

# Spatio Temporal Data Analysis HW4

Hyeonki Seo

Due: Nov 28th 2021

## 1 Simulate a Strauss point process $X$

a Strauss point process with  $\beta = 0.1, \gamma = 0.5$  is generated by r function rStrauss

```
1 set.seed(2021)
2 Ran = owin(xrange = c(0,50), yrange = c(0,50))
3 X = rStrauss(beta = 0.1, gamma = 0.5, R = 1.5, W = Ran )
```

Listing 1: codes for generating Strauss point process

**Strauss process  $X$**

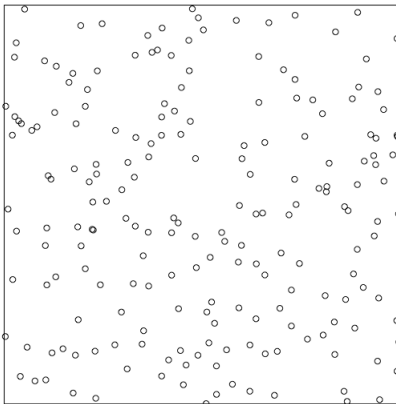


Figure 1: Strauss process

## 2 explain which part makes MCMC for a Strauss process challenging

According to the condition given by question,

$$\begin{aligned}
 P(\beta, \gamma | \mathbf{X}) &= \frac{P(\mathbf{X} | \beta, \gamma) P(\beta, \gamma)}{P(\mathbf{X})} \\
 &= \frac{P(\mathbf{X} | \beta, \gamma) P(\beta) P(\gamma)}{P(\mathbf{X})} (\because \beta, \gamma \text{ are independent}) \\
 &= \frac{1}{P(\mathbf{X})} \frac{\beta^{n(\mathbf{X})} \gamma^{s(\mathbf{X})}}{Z(\beta, \gamma)} \frac{1}{100}
 \end{aligned}$$

In order to do Metropolis-Hasting algorithm with proposal density  $q$

$$\begin{aligned}
 \text{updating ratio} &= \frac{L(\theta^*; \mathbf{X}) \pi(\theta^*) q(\theta^{(i-1)}, \theta^*)}{L(\theta^{(i-1)}; \mathbf{X}) \pi(\theta^{(i-1)}) q(\theta^*, \theta^{(i-1)})} \\
 &= \frac{\beta^{*n(\mathbf{X})} \gamma^{*s(\mathbf{X})} Z(\beta^{(i-1)}, \gamma^{(i-1)}) q(\theta^{(i-1)}, \theta^*)}{\beta^{(i-1)n(\mathbf{X})} \gamma^{(i-1)s(\mathbf{X})} Z(\beta^*, \gamma^*) q(\theta^*, \theta^{(i-1)})}
 \end{aligned}$$

But in this case,  $Z$  is intractable.

$$Z(\beta, \gamma) = \int \dots \int_{\mathbf{X}} \beta^{n(\mathbf{X})} \gamma^{s(\mathbf{X})} dX_1 \dots dX_n$$

Therefore,  $Z(\beta^*, \gamma^*)$  parts makes MCMC samples challenging

### 3 Single variable exchange algorithm

Make Single variable exchange algorithm which makes auxiliary variable  $W$  generated by same process but different parameter in order to replace normalizing part.

$$\text{updating ratio} = \frac{\beta^{*n(\mathbf{X})} \gamma^{*s(\mathbf{X})} \beta^{(i-1)n(\mathbf{W})} \gamma^{(i-1)s(\mathbf{W})} q(\theta^{(i-1)}, \theta^*)}{\beta^{(i-1)n(\mathbf{X})} \gamma^{(i-1)s(\mathbf{X})} \beta^{*n(\mathbf{W})} \gamma^{*s(\mathbf{W})} q(\theta^*, \theta^{(i-1)})}$$

Implemented code is

```

1 h <- function(X, beta, gamma) {
2   n <- npoints(X)
3   s <- (sum(pairedist(X) < 1.5) - n) / 2
4   return (beta^n * gamma ^s)
5 }
6
7 # Single variable exchange algorithm
8 sve <- function(X, beta_init, gamma_init, iter, beta_sd, gamma_sd){
9   ## empty seq
10  beta <- c()
11  gamma <- c()
12  accept_count <- 0
13
14  beta <- append(beta, beta_init)
15  gamma <- append(gamma, gamma_init)
16
17  # run MCMC
18  for (i in 1:(iter -1)) {
19
20    if (i %% 1000 == 0) {
21      print(i / iter)
22    }
23
24    # para candidate
25    temp_beta <- rnorm(1, mean = beta[i], sd = beta_sd)
26    temp_gamma <- rnorm(1, mean = gamma[i], sd = gamma_sd)
27    while (temp_beta < 0 | temp_gamma < 0 | temp_gamma > 1) {
28      temp_beta <- rnorm(1, mean = beta[i], sd = beta_sd)
29      temp_gamma <- rnorm(1, mean = gamma[i], sd = gamma_sd)
30    }
31
32    # auxiliary varriable
33    W <- rStrauss(beta = temp_beta, gamma = temp_gamma, R = 1.5, W = owin(xrange = c
34      (0,50), yrange = c(0,50)))
35
36    # density of q
37    q_n <- dnorm(x=beta[i], mean = temp_beta, sd = beta_sd, log = TRUE) +
38      dnorm(x = gamma[i], mean = temp_gamma, sd = gamma_sd, log = TRUE)
39    q_d <- dnorm(x = temp_beta, mean = beta[i], sd = beta_sd, log = TRUE)+
40      dnorm(x = temp_gamma, mean = gamma[i], sd = gamma_sd, log = TRUE)
41
42    #prior is always same because they follows uniform dist
43
44    # obs h(x)
45    h_obs_n <- log(h(X, temp_beta, temp_gamma))
46    h_obs_d <- log(h(X, beta[i], gamma[i]))
47
48    # aux h(x)
49    h_aux_n <- log(h(W, beta[i], gamma[i]))
50    h_aux_d <- log(h(W, temp_beta, temp_gamma))
51
52    # log ratio
53    log_ratio <- q_n + h_obs_n + h_aux_n - (q_d + h_obs_d + h_aux_d)
54
55    if (is.na(log_ratio)){
56      beta <- append(beta, beta[i])
57      gamma <- append(gamma, gamma[i])
58      next
59    }

```

```

60
61 # update
62 r <- runif(1)
63 log_r <- log(r)
64
65 if(log_r < log_ratio) {
66   beta <- append(beta,temp_beta)
67   gamma <- append(gamma,temp_gamma)
68   accept_count <- accept_count+1
69 } else {
70   beta <- append(beta,beta[i])
71   gamma <- append(gamma,gamma[i])
72 }
73 }
74
75 # acceptance_ratio
76 acceptance_ratio <- accept_count / iter
77
78 result <- list(beta = beta, gamma = gamma, acceptance_ratio = acceptance_ratio)
79
80 return(result)
81 }
82
83 # run MCMC
84 result <- sve(X,beta_init = 0.1, gamma_init = 0.5, iter = 10000, beta_sd = 0.05, gamma
85   _sd = 0.05 )
86
87 # trace plot
88 ## beta
89 pdf("traceplot.pdf", width=8, height=5)
90 par(mar=c(3,3,3,3),mfrow = c(1,2) )
91 plot.ts(result$beta, main = 'Trace plot of Beta')
92 abline(h = 0.1, col = 'red')
93 plot.ts(result$gamma, main = 'Trace plot of Gamma')
94 abline(h = 0.5, col = 'red')
95 dev.off()
96
97 # acceptance rate
98 result$acceptance_ratio
99
100 # posterior mean
101 mean(result$beta)
102 mean(result$gamma)
103
104 #HPD
105 quantile(result$beta, probs = c(0.025, 0.975))
106 quantile(result$gamma, probs = c(0.025, 0.975))

```

Listing 2: codes of Single variable exchange algorithm

By use this algorithm, Trace plot of beta and gamma converge well around true value

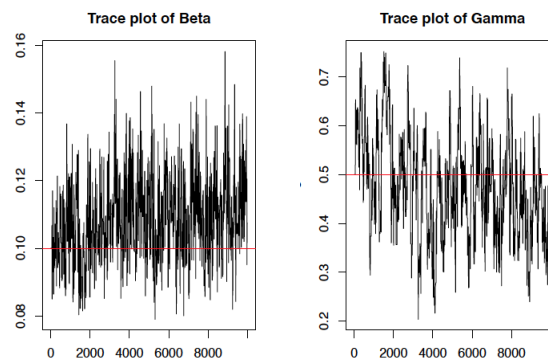


Figure 2: Trace plot of beta and gamma by Single variable exchange

Acceptance rate, posterior means, 95% HPD intervals are below

	Acceptance rate	posterior mean	95% lower bound	95% upper bound
$\beta$	0.1422	0.1084465	0.08645546	0.13406545
$\gamma$		0.4750208	0.2807820	0.6823808

Table 1: result of single variable exchange MCMC

## 4 Birth-Death algorithm

If r doesn't have perfect sampler function, Birth-death algorithm can be a good alternative. If It takes parameter values( $\beta, \gamma$ ), then it generate or delete point by given function iteratively. Consequently, The number of points converged around true process.

```

1 birth_death_MCMC <- function(X, beta, gamma, iter, domain_area){
2   n <- npoints(X)
3   s <- (sum(pairdist(X) < 1.5) - n)
4
5   accept_count <- 0
6   birth_count <- 0
7   death_count <- 0
8
9   # run MCMC
10  for (i in 1:(iter -1)){
11
12    if (i %% 1000 == 0){
13      print(i / iter)
14    }
15    # decide birth or death
16    birth <- rbinom(1,1,0.5)
17
18    if (birth == 1) {
19      # birth
20      birth_count <- birth_count +1
21
22      # generate new point
23      new_pt <- rpoint(1, win = owin(xrange = c(0,50), yrange = c(0,50)))
24
25      # insert new point
26      tempX <- X
27      tempX$n <- tempX$n +1
28      tempX$x[X$n+1] <- new_pt$x
29      tempX$y[X$n+1] <- new_pt$y
30
31      # accept prob
32      numerator <- log(h(tempX, beta = beta, gamma = gamma)) + log(domain_area)
33      denom <- log(h(X, beta = beta, gamma = gamma)) + log(X$n +1)
34      log_ratio <- numerator - denom
35
36      r <- runif(1)
37      log_r <- log(r)
38
39      if(log_r < log_ratio) {
40        X <-tempX
41        accept_count <- accept_count+1
42      }
43
44    }else{
45      # death
46      death_count <- death_count + 1
47
48      # remove one point
49      tempX <- X[-sample(1:X$n,1)]
50
51      # accept prob
52      numerator <- log(h(tempX, beta = beta, gamma = gamma)) + log(X$n-1)
53      denom <- log(h(X, beta = beta, gamma = gamma)) + log(domain_area)
54      log_ratio <- numerator - denom
55      ratio = exp(log_ratio)
56
57      r <- runif(1)

```

```

58     log_r <- log(r)
59
60     if (log_r < log_ratio){
61         X <- tempX
62         accept_count <- accept_count +1
63     }
64 }
65
66 n <- append(n, npoints(X))
67 s <- append(s, (sum(pairedist(X) < 1.5) - npoints(X))/2)
68 }
69
70 accept_ratio <- accept_count / iter
71 result <- list(n = n, s = s, accept_ratio = accept_ratio,
72               birth_count = birth_count, death_count = death_count,
73               X = X)
74 return(result)
75 }
76
77 # trace plot of n(X)
78 par(mar = c(3,3,3,3), mfrow = c(1,1))
79 plot.ts(result2$n, main = 'Traceplot of n(X)')

```

Listing 3: code of Birth-death algorithm

Start with 100 number of random generated points and 10,000 iteration. Trace plot is converged well around 190 which is almost same as number of points generated by rStrauss function

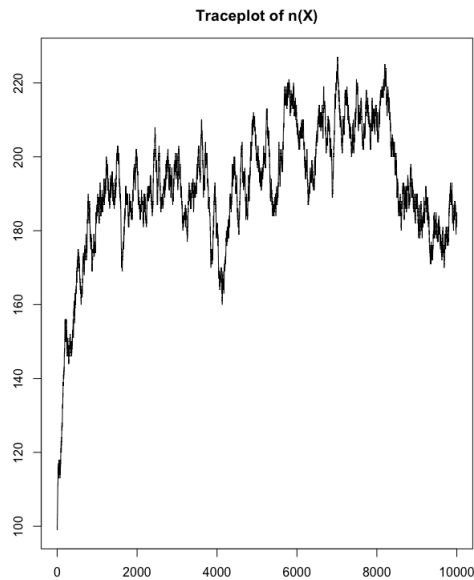


Figure 3: Trace plot of number of points by Birth-death algorithm

## 5 Double Metropolis Hasting

By Combining Single value exchange algorithm and Birth-death algorithm, Double metropolis hasting algorithm can be made. Auxiliary variable part(which is generated by rStrauss function) is replaced by Birth-death algorithm which have 200 number of randomly generated starting points and 500 iteration.

```

1 double_MH <- function(X, beta_init,gamma_init,iter1,iter2,beta_sd, gamma_sd, domain_
  area){
2   ## empty seq
3   beta <- c()
4   gamma <- c()
5   n_W <- matrix(0,ncol = iter1, nrow = iter2)
6   accept_count <- 0
7
8   beta <- append(beta,beta_init)
9   gamma <- append(gamma,gamma_init)
10
11  # outer MCMC
12  for (i in 1:(iter1 -1)) {
13
14    if (i %% 100 == 0) {
15      print(i / iter1)
16    }
17
18    # para candidate
19    temp_beta <- rnorm(1,mean = beta[i], sd = beta_sd)
20    temp_gamma <- rnorm(1,mean = gamma[i], sd = gamma_sd)
21    while (temp_beta < 0 | temp_gamma < 0 | temp_gamma > 1) {
22      temp_beta <-rnorm(1,mean = beta[i], sd = beta_sd)
23      temp_gamma <- rnorm(1,mean = gamma[i], sd = gamma_sd)
24    }
25
26    # auxiliary variable MCMC (inner MCMC)
27    tempW <- rpoint(200, win = owin(xrange = c(0,50),yrange = c(0,50)))
28    W_result <- birth_death_MCMC(tempW,beta = temp_beta,
29                                gamma = temp_gamma,iter = iter2,domain_area = domain_area)
30    n_W[,i+1] <- W_result$n
31    W <- W_result$X
32
33
34    # density of q
35    q_n <- dnorm(x=beta[i], mean = temp_beta, sd = beta_sd, log = TRUE) +
36          dnorm(x = gamma[i],mean = temp_gamma, sd = gamma_sd, log = TRUE)
37    q_d <- dnorm(x = temp_beta, mean = beta[i], sd = beta_sd, log = TRUE)+
38          dnorm(x = temp_gamma, mean = gamma[i], sd = gamma_sd, log = TRUE)
39
40    #prior is always same because they follow uniform dist
41
42    # obs h(x)
43    h_obs_n <- log(h(X,temp_beta,temp_gamma))
44    h_obs_d <- log(h(X,beta[i], gamma[i]))
45
46    # aux h(x)
47    h_aux_n <-log(h(W,beta[i], gamma[i]))
48    h_aux_d <-log(h(W,temp_beta,temp_gamma))
49
50    # log ratio
51    log_ratio <- q_n + h_obs_n + h_aux_n - (q_d + h_obs_d + h_aux_d)
52
53    if (is.na(log_ratio)){
54      beta <- append(beta,beta[i])
55      gamma <- append(gamma,gamma[i])
56      next
57    }
58
59    # update
60    r <- runif(1)
61    log_r <- log(r)
62
63
64    if(log_r < log_ratio) {

```

```

65     beta <- append(beta,temp_beta)
66     gamma <- append(gamma,temp_gamma)
67     accept_count <- accept_count+1
68   } else {
69     beta <- append(beta,beta[i])
70     gamma <- append(gamma,gamma[i])
71   }
72 }
73
74 # acceptance_ratio
75 acceptance_ratio <- accept_count / iter1
76
77 result <- list(beta = beta, gamma = gamma,
78               n_W = n_W, acceptance_ratio = acceptance_ratio)
79
80 return(result)
81 }
82
83 result3 <- double_MH(X, 0.1,0.5,10000,500,0.01,0.01,2500)
84 result3
85
86 # trace plot
87
88 pdf("5traceplot.pdf", width=8, height=5)
89 par(mar=c(3,3,3,3),mfrow = c(1,2) )
90 plot.ts(result3$beta, main = 'Trace plot of Beta')
91 abline(h = 0.1, col = 'red')
92 plot.ts(result3$gamma, main = 'Trace plot of Gamma')
93 abline(h = 0.5, col = 'red')
94 dev.off()
95 #acceptance ratio
96 result3$acceptance_ratio
97
98 # posterior mean
99 mean(result3$beta)
100 mean(result3$gamma)
101
102 #95% HPD
103 quantile(result3$beta, probs = c(0.025,0.975))
104 quantile(result3$gamma, probs = c(0.025,0.975))

```

Listing 4: code of Double Metropolis Hasting



Trace plots are below. Because I started with 200 pts which is almost same as number of rStrauss function generating pts, Trace plots are almost same as Single value exchange algorithm

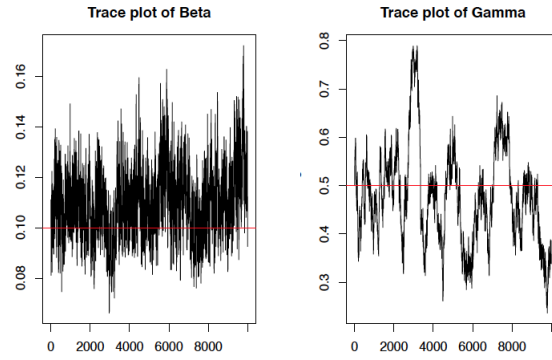


Figure 4: Trace plot of  $\beta$  and  $\gamma$  by double M-H algorithm

Acceptance rate, posterior means, 95% HPD intervals are also similar to that of Single variable exchange

	Acceptance rate	posterior mean	95% lower bound	95% upper bound
$\beta$	0.62	0.1098393	0.08503595	0.13929871
$\gamma$		0.4780175	0.3137863	0.7392927

Table 2: result of Double Metropolis-Hasting MCMC