Lecture 24 Rscript

Attempting Bayesian inference for a Gaussian process.

Noisy observations, single realisation from a Gaussian process prior.

In this example, we will assume the following,

$$\begin{aligned} p(\mathbf{y}|\boldsymbol{\mu}(\mathbf{x})) &= \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \sigma^2\mathbf{I}) \\ p(\boldsymbol{\mu}(\mathbf{x})) &= \mathcal{N}(\boldsymbol{m}(\mathbf{x}), \boldsymbol{k}(\mathbf{x}, \mathbf{x})), \end{aligned}$$

where $\boldsymbol{m}(\mathbf{x}) = 0$, $\boldsymbol{k}(x_i, x_j) = \sigma_K^2 e^{-\beta \sin^2(\pi(x_i - x_j)/12)}$. The parameters we want to make inference on are:

- $\boldsymbol{\mu}(\tilde{x})$, where $\tilde{\mathbf{x}}$ may include points where we observe $\mathbf{y}$.
- σ^2. From the lecture slides, we can do this using a Gibbs step.
- σ_K^2. From the lecture slides, we can do this using a Gibbs step.
- β. This will require a Metropolis-Hastings or HMC step.

```
t<-1:23*0.5 #Set of points where function is evaluated
#Values for parameters.
beta <- 2.1
sigma2<-2.1
sigma2k<-1.3

#constructing k.
n<-length(t)
tmat<-matrix(t,n,n)
tdiff<-tmat-t(tmat)
k<- sigma2k*exp(-beta*sin(pi*tdiff/12)^2)

#simulating mu
library(mvtnorm)
```

Simulating data

```
## Warning: package 'mvtnorm' was built under R version 4.3.1
```

```
mu.t <- rmvnorm(1,mean=rep(0,n),sigma=k)

#simulating y
y    <- mu.t + rnorm(n)*sqrt(sigma2)
y    <-as.numeric(y)
```

Estimating parameters For this, we will assume a flat prior for β, and vague gamma(0,0) priors for $\tau = (\sigma^2)^{-1}$ and $\tau_K = (\sigma_K^2)^{-1}$.

```
#Arguments
#y, vector of responses
#t, time points where responses were observed.
#tpred, time points where predictions of \mu(x) are wanted.
#tau0, initial value for precision.
#sigma_K: initial value for parameter controlling scale of $k(t,t)$.
#beta: parameter controlling decay in periodic function.
#Iter: number of iterations
#burnin: number of initial iterations to remove
Gibbs.Gp<-function(y,t,tpred,tau0,sigma_K,beta,iter,burnin){
  n <- length(y)      #number of points
  t.all <-c(t,tpred)
  p <-length(tpred)
  mT<-matrix(t.all,n+p,n+p)
  #Note from properties of exponential function, the following never changes
  Kc<- exp(-sin(pi*(mT-t(mT))/12)^2)
  Kall<-Kc^beta

  tau    <-tau0
  library(mvtnorm)



  par<-matrix(0,iter,n+p+4)
   #storing iterations, mu (length p) + sigma, sigma_K, beta and acceptance indicator (length 4)
  for( i in 1:iter){
    #Updating mu for both t and t outside.
    p1   <-Kall[1:n,1:n]*sigma_K^2+diag(n)/tau
    pinv <-solve(p1)
    pred.mean<-sigma_K^2*Kall[,1:n]%*%pinv%*%y
    pred.var <-Kall*sigma_K^2 - Kall[,1:n]%*%pinv%*%Kall[1:n,]*sigma_K^4
    mu <- rmvnorm(1,mean=pred.mean,sigma=pred.var) #sample mu(x)
    mu <- as.numeric(mu)

    #updating sigma
    err  <- y-mu[1:n]           #errors for mu where values were observed.
    tau  <- rgamma(1,0.5*n,0.5*sum(err^2))     #sample tau.

    #updating sigma_K
    Kinvo<-solve(Kall[1:n,1:n]) #constructing inverse needed to update sigma_K,
    #note this assume all points where t was observed were listed first.
    muKmuinv <- t(mu[1:n])%*%Kinvo%*%mu[1:n]
    tauK<- rgamma(1,0.5*n,0.5*muKmuinv)
    sigma_K<-1/sqrt(tauK)

    #Updating beta.
    beta.cand <-runif(1,1.5,2.5)
    r1        <-dmvnorm(mu[1:n],mean=rep(0,n),sigma=sigma_K^2*Kc[1:n,1:n]^beta.cand,log=TRUE)
    r2        <-dmvnorm(mu[1:n],mean=rep(0,n),sigma=sigma_K^2*Kc[1:n,1:n]^beta,log=TRUE)
    r         <-r1-r2 #log of ratio.
    ind       <-rbinom(1,1,exp(min(c(r,0))))
```

```
    beta      <-ind*beta.cand + (1-ind)*beta

    #Update k
    Kall<-Kc^beta

    par[i,] <-c(mu,1/sqrt(tau),sigma_K,beta,ind)  #store current round of mu, sigma in par.
  }

  par <-par[-c(1:burnin),]      #removing the first iterations
  colnames(par)<-c(paste('mu',c(t,tpred),sep=''),'sigma','sigma_K','beta','accept')
  return(par)
}

system.time(chain1<-Gibbs.Gp(y=y,t=t,tpred=12:23,tau0=1,sigma_K=0.33,beta=1.5,
          iter=10000,burnin=2000))
```

```
##    user  system elapsed
##   5.490   0.261   5.754
```

```
system.time(chain2<-Gibbs.Gp(y=y,t=t,tpred=12:23,tau0=5,sigma_K=0.75,beta=2,
          iter=10000,burnin=2000))
```

```
##    user  system elapsed
##   5.447   0.228   5.678
```

```
system.time(chain3<-Gibbs.Gp(y=y,t=t,tpred=12:23,tau0=0.2,sigma_K=1.5,beta=2.5,
          iter=10000,burnin=2000))
```

```
##    user  system elapsed
##   5.434   0.239   5.677
```

```
library(coda)
```

```
## Warning: package 'coda' was built under R version 4.3.1
```

```
#Estimating Gelman -Rubin diagnostics. Remove last column because it is acceptance indictor
dim2<-dim(chain1)[2]
#Note 8000 iterations were retained, so 50:50 split is iteration 1:4000 and iteration 4001:8000


#However first we must convert the output into mcmc lists for coda to interpret.
ml1<-as.mcmc.list(as.mcmc((chain1[1:4000,-dim2])))
ml2<-as.mcmc.list(as.mcmc((chain2[1:4000,-dim2])))
ml3<-as.mcmc.list(as.mcmc((chain3[1:4000,-dim2])))
ml4<-as.mcmc.list(as.mcmc((chain1[4000+1:4000,-dim2])))
ml5<-as.mcmc.list(as.mcmc((chain2[4000+1:4000,-dim2])))
ml6<-as.mcmc.list(as.mcmc((chain3[4000+1:4000,-dim2])))
estml<-c(ml1,ml2,ml3,ml4,ml5,ml6)

#Gelman-Rubin diagnostic.
gelman.diag(estml)[[1]]
```

```
##          Point est. Upper C.I.
## mu0.5     1.0007419  1.0021719
## mu1       1.0007870  1.0022762
## mu1.5     1.0004108  1.0013350
## mu2       1.0001418  1.0006836
## mu2.5     1.0005582  1.0017356
## mu3       1.0012165  1.0033123
## mu3.5     1.0013679  1.0036418
## mu4       1.0010289  1.0028202
## mu4.5     1.0006702  1.0019135
## mu5       1.0005698  1.0016888
## mu5.5     1.0007585  1.0022100
## mu6       1.0010149  1.0028129
## mu6.5     1.0010762  1.0028521
## mu7       1.0008841  1.0022329
## mu7.5     1.0005480  1.0013815
## mu8       1.0003290  1.0008607
## mu8.5     1.0001772  1.0005641
## mu9       1.0000279  1.0003046
## mu9.5     0.9999940  1.0001661
## mu10      0.9999334  1.0000261
## mu10.5    0.9999080  0.9999719
## mu11      0.9999041  0.9999967
## mu11.5    1.0000361  1.0003621
## mu12      1.0003950  1.0012671
## mu13      1.0007870  1.0022762
## mu14      1.0001418  1.0006836
## mu15      1.0012165  1.0033123
## mu16      1.0010289  1.0028202
## mu17      1.0005698  1.0016888
## mu18      1.0010149  1.0028129
## mu19      1.0008841  1.0022329
## mu20      1.0003290  1.0008607
## mu21      1.0000279  1.0003046
## mu22      0.9999334  1.0000261
## mu23      0.9999041  0.9999967
## sigma     1.0006682  1.0015401
## sigma_K   1.0024175  1.0052804
## beta      1.0168877  1.0416882
```

```
#effective sample size.
effectiveSize(estml)
```

```
##      mu0.5        mu1      mu1.5        mu2      mu2.5        mu3      mu3.5
## 19744.3776 19021.0664 24000.0000 24000.0000 16276.7047 10001.3370  9387.9138
##        mu4      mu4.5        mu5      mu5.5        mu6      mu6.5        mu7
## 14987.1592 22128.0435 14973.3698  7854.2295  5924.8902  5768.5758  6719.8484
##      mu7.5        mu8      mu8.5        mu9      mu9.5       mu10     mu10.5
##  9086.3915 12180.4228 14717.1594 16742.9991 19942.3687 21995.5576 22436.2005
##       mu11     mu11.5       mu12       mu13       mu14       mu15       mu16
## 24000.0000 22518.4046 20705.4905 19021.0664 24000.0000 10001.3369 14987.1591
##       mu17       mu18       mu19       mu20       mu21       mu22       mu23
## 14973.3698  5924.8901  6719.8484 12180.4229 16742.9993 21995.5578 24000.0000
##      sigma    sigma_K       beta
```

```
## 12618.0127  2236.1030   463.7222
```

```r
#proportion accepted.
sum(chain1[,dim2])/8000
```

```
## [1] 0.267875
```

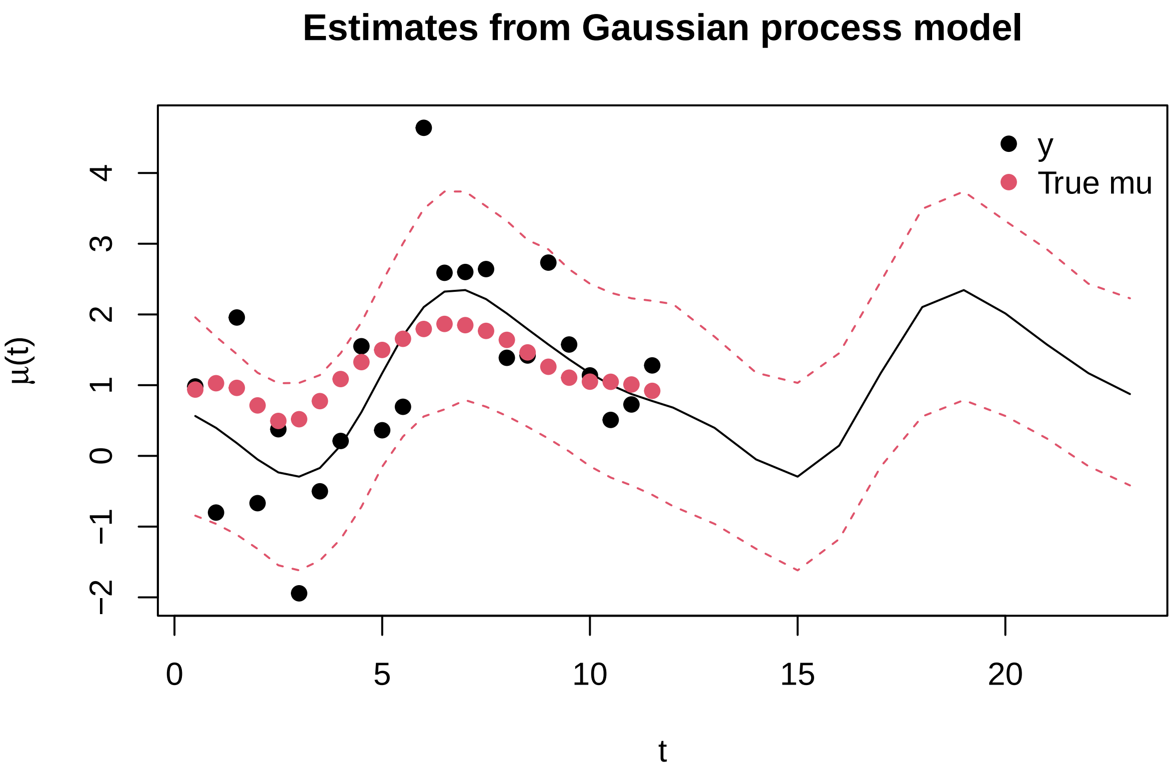
```r
sum(chain2[,dim2])/8000
```

```
## [1] 0.281125
```

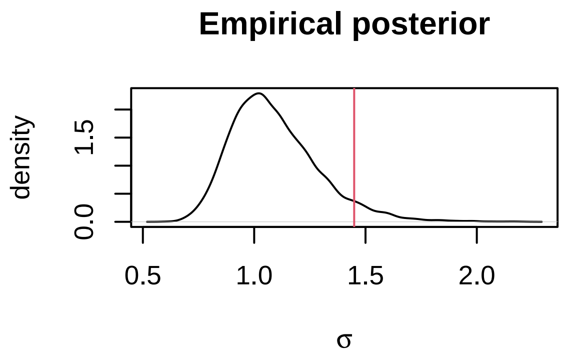
```r
sum(chain3[,dim2])/8000
```

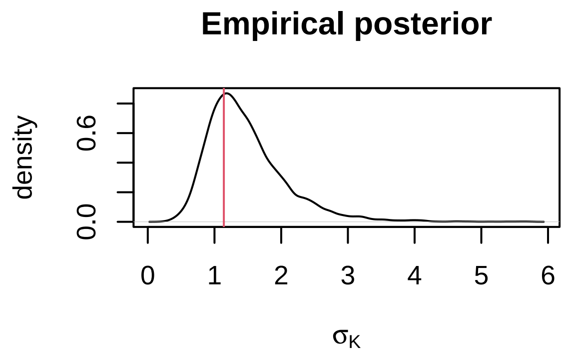
```
## [1] 0.279
```

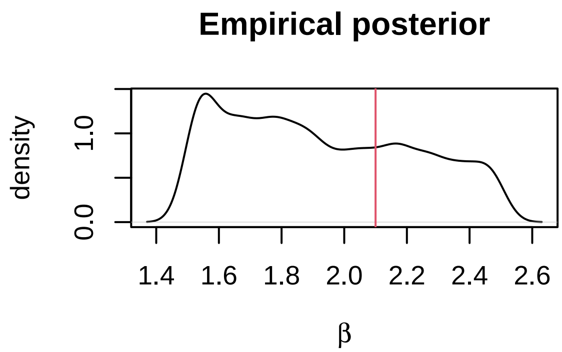
```r
#plot predictions with 99 \% credible interval for data.

chain.all<-cbind(chain1,chain2,chain3)
LL       <-apply(chain.all,2,FUN= function(x) quantile(x,0.005))
post.mean<-colMeans(chain.all)
UL       <-apply(chain.all,2,FUN= function(x) quantile(x,0.995))

ylims=c(min(c(LL,y))-0.05,max(c(UL,y))+0.05 )
plot(c(t,12:23),post.mean[1:(length(t)+12)],type='l',ylim=ylims,xlab='t',
     ylab=expression(paste( mu,'(t)',sep='') ), main='Estimates from Gaussian process model')
lines(c(t,12:23),LL[1:(length(t)+12)],col=2,lty=2)
lines(c(t,12:23),UL[1:(length(t)+12)],col=2,lty=2)
points(t,y,pch=19)
points(t,mu.t,pch=19,col=2)
legend('topright',c('y','True mu'),pch=19,col=1:2,bty='n')
```




```
#Estimates of sigma, sigma_K,beta
par(mfrow=c(2,2))
plot(density(chain.all[,36]),xlab=expression(sigma),ylab='density',main='Empirical posterior')
abline(v=sqrt(sigma2),col=2)
plot(density(chain.all[,37]),xlab=expression(sigma[K]),ylab='density',main='Empirical posterior')
abline(v=sqrt(sigma2k),col=2)
plot(density(chain.all[,38]),xlab=expression(beta),ylab='density',main='Empirical posterior')
abline(v=beta,col=2)
```


Empirical posterior
density
0.0
1.5
0.5
1.0
1.5
2.0
σ


Empirical posterior
density
0.0
0.6
0
1
2
3
4
5
6
σK


Empirical posterior
density
0.0
1.0
1.4
1.6
1.8
2.0
2.2
2.4
2.6
β