Exercise F: Metropolis within Gibbs Sampler

Benedikt Ehinger

12. Januar 2017

Videos L,M: Gibbs

1 2D Metropolis

[1]

We want to sample from the following target function (you might want to think of a posterior with two continuous variables X1 and X2. The function could come from a complicated 'blackbox' model).

$g(x_1, x_2) = |cos(\sqrt{x_1 * x_2})|^{50}$

With $0 \leq x_1 \leq 20$ and $0 \leq x_2 \leq 20$

We will start with a 2D metropolis sampler. The only difference to last homework is, that the proposal distribution has to be 2-Dimensional now.

You can start with the following code:

```
g = function(x) {
  if ( (x[1]<0) || (x[1]>20) || (x[2]<0) || (x[2]>20) )
              return(0)
        return( abs( cos(sqrt(x[1]*x[2]) ) )^50)
}



M = 10000  # run length (number of scans)
X = c( ... , ... ) # starting values for X1 and X2
n_reject = 0
x1list = x2list = rep(NA,M) # allocate memory
for (i in 1:M) {
    X_star = X + ...  # New proposal sample in 2D
    U = runif(1)  # for accept/reject
    alpha = g(X_star) / g(X)  # for accept/reject
    if (U < alpha)
              X = X_star  # accept proposal
```

[1] This exercise is based on code from Jeffrey S. Rosenthal

```
    else
        n_reject = n_reject + 1
    x1list[i] = X[1]
    x2list[i] = X[2]
}
```

1. Fill in all the gaps in the example code. Add a 2D proposal function to the code. For now we can use rnorm(2).
2. plot the resulting x1,x2 pairs of the posterior against each other. Use a line-plot (r: plot(x,y,type='l'), matlab: plot(x,y,'-')) to see how the chain evolves in the 2D Parameter space
3. Plot the density, use e.g. ggplot-histogram2d or histbin, or use matlabs 2D histogram function
4. We want to change the stepsize, i.e. change the proposal function a bit. Multiply the normal-distribution sample of your proposal function with 0.1 - What do you observe? What happens if you let your programm run multiple times with different initial values (just observe your plot multiple times).

1.1 Metropolis-within-Gibbs

We will now use Metropolis-within-Gibbs. This means, that the parameters are not updated at the same time, i.e. a 2D-new sample is proposed and evaluated. But instead we update each parameter (x_1 and x_2) sequentially. Sometimes this is also caled blocked metropolis. Now we do not need a 2D proposal, but a single 1D one. In the best case, and with some smart math-skills (or a good book) you could possibly find out, that you can sample from on of your parameters marginal densities using for example inverse cumulative. This will speed up the algorithm (because you don't need to to probabilistic accept-reject). In our case, we are not that smart (or someone might be?) and we need to do a metropolis step for each parameter/each gibbs step.

```
M = 10000  # run length (number of scans)
X = c( ... , ... ) # starting values for X1 and X2
n_reject = 0
x1list = x2list = rep(NA,2*M) # allocate memory (twice as much!)
for (i in 1:M) {
  for (gibbsStep in seq(1,2)){
    X_star = X
    X_star[...] = X[...] + ...  # New proposal sample in 1D
    U = runif(1)  # for accept/reject
    alpha = g(X_star) / g(X)  # for accept/reject
    if (U < alpha)
```

```
            X = X_star  # accept proposal
    else
        n_reject = n_reject + 1
    x1list[i*2+gibbsStep-1] = X[1]
    x2list[i*2+gibbsStep-1] = X[2]
  }
}
```

1. Fill in the code above.
2. Again, plot the chain in 2D with a line plot. Why does it look like it looks?
3. Calculate the accept-reject ratio. Which algorithm is more efficient?

2 Fish-Sensitive Neurons in the jellyfish [2]

Imagine you are interested in how many fish-selective neurons (n)there are in the jellyfish[3]. Fortunately, you find old data from some student projects that mention they found 16,18,22,25 and 27 of such neurons. Unfortunately, you have no idea how many neurons they tried (you don't know their success-rate) nor in what time interval they tried them (you cannot use poisson statistics).

What is a good bayesian statistician to do? From previous experiences, you know that students usually have a 10% chance to hit such a neuron. You formalize this as a $beta(2, 15)$ prior on θ. You also know that the maximal number of neurons (n_{max}) cannot be higher than 5600[4]. You truly do not know how many fish-selective neurons (n) there are, so you assume a constant/flat prior for the number.

We can use metropolis-within-gibbs. In this case, we can sample directly the θ parameter (as we know it is coming from a beta distribution). But it is a lot more difficult to sample the n parameter. Thus in that case we use a metropolis stepp from the joint-binomial distribution.

We could try to implement this in STAN, but it is difficult due to the discreteness of the n-parameter [5]

1. Fill in the gaps in the model below.
2. How can you summarize the posterior?
3. Play around with the data, what happens if you keep the mean of the data constant (e.g. 1,42,20,18,41), but change the variance?

[2] this example is modified from Bayesian Cognitive Modeling, Joint Posteriors

[3] this is totally not a made up research question

[4] `https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons`

[5] but possible, if you are interested see: `https://github.com/stan-dev/example-models/blob/master/Bayesian_Cognitive_Modeling/ParameterEstimation/Binomial/Survey.stan` and consult the STAN-manual

```
g = function(data,theta,n){
  if(n>5600 | n<max(data))
    return(0)

  dens = sapply(data,function(x) ... )
  dens = prod(dens) # the samples are independent,
    #we can simply multiply their densitie value
  return(dens)
}
logit    =  function(x){return(log(x/(1-x)))}
invlogit =  function(x){return(1/(1+exp(-x)))}

M = 10^5  # run length
THETA = 0.5 # starting value
N = 50      # starting value

thetalist = nlist = rep(NA,2*M)
data = (c(16,18,22,25,27))
for (i in 1:M) {

    for (gibbsStep in 1:2) {

      if(gibbsStep==1){
        # Gibbs Stepp
        # Instead of rbeta, we could also use the inverse cumulative
        # method here!
        THETA = rbeta(1, ... , ... ) # defines the prior + the current N.
        # From Jarad Niemi video c/03 we know that:
        # p(theta|data) = Beta(a+successes,b + N - successes)
        # Due to independence & conjugacy you can simply sum the data
        # Be sure to adjust N to your summed data!

      }
      if(gibbsStep==2){
        # Metropolis Step
        # This is very close to the previous examples

        # First we need to transform N from max(data):5600, to 0:1.
        Ntrans = (N-max(data))/(5600-max(data))

        # Now we make a small stepp
        N_star_trans = logit(Ntrans) + rnorm(1)

        # Now we transform back from the range 0:1 to max(data):5600
        N_star = round(invlogit(N_star_trans)*(5600-max(data)) + max(data))
```

```
        #By this procedure, N_star has to be between max(data) and 5600


        U = runif(1)  # for accept/reject
        alpha = g(data,THETA,N_star) / g(data,THETA,N)
        if (U < alpha)
           N = N_star  # accept proposal

      }
      # save the current values
      thetalist[2*i-2+gibbsStep] = THETA
      nlist[2*i-2+gibbsStep] = N
    }
}

library(ggplot2)
# we want to extract the most likely value (MAP, Maximum A Posteriori)
a = hexbin::hexbin(nlist,thetalist) # you need the hexbin package for this one
nMAP = a@xcm[which.max(a@count)]
tMAP = a@ycm[which.max(a@count)]
# plot a 2D histogram + the MAP value.
qplot(nlist,thetalist,geom="hex")+
  geom_point(inherit.aes = F,aes(x=nMAP,y=tMAP),color='red'))+
  xlim(c(0,2500))+
  ylim(c(0,1))
```

PS: Hexbin Package: install.packages("hexbin"). It's just a fancy 2d-Histogram