Assignment4

Seungjun Lee

2022 11 21

Feel free to work together, but your answers/code should be your own. You must write up your solutions using LaTeX. You should submit one pdf file containing solutions/codes. In this assignment you will do Bayesian inference for an attraction-repulsion point process. Consider an attraction-repulsion point process $X = \{x_1, \cdots, x_{192}\}$ defined over circle spatial domain with radius 337.5 with center (500, 500). We set the R = 0, and we have 4 parameters, $(\lambda, \theta_1, \theta_2, \theta_3)$ to be estimated. The model is defined in [1]. We will use priors as

$$\lambda \sim U(2e-4, 6e-4), \quad \theta_1 \sim U(1,2)$$
$$\theta_2 \sim U(0,20), \quad \theta_3 \sim U(0,1)$$

1. Due to the intractable normalizing functions, we need to implement DMH algorithm. To do that, we first need to construct birth-death MCMC algorithm for inner sampler. The birth-death MCMC proposes adding a new point (birth) or removing an existing point (death). Therefore, the number of points will be changed with each iteration of birth-death MCMC algorithm. Following the description in [1], construct a birth-death MCMC sampler. Run the birth-death MCMC sampler for 10,000 iterations given $\lambda = 4e-4$, $\theta_1 = 1.5$, $\theta_2 = 10$, $\theta_3 = 0.5$ and draw a trace plot of the number of points with each iteration. Since the sampler proposes adding (or removing) a point, the number of points will be converged if your code works well. Furthermore, draw the pair correlation function of the simulated process. You can use pcf function in R. Interpret the pair correlation function from your simulated point process.

Here is the Birth and Death MCMC algorithm. I just write down the essential codes for the algorithm. Full codes can be found in Appendix 1.

```
# Propose a point uniformly in the circle domain.
unif_circle = function(center=c(500,500),rad=radius){
  r = rad * sqrt(runif(1))
  theta = runif(1) * 2 * pi
  x = center[1] + r * cos(theta)
  y = center[2] + r * sin(theta)
  return(c(x,y))
}

# numerator of likelihood function. = log(h(X|theta))
logh_fn = function(lamb=4e-4,th1=1.5,th2=10,th3=0.5,k=1.2,R=0,samples,r1,r2){ # assume k =1.2

  n = length(samples)
```

```
  samples = matrix(unlist(samples),ncol=2,byrow=TRUE)
  rr = rdist(samples)
  res = n*log(lamb)
  temp =  phi_r(th1=th1,th2=th2,th3=th3,R=R,r=rr,r1=r1,r2=r2) + diag(nrow(samples))
  res = res + sum(pmin(k,rowSums(log(temp))))
  return(res)
}

# Birth and Death MCMC function; return the length list and final samples.
Inner_sampler =function(p1=0.3,radius=337.5,iter=10,
                        lamb=4e-4,th1=1.5,th2=10,th3=0.5,k=1.2,R=0,samples,r1,r2){
  len_lst = rep(NA,iter)
  b_fn = 1/(pi*(radius^2))


  for(i in 1:iter){
    u = runif(n=1,min=0,max=1)

    if(u < p1){

      # add a point
      v1 = runif(n=1,min=0,max=1)
      add_samples = samples
      add_samples[[length(samples)+1]] = unif_circle(rad=radius)
      # acceptance ratio for adding a point
      add_ratio = (log((1-p1)) + logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                         k=k,R=R,samples=add_samples,r1=r1,r2=r2)
                   + log(d_fn(samples=add_samples))
                   -log(p1)    - logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                         k=k,R=R,samples=samples,r1=r1,r2=r2)
                   - log(b_fn) )

      if( log(v1) < add_ratio){
        samples = add_samples
      }
    }
    else if((u>=p1)&(length(samples)>0)){ # u > p1
      # delete a point
      v2 = runif(1)
      del_samples = delete_sample(samples=samples)
      # acceptance ratio for deleting a point
      del_ratio = (log((p1))    + logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                          k=k,R=R,samples=del_samples,r1=r1,r2=r2)
                   + log(b_fn)
                   -log(1-p1)    - logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                           k=k,R=R,samples=samples,r1=r1,r2=r2)
                   - log(d_fn(samples=samples))  )
      if( log(v2) < del_ratio){
        samples = del_samples
      }
    }
    len_lst[i] = length(samples)
    if(i %% (iter/10) == 0) print(i)
```

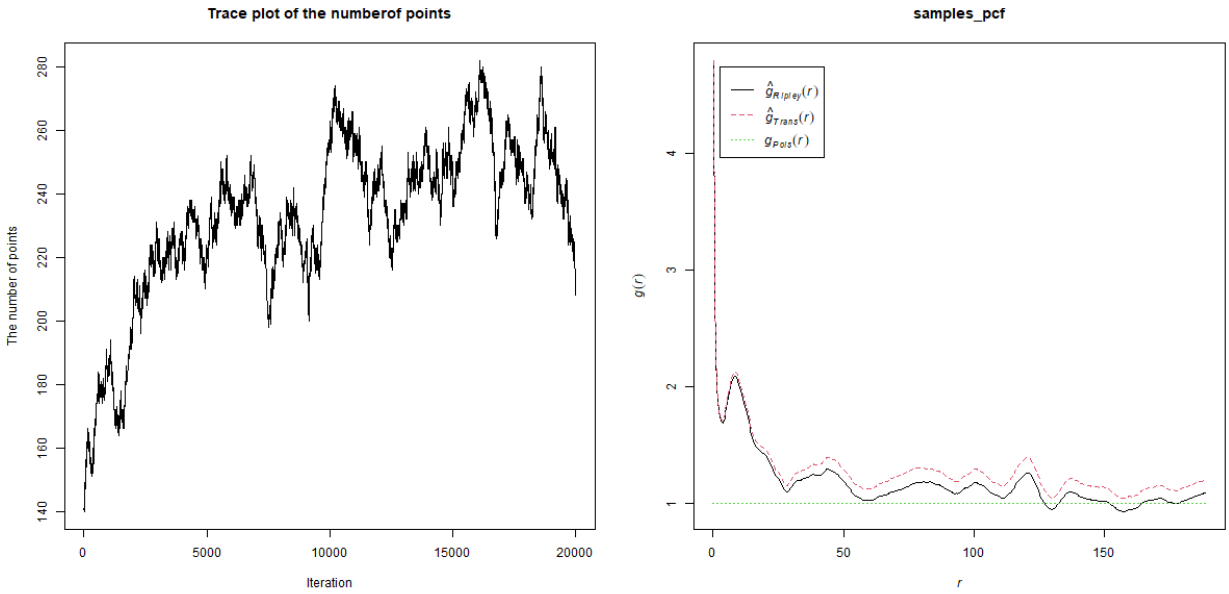
```
  }
  return(list("len_lst"=len_lst,"samples"=samples))
}

# initial points
set.seed(4)
center <- c(500,500); radius <- (1350/4)
W <- disc(radius, center)
X <- rpoispp(lambda=4e-4, win= W )
initial <- cbind(X$x,X$y)
initialdist <- rdist( initial )

samples = list()
for(i in 1:X$n){
  samples[[i]] = c(X$x[i],X$y[i])
}

# Simulation
res1 = Inner_sampler(p1=0.3,radius=337.5,iter=10000,
                     lamb=4e-4,th1=1.5,th2=10,th3=0.5,k=1.2,R=0,samples,r1,r2)
```

```
# trace plot of n
ts.plot(res1$len_lst)
# pcf plot
samples_ppp = ppp(x_coords,y_coords,window=disc(radius=377.5,centre=c(500,500)))
samples_pcf = pcf(samples_ppp)
plot(samples_pcf)
```



Except the r=0 ~ 10, as the distance increases, the correlation decreases(close to 1). Around r=0~10, There is a increasing correlation relationship when the distance is large.

2. Using the birth-death MCMC sampler as an inner sampler, construct a double Metropolis Hastings [2] for the attraction-repulsion point process model. This is a nested MCMC algoritm, because for each $\lambda, \theta_1, \theta_2, \theta_3$ update (outer MCMC), we need to simulate a point process with birth-death MCMC (inner MCMC). In hw4.RData, I provide the dataset X, which is simulated from the attraction repulsion point process model. Fit the attraction-repulsion point process model for this simulated dataset. Here, run the birth-death MCMC for 2000 iterations to generate an auxiliary variable in the algorithm. Report trace plots, acceptance rate, posterior means, 95% HPD intervals of the model parameters.

Here is Double Metropolis Hastings(DMH) algorithm. Same as problem 1, I just write down the essential codes for the algorithm. Full codes can be found in Appendix 2.

```
# Proposal distribution for DMH.
proposal = function(curr_ths,step_size){
  condition = FALSE
  while(!condition){
    next_ths = rmvnorm(1,mean=curr_ths,sigma=diag(step_size))
    condition = (2e-4 <= next_ths[1])*(next_ths[1] <= 6e-4)*
      (1 <= next_ths[2])*(next_ths[2]<=2)*
      (0 <= next_ths[3])*(next_ths[3] <=20)*(0 <= next_ths[4])*(next_ths[4] <= 1)
  }
  return(next_ths)
}

# Double Metropolis Hastings(DMH) function.
outer_sampler = function(out_iter=10,inner_iter=2000,obs=X_pattern){
  theta_lst = matrix(NA,ncol=4,nrow=out_iter)
  # init
  prior_lamb = runif(1,2e-4,6e-4)
  prior_th1 =runif(1,1,2)
  prior_th2 =runif(1,0,20)
  prior_th3 =runif(1,0,1)
  prior_k = 1.2 # not r.v ?
  curr_ths = c(prior_lamb,prior_th1,prior_th2,prior_th3)

  r = solving_r(th1=curr_ths[2],th2=curr_ths[3],th3=curr_ths[4])
  curr_r1 = r$r1
  curr_r2 = r$r2

  step_size = c((6e-4 - 2e-4)/100,1/100,20/100,1/100)
  theta_lst[1,] = curr_ths

  for(i in 2:out_iter){
    next_ths = proposal(curr_ths=curr_ths,step_size=step_size)
    print("************************")
    print(paste0(i,"th iteration"))
    print("************************")

    next_r = solving_r(th1=next_ths[2],th2=next_ths[3],th3=next_ths[4])
```

```
    next_r1 = next_r$r1 ;  next_r2 = next_r$r2

    Y_info = Inner_sampler(p1=0.5,radius=337.5,iter=inner_iter,lamb=next_ths[1],
                           th1=next_ths[2],th2=next_ths[3],th3=next_ths[4],
                           k=1.2,R=0,samples=X_pattern,r1=next_r1,r2=next_r2)
    Y = Y_info$samples

    log_acc = logh_fn(lamb=next_ths[1],th1=next_ths[2],th2=next_ths[3],th3=next_ths[4],
                      k=1.2,R=0,samples=obs,r1=next_r1,r2=next_r2) +
      logh_fn(lamb=curr_ths[1],th1=curr_ths[2],th2=curr_ths[3],th3=curr_ths[4],
              k=1.2,R=0,samples=Y,r1=curr_r1,r2=curr_r2) -
      logh_fn(lamb=curr_ths[1],th1=curr_ths[2],th2=curr_ths[3],th3=curr_ths[4],
              k=1.2,R=0,samples=obs,r1=curr_r1,r2=curr_r2) -
      logh_fn(lamb=next_ths[1],th1=next_ths[2],th2=next_ths[3],th3=next_ths[4],
              k=1.2,R=0,samples=Y,r1=next_r1,r2=next_r2)

    if(log(runif(1)) < log_acc){
      theta_lst[i,] = next_ths
      curr_ths = next_ths
      curr_r1 = next_r1
      curr_r2 = next_r2
    }
    else{
      theta_lst[i,] = curr_ths
    }
  }
  return(theta_lst)
}
out_res = outer_sampler(out_iter=2000,inner_iter=2000,obs=X_pattern)
```

{r,include=FALSE} # load(out_res) #

```
ts.plot(out_res[,1],xlab="Iteration",ylab="Theta",main="Trace plot of parameters")
ts.plot(out_res[,2],xlab="Iteration",ylab="Theta",main="Trace plot of parameters")
ts.plot(out_res,xlab="Iteration",ylab="Theta",main="Trace plot of parameters",
        col=c(1,2,3,4),lty=c(1,2,3,4),lwd=1.5)
legend("bottomleft",fill=c(1,2,3,4),col=c(1,2,3,4),legend=c("A","B","C","D"))
```

{r} # theta1 = out_res[,1] # # Accepatnce rate # sum(diff(theta1) !=0)/length(theta1) # # Posterior mean # apply(out_res,2,mean) # # 95% HPD interval # apply(out_res,2,quantile,probs=c(0.025,0.975)) # #

Appendix(full codes)

1.

```
# Propose a point uniformly in the circle domain.
unif_circle = function(center=c(500,500),rad=radius){
  r = rad * sqrt(runif(1))
  theta = runif(1) * 2 * pi
  x = center[1] + r * cos(theta)
  y = center[2] + r * sin(theta)
  return(c(x,y))
}

# solution for r1 r2 solving two equations (see Goldstein et al, 2015)
solving_r = function(th1=1.5,th2=10,th3=0.5,R=0){
  a = (th2 - R)^2 / (th1*th3^2)
  b = th3^2*(th1 - 1)
  d = 27*a*b^2
  c = (d + sqrt((d+2)^2-4) + 2)^(1/3)
  deltar = 1/(3*b) * ( c/2^(1/3) + 2^(1/3)/c + 1 )

  r1 = a / deltar^1.5 + th2
  r2 = r1 - sqrt(deltar)
  return(list("r1"=r1,"r2"=r2))
}

r = solving_r(th1=1.5,th2=10,th3=0.5)
r1 = r$r1
r2 =r $r2

# correlation function using the distance.
phi_r = function(th1=1.5,th2=10,th3=0.5,R=0,r,r1,r2){ # R=0 is default in this problem.

  result = 0 + (r>100)*1 + ((r>r1) & (r<=100))*(1 + 1/( th3*(r-r2) )^2) +
    ((r>R) & (r<= r1)) * (th1 - ( sqrt(th1)*(r-th2)/(th2-R) )^2)
  return(result)
}

x = seq(-5,50,by=0.1)
plot(x,phi_r(r=x,r1=r1,r2=r2),type="l")

phi_r(r=0,r1=r1,r2=r2)

# numerator of likelihood function. = log(h(X|theta))
logh_fn = function(lamb=4e-4,th1=1.5,th2=10,th3=0.5,k=1.2,R=0,samples,r1,r2){ # assume k =1.2

  n = length(samples)
  samples = matrix(unlist(samples),ncol=2,byrow=TRUE)
  rr = rdist(samples)
  res = n*log(lamb)
  temp =  phi_r(th1=th1,th2=th2,th3=th3,R=R,r=rr,r1=r1,r2=r2) + diag(nrow(samples))
  res = res + sum(pmin(k,rowSums(log(temp))))
  return(res)
```

```
}


# used in Birth and Death MCMC
d_fn = function(samples){
  return(1/(length(samples)))
}

# For death part of the Birth and Death MCMC.
# birth part is nested in the Inner_sampler function.
delete_sample = function(samples){
  n = length(samples)
  drop_idx = sample(1:n,1)
  return(samples[-drop_idx])
}

del_samples = delete_sample(samples=samples)

# Birth and Death MCMC function; return the length list and final samples.
Inner_sampler =function(p1=0.3,radius=337.5,iter=10,lamb=4e-4,
                        th1=1.5,th2=10,th3=0.5,k=1.2,R=0,samples,r1,r2){
  len_lst = rep(NA,iter)
  b_fn = 1/(pi*(radius^2))


  for(i in 1:iter){
    u = runif(n=1,min=0,max=1)

    if(u < p1){

      # add a point
      v1 = runif(n=1,min=0,max=1)

      add_samples = samples
      add_samples[[length(samples)+1]] = unif_circle(rad=radius)

      add_ratio = (log((1-p1)) + logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                         k=k,R=R,samples=add_samples,r1=r1,r2=r2)
                   + log(d_fn(samples=add_samples))
                   -log(p1)    - logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                         k=k,R=R,samples=samples,r1=r1,r2=r2)
                   - log(b_fn) )

      if( log(v1) < add_ratio){
        samples = add_samples
      }

    }
    else if((u>=p1)&(length(samples)>0)){ # u > p1
      # delete a point

      v2 = runif(1)
      del_samples = delete_sample(samples=samples)
```

```
      del_ratio = (log((p1))    + logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                          k=k,R=R,samples=del_samples,r1=r1,r2=r2)
                   + log(b_fn)
                   -log(1-p1)    - logh_fn(lamb=lamb,th1=th1,th2=th2,th3=th3,
                                           k=k,R=R,samples=samples,r1=r1,r2=r2)
                   - log(d_fn(samples=samples))  )


      if( log(v2) < del_ratio){
        samples = del_samples
      }
    }
    len_lst[i] = length(samples)
    if(i %% (iter/10) == 0) print(i)
  }
  return(list("len_lst"=len_lst,"samples"=samples))
}
```

2.

```
load("hw4.RData")
plot(X)
polygon(500+x,500+y)

# obs data : matrix -> list
X_pattern = list()
for(i in 1:nrow(X)){
  X_pattern[[i]] = X[i,]
}

# Proposal distribution for DMH.
proposal = function(curr_ths,step_size){
  condition = FALSE
  while(!condition){
    next_ths = rmvnorm(1,mean=curr_ths,sigma=diag(step_size))
    condition = (2e-4 <= next_ths[1])*(next_ths[1] <= 6e-4)*(1 <= next_ths[2])*(next_ths[2]<=2)*
      (0 <= next_ths[3])*(next_ths[3] <=20)*(0 <= next_ths[4])*(next_ths[4] <= 1)
  }
  return(next_ths)
}

# Double Metropolis Hastings(DMH) function.
outer_sampler = function(out_iter=10,inner_iter=2000,obs=X_pattern){
  theta_lst = matrix(NA,ncol=4,nrow=out_iter)
  # init
  prior_lamb = runif(1,2e-4,6e-4)
  prior_th1 =runif(1,1,2)
  prior_th2 =runif(1,0,20)
  prior_th3 =runif(1,0,1)
  prior_k = 1.2 # not r.v ?
  curr_ths = c(prior_lamb,prior_th1,prior_th2,prior_th3)
```

```
  r = solving_r(th1=curr_ths[2],th2=curr_ths[3],th3=curr_ths[4])
  curr_r1 = r$r1
  curr_r2 = r$r2

  step_size = c((6e-4 - 2e-4)/100,1/100,20/100,1/100)
  theta_lst[1,] = curr_ths

  for(i in 2:out_iter){
    next_ths = proposal(curr_ths=curr_ths,step_size=step_size)
    print("************************")
    print(paste0(i,"th iteration"))
    print("************************")

    next_r = solving_r(th1=next_ths[2],th2=next_ths[3],th3=next_ths[4])
    next_r1 = next_r$r1 ;  next_r2 = next_r$r2

    Y_info = Inner_sampler(p1=0.5,radius=337.5,iter=inner_iter,
                           lamb=next_ths[1],th1=next_ths[2],th2=next_ths[3],th3=next_ths[4],
                           k=1.2,R=0,samples=X_pattern,r1=next_r1,r2=next_r2)
    Y = Y_info$samples

    log_acc = logh_fn(lamb=next_ths[1],th1=next_ths[2],th2=next_ths[3],th3=next_ths[4],
                      k=1.2,R=0,samples=obs,r1=next_r1,r2=next_r2) +
      logh_fn(lamb=curr_ths[1],th1=curr_ths[2],th2=curr_ths[3],th3=curr_ths[4],
              k=1.2,R=0,samples=Y,r1=curr_r1,r2=curr_r2) -
      logh_fn(lamb=curr_ths[1],th1=curr_ths[2],th2=curr_ths[3],th3=curr_ths[4],
              k=1.2,R=0,samples=obs,r1=curr_r1,r2=curr_r2) -
      logh_fn(lamb=next_ths[1],th1=next_ths[2],th2=next_ths[3],th3=next_ths[4],
              k=1.2,R=0,samples=Y,r1=next_r1,r2=next_r2)

    if(log(runif(1)) < log_acc){
      theta_lst[i,] = next_ths
      curr_ths = next_ths
      curr_r1 = next_r1
      curr_r2 = next_r2
    }
    else{
      theta_lst[i,] = curr_ths
    }
  }
  return(theta_lst)
}
out_res = outer_sampler(out_iter=2000,inner_iter=2000,obs=X_pattern)
save(out_res,file="DMH_sample.RData")

out_res

#png(filename="hw4_traceplot.png",width=600,height=600)
ts.plot(out_res[,1],xlab="Iteration",ylab="Theta",main="Trace plot of parameters")
ts.plot(out_res[,2],xlab="Iteration",ylab="Theta",main="Trace plot of parameters")
ts.plot(out_res,xlab="Iteration",ylab="Theta",main="Trace plot of parameters",
        col=c(1,2,3,4),lty=c(1,2,3,4),lwd=1.5)
legend("bottomleft",fill=c(1,2,3,4),col=c(1,2,3,4),legend=c("A","B","C","D"))
```

```r
#dev.off()

theta1 = out_res[,1]
sum(diff(theta1) !=0)/length(theta1) # acc rate
apply(out_res,2,mean) # posterior mean
apply(out_res,2,quantile,probs=c(0.025,0.975)) # 95% HPD interval
```