Vignette for Bayesian Spatiotemporal Differential Process Assessment

Simulated Experimental Data

1 Installing & loading the package

```
# Install the R-package
if("devtools" %in% rownames(installed.packages()) == FALSE){
    install.packages("devtools")
}
devtools::install_github('arh926/sptwombling')
# Load the R-package
require(sptwombling)
```

2 Generating the simulated data: Pattern 1

```
####################################
# Generate Spatiotemporal Process  #
####################################
# Create Plots
Ns = 100
Nt = 9
N = Ns * Nt
pattern = "2"
seed = 1234
spt.sim1 = generate_spt_data(Ns = Ns, Nt = Nt,
                             pattern.type = pattern,
                             tau = 1,
                             gradients = TRUE,
                             derived.geom = TRUE,
                             seed = seed)
####################################
# Descriptive statistics for data #
####################################
mean(spt.sim1$y) # true value for beta0
var(spt.sim1$y); range(spt.sim1$y)
```

```
 [1] 74.59779
[1] -20.41791  20.84130
```

2.1 Plotting the simulated response and true differential processes

Using the `spt_plot()` for plotting the simulated data we obtain Figure 1. Interpolated spatial plots for the differential processes can be generated similarly using the function `spt_gradients_plot()`. If the process specification allows only the gradient and no curvature to exist then we set `only.grad.no.curv` to be `TRUE`. We show the generated plot in Figure 2.

```
###############
# == Plots == #
###############
spt_plot(coords = spt.sim1$coords,
         Nt = Nt,
         data_frame = spt.sim1$sim.pattern)



plots = list()
only.grad.no.curv = FALSE
for(i.t in 1:Nt){
  plots.time = spt_gradients_plot(coords = spt.sim1$coords,
                             data_frame =  spt.sim1$sim.pattern[, i.t],
                             grid.points = spt.sim1$grid.points,
                             gradients = spt.sim1$gradients[[i.t]],
                             only.grad.no.curv = only.grad.no.curv)
  if(only.grad.no.curv){
    plots[[i.t]] =  plot_grid(plotlist = plots.time,
                              labels = LETTERS[1:6],
                              label_size = 6,
                              nrow = 2, ncol = 3, hjust = 0.1)
  }else{
    plots[[i.t]] = plot_grid(plotlist = plots.time,
                             labels = LETTERS[1:18],
                             label_size = 6,
                             nrow = 6, ncol = 3, hjust = 0.1)
  }
}

plots[[1]]
```

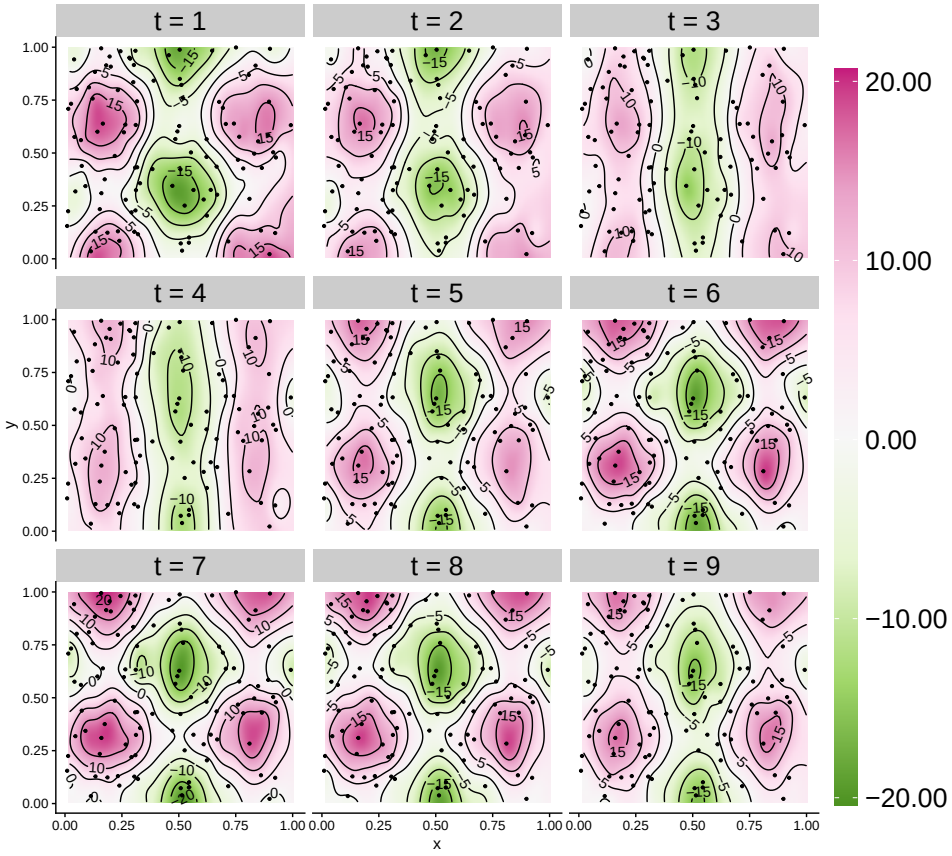

Figure 1: Spatiotemporal surfaces of patterned data (Pattern 1) used for experiments.

3 Bayesian hierarchical model fitting

```
################################
# Prior and Chain Parameters #
################################
niter = 5e3; nburn = 0.5 * niter; report = 1e2

coords = spt.sim1$coords
t = spt.sim1$t
y = spt.sim1$y

#####################################################################
# ========= if running only matern with prior on nu ========= #
#####################################################################
# Note:: Defaults to a matern if
# cov.type is not specified (estimates smoothness, nu)
# chain_y = hlmBayes_mh.spt(coords = coords, t = t,
```

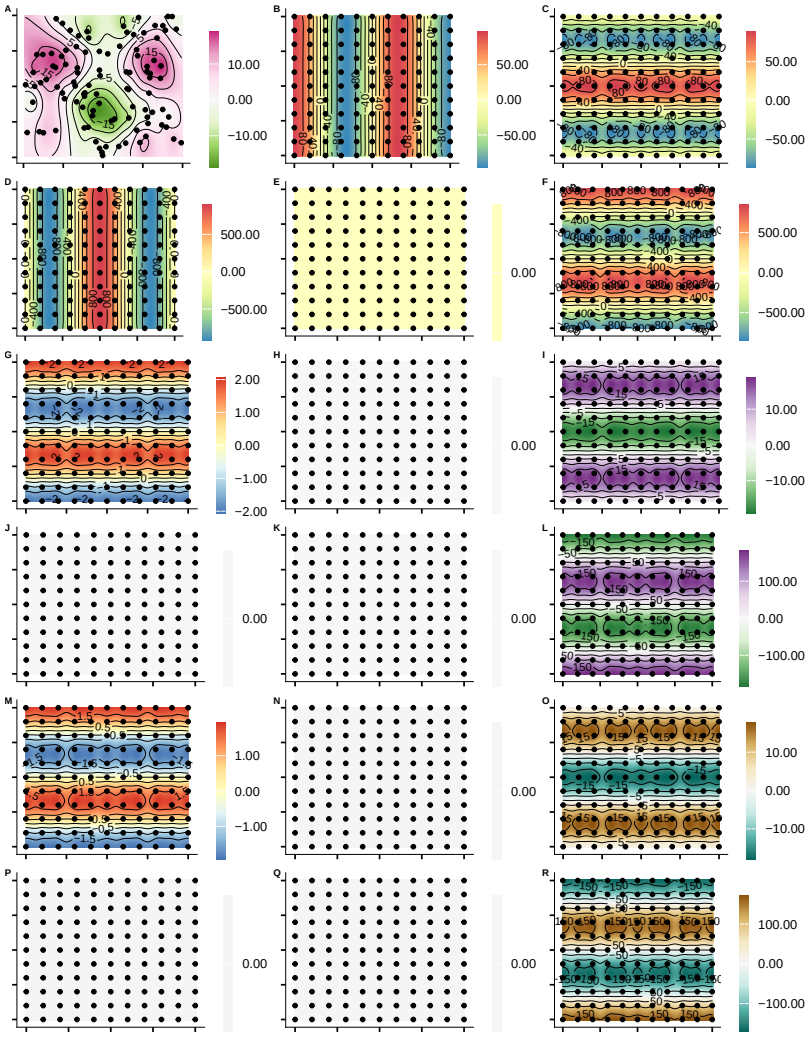

Figure 2: Spatiotemporal surfaces for true gradients and curvature for time point 1.

```
# y = y, niter = niter, nburn = nburn, report = report)
#
# plot_mcmc(samples = chain_y$nu[thin_id], cnames = "nu")
# ============================================================
ncores = detectCores() - 1
idx = c("matern1", "matern2", "gaussian")
results = mclapply(idx, function(x){
    set.seed(seed)
    sink(paste0("out/", sys.name,"-", x,"-pattern-", pattern, "-out.txt"))
    chain_y = hlmBayes_mh.spt(coords = coords, t = t, y = y, cov.type = x,
                              niter = niter, nburn = nburn, report = report)
    sink()
```

```
    return(chain_y)
    }, mc.cores = 3)


####################################
# Post-MCMC Sampling for Beta and Z #
####################################
sptRecover = sapply(idx, function(x){
    set.seed(seed)
    if(x == "matern1"){
        y = results[[1]]$y

        coords = results[[1]]$coords
        t = results[[1]]$t

        phis = results[[1]]$phis
        phit = results[[1]]$phit
        sig2 = results[[1]]$sig2
        tau2 = results[[1]]$tau2

        z.beta.sample = spsample(y = y, coords = coords, t = t,
                                 phis = phis, phit = phit, sig2 = sig2,
                                 tau2 = tau2,
                                 cov.type = x, silent = FALSE)
    }
    if(x == "matern2"){
        y = results[[2]]$y

        coords = results[[2]]$coords
        t = results[[2]]$t


        phis = results[[2]]$phis
        phit = results[[2]]$phit
        sig2 = results[[2]]$sig2
        tau2 = results[[2]]$tau2

        z.beta.sample = spsample(y = y, coords = coords, t = t,
                                 phis = phis, phit = phit, sig2 = sig2,
                                 tau2 = tau2,
                                 cov.type = x, silent = FALSE)
    }
```

```
    if(x == "gaussian"){
        y = results[[3]]$y

        coords = results[[3]]$coords
        t = results[[3]]$t

        phis = results[[3]]$phis
        phit = results[[3]]$phit
        sig2 = results[[3]]$sig2
        tau2 = results[[3]]$tau2

        z.beta.sample = spsample(y = y, coords = coords, t = t,
                                 phis = phis, phit = phit, sig2 = sig2,
                                 tau2 = tau2,
                                 cov.type = x, silent = FALSE)

    }
    return(z.beta.sample)
})
########################
# Save Results in File #
########################
results[[1]]$z = sptRecover["z", "matern1"][[1]]
results[[1]]$beta = sptRecover["beta", "matern1"][[1]]

results[[2]]$z = sptRecover["z", "matern2"][[1]]
results[[2]]$beta = sptRecover["beta", "matern2"][[1]]

results[[3]]$z = sptRecover["z", "gaussian"][[1]]
results[[3]]$beta = sptRecover["beta", "gaussian"][[1]]


# thinning
thin_id = seq(1 , (niter - nburn), by = 20)

spt.est = lapply(results, function(x){
    z.est = t(apply(x$z[thin_id,], 2, function(x){
        c(median(x), coda::HPDinterval(coda::as.mcmc(x)))
    }))
    z.est
})
```

```
pdf(paste0("plots/diagnostics-hlm-pattern", pattern, ".pdf"),
    width = 8, height = 6)
############################
# Observed vs. Fitted Plots #
############################
obvf = data.frame(rbind(cbind(y = results[[1]]$y, spt.est[[1]],
                        cov.type = 1),
                        cbind(y = results[[2]]$y, spt.est[[2]],
                        cov.type = 2),
                        cbind(y = results[[3]]$y, spt.est[[3]],
                        cov.type = 3)))
obvf$cov.type = as.factor(obvf$cov.type)
levels(obvf$cov.type) = c("matern.1.5", "matern.2.5", "gaussian")
colnames(obvf) = c("observed", "fitted", "lower.hpd",
                   "upper.hpd","cov.type")

ggplot(obvf, aes(x = fitted, y = observed)) +
    geom_point() +
    theme_bw() +
    labs(x = "Fitted", y = "Observed") +
    geom_ribbon(aes(ymin = lower.hpd, ymax = upper.hpd),
                fill = "blue", alpha = 0.3) +
    geom_line(aes(y = fitted), col = "red") +
    facet_wrap(~cov.type, ncol =  3, nrow = 1)

    ######################
    # check convergence #
    ######################
    par(mfrow = c(5,3))
    par(mar = rep(2.2, 4))

    plot_mcmc(samples = results[[2]]$beta[thin_id], cnames = "$\\beta_0$",
              true = mean(spt.sim1$y))
    plot_mcmc(samples = results[[2]]$sig2[thin_id], cnames = "$\\sigma^2$")
    plot_mcmc(samples = results[[2]]$tau2[thin_id], cnames = "$\\tau^2$",
              true = 1)
    plot_mcmc(samples = results[[2]]$phis[thin_id], cnames = "$\\phi_s$")
    plot_mcmc(samples = results[[2]]$phit[thin_id], cnames = "$\\phi_t$")

    dev.off()
```

4 Estimating Spatiotemporal Differential processes

The spatiotemporal differential processes (STDP) are estimated using a grid overlaid onto the spatial domain for every time point. The STDP are estimated at every grid location.

```
  ###################################
  # Differential Process Assessment #
  ###################################
  # Generating a Grid
  grid.points <- expand.grid(x = seq(0, 1, by = 0.1),
              y = seq(0, 1, by = 0.1))

  ########################
  # True Values for      #
  # Pattern-1            #
  ########################
  for(i.t in t){
      true.grad.sx[[i.t]] <- 30 * pi * cos(3 * pi * grid.points[,1])
      true.grad.sy[[i.t]] <- -30 * pi *
                           sin(3 * pi * grid.points[,2]) *
                           cos(i.t * pi/7)
      true.curv.sxx[[i.t]] <- -90 * pi^2 * sin(3 * pi * grid.points[,1])
      true.curv.sxy[[i.t]] <- rep(0, nrow(grid.points))
      true.curv.syy[[i.t]] <- -90 * pi^2 *
                              cos(3 * pi * grid.points[,2]) *
                              cos(i.t * pi/7)
      true.grad.t[[i.t]] <- -10 * pi *
                              cos(3 * pi * grid.points[,2]) *
                              sin(i.t * pi/7)/7
      true.grad.sxt[[i.t]] <- rep(0, nrow(grid.points))
      true.grad.syt[[i.t]] <- 30 * pi^2 *
                              sin(3 * pi * grid.points[,2]) *
                              sin(i.t * pi/7)/7
      true.curv.sxxt[[i.t]] <- rep(0, nrow(grid.points))
      true.curv.sxyt[[i.t]] <- rep(0, nrow(grid.points))
      true.curv.syyt[[i.t]] <- 90 * pi^3 *
                              cos(3 * pi * grid.points[,2]) *
                              sin(i.t * pi/7)/7
      true.grad.tt[[i.t]] <- -10 * pi^2 *
                              cos(3 * pi * grid.points[,2]) *
                              cos(i.t * pi/7)/49
      true.grad.sxtt[[i.t]] <- rep(0, nrow(grid.points))
      true.grad.sytt[[i.t]] <- 30 * pi^3 *
```

```
                            sin(3 * pi * grid.points[,2]) *
                            cos(i.t * pi/7)/49
    true.curv.sxxtt[[i.t]] <- rep(0, nrow(grid.points))
    true.curv.sxytt[[i.t]] <- rep(0, nrow(grid.points))
    true.curv.syytt[[i.t]] <- 90 * pi^4 *
                            cos(3 * pi * grid.points[,2]) *
                            cos(i.t * pi/7)/49
}

true = list(true.grad.sx = true.grad.sx,
            true.grad.sy = true.grad.sy,
            true.curv.sxx = true.curv.sxx,
            true.curv.sxy = true.curv.sxy,
            true.curv.syy = true.curv.syy,
            true.grad.t = true.grad.t,
            true.grad.sxt = true.grad.sxt,
            true.grad.syt = true.grad.syt,
            true.curv.sxxt = true.curv.sxxt,
            true.curv.sxyt = true.curv.sxyt,
            true.curv.syyt = true.curv.syyt,
            true.grad.tt = true.grad.tt,
            true.grad.sxtt = true.grad.sxtt,
            true.grad.sytt = true.grad.sytt,
            true.curv.sxxtt = true.curv.sxxtt,
            true.curv.sxytt = true.curv.sxytt,
            true.curv.syytt = true.curv.syytt)

########################################
# Estimating the Differential Processes #
########################################
spt.gradients = spt_gradients(model = results.sim[[k]],
                            cov.type.s = cov.type.s,
                            grid.points = grid.points,
                            true = true, plots = FALSE)
# root mean square predictive error
MSPE = spt.gradients$mspe
# coverage probability
CP = 1 - do.call(rbind,
            lapply(spt.gradients$plot.fn.cp,
            function(x) x$CP))
colnames(CP) = colnames(MSPE)
```

```
spt.gradients$d.ggplot[[1]]
```

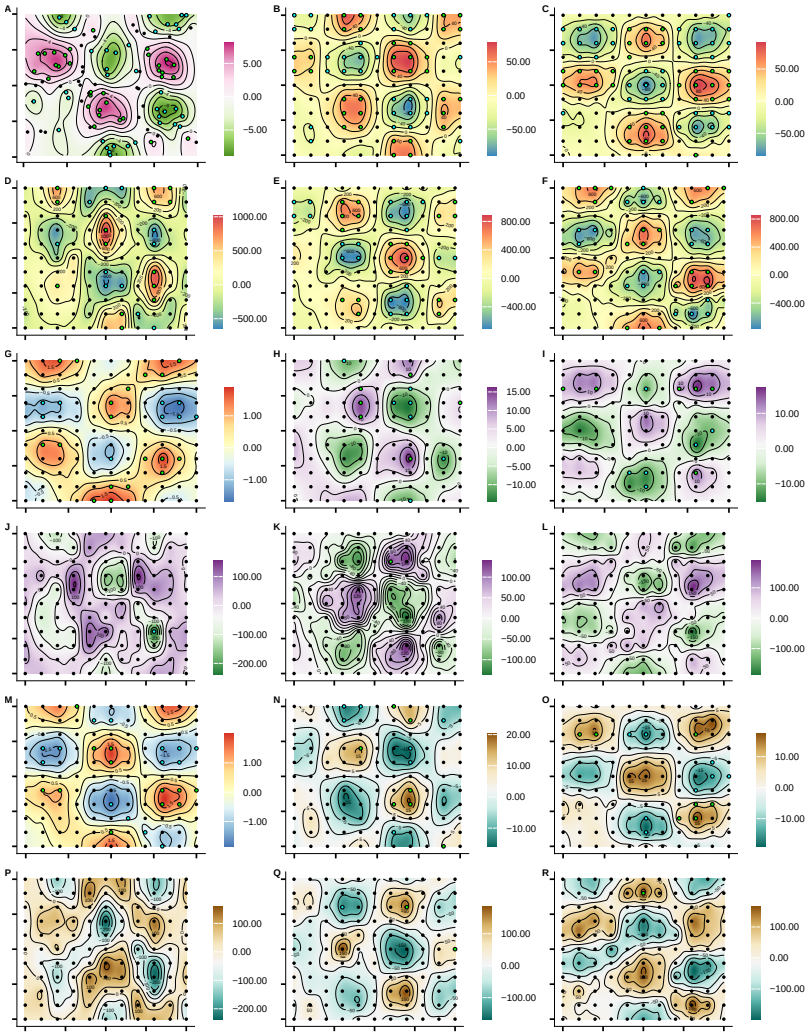

Figure 3: Spatiotemporal surfaces for estimated gradients and curvature for time point 1.

References