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

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

Listing 1: codes for generating Strauss point process

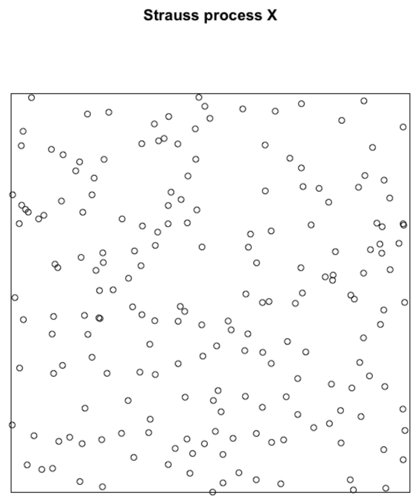

Figure 1: Strauss process

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.

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

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

```
h <- function(X, beta, gamma) {
  n <-  npoints(X)
  s <- (sum(pairdist(X) < 1.5) - n) / 2
  return (beta^n * gamma ^s)
}

# Single variable exchange algorithm
sve <- function(X,beta_init,gamma_init,iter,beta_sd, gamma_sd){
  ## empty seq
  beta <- c()
  gamma <- c()
  accept_count <- 0

  beta <- append(beta,beta_init)
  gamma <- append(gamma,gamma_init)

  # run MCMC
  for (i in 1:(iter -1)) {

    if (i %% 1000 == 0) {
      print(i / iter)
    }

    # para candidate
    temp_beta <- rnorm(1,mean = beta[i], sd = beta_sd)
    temp_gamma <- rnorm(1,mean = gamma[i], sd = gamma_sd)
    while (temp_beta < 0 | temp_gamma < 0 | temp_gamma > 1 ) {
      temp_beta <-rnorm(1,mean = beta[i], sd = beta_sd)
      temp_gamma <- rnorm(1,mean = gamma[i], sd = gamma_sd)
    }

    # auxiliary varriable
    W <- rStrauss(beta = temp_beta, gamma = temp_gamma, R = 1.5, W = owin(xrange = c
    (0,50),yrange = c(0,50)))


    # density of q
    q_n <-  dnorm(x=beta[i], mean = temp_beta, sd = beta_sd, log = TRUE) +
      dnorm(x = gamma[i],mean = temp_gamma, sd = gamma_sd, log = TRUE)
    q_d <-  dnorm(x = temp_beta, mean = beta[i], sd = beta_sd, log = TRUE)+
      dnorm(x = temp_gamma, mean = gamma[i], sd = gamma_sd, log = TRUE)

    #prior is always same because they follows uniform dist

    #  obs h(x)
    h_obs_n <- log(h(X,temp_beta,temp_gamma))
    h_obs_d <- log(h(X,beta[i], gamma[i]))

    # aux h(x)
    h_aux_n <-log(h(W,beta[i], gamma[i]))
    h_aux_d <-log(h(W,temp_beta,temp_gamma))

    # log ratio
    log_ratio <- q_n + h_obs_n + h_aux_n - (q_d + h_obs_d + h_aux_d)

    if (is.na(log_ratio)){
      beta <- append(beta,beta[i])
      gamma <- append(gamma,gamma[i])
      next
    }
```

```r

    # update
    r <- runif(1)
    log_r <- log(r)

    if(log_r < log_ratio) {
      beta <- append(beta,temp_beta)
      gamma <- append(gamma,temp_gamma)
      accept_count <- accept_count+1
    } else {
      beta <- append(beta,beta[i])
      gamma <- append(gamma,gamma[i])
    }
  }

  # acceptance_ratio
  acceptance_ratio <-  accept_count / iter

  result <- list(beta = beta, gamma = gamma, acceptance_ratio = acceptance_ratio)

  return(result)
}

# run MCMC
result <- sve(X,beta_init = 0.1, gamma_init = 0.5, iter = 10000, beta_sd = 0.05, gamma
    _sd = 0.05 )

# trace plot
## beta
pdf("traceplot.pdf", width=8, height=5)
par(mar=c(3,3,3,3),mfrow = c(1,2) )
plot.ts(result$beta, main = 'Trace plot of Beta')
abline(h = 0.1, col = 'red')
plot.ts(result$gamma, main = 'Trace plot of Gamma')
abline(h = 0.5, col = 'red')
dev.off()

# acceptance rate
result$acceptance_ratio

# posterior mean
mean(result$beta)
mean(result$gamma)

#HPD
quantile(result$beta, probs = c(0.025, 0.975))
quantile(result$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

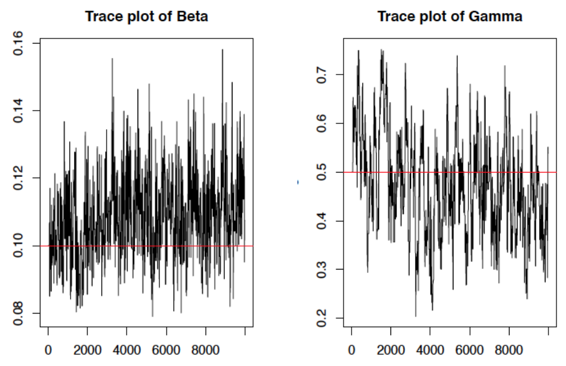

Figure 2: Trace plot of beta and gamma by Single variable exchange

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

	Acceptance rate	posterior mean	95% lower bound	95% upper bound
β	0.1422	0.1084465	0.08645546	0.13406545
γ		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(β, γ), then it generate or delete point by given function iteratively. Consequently, The number of points converged around true process.

```r
birth_death_MCMC <- function(X, beta, gamma, iter, domain_area){
  n <- npoints(X)
  s <- (sum(pairdist(X) < 1.5) - n)

  accept_count <- 0
  birth_count <- 0
  death_count <- 0

  # run MCMC
  for (i in 1:(iter -1)){

    if (i %% 1000 == 0){
      print(i / iter)
    }
    # decide birth or death
    birth <- rbinom(1,1,0.5)

    if (birth == 1) {
      # birth
      birth_count <- birth_count +1

      # generate new point
      new_pt <- rpoint(1, win = owin(xrange = c(0,50), yrange = c(0,50)))

      # insert new point
      tempX <- X
      tempX$n <- tempX$n +1
      tempX$x[X$n+1] <- new_pt$x
      tempX$y[X$n+1] <- new_pt$y

      # accept prob
      numerator <-  log(h(tempX, beta = beta, gamma = gamma)) + log(domain_area)
      denom <- log(h(X, beta = beta, gamma = gamma)) + log(X$n +1)
      log_ratio <- numerator - denom

      r <- runif(1)
      log_r <- log(r)

      if(log_r < log_ratio) {
        X <-tempX
        accept_count <- accept_count+1
      }

    }else{
      # death
      death_count <- death_count + 1

      # remove one point
      tempX <- X[-sample(1:X$n,1)]

      # accept prob
      numerator <- log(h(tempX,beta = beta, gamma = gamma)) + log(X$n-1)
      denom <- log(h(X,beta = beta, gamma = gamma)) + log(domain_area)
      log_ratio <- numerator - denom
      ratio = exp(log_ratio)

      r <- runif(1)
      log_r <- log(r)

      if (log_r < log_ratio){
        X <-  tempX
        accept_count <- accept_count +1
      }
    }

    n <-  append(n, npoints(X))
    s <- append(s, (sum(pairdist(X) < 1.5) - npoints(X))/2)
  }

  accept_ratio <- accept_count / iter
  result <- list(n = n, s = s, accept_ratio = accept_ratio,
                 birth_count = birth_count, death_count = death_count,
                 X = X)
  return(result)
}

# trace plot of n(X)
par(mar = c(3,3,3,3), mfrow = c(1,1))
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

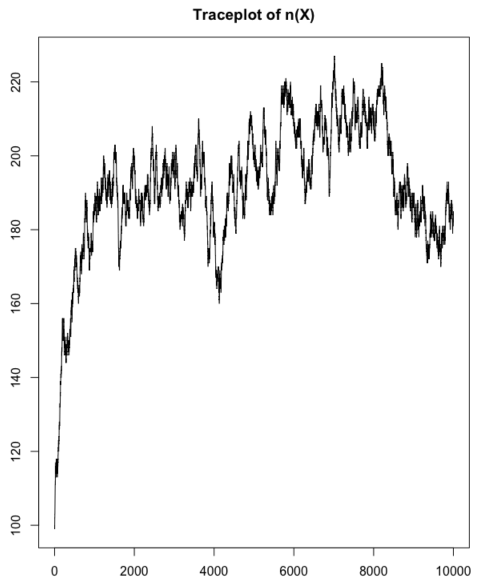

Figure 3: Trace plot of number of points by Birth-death algorithm

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.

```r
double_MH <- function(X, beta_init,gamma_init,iter1,iter2,beta_sd, gamma_sd, domain_
    area){
  ## empty seq
  beta <- c()
  gamma <- c()
  n_W <- matrix(0,ncol = iter1, nrow = iter2)
  accept_count <- 0

  beta <- append(beta,beta_init)
  gamma <- append(gamma,gamma_init)

  # outer MCMC
  for (i in 1:(iter1 -1)) {

    if (i %% 100 == 0) {
      print(i / iter1)
    }

    # para candidate
    temp_beta <- rnorm(1,mean = beta[i], sd = beta_sd)
    temp_gamma <- rnorm(1,mean = gamma[i], sd = gamma_sd)
    while (temp_beta < 0 | temp_gamma < 0 | temp_gamma > 1 ) {
      temp_beta <-rnorm(1,mean = beta[i], sd = beta_sd)
      temp_gamma <- rnorm(1,mean = gamma[i], sd = gamma_sd)
    }

    # auxiliary variable MCMC (inner MCMC)
    tempW <- rpoint(200, win = owin(xrange = c(0,50),yrange = c(0,50)))
    W_result <- birth_death_MCMC(tempW,beta = temp_beta,
                                 gamma = temp_gamma,iter = iter2,domain_area = domain_area)
    n_W[,i+1] <- W_result$n
    W <- W_result$X


    # density of q
    q_n <-  dnorm(x=beta[i], mean = temp_beta, sd = beta_sd, log = TRUE) +
      dnorm(x = gamma[i],mean = temp_gamma, sd = gamma_sd, log = TRUE)
    q_d <-  dnorm(x = temp_beta, mean = beta[i], sd = beta_sd, log = TRUE)+
      dnorm(x = temp_gamma, mean = gamma[i], sd = gamma_sd, log = TRUE)

    #prior is always same because they follow uniform dist

    #  obs h(x)
    h_obs_n <- log(h(X,temp_beta,temp_gamma))
    h_obs_d <- log(h(X,beta[i], gamma[i]))

    # aux h(x)
    h_aux_n <-log(h(W,beta[i], gamma[i]))
    h_aux_d <-log(h(W,temp_beta,temp_gamma))

    # log ratio
    log_ratio <- q_n + h_obs_n + h_aux_n - (q_d + h_obs_d + h_aux_d)

    if (is.na(log_ratio)){
      beta <- append(beta,beta[i])
      gamma <- append(gamma,gamma[i])
      next
    }

    # update
    r <- runif(1)
    log_r <- log(r)


    if(log_r < log_ratio) {
      beta <- append(beta,temp_beta)
      gamma <- append(gamma,temp_gamma)
      accept_count <- accept_count+1
    } else {
      beta <- append(beta,beta[i])
      gamma <- append(gamma,gamma[i])
    }
  }

  # acceptance_ratio
  acceptance_ratio <-  accept_count / iter1

  result <- list(beta = beta, gamma = gamma,
                 n_W = n_W, acceptance_ratio = acceptance_ratio)

  return(result)
}

result3 <- double_MH(X, 0.1,0.5,10000,500,0.01,0.01,2500)
result3

# trace plot

pdf("5traceplot.pdf", width=8, height=5)
par(mar=c(3,3,3,3),mfrow = c(1,2) )
plot.ts(result3$beta, main = 'Trace plot of Beta')
abline(h = 0.1, col = 'red')
plot.ts(result3$gamma, main = 'Trace plot of Gamma')
abline(h = 0.5, col = 'red')
dev.off()
#acceptance ratio
result3$acceptance_ratio

# posterior mean
mean(result3$beta)
mean(result3$gamma)

#95% HPD
quantile(result3$beta, probs = c(0.025,0.975))
quantile(result3$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

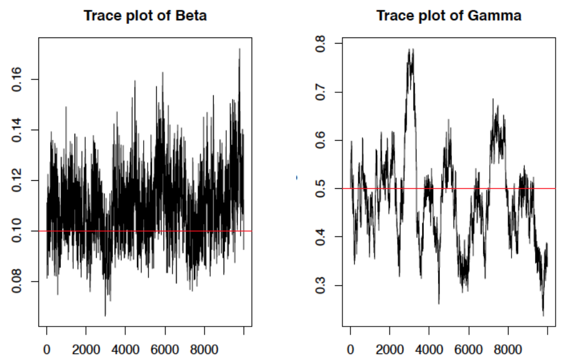

Figure 4: Trace plot of β and γ by double M-H algorithm

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
β	0.62	0.1098393	0.08503595	0.13929871
γ		0.4780175	0.3137863	0.7392927

Table 2: result of Double Metropolis-Hasting MCMC