Monte Carlo Methods Assignment 2

Eavan Pattie

30/11/19

The Density

We are given the following density.

$$\begin{aligned} f(x) \propto s_7 \exp\{-\sin(\frac{s_1 \cdot x^2}{15 - s_1}) - \frac{(x - 3 - s_2\pi)^2}{2 \cdot (5 + s_3)^2}\} \\ + 2 \cdot (1 + s_7) \cdot \exp\{-\frac{x^2}{32}\} \\ + (10 - s_7) \cdot \exp\{-\cos(\frac{s_4 \cdot x^2}{15 + s_4}) - \frac{(x + 3 + s_5\pi)^2}{2 \cdot (5 + s_6)^2}\} \end{aligned}$$

Where s_i is the ith digit of my student number 1630022. Applying the values of s_i and simplifying we get the reduced form.

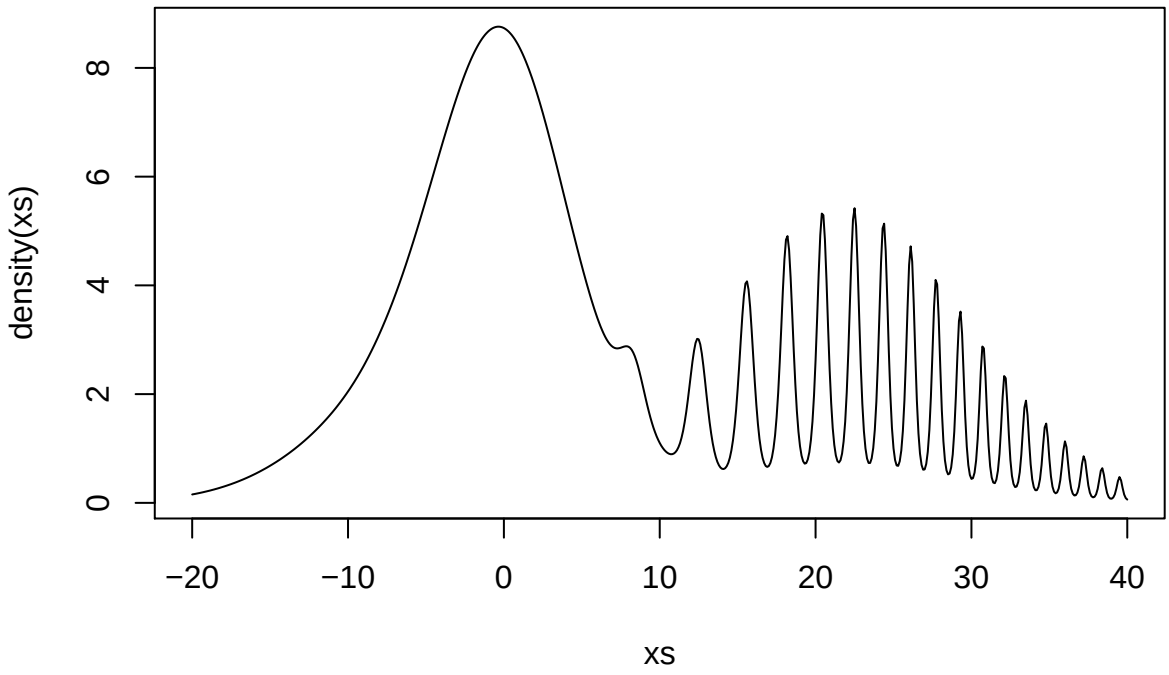
$$\begin{aligned} f(x) \propto 2 \cdot \exp\{-\sin(\frac{x^2}{14}) - \frac{(x - 3 - 6\pi)^2}{128}\} \\ + 6 \cdot \exp\{-\frac{x^2}{32}\} \\ + 8 \cdot \exp\{-1 - \frac{(x + 3)^2}{98}\} \end{aligned}$$

Notice that the cosine term drops out entirely, which is very convenient. To evaluate this density we can use the following R code.

```
ID <-  c(1,6,3,0,0,2,2)
density <- function(x) {
    t1 <- ID[7] * exp(-sin((ID[1] * x ** 2)/(15 - ID[1]))
                      - (x - 3 - ID[2] * pi) ** 2 / (2 * (5 + ID[3]) ** 2))
    t2 <- 2 * (1 + ID[7]) * exp(- x**2 /32)
    t3 <- (10 - ID[7]) * exp(-cos(ID[4] * x ** 2 / (15 + ID[4]))
                      - (x + 3 + ID[5] * pi)**2/(2*(5+ID[6])**2))
    return(t1 + t2 + t3)
}
```

Plotting this density we see that has many modes.

```
xs <- seq(-20, 40, 0.1)
plot(xs, density(xs), type="l")
```



Sampling

Simulated Annealing and Metropolis Hastings

We can implement a random walk metropolis hastings sampler as follows.

```
MH <- function(num_samples, x0 = 0, temperature_scheme = rep(1, num_samples), sd = 1){
    samples <- matrix(nrow = num_samples + 1)
    samples[1] <- x0
    for (i in 1:num_samples) {
        proposal <- rnorm(1, mean = samples[i], sd = sd)
        if (runif(1) <= (density(proposal) / density(samples[i])) ** temperature_scheme[i]) {
            samples[i+1] <- proposal
        } else {samples[i+1] <- samples[i]}
    }
    return(samples[2:num_samples+1])
}
```

Notice that we've left in an optional temperature scheme. If this is left as a sequence of 1s, then it is equivalent to standard random walk metropolis hastings. However, we can leverage this to implement our mode finding algorithm using simulated annealing.

```
simulated_annealing <- function(num_samples, max_temp = 10, x0 = 0, sd = 1) {
    temperature_scheme = seq(1, max_temp, length.out = num_samples)
    samples <- MH(num_samples, x0 = x0, temperature_scheme = temperature_scheme, sd = sd)

    mode <- NULL
    density_mode <- -1
    for (sample in samples){
        if (density(sample) > density_mode){
            mode <- sample
            density_mode <- density(sample)
        }
```

```
    }
    return(mode)
}
```

Which yeilds the following mode for the distribution.

```
simulated_annealing(10000)
```

```
## [1] -0.3514037
```

Since we expect the mode to be close to 0, we set x0 to be 0 and the standard deviation of the kernel to be 1. When using random walk metropolis hastings to sample the distribution instead of finding the mode it makes sense to choose the standard deviation to be much larger. We can choose the value for standard deviation that maximizes the expected square jumping distance.

```
esjd <- function(samples) {
    total_distance <- 0
    for (i in 2:length(samples)){
        total_distance <- total_distance + (samples[i] - samples[i-1]) ** 2
    }
    return(total_distance / (length(samples) - 1))
}
```

Iterating over a few candidates for standard deviation.

```
for (sd in c(1,2,4,8,16,32,64)) {
    samples <- MH(1000, sd = sd)
    print(paste("Standard Deviation: ", sd, " | ESJD:", esjd(samples)))
}
```

```
## [1] "Standard Deviation:  1  | ESJD: 0.793946571538279"
## [1] "Standard Deviation:  2  | ESJD: 2.89473719940428"
## [1] "Standard Deviation:  4  | ESJD: 8.96419137150271"
## [1] "Standard Deviation:  8  | ESJD: 27.6542348367922"
## [1] "Standard Deviation:  16  | ESJD: 65.565160363684"
## [1] "Standard Deviation:  32  | ESJD: 79.093503334347"
## [1] "Standard Deviation:  64  | ESJD: 62.729909982293"
```

We can see that setting the standard deviation to 32 would lead to a better sampler. This could be improved by using a finer search, however we will abandon random walk metropolis hastings for now.

Slice Sampler

If we wished to implement a slice sampler for this density we would need to be able to sample from the level sets of the distribution. Computing the level sets for our density exactly would be a futile exercise, so we will not attempt it. We will instead derive loose bounds on the level set and then use rejection sampling to sample from the level set within our bounds.

First, notice that level sets for gaussians are easy to compute $S = \{x \mid \mu - \sqrt{-2\sigma^2(\log k - \alpha)} \leq x \leq \mu + \sqrt{-2\sigma^2(\log k - \alpha)}\}$ where k is the level and α is some constant of proportionality. We can therefore bound the distribution by the max of some gaussians upto a constant of proportionality, compute their level sets and find some uniform area that contains those level sets. We can then rejection sample from this area until we have a sample from the true level set. The implementation is given below.

```
slice_sampler <- function(num_samples, x0 = 0, y0 = 1, show_ar = TRUE) {
    samples <- matrix(nrow = num_samples + 1, ncol = 2)
    samples[1,] <- c(x0,y0)
    num_proposals <- 0
```

```
    num_accepted <- 0
    for (i in 1:num_samples){
        lskew <- sqrt(-130 * (log(samples[i,2]) - 10))
        rskew <- sqrt(-128 * (log(samples[i,2]) - 2 * exp(1)))
        min_x = min(0.35 - lskew, 3 + 6 * pi - rskew)
        max_x = max(0.35 + lskew, 3 + 6 * pi + rskew)

        new_x <- NULL
        while (TRUE) {
            proposal <- runif(1, min_x, max_x)
            num_proposals <- num_proposals + 1
            if (density(proposal) > samples[i,2]) {
                new_x <- proposal
                num_accepted <- num_accepted + 1
                break
            }
        }

        new_y <- runif(1, 0, density(new_x))
        samples[i+1,] <- c(new_x, new_y)
    }
    if (show_ar){print(paste("Acceptance Rate: ", num_accepted / num_proposals))}
    return(samples[2:num_samples+1,])
}
```

Inspecting this sampler we can see the following

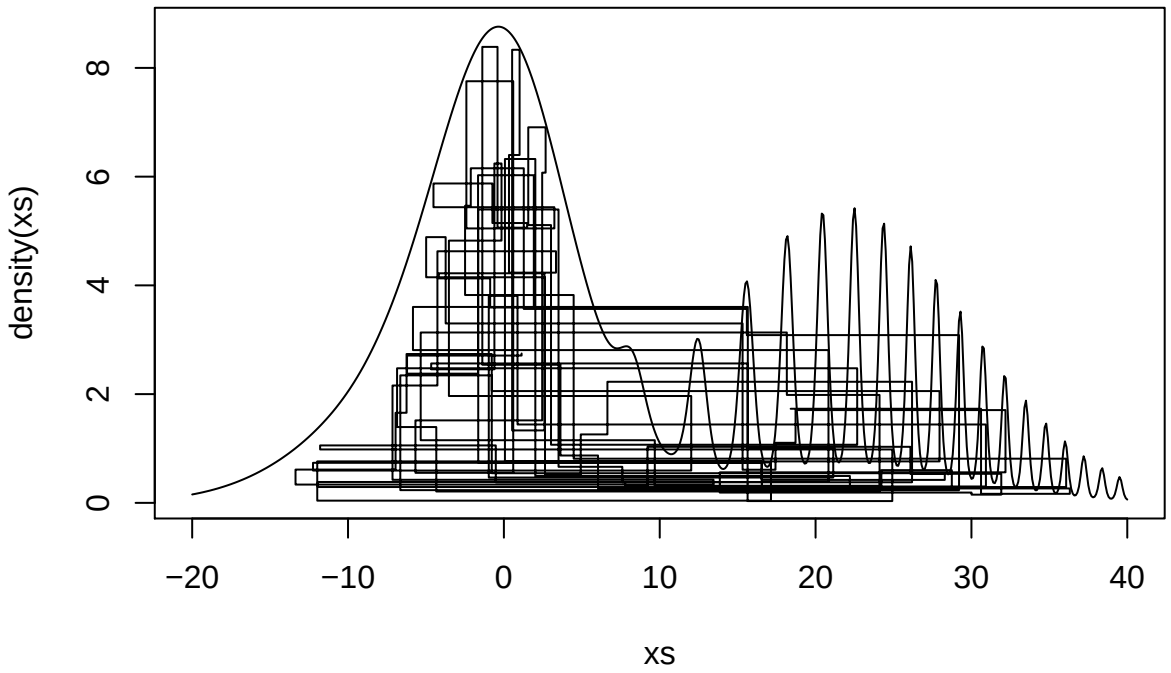
```
samples <- slice_sampler(10000)
```

```
## [1] "Acceptance Rate:  0.231294090435989"
```

```
esjd(samples[,1])
```

```
## [1] 285.1701
```

```
plot(xs, density(xs), type = "l")
lines(samples[1:100,1], samples[1:100,2], type="s")
```



The expected square jumping distance is around 3 times higher than our best result from random walk metropolis hastings although our average acceptance rate when rejection sampling the level set is below a quarter. Furthermore, qualitatively we can see that the slice sampler is moving around the distribution quite quickly, as is reflected in the jumping distances.

Other Sampler

As we saw from the slice sampler, the density of interest can be bounded by a mixture of gaussians. It is therefore natural to consider a transition kernel that is exactly this mixture. This would be implementing an independance sampler.

To arrive at the mixture of gaussians we need simply to replace our sin term with its maximum 1. After simplyfing the result we arrive at the the following.

$$\begin{aligned} g(x) \propto 2e \exp\{-\frac{(x-3-6\pi)^2}{128}\}+ \\ 6\exp\{-\frac{x^2}{32}\}+ \\ \frac{8}{e}\exp\{-\frac{(x+3)^2}{98}\} \end{aligned}$$

The constant of proportionality is $\sqrt{2\pi}(16e + 24 + \frac{56}{e}) \approx 220.82$. The mixture weights are therefore $0.4937, 0.2724, 0.2339$ for the first, second and third component respectively. The independence sampler can then be implemented as follows.

```
rgmm <- function(num = 1) {
    choice <- runif(num)
    samples <- c(rnorm(length(choice[choice < 0.4937]), mean = 3 + 6 * pi, sd = 8))
    samples <- c(samples, rnorm(length(choice[0.4937 <= choice & choice < 0.7661]), mean = 0, sd = 4))
    samples <- c(samples, rnorm(length(choice[0.7661 <= choice]), mean = -3, sd = 7))
    return(samples)
}

dgmm <- function(x) {
    return(0.4937 * dnorm(x, mean = 3 + 6 * pi, sd = 8) +
```

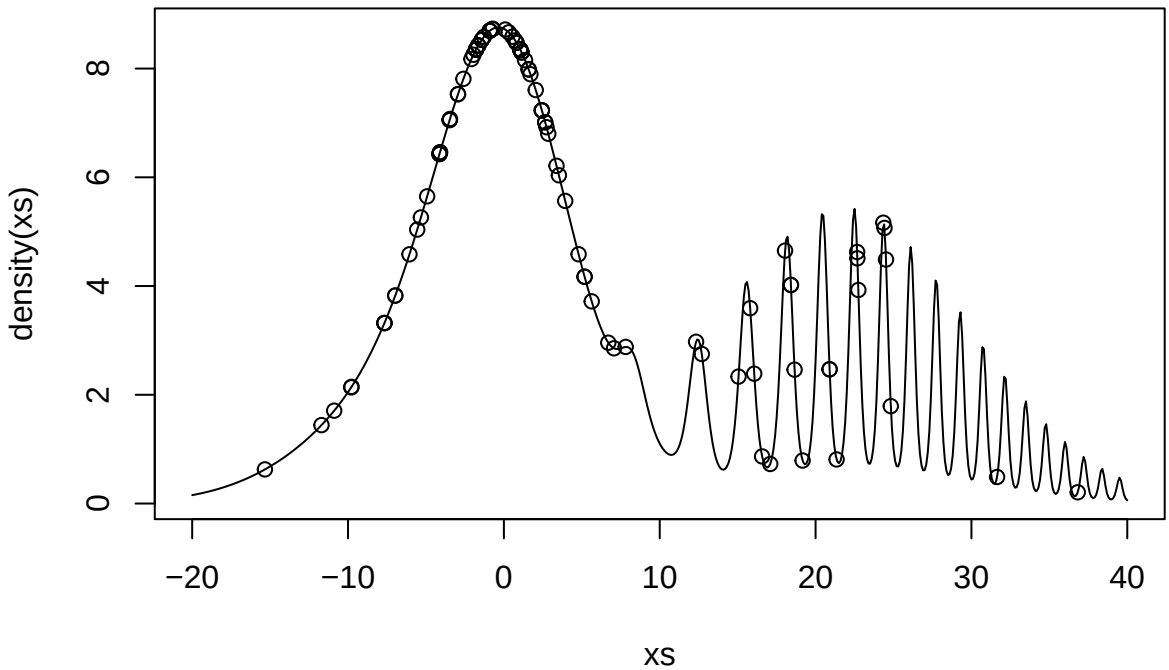
```
           0.2724 * dnorm(x, mean = 0, sd = 4) +
           0.2339 * dnorm(x, mean = -3, sd = 7))
}

indep_sampler <- function(num_samples, x0 = 0){
    samples <- matrix(nrow = num_samples + 1)
    samples[1] <- x0
    for (i in 1:num_samples) {
        proposal <- rgmm(1)
        acceptance_prob <- (density(proposal) / density(samples[i])) * (dgmm(samples[i]) / dgmm(proposal
        if (runif(1) < acceptance_prob) {
            samples[i + 1] <- proposal
        } else {
            samples[i + 1] <- samples[i]
        }
    }
    return(samples[2:num_samples+1])
}
```

Inspecting this sampler we can see

```
samples <- indep_sampler(10000)
esjd(samples)
```

```
## [1] 244.7459
```

```
plot(xs, density(xs), type = "l")
points(samples[1:100], density(samples[1:100]))
```



We have slightly lower jumping distances compared to the slice sampler, but we don't have to worry about the acceptance rate of sampling from a level set. Taking this into account, the independence sampler seems like the best sampler that has been tried.

New Acceptance Rate

To show that the new acceptance probability is valid, we can show that the detailed balance condition holds for the new acceptance probability. For the $x \neq y$ case we want to show:

$$\begin{aligned}\int_A f(x)K(x,y)dx &= \int_A f(y)K(y,x)dy \\ &= \int_A f(y)q(x|y)\frac{f(x)q(y|x)}{f(x)q(y|x)+f(y)q(x|y)} + \\ & f(y) + \delta_y(x)\frac{f(y)q(x|y)}{f(x)q(y|x)+f(y)q(x|y)}dy \\ & \int_A \frac{f(x)f(y)q(x|y)q(y|x) + f(y)^2\delta_y q(x|y)}{f(x)q(y|x)+f(y)q(x|y)} \\ & \int_A \frac{f(x)f(y)q(x|y)q(y|x)}{f(x)q(y|x)+f(y)q(x|y)}\end{aligned}$$

where A is the state sapce, f - invariant distribution and K - the transition kernel. We apply the definition of the transition kernel and the acceptance probability, then we simplify to get the following:

$$\int_A f(x)K(x,y)dx = \int_A \frac{f(x)f(y)q(x|y)q(y|x)}{f(x)q(y|x)+f(y)q(x|y)}$$

Note: We know the case where $x = y$ is trivially equal, hence $\int_A f(x)K(x,y)dx = \int_A f(y)K(y,x)dy$ in all cases. Therefore, f is the invariant distribution and the new acceptance probability is valid.

Improvements over standard acceptance rate

It is clear from the two acceptance rates that

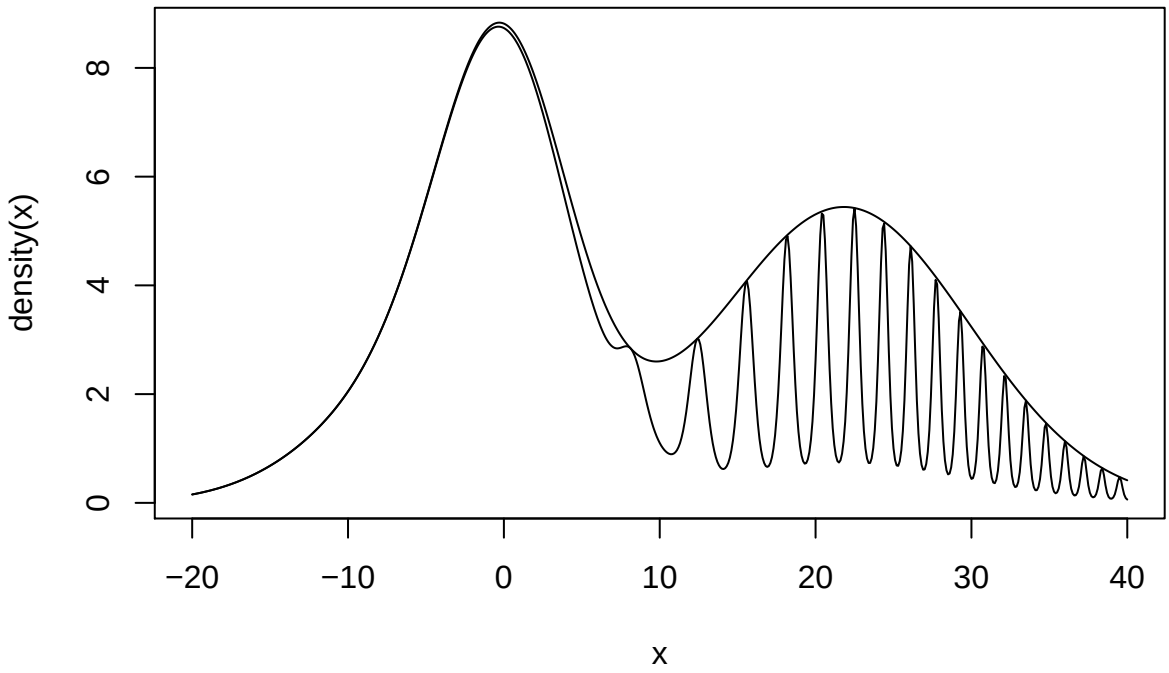
$$\frac{f(y)q(x|y)}{f(y)q(x|y)+f(x)q(y|x)} \leq \frac{f(y)q(x|y)}{f(x)q(y|x)}$$

therefore, the acceptance rate within the new acceptance probability will be always less than or equal to the Metropolis-Hasting's acceptance probability. This leads to higher autocorrelation and lower expected square jumping distance. We should therefore conclude that the new acceptance probability will give only slower convergence.

My sampler

We can see that our ditribution has a close upper bound with a Gaussian mixture model, therefore we can consider a rejection sampler that uses the Gaussian mixture model as a proposal density. The proposal distribution and our target density are plotted below.

```
gmm_bound <- function(x) {
    t1 <- ID[7] * exp(1 - (x - 3 - ID[2] * pi) ** 2 / (2 * (5 + ID[3]) ** 2))
    t2 <- 2 * (1 + ID[7]) * exp(- x**2 /32)
    t3 <- (10 - ID[7]) * exp(-1 - (x + 3 + ID[5] * pi)**2/(2*(5+ID[6])**2))
    return(t1 + t2 + t3)
}
x <- seq(-20,40, 0.1)
plot(x, density(x), type="l")
lines(x, gmm_bound(x))
```



Our rejection sampler is then as follows:

```
rej_sampler <- function(num_samples = 1){
    samples <- c()
    pseudo_m <- 1
    while (length(samples) < num_samples) {
        quantity <- pseudo_m * (num_samples - length(samples))
        proposals <- rgmm(quantity)
        accepted_samples <- proposals[density(proposals) > runif(quantity, 0, gmm_bound(proposals))]
        samples <- c(samples, accepted_samples)

        pseudo_m <- quantity / length(accepted_samples)
        print(paste("Acceptance Rate: ", 1 / pseudo_m))
    }
    return(samples[1:num_samples])
}
```

Since our rejection sampler gives iid draws we do not need to concern ourselves with the rate of convergence. We only need to consider the acceptance rate.

```
start_time <- Sys.time()
samples <- rej_sampler(1000000)
```

```
## [1] "Acceptance Rate:  0.734856"
## [1] "Acceptance Rate:  0.735341018706816"
```

```
end_time <- Sys.time()
end_time-start_time
```

```
## Time difference of 0.8923559 secs
```

Since our acceptance rate is above 70% and it takes less than a second to generate one million of iid samples, there is little to be gained by using a Markov Chain based sampler.