Slice Sample

Wang,Yue,yue18

Description

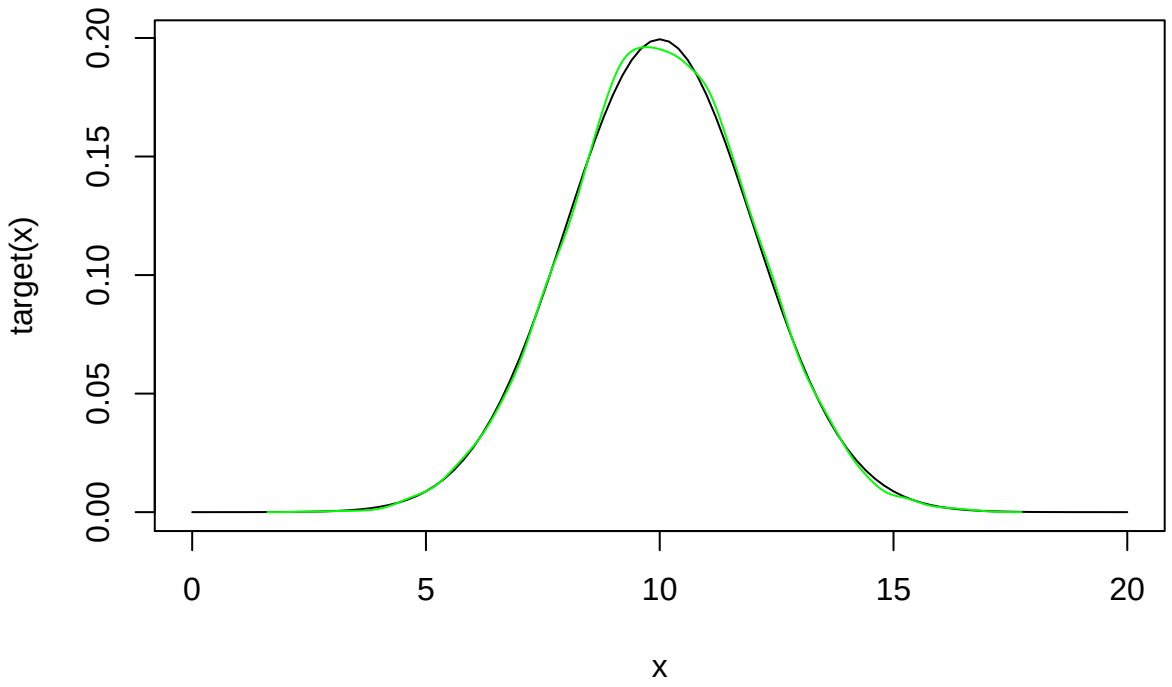
Two-dimensional sampling (x, y) is performed from the shaded area between the probability curve $f(x)$ and $y = 0$, and then the point is projected onto the x-axis. Use Gibbs sampling to sample from $P(x|y)$ $P(y|x)$.$P(x|y)$ is defined as a uniform distribution on the horizontal line, and $P(y|x)$ is defined as a uniform distribution on the vertical line. Given initial point x_0.

The basic method is as follows (Bandyopadhyay & Bhattacharya, 2014):
Generate a uniformly distributed random number x.
Select a random curve f(x).
Choose a starting point x0, where f(x0) $>$ 0.
Sample a y-value, 0$<$ y $<$ f(x0).
Slice the curve horizontally at y.
Sample a point (x,y) from within the line segment you just created.
Repeat the steps.

Code

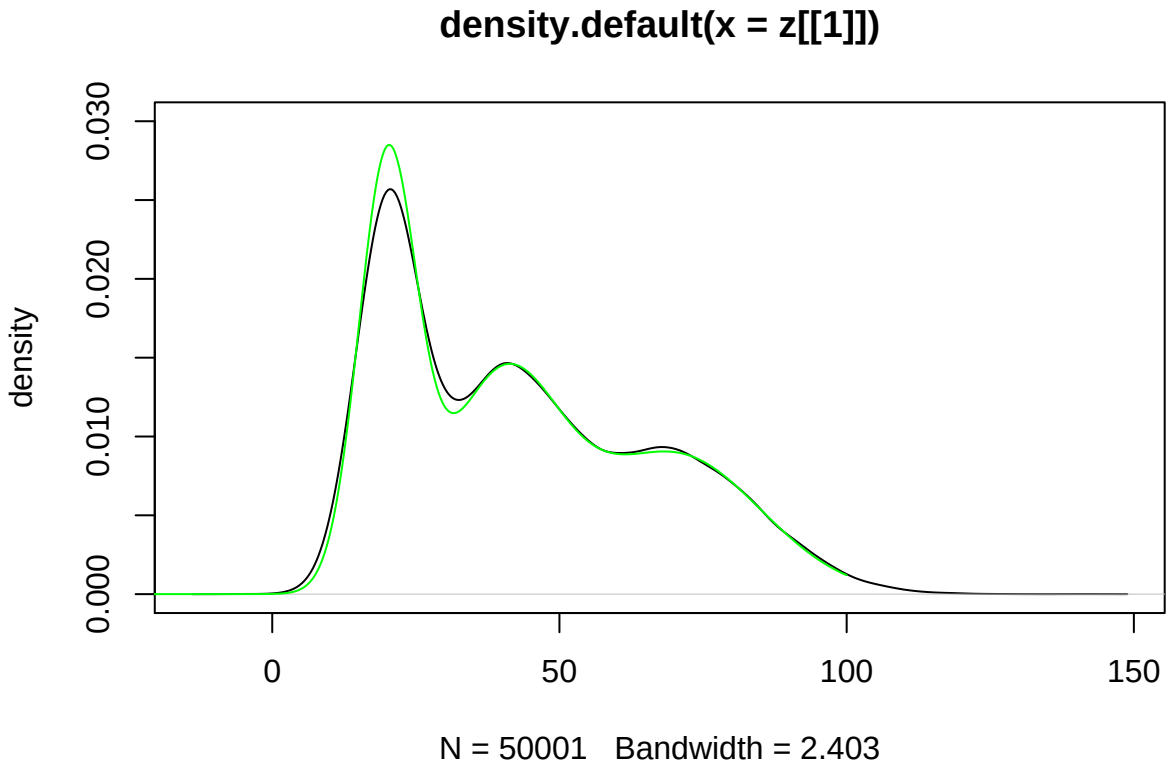
```
sliceSample = function (n, f, x.interval = c(0, 1), root.accuracy = 0.01) {
  pts = vector("numeric", n)
  x = runif(1, x.interval[1], x.interval[2])

  for (i in 1:n) {
    pts[i] = x
    y = runif(1, 0, f(x))

    fshift = function (x) { f(x) - y }
    roots = c()
    for (j in seq(x.interval[1], x.interval[2] - root.accuracy, by = root.accuracy)) {
      if ((fshift(j) < 0) != (fshift(j + root.accuracy) < 0)) {
        root = uniroot(fshift, c(j, j + root.accuracy))$root
        roots = c(roots, root)
      }
    }
    roots = c(x.interval[1], roots, x.interval[2])

    segments = matrix(ncol = 2)
    for (j in 1:(length(roots) - 1)) {
      midpoint = (roots[j + 1] + roots[j]) / 2.0
      if (f(midpoint) > y) {
        segments = rbind(segments, c(roots[j], roots[j + 1]))
      }
    }
    total = sum(sapply(2:nrow(segments), function (i) {
```

```
      segments[i, 2] - segments[i, 1]
    }))
    probs = sapply(2:nrow(segments), function (i) {
      (segments[i, 2] - segments[i, 1]) / total
    })

    p = runif(1, 0, 1)
    selectSegment = function (x, i) {
      if (p < x) return(i)
      else return(selectSegment(x + probs[i + 1], i + 1))
    }
    seg = selectSegment(probs[1], 1)

    x = runif(1, segments[seg + 1, 1], segments[seg + 1, 2])
  }

  return(pts)
}

target = function (x) { dnorm(x, 10, 2) }
points = sliceSample(n = 10000, target, x.interval=c(0, 20), root.accuracy = 0.1)
curve(target, from = 0, to = 20)
lines(density(points), col="green")
```




```
p=function(x,u1,sig1,u2,sig2){
(1/3)*(1/(sqrt(2*pi)*15)*exp(-0.5*(x-70)^2/15^2)+1/(sqrt(2*pi)*sig1)*exp(-0.5*(x-u1)^2/sig1^2)+1/(sqrt(
}
sample=function(init,time,w,u1,sig1,u2,sig2){
X=NULL
Y=NULL
X[1]=init
```

```
for (i in 1:time){
    Y[i]=runif(1,0,p(X[i],u1,sig1,u2,sig2))
    xl=X[i]-runif(1,0,w)
    xr=xl+w
    while(!((p(xl,u1,sig1,u2,sig2)<Y[i])&(p(xr,u1,sig1,u2,sig2)<Y[i]))){
        xl=xl-w
        xr=xr+w
    }
    x=runif(1,xl,xr)
    while(p(x,u1,sig1,u2,sig2)<Y[i]){
        if(x>X[i]){xr=x}else{xl=x}
        x=runif(1,xl,xr)
    }
    if(p(x,u1,sig1,u2,sig2)>Y[i]){X[i+1]=x} else{print('error')}
}
return(list(X,Y))
}


z=sample(20,50000,10,20,5,40,10)
plot(density(z[[1]]),ylim=c(0,0.03),ylab='density')
x=seq(-100,100,0.01)
lines(x,p(x,20,5,40,10),col="green")
```



Discussion

The two transition kernel that look for the horizontal line (stepping and contraction procedures and multiplication procedures) are adaptive and therefore more effective. In MH algorithm, there is only one given transition function, while slice sampling is different uniform distribution at different points.

Better suppression of random walk behavior and better adaptation to the correlation between variables.
Easier to implement than Metropolis Hastings
Faster than Rejection sampling
The step-size feature of MH is tough to get right, and is constant. Slice-sampling auto-adjusts the step size for you.
In the cases of slow sampling because of the shape of the distribution (the bigger the x interval, the slower it is to find the roots of the intersections), you can fit slice sampling with an ellipcal curve for say, Guassian priors.

Reference

Neal, Radford M. Slice sampling. Ann. Statist. 31 (2003), no. 3, 705–767. doi:10.1214/aos/1056562461. https://projecteuclid.org/euclid.aos/1056562461
Bandyopadhyay, S. & Bhattacharya, R. (2014). Discrete and Continuous Simulation: Theory and Practice. CRC Press. Huzurbazar, A. (2004). Flowgraph Models for Multistate Time-to-Event Data. John Wiley & Sons.
https://www.zoology.ubc.ca/~fitzjohn/diversitree.docs/mcmc.html