A monotone single-index model for spatially-referenced multistate current status data

Implementation of modeling framework proposed in the paper, Das, S., Chae, M., Pati, D., Bandyopadhyay, D. (2024+) "A monotone single-index model for spatially-referenced multistate current status data".

Overview

Assessment of multistate disease progression is commonplace in biomedical research, such as, in periodontal disease (PD). However, the presence of current status endpoints, where subjects' teeth transitioning through various PD states are observed at a single random inspection time, complicates the inferential framework. In addition, these endpoints are clustered (teeth within subjects), and spatially associated i.e., proximal teeth share similar PD status compared to distal ones.

We propose a Bayesian semiparametric accelerated failure time model with (spatial) random effects, and flexible errors that follow a Dirichlet process (DP) mixture of Gaussians. For elegant interpretation, we formulate a monotone single index model, whose (unknown) link function is estimated via a novel integrated basis expansion with constrained Gaussian process on the basis coefficients. Efficient posterior computing is achieved using elliptical slice sampling and fast circulant embedding techniques.

Main Functions

- **data_generate.R** - Functions to generate clustered multistate currentstatus data. The functions `data_model` and `data_misspecification` generate data from the correctly specified and misspecified models respectively.
- **SPMSM.R** - Wrapper function in R to implement our proposed Bayesian modeling framework. The function `GP_MSM` performs posterior sampling of parameters of our proposed model.
- **SOP_TP.R** - Functions to calculate the state occupation probability (SOP) and transition probability (TP) of the disease states over time, corresponding to a supplied choice of covariates. The function `est_SOP_TP` generates the SOP and TP, using posterior samples of parameters obtained from `GP_MSM`.
- **SOP_TP_SI.R** - Functions to calculate the state occupation probability (SOP) and transition probability (TP) of the disease states over time, corresponding to a supplied single index score that combining all covariates. The function `est_SOP_TP_SI` generates the SOP and TP for a given single index, using posterior samples of parameters obtained from `GP_MSM`.

Additional Functions

- **SPMSM.cpp** - Functions to implement our proposed Bayesian modeling framework. The Cpp function `GP_MSM_c` performs posterior sampling of parameters of our proposed model.
- **singleIndex.h** - Functions to fit a Bayesian monotone single index model using constrained Gaussian process and Bernstein polynomial basis functions.

- **circulantembedding.R**, **inv__chol.cpp** - Functions to estimate a monotone function using a constrained Gaussian process basis expansion, by employing the algorithm proposed by Ray et al. (2020). Code is taken from the repository, raypallavi/BNP-Computations.
- **DPM.h** - Functions to fit a Dirichlet process mixture of Gaussians, using the blocked Gibbs sampler proposed by Ishwaran & James (2001).
- **rTruncDist.h** - Functions to generate samples from truncated distributions.
- **LogLik.h** - Functions to evaluate the log-likelihood of the fitted models.
- **RcppBasic.h** - Functions for basic vector and matrix operations in Rcpp.

Additional Materials

- The **data** folder contains the motivating GAAD data set used in the application section of the paper.
- The **simulations** folder contains the code to reproduce the simulations results and generate the plots using MCMC output, as shown in the paper.

Example

We consider a sample of $n = 100$ subjects, $m = 10$ teeth for each subject, and disease states $0, 1, \ldots, K$ with $K = 3$ for each teeth. Each subject has a p-dimensional vector of covariates. Here, we take $p = 5$ with two subject level covariates (one binary and one continuous) and one tooth level covariate indicating whether a tooth is present in the upper or lower jaw.

- True regression parameter : $\beta = (-1, 1, -1)/\sqrt{3}$.
- True monotone link function : $g(x) = c \cdot (\Phi(\frac{(x+1)/2\ -\ 0.5}{0.2}) - \Phi(-\ 0.5/0.2))$, $x \in [-1, 1]$, where c is a constant chosen in such a way that the signal-to-noise ratio (SNR) is fixed at 5. Here, Φ denotes the CDF of a standard normal distribution.
- Random effects : generated from $\mathcal{N}(0, \Sigma_b)$ with $\Sigma_b = (0.1)^2(E_W - 0.9W)^{-1}$, where W and E_W respectively denote the adjacency matrix and a diagonal matrix with the jth diagonal entry representing the number of neighbors at location j of a graph connecting neighbouring teeth.
- Random errors : generated from a mixture of Gaussians, $\frac{1}{3}\mathcal{N}(-0.5, 0.1^2) + \frac{1}{3}\mathcal{N}(0, 0.1^2) + \frac{1}{3}\mathcal{N}(0.5, 0.1^2)$.
- True time to each disease state is generated after generating the relative disease increment times from a Dirichlet $(3, 4, 2)$ distribution and the random inspection times from a Gamma $(1, 0.2)$ distribution.

Data generation :

```r
library(tidyverse); library(ggpubr)
source("SPMSM.R")

set.seed(2024)
n = 100; m = 10; p = 3 ; N = n*m ; K = 3

# true regression parameter
tmp = c(-1, 1, -1) ; true.beta = tmp/norm(tmp,"2")

# true link function
true.g = function(x, a = 1){
  y = (x+1)/2
  a*(pnorm(y, mean = 0.5, sd = 0.2) - pnorm(0, mean = 0.5, sd = 0.2))
}

# parameters for generating the relative disease increment times
true.alpha = c(3, 4, 2)

# means for the mixture distribution of errors
mu_eps = c(-0.5, 0, 0.5)

# parameters for generating the random inspection times
C.param = c(1, 0.2)

# generate the data by setting SNR = 5
data_all = data_model(n = n, m = m, p = p, snr = 5, true.alpha = true.alpha,
                      true.beta = true.beta, mu_eps = mu_eps, C.param = C.param)
obs.full = data_all$obs.full
const.g = data_all$const.g
```

Specify hyperparameters :

```
# Coefficients of the constrained GP basis functions.
# We use 30 basis functions on equidistant knot points in [-1,1] for illustration.
L = 30 ; u = seq(-1, 1, length = L+1) ; a.xi = 0.01 ; b.xi = 0.01 ; eta.xi = 100 ;
# smoothness and length-scale parameters of the Matern covariance Kernel
nu.matern = 0.75 ; l.matern = l_est(nu.matern, c(u[1], u[length(u)]), 0.05)

# Variance of the regression parameters
sigma.sq.beta = 100

# Dirichlet parameters for the relative disease increment times
alpha.a = 1; alpha.lambda = 0.1; alpha.tune = 0.08

# Covariance matrix of the random effects
rho = 0.9
tmp=diag(rep(2,m/2)); tmp[1,1]=tmp[m/2,m/2]=1
for(j in 1:(m/2-1))
  tmp[j,j+1] = tmp[j+1,j] = -rho ;
S.Omega = solve(tmp) ; c.Omega = m/2 + 2

# DP mixture of Gaussians for the error density
f.mu = 0; f.nu = 1; f.lambda = 0.1; f.alpha = 2.001;
# level of truncation for running the blocked Gibbs sampler for DP
L.max = 5

# Number of MCMC samples, burn-in period and thinning interval.
M = 5000; M.burn = 1000; M.thin = 5
```

Simulate posterior samples using a Gibbs sampling algorithm :

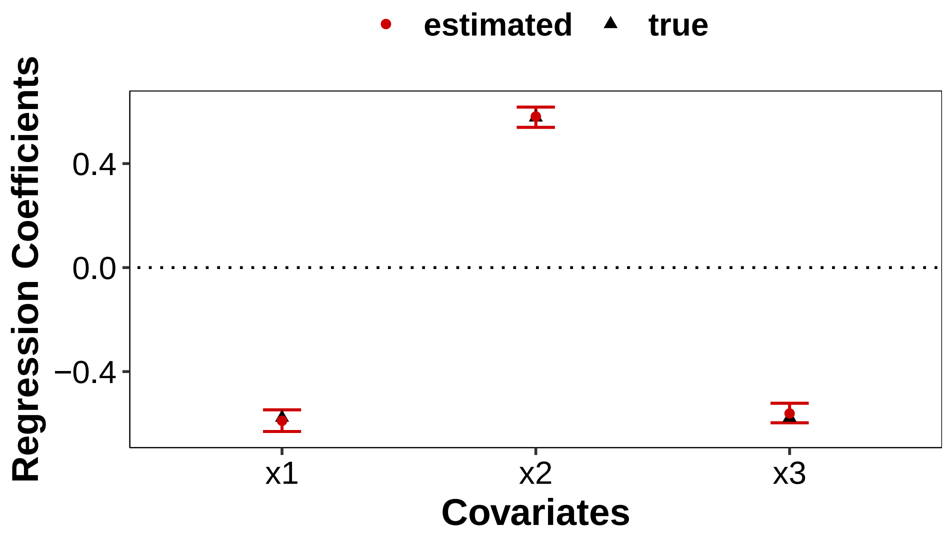
```
# run the Gibbs sampler to get posterior samples
out = GP_MSM(id = obs.full[ , "id"], tooth = obs.full[ , "tooth"], C  = obs.full[ , "C"],
             State = obs.full[ , "State"], X = obs.full[ , paste0("x", 1:p)],
             sigma.sq.beta = sigma.sq.beta, S.Omega = S.Omega, c.Omega = c.Omega,
             u = u, l.matern = l.matern, nu.matern = nu.matern, eta.xi = eta.xi,
             a.xi = a.xi, b.xi = b.xi, L = L,
             alpha.a = alpha.a, alpha.lambda = alpha.lambda, alpha.tune = alpha.tune,
             f.mu = f.mu, f.nu = f.nu, f.lambda = f.lambda, f.alpha = f.alpha, L.max = L.max,
             M = M, M.burn = M.burn, M.thin = M.thin)
```

Posterior estimate of regression parameters :

```
# Posterior mean of beta
sam.beta = out$beta
hat.beta = colMeans(out$beta)
hat.beta = hat.beta / norm(hat.beta, "2")

# 95% credible interval of beta
CI.beta = apply(sam.beta, 2, quantile, probs=c(0.025,0.975))
```

Plot showing the true and estimated regression parameter along with the 95% credible intervals :

```
# all plots follow a pre-specified theme stored in "themegg",
# which can be found in the README.Rmd file.

# data frame for plotting the posterior estimates of beta
df_beta = data.frame(x = paste0("x", 1:p), beta = c(true.beta, hat.beta),
                     type = rep(c("true", "estimated"), each = p),
                     beta_u = CI.beta[2,], beta_l = CI.beta[1, ])

# plot the posterior mean and 95% CI of beta, along with the true parameters
g.beta = ggplot(df_beta, aes(x = x, y = beta)) +
  geom_errorbar(aes(ymin=beta_l, ymax=beta_u), width=.15, color = "red3") +
  geom_point(aes(color = type, shape = type)) +
  scale_color_manual(values = c("red3", "black"))+
  geom_abline(slope = 0, linetype = "dotted")+
  xlab("Covariates") + ylab("Regression Coefficients") + themegg
g.beta
```

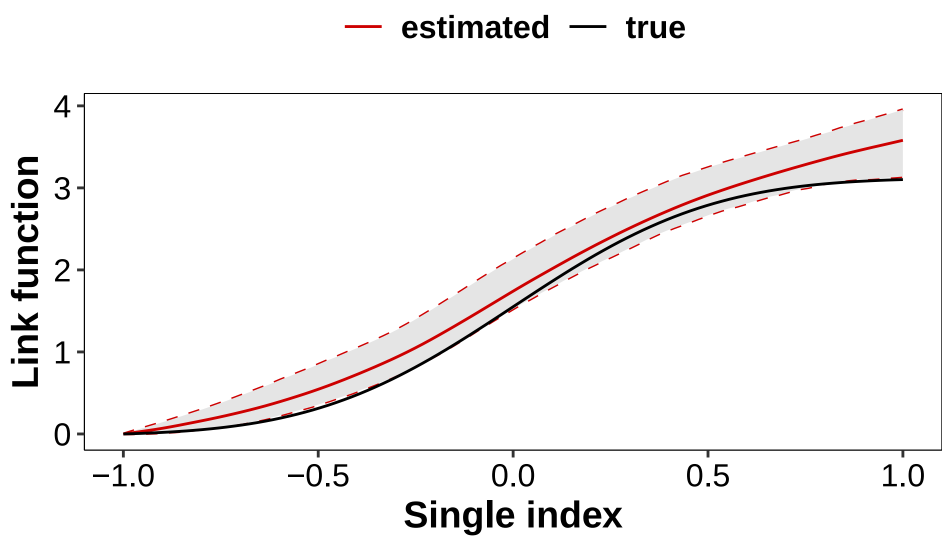


Posterior estimates of the monotone link function :

```r
# get the true link function at 100 equidistant grid points on [-1,1].
grid.g = seq(-1, 1, length= 100)
g.true = rep(0, length(grid.g))
for(i in 1:length(grid.g))
  g.true[i] = true.g(grid.g[i], a = const.g)

# posterior mean of xi
sam.xi = out$xi
hat.xi = colMeans(sam.xi)

# posterior estimates of link function
sam.g = g.est(sam.xi, grid.g, u)
hat.g = colMeans(sam.g)
band.g = apply(sam.g, 2, quantile, probs=c(0.025,0.975))
```

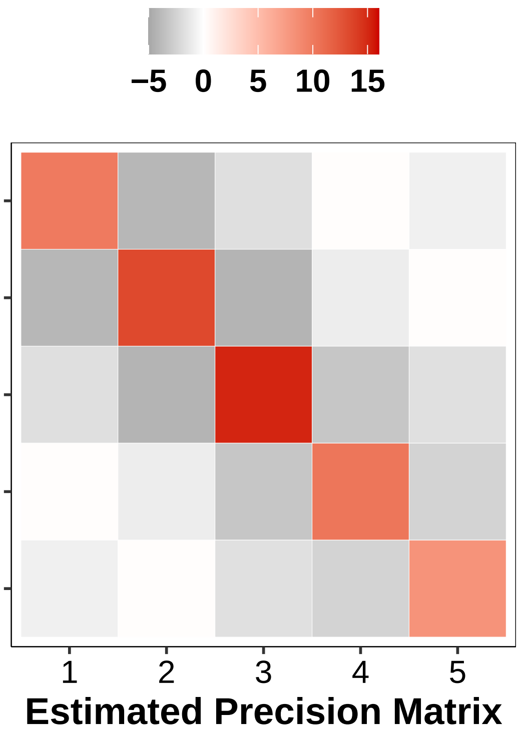
Plot showing the true and estimated link function along with the 95% credible band :

```r
# plot the posterior mean and 95% CI of estimated link function
df_g = data.frame(g = c(hat.g, g.true), type = rep(c("estimated", "true"), each = 100),
                  x = grid.g, g_u = band.g[2,], g_l = band.g[1,])

g.link = ggplot() +
  geom_line(df_g, mapping = aes(x=x, y = g_u), color = "red3", linetype = "dashed") +
  geom_line(df_g, mapping = aes(x=x, y = g_l), color = "red3", linetype = "dashed") +
  geom_ribbon(df_g, mapping = aes(x = x, ymin = g_l, ymax = g_u), fill = "grey90")+
  geom_line(df_g, mapping = aes(x= x, y = g, color = type))+
  scale_color_manual(values = c("red3", "black"))+
  xlab("Single index") +
  ylab("Link function") + themegg
g.link
```



Heatmap of the estimated precision matrix the (spatial) tooth-level random effects shared between the upper and lower jaws :

```r
m_jaw = m/2
var1 = factor(1:m_jaw)
var2 = factor(1:m_jaw, levels = m_jaw:1)
df_prec = expand.grid(var1 = var1, var2 = var2)
prec_mat = solve(out$hat_Sigma_b)
df_prec$prec = c(prec_mat)

s1 = floor(min(prec_mat))
s2 = ceiling(max(prec_mat))

g2 = ggplot(df_prec, aes(x = var1, y = var2)) +
  geom_tile(aes(fill = prec), color='white') +
  scale_fill_gradient2(low = "black", high = "red3", mid = "white",
                       midpoint = 0 ,
                       limit = c(s1, s2),
                       space = "Lab") +
  themegg + xlab("Estimated Precision Matrix")  + ylab("") + coord_fixed()
g2
```



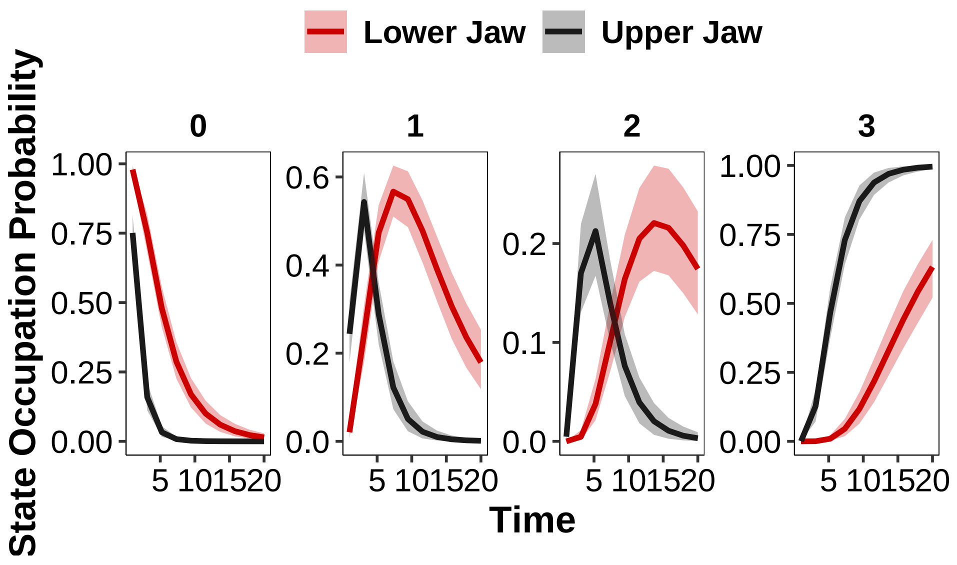
Calculate the state occupation and transition probabilities for the first tooth in the upper and lower jaws of a subject with a user-supplied choice of covariates:

```r
source("SOP_TP.R")

# covariate values of the new subject with teeth in the upper and lower jaws (x3 = 0,1)
x.new = expand.grid(x2 = 1, x3 = 0:1)
x.new = cbind(x1 = 0, x.new)
x.new.scaled = scale.new.covariate(x.new = x.new, x.data = data_all$x.scaled,
                                   wts = data_all$x.scaled.wts)
```

```
# Number of Monte Carlo samples to calculate the probabilities
M_mc = 5000
#choosing the first tooth in each jaw
tooth_num = c(1,1)

# time intervals over which the probabilities are calculated
t0 = 5; t_vec = seq(1, 20, length = 10)
t_vec_tp = t0 + seq(1, 5, length = 10)

# calculate the probabilties
out.sop.tp = est_SOP_TP(M_mc = M_mc, out = out, x.new.scaled = x.new.scaled,
                        tooth_num = tooth_num, t0 = t0,
                        t_vec = t_vec, t_vec_tp = t_vec_tp)
```

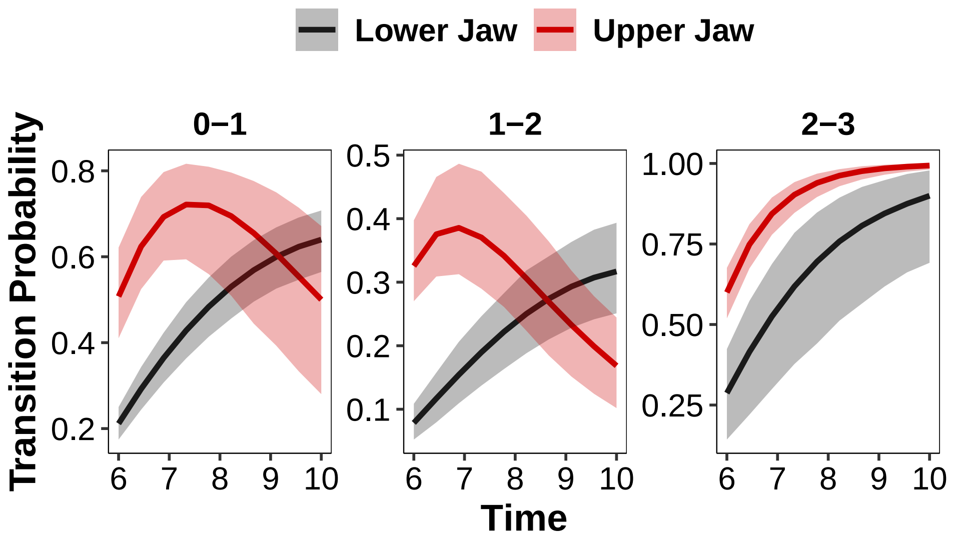
Plot the estimated state occupation probabilities, along with their 95% credible bands :

```
# plot the state occupation probability
n_time = length(t_vec) ; states = 0:K; n_state = length(states)
cov = c("Lower Jaw", "Upper Jaw")
n_cov = length(cov)
df_SOP = data.frame(SOP = out.sop.tp$plot.SOP,
                    CB.SOP.1 = out.sop.tp$CB.SOP.1,
                    CB.SOP.2 = out.sop.tp$CB.SOP.2,
                    State = rep(states, each = n_time, times = n_cov),
                    cov = rep(cov, each = n_time*n_state),
                    Time = rep(t_vec, times = n_state*n_cov))
g.SOP = ggplot(df_SOP, aes(x = Time, y = SOP)) +
  geom_line(aes(color = cov), lwd = 1) +
  geom_ribbon(aes(x = Time, ymin = CB.SOP.1, ymax = CB.SOP.2, fill = cov),
              alpha = 0.3, color = NA) +
  xlab("Time") + ylab("State Occupation Probability")+
  facet_wrap(~State, scales = "free", nrow = 1) + themegg +
  scale_fill_manual(values = c("red3", "grey10"), aesthetics = c("color", "fill"))
g.SOP
```



Plot the estimated transition probabilities, along with their 95% credible bands :

```
# plot the transition probability
trans = paste0(0:(K-1), "-", 1:K)
n_trans = length(trans)
df_TP = data.frame(TP = out.sop.tp$plot.TP,
                   CB.TP.1 = out.sop.tp$CB.TP.1,
                   CB.TP.2 = out.sop.tp$CB.TP.2,
                   State = rep(trans, each = n_time, times = n_cov),
                   cov = rep(cov, each = n_time*n_trans),
                   Time = rep(t_vec_tp, times = n_trans*n_cov))
g.TP = ggplot(df_TP, aes(x = Time, y = TP)) + ylab("TP")+
  geom_line(aes(color = cov), lwd = 1) +
  geom_ribbon(aes(x = Time, ymin = CB.TP.1, ymax = CB.TP.2, fill = cov),
            alpha = 0.3, color = NA) +
  xlab("Time") + ylab("Transition Probability")+
  facet_wrap(~State, scales = "free") + themegg +
  scale_fill_manual(values = c("grey10", "red3"), aesthetics = c("color", "fill"))
g.TP
```


Lower Jaw
Upper Jaw
Transition Probability
Time