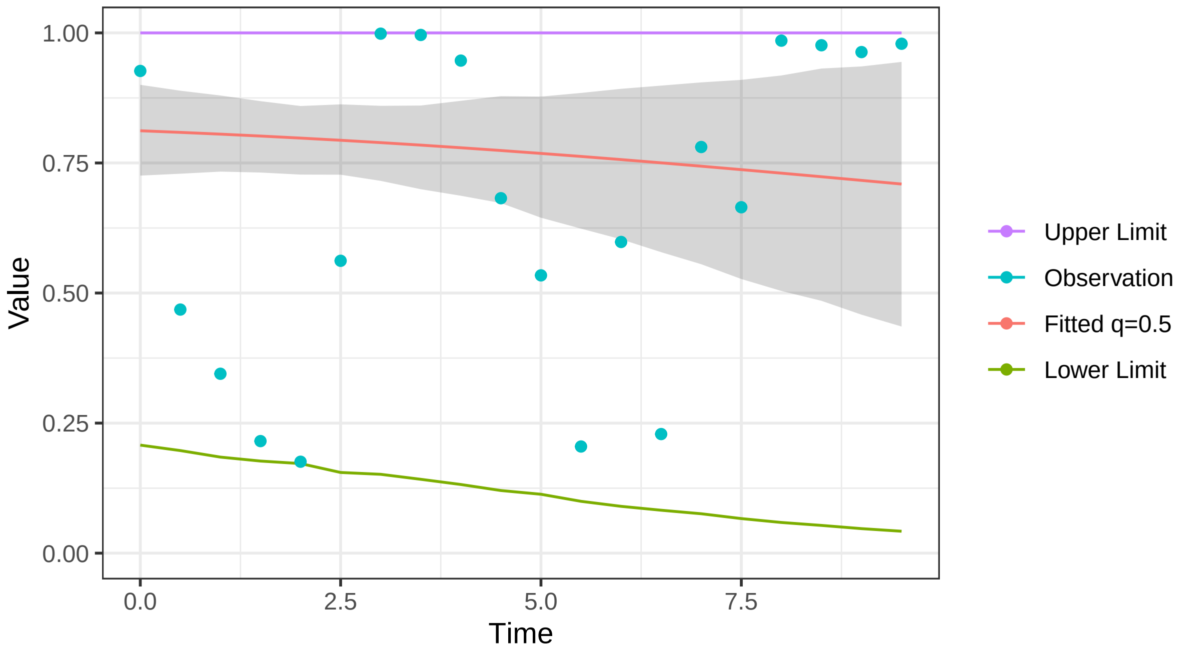


The counter-factual is what the model would have expected to see from this subject had they been in the other group. This is useful in causal inference. We can drop it for further graphs though as it is only of interest in select cases.

A question arises in that we can clearly see the uncertainty in limits, but is this interesting. If so, we can leave it as is - showing the MCMC error - otherwise we must address it somehow.

```r
rolling_mean <- function(x, k = 15) {
  n <- length(x)
  y <- x
  x <- c(x, rep(x[n], k))
  for (i in 1:n) {
    y[i] <- x[i:(i+k-1)] |> mean(na.rm = TRUE)
  }
  y
}
```

```r
plot_line_names <- plot_line_names[-5]
sbj_plot_data <- data.frame(
  Item = rep(plot_line_names, each = nrows_sbj),
  Time = rep(long_sbj$time, times = length(plot_line_names)),
  Value = c(long_sbj$Y,
            fitted_sbj |> plogis() |> rowMeans(),
            sbj_pred_intervals[1,] |> rolling_mean(5),
            sbj_pred_intervals[2,] |> rolling_mean(5)
            )
  )
sbj_plot_data |> ggplot(aes(x = Time)) +
  geom_ribbon(aes(ymin = Low, ymax = High), data = sbj_shade_data, alpha = 0.2) +
  geom_line(aes(y = Value, group = Item, colour = Item), data = \(x) filter(x, Item != 'Observation'
  geom_point(aes(y = Value, group = Item, colour = Item), data = \(x) filter(x, Item == 'Observatio
  scale_colour_discrete(breaks = plot_line_names[c(4,1,2,5,3)]) +
  theme(legend.title = element_blank()) +
  ylim(0,1)
```



Extending the plot past the censoring point

We wish to plot predictions to a specified end point, not necessarily stop at the censoring point.

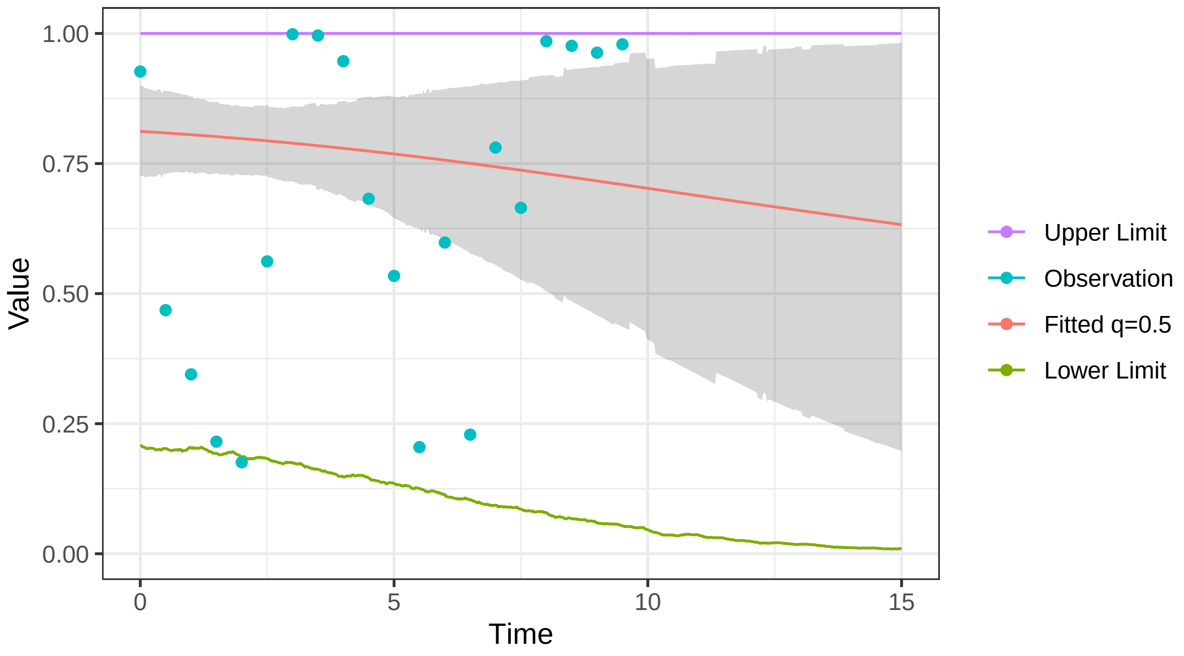
First we rework the plotting code to have an extended endpoint. We use a very fine scale to ensure accurate integration later.

```
time_seq <- seq(0, ParamList$endpoint, ParamList$gap)
n_time_seq <- length(time_seq)
int_seq <- rep(1, n_time_seq)
```

```
sbj_obs_data <- data.frame(
  Item = rep(plot_line_names[1], nrows_sbj),
  Time = long_sbj$time,
  Value = long_sbj$Y
  )
```

```r
sbj_explanatory_values <- long_sbj[1,3:(ncol(long_sbj)-1)] |> unlist()
beta_pred_sbj <- (c(int_seq,
                    time_seq,
                    time_seq*sbj_explanatory_values[1],
                    rep(sbj_explanatory_values,
                        each = n_time_seq)
                   ) |> matrix(n_time_seq)
                  ) %*% beta_effects
gamma_pred_sbj <- (c(int_seq,
                     rep(sbj_explanatory_values,
                        each = n_time_seq)
                    ) |> matrix(n_time_seq)
                   ) %*% gamma_effects
fitted_sbj <- beta_pred_sbj +
              int_seq %*% t(random_effects_level_sbj) +
              time_seq %*% t(random_effects_slope_sbj)
predicted_sbj <- seq_len(n_post_sims) |> sapply(\(sim) {
  rkum(n_time_seq, plogis(fitted_sbj[,sim]), pred_raw_prec[sim], ParamList$q)
})
sbj_intervals <- fitted_sbj |> plogis() |> apply(1, shortestinterval)
sbj_pred_intervals <- predicted_sbj |> apply(1, shortestinterval)
# expected_counter_sbj <- (c(int_seq,
#                           time_seq,
#                           rep(c(1-2*long_sbj[1,3], unlist(long_sbj[1,4:(ncol(long_sbj)-1)])),
#                              each = n_time_seq)
#                           ) |> matrix(n_time_seq)
#                          ) %*% beta_effects +
#                          int_seq %*% t(random_effects_level_sbj) +
#                          time_seq %*% t(random_effects_slope_sbj)
sbj_shade_data <- data.frame(
  Time = time_seq,
  Low = sbj_intervals[1,],
  High = sbj_intervals[2,]
  )
sbj_line_data <- data.frame(
  Item = rep(plot_line_names[-1], each = n_time_seq),
  Time = rep(time_seq, times = length(plot_line_names[-1])),
  Value = c(fitted_sbj |> plogis() |> rowMeans(),
            sbj_pred_intervals[1,] |> rolling_mean(),
            sbj_pred_intervals[2,] |> rolling_mean()
            )
  )
```

```r
sbj_line_data |> ggplot(aes(x = Time)) +
  geom_ribbon(aes(ymin = Low, ymax = High), data = sbj_shade_data, alpha = 0.2) +
  geom_line(aes(y = Value, group = Item, colour = Item)) +
  geom_point(aes(y = Value, group = Item, colour = Item), data = sbj_obs_data) +
  scale_colour_discrete(breaks = plot_line_names[c(4,1,2,5,3)]) +
  theme(legend.title = element_blank()) +
  ylim(0,1)
```



Plotting the survival function

To plot the survival function we begin by constructing the baseline hazard function with uncertainty. For the period being modelled we will use the existing posterior samples for estimation and uncertainty, but for the extended period we will extend the existing sample paths going forward in time.

```
bh_items <- latent_names |> startsWith("baseline.hazard") |> which()
bh_times <- JMinla$summary.random[["baseline.hazard"]]$ID
bh_sims_raw <- psims |> sapply(\(sim) {
  sim$latent[bh_items,]
}) |> t()
bh_gap <- diff(bh_times) |> mean()
```

```
sigma <- psims |> sapply(\(sim) {
  sim$hyperpar["Precision for baseline.hazard"]^(-0.5)
})
current_time <- bh_times |> last()
while (current_time < ParamList$endpoint) {
  current_time <- current_time + bh_gap
  k <- ncol(bh_sims_raw)
  new_sims <- bh_sims_raw[,k]*2 - bh_sims_raw[,k-1] + rnorm(n_post_sims, 0 , sigma)
  bh_sims_raw <- cbind(bh_sims_raw, new_sims)
  bh_times <- c(bh_times, current_time)
}
rm(current_time, k, new_sims)
```

Then we interpolate the hazard, sticking to the piecewise constant assumption.

```r
surv_intercept_column <- latent_names |> startsWith("Intercept") |> which()
bh_sims <- seq_len(n_time_seq) |> sapply(\(i) {
  bh_sims_raw[, match(TRUE, bh_times > time_seq[i]) - 1]
})
phi_sims <- psims |> sapply(\(sim) {
  sim$hyperpar["Beta for eta"]
})
```

The following step is to bring in the eta function before integrating. This step must be done carefully as it depends on the random effects, making it dependent on a specific subject (either an existing subject or a hypothetical future subject).

For now we will use our chosen subject 25.

First we obtain the area under the hazard curve. This allows us to do integrals over any or multiple regions without recalculating. However, care should be taken to only integrate over valid regions - the introduction of additional information will change the parameters.

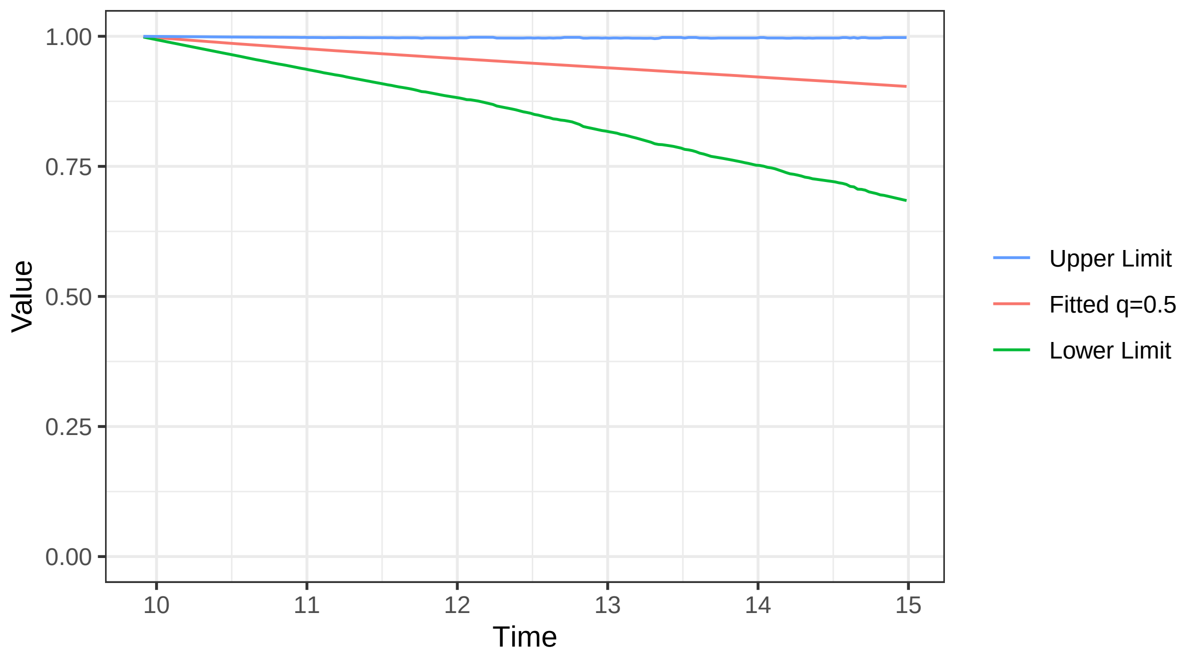
```r
h_sbj <- exp(bh_sims + t(fitted_sbj)*phi_sims + t(gamma_pred_sbj))
area_sbj <- (h_sbj[,-1] + h_sbj[,-n_time_seq])/2*ParamList$gap
area_centers <- (time_seq[-1] + time_seq[-n_time_seq])/2
```

In this case we will restrict ourselves to integrating from the event point onwards specifically.

```r
H_sbj <- area_sbj[, (area_centers > surv_sbj$eventTimes)] |> apply(1, cumsum)
S_sbj <- exp(-H_sbj)
S_sbj_times <- area_centers[(area_centers > surv_sbj$eventTimes)]
S_sbj_n_times <- length(S_sbj_times)
```

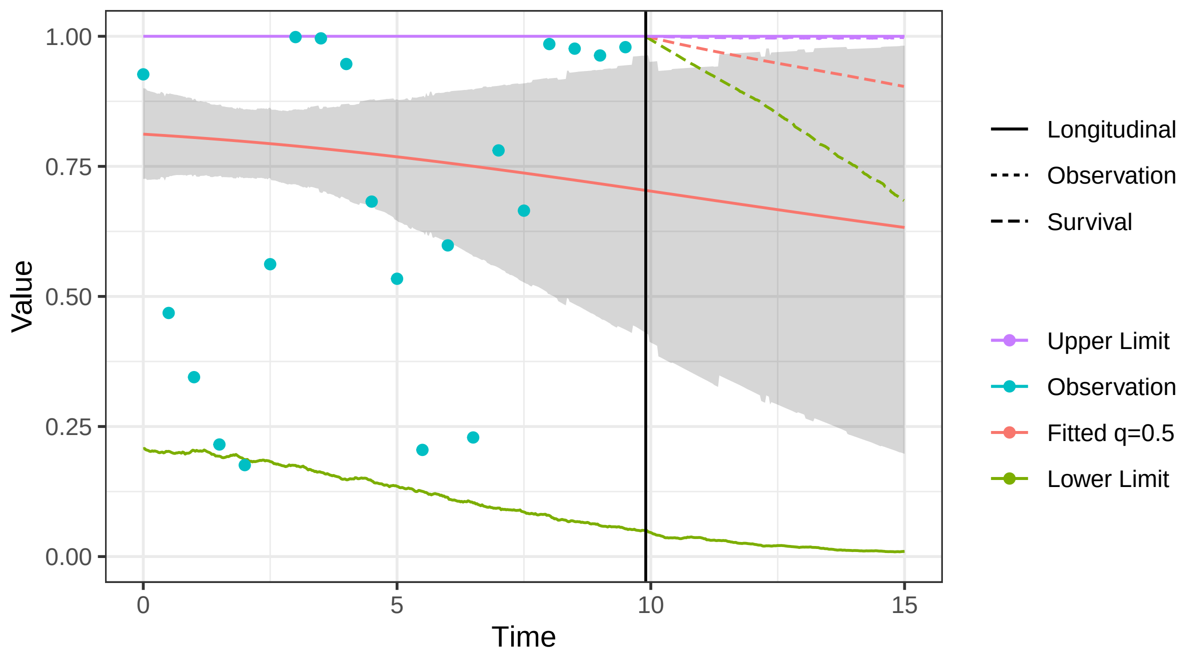
Then we can plot the survival function, after summarising across the simulations.

```r
S_plot_lines <- c("Upper Limit", "Fitted q=0.5", "Lower Limit")
S_sbj_intervals <- S_sbj |> apply(1, shortestinterval)
S_sbj_line_data <- data.frame(
  Item = rep(S_plot_lines, each = S_sbj_n_times),
  Time = rep(S_sbj_times, times = length(S_plot_lines)),
  Value = c(S_sbj_intervals[2,],
            S_sbj |> apply(1, mean),
            S_sbj_intervals[1,]
            )
)
S_sbj_line_data |> ggplot(aes(x = Time, y = Value, group = Item, colour = Item)) +
  geom_line() +
  scale_colour_discrete(breaks = S_plot_lines) +
  theme(legend.title = element_blank()) +
  ylim(0,1)
```



This can even be combined with the previous plot.

```
sbj_obs_data$Component <- "Observation"
sbj_line_data$Component <- "Longitudinal"
S_sbj_line_data$Component <- "Survival"
sbj_line_data |> ggplot(aes(x = Time, y = Value, group = Item, colour = Item, linetype = Component))
  geom_ribbon(aes(x = Time, ymin = Low, ymax = High), data = sbj_shade_data, alpha = 0.2, inherit.ae
  geom_line() +
  geom_point(data = sbj_obs_data) +
  scale_colour_discrete(breaks = plot_line_names[c(4,1,2,5,3)]) +
  theme(legend.title = element_blank()) +
  ylim(0,1) +
  geom_line(data = S_sbj_line_data) +
  geom_vline(xintercept = surv_sbj$eventTimes)
```



Plots for a new or average subject

Here we consider a future subject without observed data. For intervals we could consider either a random future subject (sampled random effects) or the average subject (zero random effects).

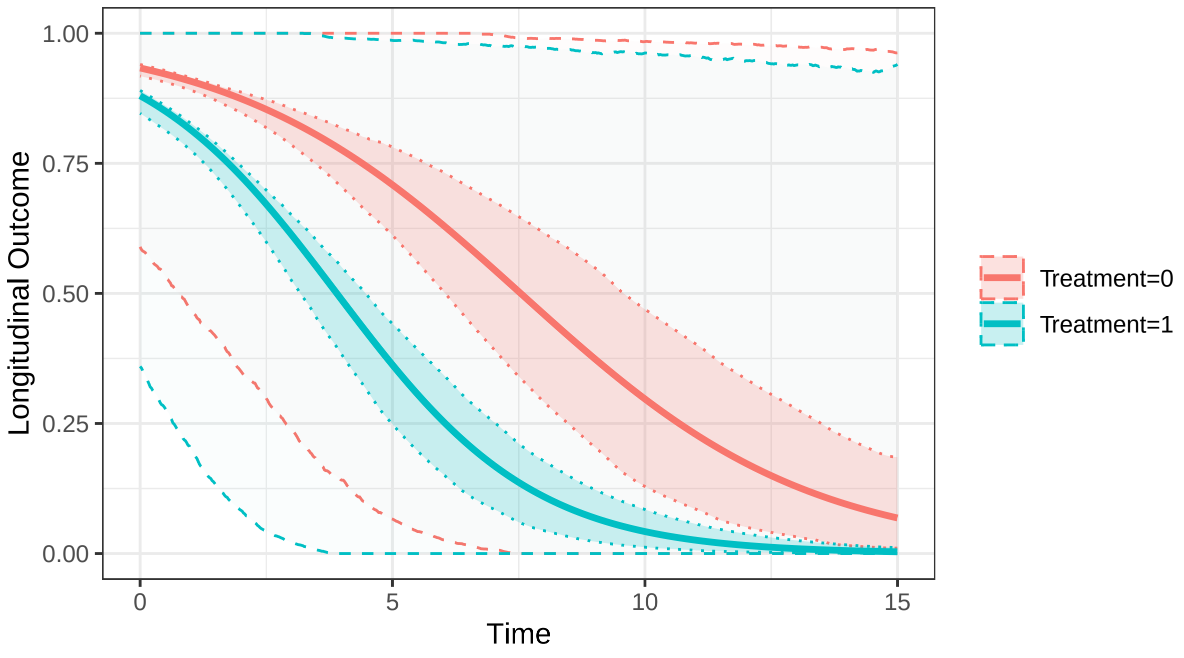
For the continuous predictor(s) we will use the mean value(s), while for the binary predictor we will consider both options on one plot.

```r
random_effects_Sigma <- function(psims) {
  sigma11_sims <- 1/(psims |> sapply(\(sim) {
    sim$hyperpar["Precision for IDY (component 1)"]
  }))
  sigma22_sims <- 1/(psims |> sapply(\(sim) {
    sim$hyperpar["Precision for IDY (component 2)"]
  }))
  sigma12_sims <- (psims |> sapply(\(sim) {
    sim$hyperpar["Rho1:2 for IDY"]
  }))*sqrt(sigma11_sims*sigma22_sims)
  cbind(sigma11_sims, sigma12_sims, sigma12_sims, sigma22_sims)
}
random_effects_future <- psims |>
  random_effects_Sigma() |>
  apply(1, \(sigmavec) {
  MASS::mvrnorm(1, matrix(0, 1, 2), sigmavec |> matrix(2))
})
averages <- d$longDat[,4:(ncol(d$longDat)-1), drop = FALSE] |> colMeans()
n_treatments <- 2
treatments <- seq_len(n_treatments)
treatment_plot_lines <- paste0("Treatment=", treatments-1)
beta_binary <- treatments |> lapply(\(tr) {
  (c(int_seq,
    time_seq,
    rep(tr - 1, n_time_seq)*time_seq,
    rep(tr - 1, n_time_seq),
    rep(averages, each = n_time_seq)
   ) |> matrix(n_time_seq)
  ) %*% beta_effects
})
fitted_future <- treatments |> lapply(\(tr) {
  beta_binary[[tr]] +
  int_seq %*% t(random_effects_future[1,]) +
  time_seq %*% t(random_effects_future[2,])
})
predicted <- treatments |> lapply(\(tr) {
  seq_len(n_post_sims) |> sapply(\(sim) {
    rkum(n_time_seq, plogis(beta_binary[[tr]][,sim]), pred_raw_prec[sim], ParamList$q)
  })
})
beta_intervals <- treatments |> lapply(\(tr) {
  beta_binary[[tr]] |> plogis() |> apply(1, shortestinterval)
})
pred_intervals <- treatments |> lapply(\(tr) {
  predicted[[tr]] |> apply(1, shortestinterval)
})
gamma_pred <- treatments |> lapply(\(tr) {
  (c(int_seq,
    rep(tr - 1, n_time_seq),
    rep(averages, each = n_time_seq)
   ) |> matrix(n_time_seq)
  ) %*% gamma_effects
})
hazard <- treatments |> lapply(\(tr) {
  exp(bh_sims + t(beta_binary[[tr]])*phi_sims + t(gamma_pred[[tr]]))
})
h_future <- treatments |> lapply(\(tr) {
  exp(bh_sims + t(fitted_future[[tr]])*phi_sims + t(gamma_pred[[tr]]))
})
h_intervals_future <- treatments |> lapply(\(tr) {
  h_future[[tr]] |> apply(2, shortestinterval)
})
h_intervals <- treatments |> lapply(\(tr) {
  hazard[[tr]] |> apply(2, shortestinterval)
})
S_future <- treatments |> lapply(\(tr) {
  exp(-(((h_future[[tr]][,-1] + h_future[[tr]][,-n_time_seq])/2*ParamList$gap) |> apply(1, cumsum))
})
S <- treatments |> lapply(\(tr) {
  exp(-(((hazard[[tr]][,-1] + hazard[[tr]][,-n_time_seq])/2*ParamList$gap) |> apply(1, cumsum)))
})
```

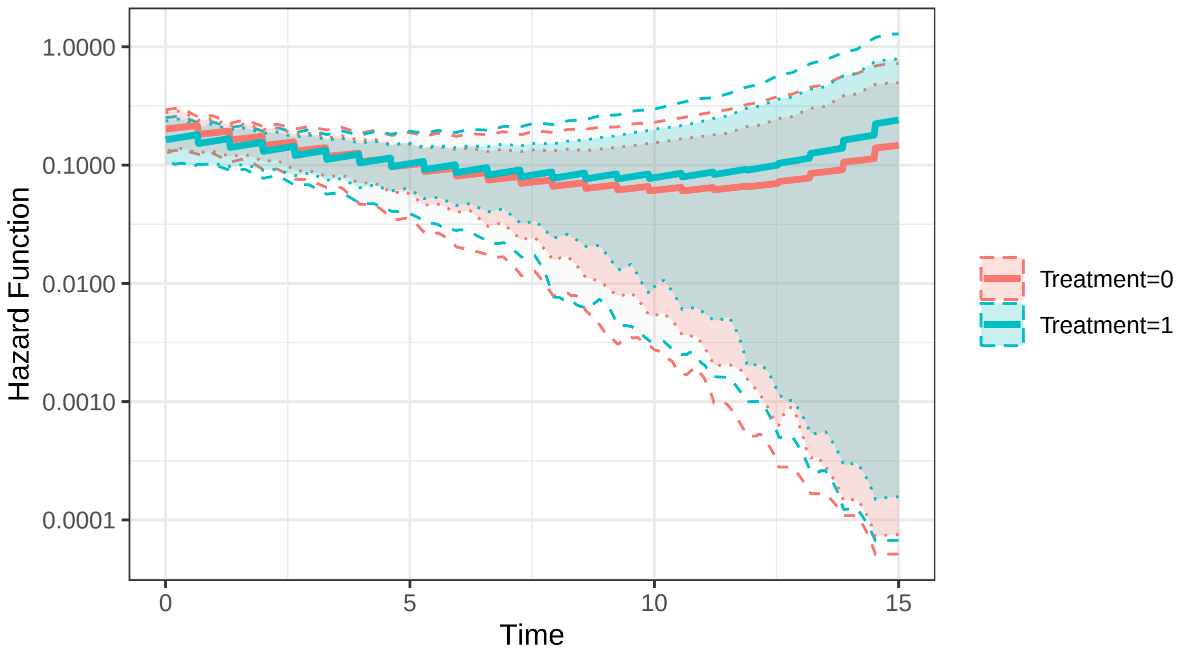
```
S_intervals_future <- treatments |> lapply(\(tr) {
  S_future[[tr]] |> apply(1, shortestinterval)
})
S_intervals <- treatments |> lapply(\(tr) {
  S[[tr]] |> apply(1, shortestinterval)
})
```

In the plots below each line has two intervals: a narrower interval representing the average subject, and a wider interval representing a random future subject.

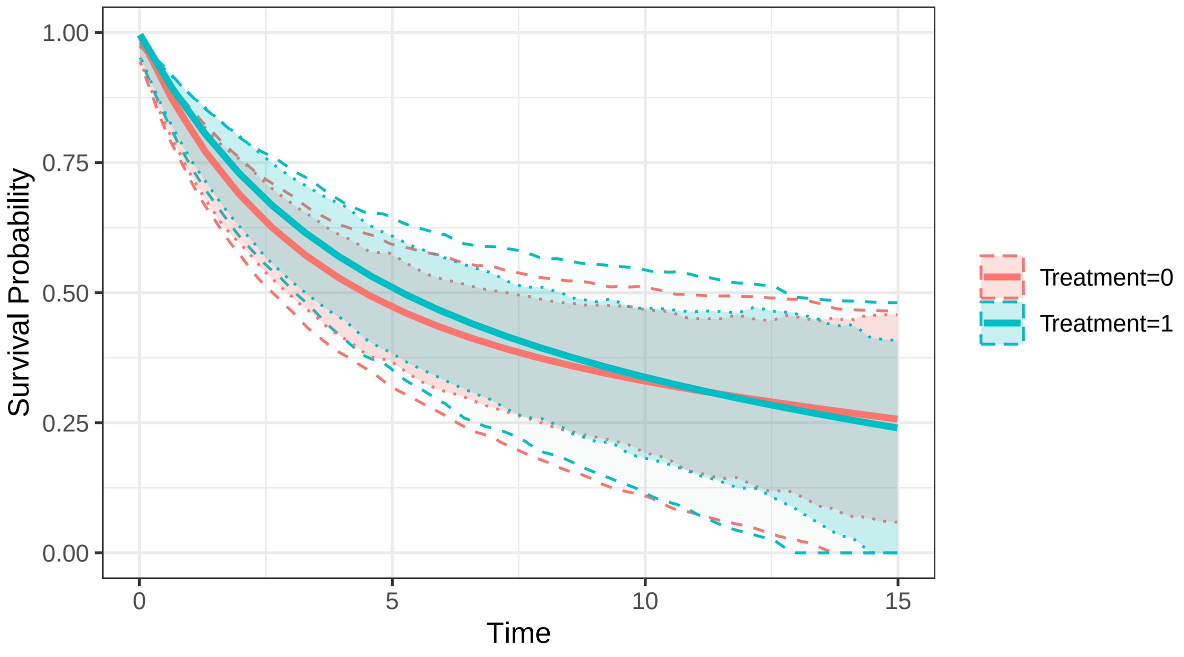
```
long_plot_data <- data.frame(
  Treatment = rep(treatment_plot_lines, each = n_time_seq),
  Time = rep(time_seq, times = n_treatments),
  Estimate = c(beta_binary |> sapply(rowMeans)) |> plogis(),
  Cred_Lower = c(beta_intervals |> sapply(\(x) x[1,] |> rolling_mean())),
  Cred_Upper = c(beta_intervals |> sapply(\(x) x[2,] |> rolling_mean())),
  Pred_Lower = c(pred_intervals |> sapply(\(x) x[1,] |> rolling_mean())),
  Pred_Upper = c(pred_intervals |> sapply(\(x) x[2,] |> rolling_mean())),
  Component = "Longitudinal"
  )

long_plot_data |> ggplot(aes(x = Time, group = Treatment, colour = Treatment, fill = Treatment)) +
  geom_ribbon(aes(ymin = Pred_Lower, ymax = Pred_Upper), alpha = 0.02, linetype = 2) +
  geom_ribbon(aes(ymin = Cred_Lower, ymax = Cred_Upper), alpha = 0.2, linetype = 3) +
  geom_line(aes(y = Estimate), linewidth = 1.2) +
  scale_colour_discrete(breaks = treatment_plot_lines) +
  theme(legend.title = element_blank()) +
  ylab("Longitudinal Outcome")
```



```r
h_plot_data <- data.frame(
  Treatment = rep(treatment_plot_lines, each = n_time_seq),
  Time = rep(time_seq, times = n_treatments),
  Estimate = c(hazard |> sapply(colMeans)),
  Cred_Lower = c(h_intervals |> sapply(\(x) x[1,] |> rolling_mean())),
  Cred_Upper = c(h_intervals |> sapply(\(x) x[2,] |> rolling_mean())),
  Pred_Lower = c(h_intervals_future |> sapply(\(x) x[1,] |> rolling_mean())),
  Pred_Upper = c(h_intervals_future |> sapply(\(x) x[2,] |> rolling_mean())),
  Component = "Hazard"
  )

h_plot_data |> ggplot(aes(x = Time, group = Treatment, colour = Treatment, fill = Treatment)) +
  geom_ribbon(aes(ymin = Pred_Lower, ymax = Pred_Upper), alpha = 0.02, linetype = 2) +
  geom_ribbon(aes(ymin = Cred_Lower, ymax = Cred_Upper), alpha = 0.2, linetype = 3) +
  geom_line(aes(y = Estimate), linewidth = 1.2) +
  scale_colour_discrete(breaks = treatment_plot_lines) +
  theme(legend.title = element_blank()) +
  ylab("Hazard Function") +
  scale_y_log10()
```



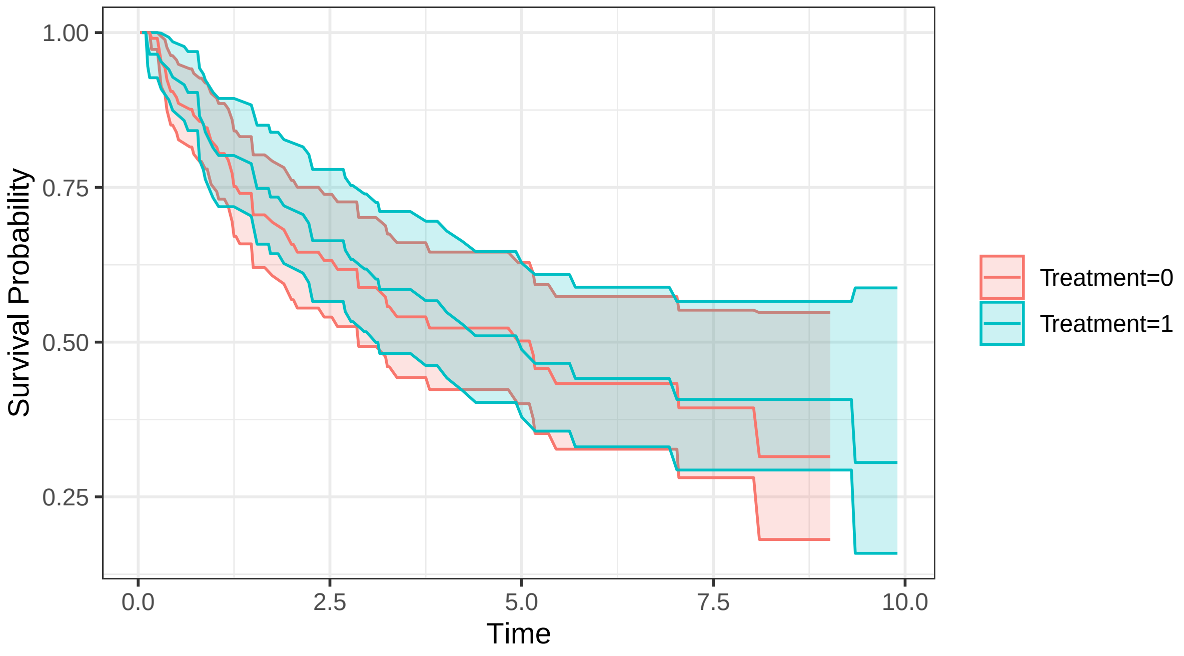
```r
S_future_line_data <- data.frame(
  Treatment = rep(treatment_plot_lines, each = length(area_centers)),
  Time = rep(area_centers, times = length(treatment_plot_lines)),
  Estimate = c(S |> sapply(rowMeans)),
  Cred_Lower = c(S_intervals |> sapply(\(x) x[1,] |> rolling_mean())),
  Cred_Upper = c(S_intervals |> sapply(\(x) x[2,] |> rolling_mean())),
  Pred_Lower = c(S_intervals_future |> sapply(\(x) x[1,] |> rolling_mean())),
  Pred_Upper = c(S_intervals_future |> sapply(\(x) x[2,] |> rolling_mean())),
  Component = "Survival"
  )

S_future_line_data |> ggplot(aes(x = Time, group = Treatment, colour = Treatment, fill = Treatment))
  geom_ribbon(aes(ymin = Pred_Lower, ymax = Pred_Upper), alpha = 0.02, linetype = 2) +
  geom_ribbon(aes(ymin = Cred_Lower, ymax = Cred_Upper), alpha = 0.2, linetype = 3) +
  geom_line(aes(y = Estimate), linewidth = 1.2) +
  scale_colour_discrete(breaks = treatment_plot_lines) +
  theme(legend.title = element_blank()) +
  ylim(0,1) + ylab("Survival Probability")
```



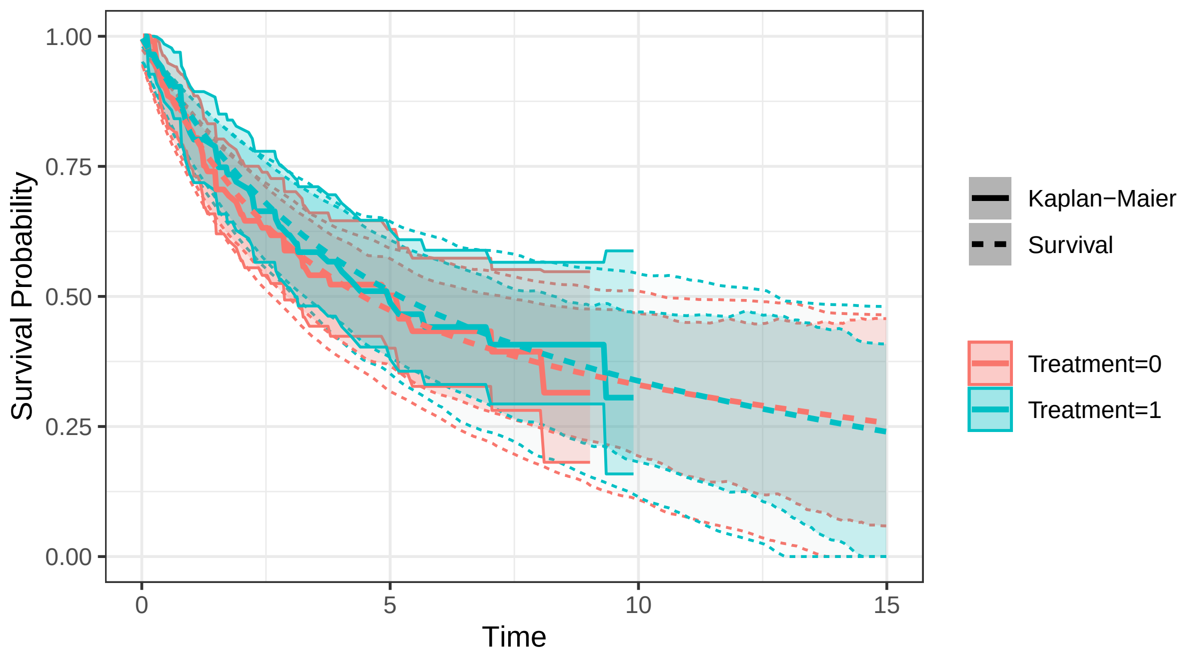
Adding Kaplan-Maier lines

As a comparison let us add the Kaplan-Maier survival function lines to the survival plot.

```
d$survDat$Treatment <- d$longDat$binX[match(d$survDat$Id, d$longDat$Id)]
km_fit <- survfit(Surv(eventTimes, Event) ~ Treatment, data = d$survDat)
km_curve_data <- data.frame(
  Time = km_fit$time,
  KM_est = km_fit$surv,
  KM_lower = km_fit$lower,
  KM_upper = km_fit$upper,
  Treatment = rep(names(km_fit$strata), km_fit$strata),
  Component = "Kaplan-Maier"
  )

km_curve_data |> ggplot(aes(x = Time, group = Treatment, colour = Treatment, fill = Treatment)) +
  geom_ribbon(aes(ymin = KM_lower, ymax = KM_upper), alpha = 0.2) +
  geom_line(aes(y = KM_est)) +
  theme(legend.title = element_blank()) +
  ylab("Survival Probability")
```



```
S_future_line_data |> ggplot(aes(x = Time,
                                 group = Treatment,
                                 colour = Treatment,
                                 fill = Treatment,
                                 linetype = Component)) +
  geom_ribbon(aes(ymin = Pred_Lower, ymax = Pred_Upper), alpha = 0.02) +
  geom_ribbon(aes(ymin = Cred_Lower, ymax = Cred_Upper), alpha = 0.2) +
  geom_ribbon(aes(ymin = KM_lower, ymax = KM_upper), data = km_curve_data, alpha = 0.2) +
  geom_line(aes(y = KM_est), data = km_curve_data, linewidth = 1) +
  geom_line(aes(y = Estimate), linewidth = 1) +
  scale_colour_discrete(breaks = treatment_plot_lines) +
  theme(legend.title = element_blank()) +
  ylim(0,1) + ylab("Survival Probability")
```



Capturing treatment effect

Since the treatment effect is time dependent, we will evaluate it at selected time points for convenience of interpretation.

```r
treatment_effect <- plogis(beta_binary[[1]]) - plogis(beta_binary[[2]])
calculate_stats <- \(x) {
  diff_dens <- density(x)
  diff_interval <- shortestinterval(x)
  c(
    Mean = mean(x),
    Median = median(x),
    Mode = diff_dens$x[which.max(diff_dens$y)],
    Lower = diff_interval[1],
    Upper = diff_interval[2]
)}
treatment_effect_stats <- treatment_effect |> apply(1, calculate_stats) |> t()
treatment_effect_df <- data.frame(Time = time_seq, treatment_effect_stats)

S_treatment_effect <- S[[1]] - S[[2]]
S_treatment_effect_stats <- S_treatment_effect |> apply(1, calculate_stats) |> t()
S_treatment_effect_df <- data.frame(Time = area_centers, S_treatment_effect_stats)

list(TreatmentEffect = treatment_effect_df, SurvivalEffect = S_treatment_effect_df) |>
  openxlsx::write.xlsx("Joint Model Treatment Effect Statistics.xlsx")
```

```r
summary_points <- seq(1, ParamList$followup/ParamList$gap , length = 5)
treatment_effect_df[summary_points, ] |> knitr::kable(digits = 3)
```

	Time	Mean	Median	Mode	Lower	Upper
1	0.000	0.054	0.054	0.054	0.030	0.078
100	2.475	0.181	0.181	0.181	0.128	0.229
200	4.975	0.342	0.344	0.348	0.215	0.467
300	7.475	0.362	0.361	0.344	0.185	0.529
400	9.975	0.259	0.251	0.213	0.088	0.447

```
S_treatment_effect_df[summary_points[-1], ] |> knitr::kable(digits = 3)
```

	Time	Mean	Median	Mode	Lower	Upper
100	2.487	-0.042	-0.042	-0.045	-0.177	0.070
200	4.988	-0.037	-0.039	-0.048	-0.202	0.109
300	7.488	-0.023	-0.024	-0.039	-0.190	0.136
400	9.988	-0.008	-0.009	-0.024	-0.176	0.150

Plotting function

Before we continue to dynamic predictions, we will compact the above processes into a simulation extraction and plotting function. This function accepts a vector of subjects for which to perform the plotting, to be supplied after the data and the INLA model fit. Additional inputs are the number of simulations to use and the parameter list.

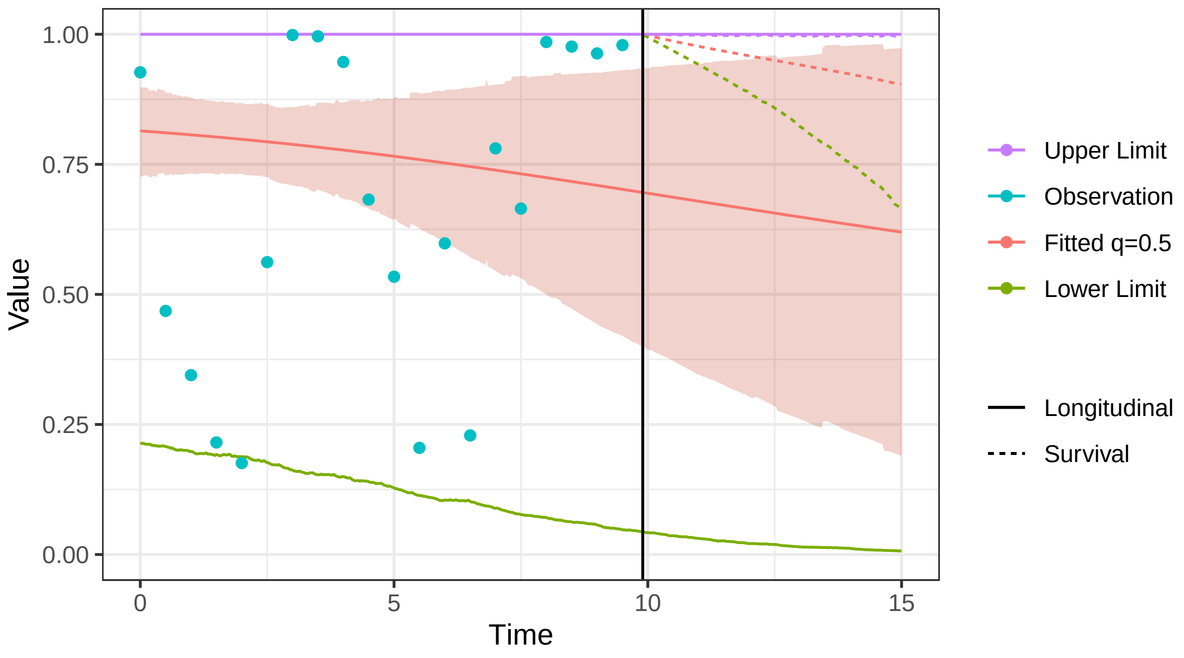
```
pred_plot <- function(d, INLAfit, parameters, subjects = 1:2, n_post_sims = 2000, draw_plots = TRUE
  # Get posterior simulations and process them
  psims <- inla.posterior.sample(n_post_sims, INLAfit)
  observed <- d$longDat$Y
  n_obs_long <- length(observed)
  pred_raw_prec <- psims |> sapply(\(sim) {
    sim$hyperpar["precision for qkumar observations"]
  })
  predicted_raw <- psims |> sapply(\(sim) {
    sim$latent[seq_len(n_obs_long)]
  })
  predicted_INLA <- seq_len(n_post_sims) |> sapply(\(sim) {
    rkum(n_obs_long, plogis(predicted_raw[,sim]), pred_raw_prec[sim], ParamList$q)
  })
  NS <- length(unique(d$survDat$Id))
  n_latent <- nrow(psims[[1]]$latent)
  latent_names <- psims[[1]]$latent |> rownames()
  latent_pos_InteY <- which(latent_names |> startsWith("InteY"))
  latent_pos_Intercept <- which(latent_names |> startsWith("Intercept"))
  # Number of predictors after intercept, time, and binary treatment indicator:
  n_other_preditors <- n_latent - latent_pos_InteY - 2
  beta_effects <- psims |> sapply(\(sim) {
    sim$latent[seq(latent_pos_InteY, n_latent),]
```

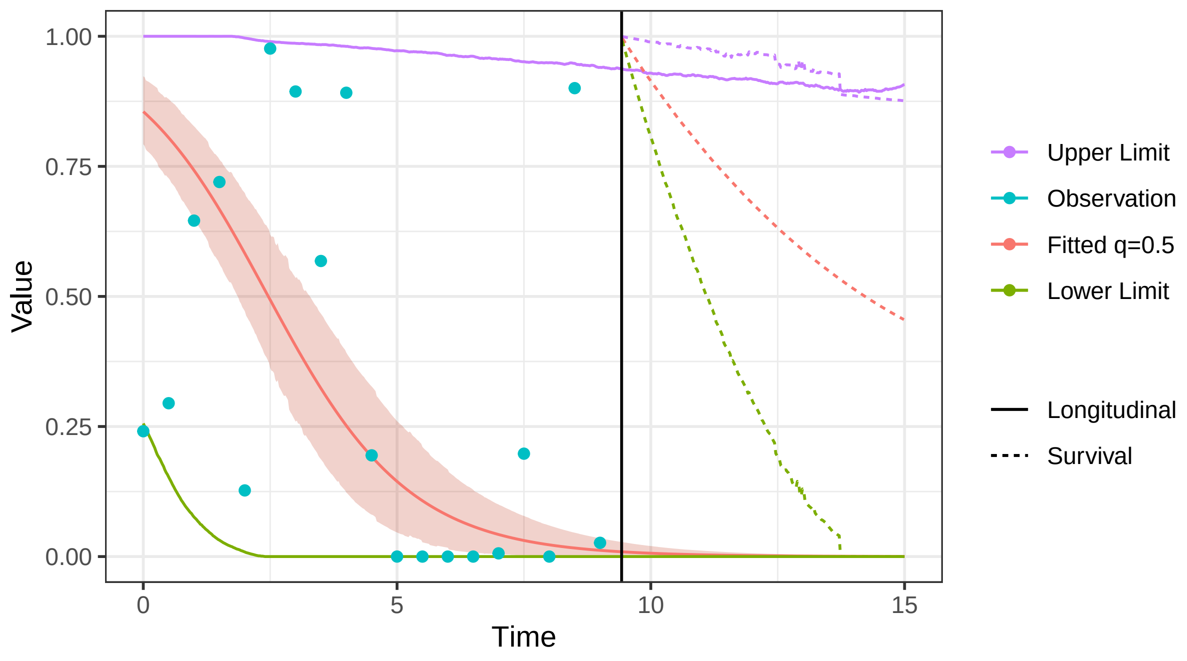
```r
})
gamma_effects <- psims |> sapply(\(sim) {
  sim$latent[seq(latent_pos_Intercept, latent_pos_InteY - 1),]
})
random_effects_level <- psims |> sapply(\(sim) {
  sim$latent[which(latent_names |> startsWith("IDY"))[seq_len(NS)],]
})
random_effects_slope <- psims |> sapply(\(sim) {
  sim$latent[which(latent_names |> startsWith("IDY"))[seq_len(NS) + NS],]
})
rolling_mean <- function(x, k = 15) {
  n <- length(x)
  y <- x
  x <- c(x, rep(x[n], k))
  for (i in 1:n) {
    y[i] <- x[i:(i+k-1)] |> mean(na.rm = TRUE)
  }
y  }
phi_sims <- psims |> sapply(\(sim) {
  sim$hyperpar["Beta for eta"]
})
plot_line_names <- c('Observation', paste0('Fitted q=', ParamList$q), 'Lower Limit', 'Upper Limit
S_plot_lines <- c("Upper Limit", "Fitted q=0.5", "Lower Limit")
# Extent timeline beyond fitted portion
time_seq <- seq(0, ParamList$endpoint, ParamList$gap)
n_time_seq <- length(time_seq)
sigma <- psims |> sapply(\(sim) {
  sim$hyperpar["Precision for baseline.hazard"]^(-0.5)
})
bh_items <- latent_names |> startsWith("baseline.hazard") |> which()
bh_times <- INLAfit$summary.random[["baseline.hazard"]]$ID
bh_sims_raw <- psims |> sapply(\(sim) {
  sim$latent[bh_items,]
}) |> t()
bh_gap <- diff(bh_times) |> mean()
current_time <- bh_times |> last()
while (current_time < ParamList$endpoint) {
  current_time <- current_time + bh_gap
  k <- ncol(bh_sims_raw)
  new_sims <- bh_sims_raw[,k]*2 - bh_sims_raw[,k-1] + rnorm(n_post_sims, 0 , sigma)
  bh_sims_raw <- cbind(bh_sims_raw, new_sims)
  bh_times <- c(bh_times, current_time)
}
surv_intercept_column <- latent_names |> startsWith("Intercept") |> which()
bh_sims <- seq_len(n_time_seq) |> sapply(\(i) {
  bh_sims_raw[, match(TRUE, bh_times > time_seq[i]) - 1]
})
area_centers <- (time_seq[-1] + time_seq[-n_time_seq])/2
plots <- vector('list', length(subjects))
```

```r
# Draw plots for each subject
for (sbj_i in seq_along(subjects)) {
  sbj <- subjects[sbj_i]
  surv_sbj <- d$survDat |> filter(Id == sbj)
  rows_sbj <- which(d$longDat$Id == sbj)
  nrows_sbj <- length(rows_sbj)
  long_sbj <- d$longDat |> filter(Id == sbj)
  predicted_sbj <- predicted_INLA[rows_sbj,]
  fitted_sbj <- predicted_raw[rows_sbj,]
  random_effects_level_sbj <- random_effects_level[sbj,]
  random_effects_slope_sbj <- random_effects_slope[sbj,]
  int_seq <- rep(1, n_time_seq)
  sbj_obs_data <- data.frame(
    Item = rep(plot_line_names[1], nrows_sbj),
    Time = long_sbj$time,
    Value = long_sbj$Y,
    Component = "Observation"
    )
  sbj_explanatory_values <- long_sbj[1,3:(ncol(long_sbj)-1)] |> unlist()
  beta_pred_sbj <- (c(int_seq,
                      time_seq,
                      time_seq*sbj_explanatory_values[1],
                      rep(sbj_explanatory_values,
                          each = n_time_seq)
                     ) |> matrix(n_time_seq)
                    ) %*% beta_effects
  gamma_pred_sbj <- (c(int_seq,
                       rep(sbj_explanatory_values,
                          each = n_time_seq)
                      ) |> matrix(n_time_seq)
                     ) %*% gamma_effects
  fitted_sbj <- beta_pred_sbj +
              int_seq %*% t(random_effects_level_sbj) +
              time_seq %*% t(random_effects_slope_sbj)
  predicted_sbj <- seq_len(n_post_sims) |> sapply(\(sim) {
    rkum(n_time_seq, plogis(fitted_sbj[,sim]), pred_raw_prec[sim], ParamList$q)
  })
  sbj_intervals <- fitted_sbj |> plogis() |> apply(1, shortestinterval)
  sbj_pred_intervals <- predicted_sbj |> apply(1, shortestinterval)
  sbj_intervals <- fitted_sbj |> plogis() |> apply(1, shortestinterval)
  sbj_pred_intervals <- predicted_sbj |> apply(1, shortestinterval)
  sbj_shade_data <- data.frame(
    Time = time_seq,
    Low = sbj_intervals[1,],
    High = sbj_intervals[2,]
  )
  sbj_line_data <- data.frame(
    Item = rep(plot_line_names[-1], each = n_time_seq),
    Time = rep(time_seq, times = length(plot_line_names[-1])),
    Value = c(fitted_sbj |> plogis() |> rowMeans(),
              sbj_pred_intervals[1,] |> rolling_mean(21),
              sbj_pred_intervals[2,] |> rolling_mean(21)
              ),
    Component = "Longitudinal"
  )
  h_sbj <- exp(bh_sims + t(fitted_sbj)*phi_sims + t(gamma_pred_sbj))
  area_sbj <- (h_sbj[,-1] + h_sbj[,-n_time_seq])/2*ParamList$gap
  H_sbj <- area_sbj[, (area_centers > surv_sbj$eventTimes)] |> apply(1, cumsum)
  S_sbj <- exp(-H_sbj)
  S_sbj_times <- area_centers[(area_centers > surv_sbj$eventTimes)]
  S_sbj_n_times <- length(S_sbj_times)
  S_sbj_intervals <- S_sbj |> apply(1, shortestinterval)
  S_sbj_line_data <- data.frame(
    Item = rep(S_plot_lines, each = S_sbj_n_times),
    Time = rep(S_sbj_times, times = length(S_plot_lines)),
    Value = c(S_sbj_intervals[2,],
              S_sbj |> apply(1, mean),
              S_sbj_intervals[1,]
              ),
    Component = "Survival"
    )
  plots[[sbj_i]] <- sbj_line_data |> ggplot(aes(x = Time)) +
    geom_ribbon(aes(ymin = Low, ymax = High),
                data = sbj_shade_data,
                alpha = 0.2, fill = '#BB2200FF') +
    geom_line(aes(y = Value, group = Item, colour = Item, linetype = Component)) +
    geom_point(aes(y = Value, group = Item, colour = Item), data = sbj_obs_data) +
    scale_colour_discrete(breaks = plot_line_names[c(4,1,2,5,3)]) +
    theme(legend.title = element_blank()) +
    ylim(0,1) +
    geom_line(aes(y = Value, group = Item, colour = Item, linetype = Component),
              data = S_sbj_line_data) +
    geom_vline(xintercept = surv_sbj$eventTimes)
}
if (draw_plots == TRUE) {
  for (sbj_i in seq_along(subjects)) {
    plots[[sbj_i]] |> print()
  }
}
post_sims_list <- list(
  INLA_sims_list = psims,
  pred_raw_prec = pred_raw_prec,
  fitted_raw = predicted_raw,
  predicted_INLA = predicted_INLA,
  beta_effects = beta_effects,
  gamma_effects = gamma_effects,
  random_effects_level = random_effects_level,
  random_effects_slope = random_effects_slope,
    phi_sims = phi_sims,
    sigma = sigma,
    time_seq = time_seq,
    bh_sims = bh_sims,
    plots = plots
  )
  post_sims_list |> invisible()
}
```

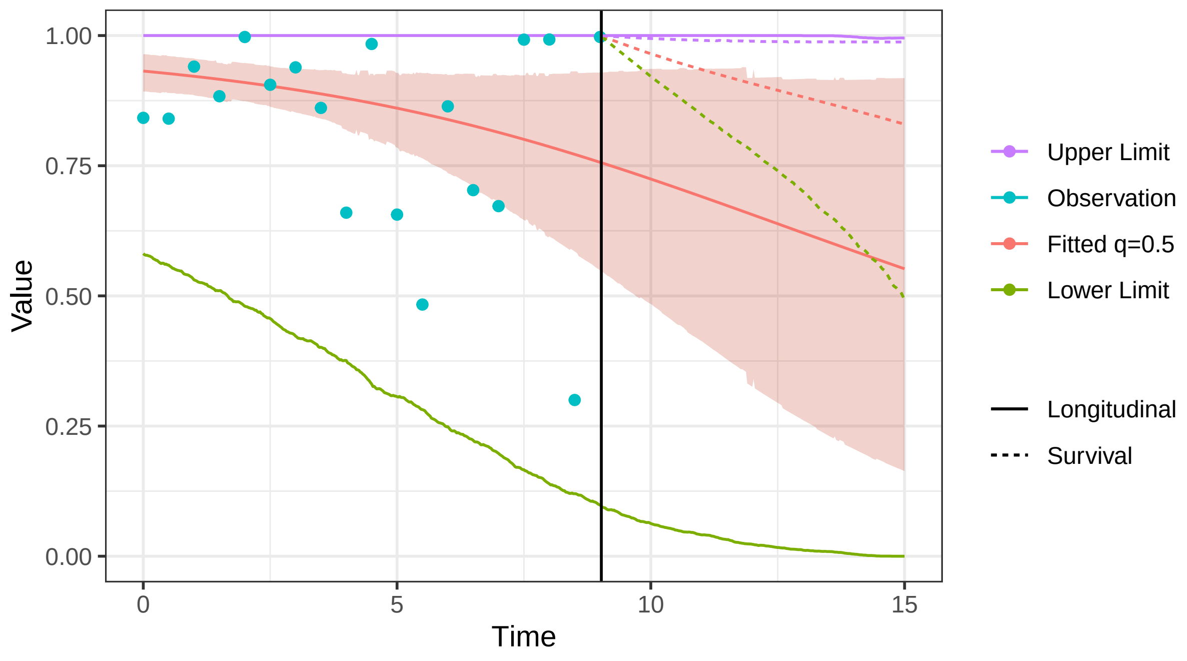
To use the function we input something like:

```
subjects_with_lots_of_data <- (d$longDat$Id |>
                                factor(levels = seq_len(ParamList$nsujet)) |>
                                table() |>
                                order(decreasing = TRUE))[1:4]
post_sims_list <- d |> pred_plot(JMinla, ParamList, subjects = subjects_with_lots_of_data)
```

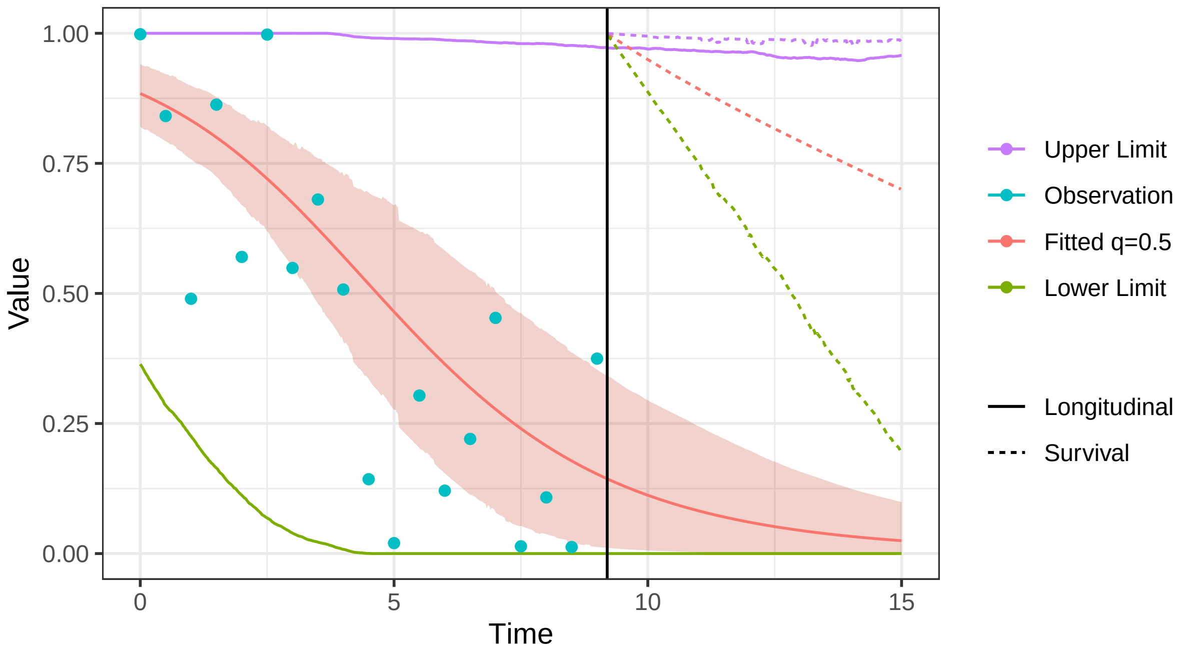




1.00
0.75
0.50
0.25
0.00
Value
0
5
10
15
Time
Upper Limit
Observation
Fitted q=0.5
Lower Limit
Longitudinal
Survival



1.00
0.75
0.50
0.25
0.00
Value
0
5
10
15
Time
Upper Limit
Observation
Fitted q=0.5
Lower Limit
Longitudinal
Survival



Dynamic predictions

The final step is to be able to update predictions for a new subject as data for that subject comes in.

One way to do this would be to construct a sampler for the random effects parameters of that specific subject given the already fitted model and the new data. The posterior of the random effects conditional on the other parameters would be proportional to the joint posterior of all parameters but with all other parameters held constant at the already simulated values. This approach would be ideal if the model was fitted on a lot of data already and the model fitting process was computationally expensive.

The better approach is to augment the existing data set with the new data, refit the model entirely, and then make predictions for that new subject as per the procedures for existing data. This approach is ideal when the model fitting was done on limited data and the model fitting process is computationally efficient.

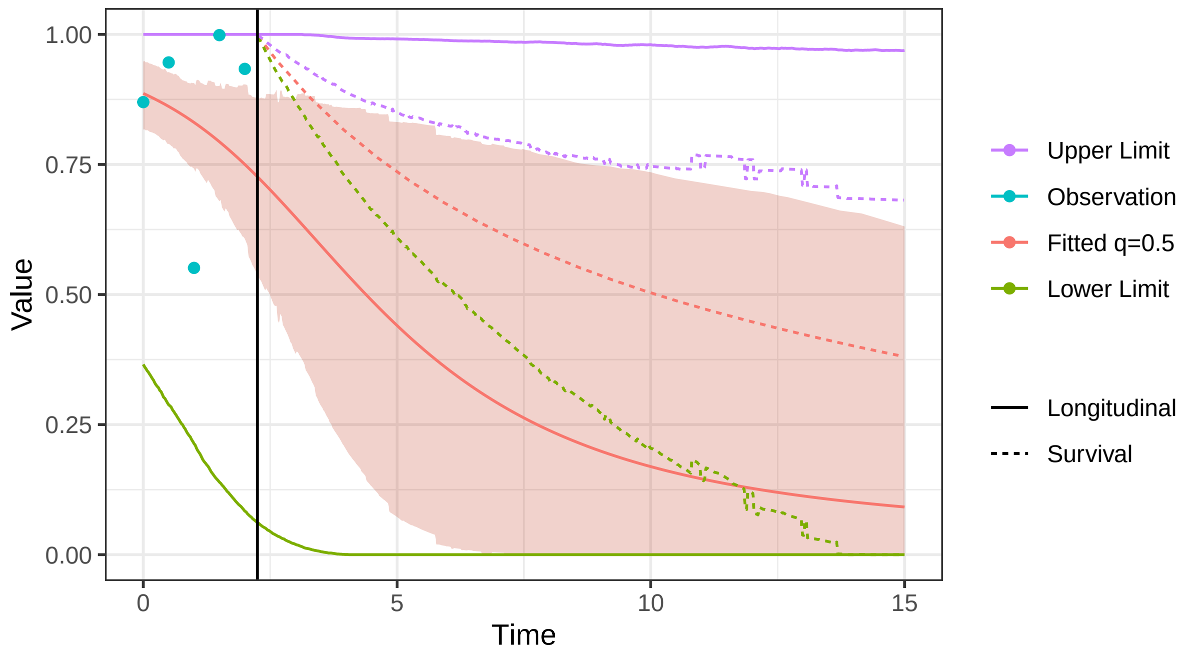
Here we will attempt to implement the refitting approach.

Adding a new subject

A new subject with a few observations is added to the end of the data set and the above procedures are rerun.

```r
new_num_times <- ceiling(runif(1,0,10))
new_times <- seq(0, (new_num_times-1))*ParamList$gapLongi
dnew <- d
newId <- max(d$survDat$Id) + 1
dnew$longDat <- dnew$longDat |> rbind(data.frame(
  Id = rep(newId, new_num_times),
  time = new_times,
  binX = rep(1, new_num_times),
  ctsX = rep(d$longDat$ctsX |> mean(), new_num_times),
  Y = rkum(new_num_times, 0.9 - (new_times)*0.07, ParamList$psi, ParamList$q)
))
dnew$survDat <- dnew$survDat |> rbind(data.frame(
  Id = newId,
  eventTimes = max(new_times) + (ParamList$gapLongi/2),
  Event = 0,
  binX = 1,
  ctsX = d$longDat$ctsX |> mean(),
  Treatment = 1
))
```

```
new_inla <- fitINLA(dnew, q = ParamList$q)
dnew |> pred_plot(new_inla, ParamList, subjects = newId)
```


Value
Time
Upper Limit
Observation
Fitted q=0.5
Lower Limit
Longitudinal
Survival