# GP-06-05-2023

Hengde Ouyang

2023-06-05

## 1. GP independent sampling with true MH ratio

Indeed, when we increase our iteration, we will eventually see the convergence pattern.

### Function

```
MH_GP_Sampling <- function(Y,delta,tau,
                           A,beta0,sigma0,var.prop,
                           m,B,eta,K,
                           Wmat_option=0){


  accept_beta = 0
  accept_lambda = 0
  beta = beta0
  lambda = lambda0
  sigma = sigma0

  # What we want to record
  BETA = matrix(0,m,dim(A)[1])
  LAMBDA = matrix(0,m,dim(A)[2])
  C_stat = c()


  # For safety m>B
  if (B>m){
    B = 0
  }


  # 0 means we use Harrell C statistics
  # 1 means we use Uno C statistics
  if (Wmat_option==0){
    Wmat <- HarrellC_Wmat(Y, delta, tau)
  }else if (Wmat_option==1){
    Wmat <- UnoC_Wmat(Y, delta, tau)
  }else{  # Other Possible C index...
```

```
  Wmat <- HarrellC_Wmat(Y, delta, tau)
}


for (i in 1:m){

  # Sample beta from proposal distribution
  beta.p = t(rmvnorm(1,beta,var.prop))


  # Compute theta from current and last iteration
  theta.p = beta.p
  theta = beta




  # Compute C-statistics from current and last iteration
  HC.p = HarrellC(theta.p, Wmat)
  HC = HarrellC(theta, Wmat)

  # Record C-statistics from last iteration
  C_stat = c(C_stat,HC)


  ##############################################
  # Compute log of MH ratio

  lrMH = eta*log(HC.p) +
        dmvnorm(as.numeric(beta.p),beta0,K,log=T)-
        eta*log(HC) -
        dmvnorm(as.numeric(beta),beta0,K,log=T)

    if (log(runif(1))<lrMH){
      beta = beta.p
      accept_beta = accept_beta + 1
    }
    BETA[i,] = beta
  ##############################################



  ###################################################################
  # Compute log of MH_lambda ratio
  lambda.p = exp(t(rnorm(dim(A)[2],log(lambda),rep(1,dim(A)[2]))))
  lambda.p = as.vector(lambda.p)
  K.p = matrix_K(A,lambda.p)

  lrMH_lambda = dmvnorm(as.numeric(beta),beta0,K.p,log=T)+
                sum(dgamma(lambda.p,alpha0,v0,log = T))  -
                dmvnorm(as.numeric(beta),beta0,K,log=T) -
                sum(dgamma(lambda,alpha0,v0,log = T))
```

```
    if (log(runif(1))<lrMH_lambda){
        lambda = lambda.p
        K = K.p
        accept_lambda = accept_lambda + 1
      }
      LAMBDA[i,] = lambda
  }
  ##########################################################################



  if (B == 0){
    return(list(BETA=BETA,
                LAMBDA = LAMBDA,
                accept_beta=accept_beta/m,
                accept_lambda=accept_lambda/m,
                C_stat = C_stat))
  }else{
    return(list(BETA=BETA[-c(1:B),],
                LAMBDA = LAMBDA[-c(1:B),],
                accept_beta=accept_beta/m,
                accept_lambda=accept_lambda/m,
                C_stat = C_stat[-c(1:B)]))
  }


}
```

## Try using lung data set first

The number of observetions is relatively small in this dataset.

```
no_na_lung = na.omit(lung)


# Input data
Y = no_na_lung$time
delta= no_na_lung$status - 1
tau = 2500
A <- model.matrix(time ~ -1+ inst+ age + sex + ph.ecog + ph.karno + pat.karno
                  +meal.cal+wt.loss,
                  data=no_na_lung)


# Gaussian Process

beta0 = rep(0,dim(A)[1])
sigma0 = rep(1,dim(A)[1])
lambda0 = rep(1,dim(A)[2])

# A relatively Small data set, can increase the iteration
m = 52000
```

```
B = 200
eta = length(Y)
Wmat_option = 0

var.prop = diag(0.01,dim(A)[1])
K = as.matrix(matrix_K(A,lambda0))
alpha0 = 1
v0 = 1
```

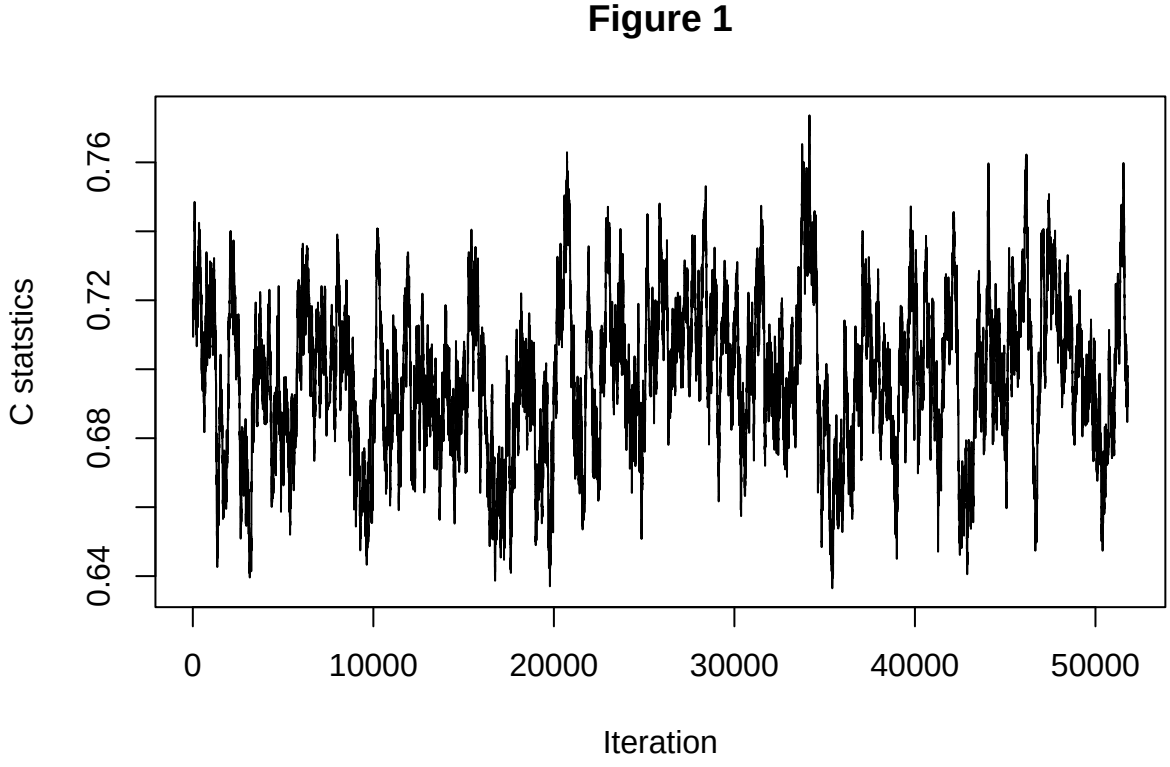
```
system.time({
  result_GP1 = MH_GP_Sampling(Y,delta,tau,
                          A,beta0,sigma0,var.prop,
                          m,B,eta,K,
                          Wmat_option)

})
```

```
##    user  system elapsed
## 1253.78   23.10 1495.66
```

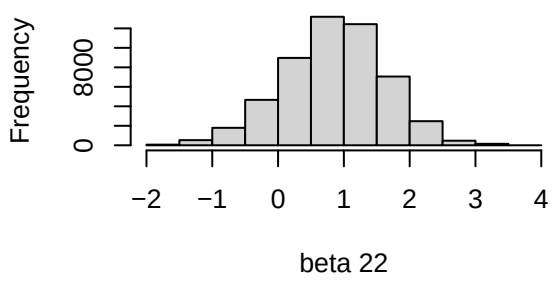
Indeed, when we increase our iteration, we will eventually see the convergence pattern.

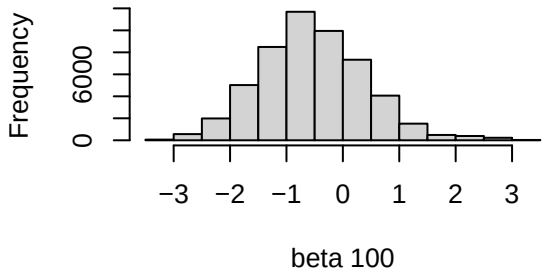
```
plot(1:(m-B),result_GP1$C_stat,type = "l",
     xlab = "Iteration",
     ylab = "C statstics",
     main = "Figure 1")
```

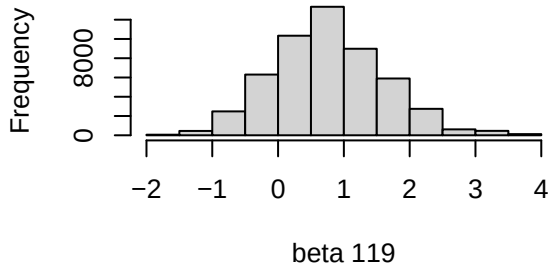


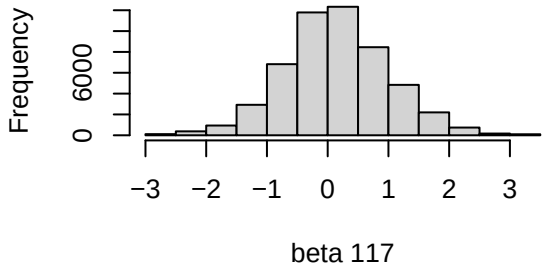
Since our $\beta$ in Gaussian Process is a nx1 vector, we randomly choose four $\beta_j$ to see what their patterns look like

```
par(mfrow=c(2,2))
index = sample(1:length(Y),size = 4)
hist(result_GP1$BETA[,index[1]],xlab=paste(expression(beta),index[1]),
     main = "")
hist(result_GP1$BETA[,index[2]],xlab=paste(expression(beta),index[2]),
     main = "")
hist(result_GP1$BETA[,index[3]],xlab=paste(expression(beta),index[3]),
     main = "")
hist(result_GP1$BETA[,index[4]],xlab=paste(expression(beta),index[4]),
     main = "")
```



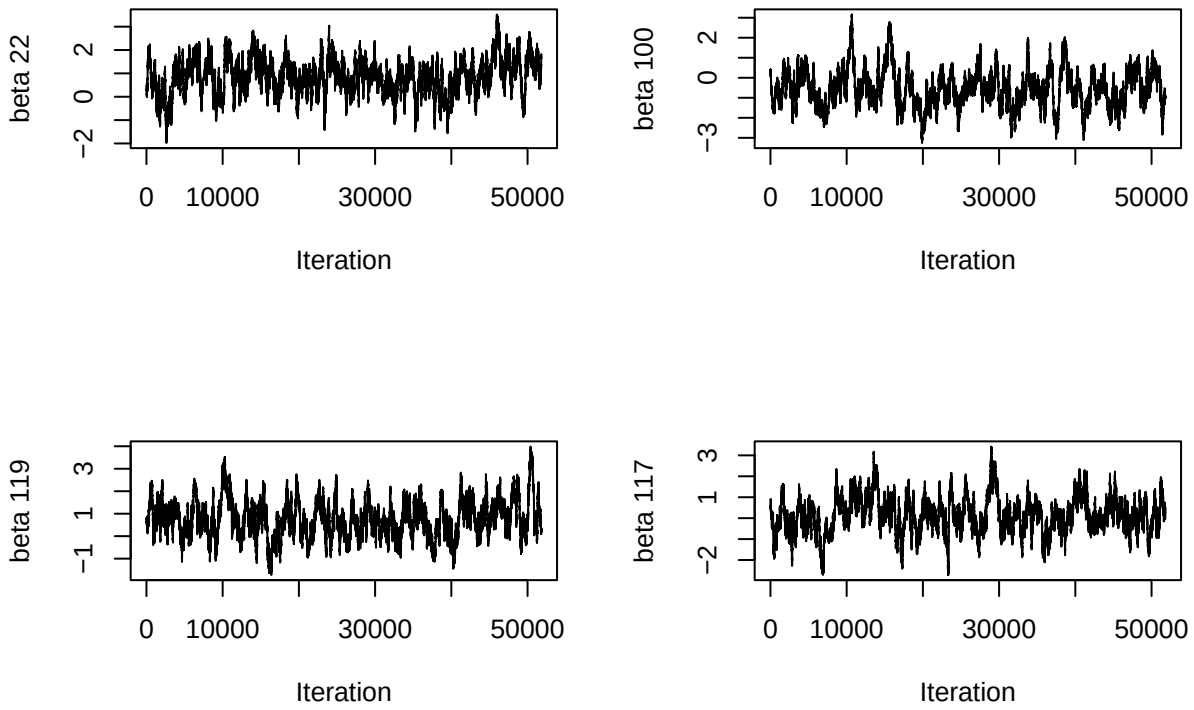




```
par(mfrow=c(2,2))
plot(1:(m-B),result_GP1$BETA[,index[1]],type = "l",
        xlab = "Iteration",
     ylab =paste(expression(beta),index[1]),main = "")
plot(1:(m-B),result_GP1$BETA[,index[2]],type = "l",
        xlab = "Iteration",
     ylab = paste(expression(beta),index[2]),main = "")
plot(1:(m-B),result_GP1$BETA[,index[3]],type = "l",
        xlab = "Iteration",
     ylab = paste(expression(beta),index[3]),main = "")
plot(1:(m-B),result_GP1$BETA[,index[4]],type = "l",
```

```
       xlab = "Iteration",
     ylab = paste(expression(beta),index[4]),main = "")
```




## Potential Problem:

### 1.

The C statistics in each iteration will converge eventually. However, when we get the average theta (average beta in each iteration), and then compute the C index. We will get a very high C index!.

```
Wmat = HarrellC_Wmat(Y,delta,tau)
HarrellC(colMeans(result_GP1$BETA),Wmat)
```

```
## [1] 0.9564311
```

Even though we never see such a high C index in each iteration (Figure 1)

Overfitting problem?

Analysis:

When we sample lambda, the algorithm seems to prefer small values of lambda..

```
colMeans(result_GP1$LAMBDA)
```

```
## [1] 0.03040511 0.03985660 0.02912638 0.03273807 0.04145842 0.05000484 0.12100906
## [8] 0.02498908
```

Recall what lambda represents:

$$K(x_i, x_j) = \sum_{k=1}^{p} \frac{(x_{ik} - x_{jk})^2}{\lambda_k}$$

Lambda is usually considered as lengthscale. When it's close to 0, it implies a mode "wiggy" function.

Based on my understanding, a "wiggy" model implies that it fits the training data well but predict poorly in the testing data..

**2.**

The Computational Issue still remains when increase the number of observations.

In our function, we cannot avoid calculating the determinant of kernel function.

In the lung data (n=167), the problem is not that big.

```
system.time({determinant(K)})
```

```
##    user  system elapsed
##       0       0       0
```

But in the gbsg data (n = 686), the problem is a big issue.

```
A2 <- model.matrix(rfstime ~ -1+ age + meno + size + grade + er + hormon,
                   data=gbsg)
lambda02 = rep(1,dim(A2)[2])
K2 = matrix_K(A2,lambda02)
system.time({determinant(K2)})
```

```
##    user  system elapsed
##    1.14    0.02    1.29
```

And we need to calculate it in each iteration! (e.g. if m=3600, that means we have to run the function for at least one hour)

(The running result for gbsg data will be shown in another file)

**Continue Investigating This Week!**