cs208 HW 3

Anthony Rentsch

4/2/2019

Note: I include the code I wrote directly in line for some questions that ask only to implement a method. For other questions, my code can be found in the Appendix and on Github. I worked with Bhaven Patel and Lipika Ramaswamy for this assignment.

Question 1

(a) To prove that this mechanism is ϵ-DP, I will show that (i) the percentile trimming transformation is 1-Lipschitz, (ii) that the Laplace noise injection mechanism is ϵ-DP, and (iii) that this implies that the entire mechanism $M(x)$ is $(1 * \epsilon)$-DP.

(i) A mapping T from dataset to dataset is c-Lipschitz iff $\forall x, x'$ $d(T(x), T(x')) \leq c * d(x, x')$. Here let's consider that x and x' only differ on one element. It follows that $d(x, x') = 1$.

Now consider the percentile trimming transformation in this mechanism. It again follows that $d(T(x), T(x')) = 1$ since the maximum number of rows that these two datasets will differ on is 1. Returning the inequality in the definition of a Lipschitz constant, we see that this transformation is 1-Lipschitz.

(ii) First, we observe that

$$\frac{1}{.9n} \sum_{P_{.05} \leq x \leq P_{0.95}} x_i$$

is simply an estimator for the mean of x after trimming the bottom and top 5% of the data. For simplicity, replace $.9n$ with n' and call this mechanism M'. Note that the global sensitivity of this query is $GS_q = D/n'$. Since the Laplace noise is scaled by $\frac{GS_q}{\epsilon}$, M' is ϵ-DP.

(iii) In class, we discussed a lemma that states that if M is ϵ-DP and T is c-Lipschitz, then $M \circ T$ is $(c * \epsilon)$-DP. Following from (i) and (ii), we then have that $M = M' \circ T$ is $(1 * \epsilon)$-DP.

Below is the implementation of this mechanism.

```
sgn <- function(x) {     # function borrowed from class
  return(ifelse(x < 0, -1, 1))
}

rlap = function(mu=0, b=1, size=1) {     # function borrowed from class
  p <- runif(size) - 0.5
  draws <- mu - b * sgn(p) * log(1 - 2 * abs(p))
  return(draws)
}

trimmedMean <- function(x, d, n, epsilon) {
  scale <- d/(epsilon*0.9*n)
  quants <- quantile(x, c(0.05,0.95))
  x_trimmed <- x[x>quants[1] & x<quants[2]]
  mean_trimmed <- (1/(0.9*epsilon*n))*sum(x_trimmed)
  mean_release <- mean_trimmed + rlap(mu=0,b=scale,size=1)
  return(mean_release)
}
```

(b) Let's first consider $d(x, x')$ for two neighboring datasets $x \sim x'$. Under the change model, $d(x, x') = 1$. In the worst case we could move the value at the 5th percentile above the 95th percentile, which shifts (a) a value inside the middle 90th percentile to the place of the 5th percentile and (b) a value from the top 5th percentile to place of the 95th percentile. However, we can view (a) as a change, so there is only one element that differs.

Let's now consider $d(T(x), T(x'))$ for $T(x) = [x]_{P_{05}}^{P_{95}}$. $T(x)$ simply changes $\frac{1}{10}^{th}$ of the data, thus $d(T(x), T(x')) = 0.1n$. By the lemma discussed in part (a), this transformation is $0.1n$-Lipschitz. Further, since we are considering the composition of an ϵ-DP mechanism with a $0.1n$-Lipschitz transformation, the resulting mechanism is $0.1n * \epsilon$-DP and **not** ϵ-DP for $n > 10$.

(c) In this mechanism, I first bin each data value into a bin and sort those bins. Then, for each bin, I calculate the utility function value for that bin, where the utility function is defined as

$$u = - \mid n * t - \#below \mid$$

where n is the number of observations, $t \in [0, 100]$ is the percentile, and $\#below$ is the number of observations with values less than or equal to the value of the current bin. The value of this utility function is close to 0 for bins that are close to the true percentile and is increasingly negative for bins that are further away from the true percentile.

These utility values are then turned into likelihoods using the exponential mechanism $\frac{e^{\epsilon * u}}{2}$ and are further weighted by the number of values between any two successive bins. The resulting bin width-weighted likelihoods are turned into probabilities of being the actual percentile.

To produce our noisy estimate of the percentile, we randomly sample a number from the standard uniform distribution and consider the first bin for which the cumulative sum of the probabilities is greater than this random number as the target true bin. For the final step, a number is sampled from the interval between this number and the next largest number and returned as the DP estimate of the true percentile.

Code for the mechanism is below.

```
exponentialPercentile <- function(x, t, epsilon){
  t <- t/100
  bins <- sort(x)
  nbins = length(bins)

  likelihoods <- rep(NA, nbins)
  for(i in 1:nbins){
    quality <- -1 * abs(nbins*t - i)
    if(i < nbins) { bin_width <- bins[i]-bins[i+1]+1 }
    else { bin_width <- 1 }
    likelihoods[i] <- (exp(epsilon * quality) / 2)*(bin_width)
  }

  probabilities <- likelihoods/sum(likelihoods)
  flag <- runif(n=1, min=0, max=1) < cumsum(probabilities)

  bin_low_ind = which(flag)[1]
  bin_high_ind = bin_low_ind + 1
  DPrelease <- runif(n=1, min=bins[bin_low_ind], max=bins[bin_high_ind])

  return(DPrelease)
}
```

(d)

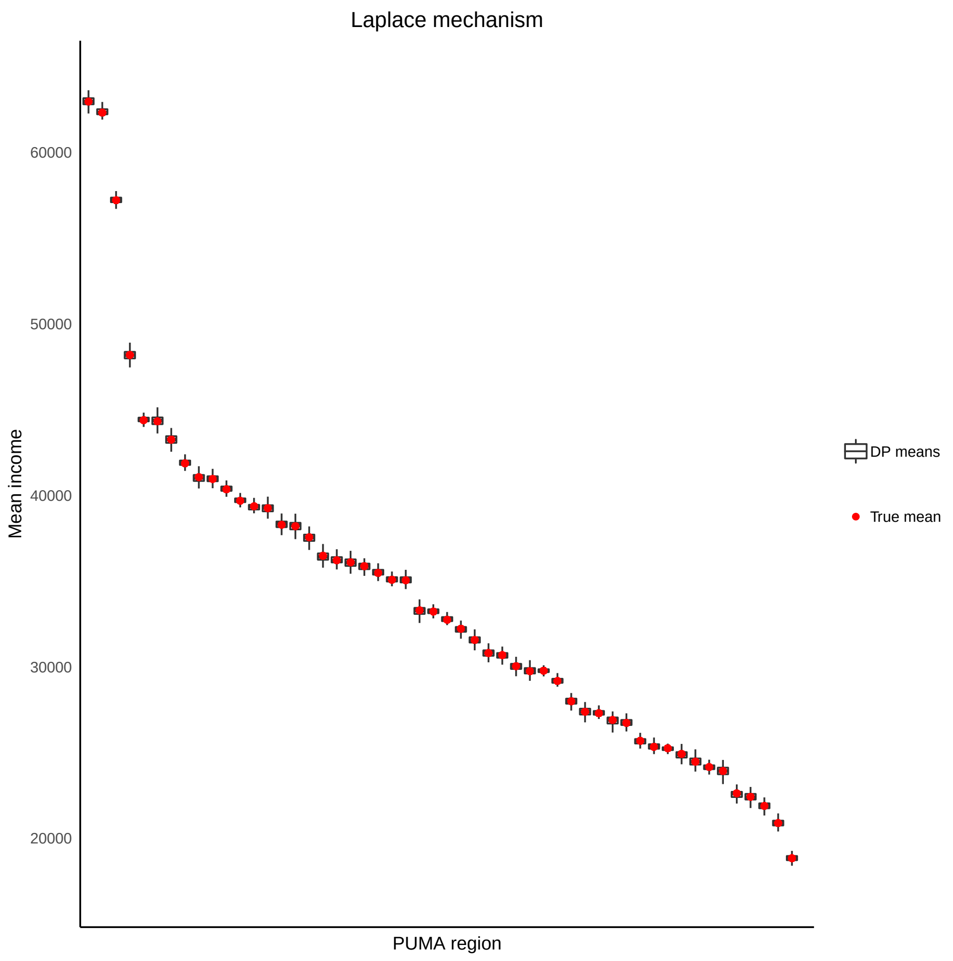
```
e3trimmedMean <- function(x, epsilon, tile_low=5, tile_high=95) {

  n <- length(x)

  tile_low_value <- exponentialPercentile(x, t=tile_low, epsilon=epsilon/3)
  tile_high_value <- exponentialPercentile(x, t=tile_high, epsilon=epsilon/3)
  x_trimmed <- x[x>tile_low_value & x<tile_high_value]
  mean_trimmed <- (1/(0.9*n))*sum(x_trimmed)

  scale <- (3*(tile_high_value-tile_low_value))/(0.9*epsilon*n)
  mean_release <- mean_trimmed + rlap(mu=0,b=scale,size=1)
  return(mean_release)
}
```

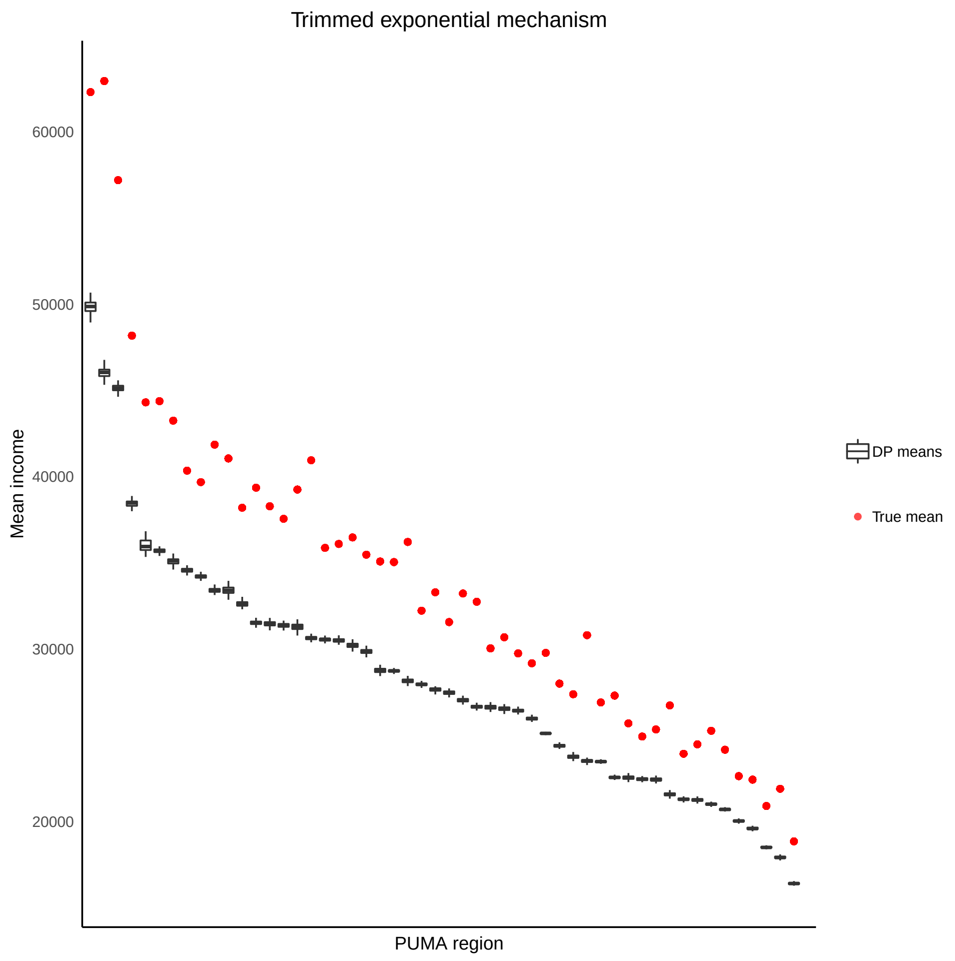
(e)

- yes
- $GS_q = \frac{\hat{P_{.95}} - \hat{P_{.05}}}{n}$ of the new, winsorized data
- by composition, adding up three $\frac{\epsilon}{3}$-DP methods gets us to ϵ-DP

(f) As the plots below show, the Laplace mechanism gives unbiased results for the actual mean while the trimmed exponential mechanism means are downwardly biased. In this case, it appears pretty obvious that the ordinary Laplace mechanism provides better performance. That is likely because the data is heavily right-skewed so that the ordinary Laplace mechanism (which only trims the data to a range based on the user's knowledge of the data domain) does not remove much of the data before computing a noisy estimate while the trimmed exponential mechanism clamps data to the 5^{th} and 95^{th} percentile, which will be design push the right tail of the income distribution downward, which will downwardly biases the estimate of the mean. In general, I would expect that the Laplace mechanism would perform better than the trimmed exponential mechanism for heavily skewed data like the PUMS income data.

On the other hand, I would expect the opposite to be true if the data was normally distributed but the user provided clip values were not symmetric around the mean of the distribution so the mean estimate would be biased in the direction in which the user's data domain varied from the empirical data range. By design, the percentiles that the trimmed exponential mechanism clamps to will be more symmetric around the center so the estimates of the means from that mechanism will be unbiased.


Laplace mechanism
Mean income
60000
50000
40000
30000
20000
PUMA region
DP means
True mean



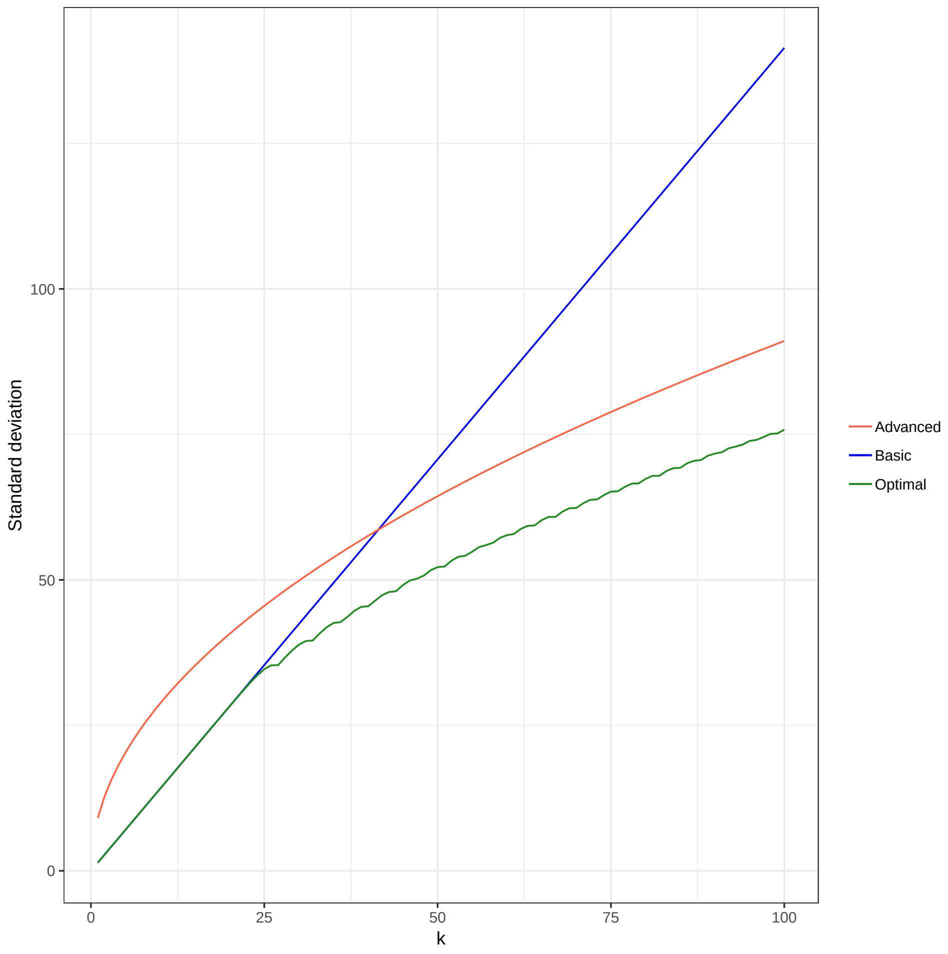
Below is a table with the average RMSE from both mechanisms for each PUMA region. In general, RMSE values are much lower for the exponential mechanism than for the ordinary Laplace mechanism, which is supported by my analysis of the skewed nature of the PUMS income data above.

PUMA Region	Average RMSE from Laplace	Average RMSE from exponential
100	170.6524	4769.752
200	104.5308	4671.822
300	156.3764	3438.994
400	371.6181	6176.201
500	162.8653	5445.659
600	256.0822	2608.996
700	192.5234	7284.758
800	228.5347	7848.513
900	246.2228	6256.800
1000	244.7632	6812.909
1100	249.7221	7955.224
1200	235.3291	3137.458
1300	256.2831	5736.057
1400	255.4670	12414.435
1500	226.4316	5616.200
1600	243.2981	5151.554
1700	266.5957	2621.115
1800	229.2759	3586.151

PUMA Region	Average RMSE from Laplace	Average RMSE from exponential
1900	222.8998	2402.932
2000	258.3219	5640.206
2100	187.7614	3320.010
2200	212.1020	5606.181
2300	145.3691	3455.776
2400	254.1259	8269.533
2500	198.7590	7669.735
2600	225.5472	16868.158
2700	135.0003	8648.496
2800	274.2385	8177.265
2900	230.6953	3228.771
3000	240.9164	3214.513
3100	222.5821	3609.384
3200	259.5572	8012.523
3301	235.1567	3983.996
3302	209.0447	10313.565
3303	251.6271	2439.831
3304	251.1535	2924.438
3305	279.6569	4137.944
3400	174.2567	12072.340
3500	242.0411	9745.336
3600	289.1059	5589.743
3700	187.6512	5349.583
3800	246.6905	6350.161
3900	201.9621	8420.846
4000	270.1566	2464.395
4100	188.8666	4262.004
4200	269.7250	3440.711
4300	248.2377	4108.175
4400	216.1265	2782.350
4500	133.3626	4244.056
4600	183.6570	6311.245
4700	179.6529	6190.813
4800	222.6085	6071.434

Question 2

The "optimal" composition theorem strictly improves upon the standard deviation of the injected noise from the basic composition theorem when $k \geq 17$ and advanced composition strictly improves upon basic composition when $k \geq 42$. Up to $k = 17$, basic and "optimal" composition correspond to roughly the same standard deviation. The standard deviation from advanced composition is strictly larger than the standard deviation from "optimal" composition, but the ratio between the two appears to remain constant as $k \to \infty$.



Question 3

Here, I examine the utility of synthetic data generated using the DP histogram approach. To do this, I (1) generate synthetic data by adding Laplace distributed noise to the *age* x *education* x *income* bins counts for the PUMS data, (2) sample 10,000 synthetic observations based on these bin counts, (3) estimate a regression model relating age and education to income, and (4) compute the mean squared error, squared bias, and variance of the coefficient estimates for age, education, and the intercept. To get a sense of the expected behavior of this procedure, I repeat this process 10 times (performing it many more times was too computationally expensive).

As a pre-process to this whole procedure, I clip the income variable to 1 so that I can work with the log of the income. I then use the log of the clipped income for the remainder of the problem, i.e., to compute the sensitive regression on the entire dataset and to produce the histogram release and corresponding synthetic data.

The table below summarizes my results. The MSE from running the regression on synthetic data is slightly larger than the MSE from running the regression on bootstrapped data, which suggests that the synthetic data approach I implement here only adds slightly more noise than we would expect to be added due to sampling. It is very likely that this DP-added error would go down even further if I binned income into more than 10 bins (I chose 10 because it made computation time reasonable).

Decomposing the MSE of the DP release itself, the bias is contributing much more to the introduced error than variance. This implies that our method will be stable while at the same time there is room to improve upon the bias, i.e., binning variables (like income) with more granularity.

Metric	Intercept	Education	Age
DP MSE	0.2181647	0.0000379	0.0004556
DP Variance	0.0067352	0.0000010	0.0000758
DP Bias^2	0.2114295	0.0000370	0.0003797
Sampling MSE	0.2174255	0.0000242	0.0010317

BONUS

Appendix

I put the code for all of my analyses here. You can also find it on Github.

```
# Set up
require(plyr); require(dplyr); require(ggplot2)
pums <- read.csv("MaPUMS5full.csv")
```

Question 1

```
# a
sgn <- function(x) {     # function borrowed from class
  return(ifelse(x < 0, -1, 1))
}

rlap = function(mu=0, b=1, size=1) {     # function borrowed from class
  p <- runif(size) - 0.5
  draws <- mu - b * sgn(p) * log(1 - 2 * abs(p))
  return(draws)
}

trimmedMean <- function(x, d, epsilon) {
  n <- length(x)
  scale <- d/(epsilon*0.9*n)
  quants <- quantile(x, c(0.05,0.95))
  x_trimmed <- x[x>quants[1] & x<quants[2]]
  mean_trimmed <- (1/(0.9*n))*sum(x_trimmed)
  mean_release <- mean_trimmed + rlap(mu=0,b=scale,size=1)
  return(mean_release)
}

# c
exponentialPercentile <- function(x, t, epsilon){
  t <- t/100
  bins <- sort(x)
  nbins = length(bins)

  likelihoods <- rep(NA, nbins)
  for(i in 1:nbins){
    quality <- -1 * abs(nbins*t - i)
    if(i < nbins) { bin_width <- bins[i]-bins[i+1]+1 }
    else { bin_width <- 1 }
    likelihoods[i] <- (exp(epsilon * quality) / 2)*(bin_width)
  }

  probabilities <- likelihoods/sum(likelihoods)
  flag <- runif(n=1, min=0, max=1) < cumsum(probabilities)

  bin_low_ind = which(flag)[1]
  bin_high_ind = bin_low_ind + 1
  DPrelease <- runif(n=1, min=bins[bin_low_ind], max=bins[bin_high_ind])

  return(DPrelease)
}

# d
e3trimmedMean <- function(x, epsilon, tile_low=5, tile_high=95) {

  n <- length(x)

  tile_low_value <- exponentialPercentile(x, t=tile_low, epsilon=epsilon/3)
  tile_high_value <- exponentialPercentile(x, t=tile_high, epsilon=epsilon/3)
  x_trimmed <- x[x>tile_low_value & x<tile_high_value]
  mean_trimmed <- (1/(0.9*n))*sum(x_trimmed)

  scale <- (3*(tile_high_value-tile_low_value))/(0.9*epsilon*n)
  mean_release <- mean_trimmed + rlap(mu=0,b=scale,size=1)
  return(mean_release)
}

# f
clip <- function(x, lower, upper){     # borrowed from class
  x.clipped <- x
  x.clipped[x.clipped<lower] <- lower
  x.clipped[x.clipped>upper] <- upper
  return(x.clipped)
}

rmse <- function(pred, true) {
  val <- sqrt(mean((pred-true)^2))
  return(val)
}

laplaceMeanRelease <- function(x, lower, upper, epsilon){     # borrowed from class
  n <- length(x)

  sensitivity <- (upper - lower)/n
  scale <- sensitivity / epsilon

  x.clipped <- clip(x, lower, upper)
  DPrelease <- mean(x.clipped) + rlap(mu=0, b=scale, size=1)

  return(DPrelease)
}

### simulations
```

```
pumas <- unique(pums$puma)
n_pumas <- length(pumas)
n_reps <- 100
means_mat <- matrix(NA, nrow=n_pumas*n_reps, ncol=6)
row_ind <- 1

for(i in 1:n_pumas){
  print(i)
  puma_region <- pumas[i]
  dat <- pums$income[pums$puma==puma_region]
  true_mean <- mean(dat)

  for(j in 1:n_reps){
    laplace_noise <- laplaceMeanRelease(x=dat, lower=0, upper=1000000 , epsilon=1)
    e3_trimmed <- e3trimmedMean(x=dat, epsilon=1)
    means_mat[row_ind,1] <- puma_region
    means_mat[row_ind,2] <- true_mean
    means_mat[row_ind,3] <- laplace_noise
    means_mat[row_ind,4] <- e3_trimmed
    means_mat[row_ind,5] <- rmse(laplace_noise, true_mean)
    means_mat[row_ind,6] <- rmse(e3_trimmed, true_mean)
    row_ind <- row_ind + 1
  }
}

means_df <- data.frame(means_mat)
names(means_df) <- c("puma","true_mean","laplace_mean","exponential_mean","rmse_laplace","rmse_exponent

### results
q1_results <- means_df %>% group_by(puma) %>% summarise(avg_lap=mean(rmse_laplace),
                                          avg_exp=mean(rmse_exponential))
write.csv(q1_results, "q1_results.csv")


q1_plot1 <- ggplot(data=means_df, aes(x=reorder(factor(puma), -laplace_mean))) +
  geom_boxplot(aes(y=laplace_mean, shape="DP means"), outlier.shape=NA, alpha=0.7) +
  geom_point(aes(y=true_mean, colour="True mean")) +
  scale_colour_manual(values=c("red")) +
  labs(x="PUMA region", y="Mean income", title="Laplace mechanism") +
  theme_bw() +
  theme(axis.text.x=element_blank(), axis.ticks.x=element_blank(), axis.ticks.y=element_blank(),
        axis.line = element_line(colour = "black"), panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(), panel.border = element_blank(),
        panel.background = element_blank(), legend.title = element_blank(),
        plot.title = element_text(hjust = 0.5))
pdf("plots/q1_plot1.pdf", width=8, height=8)
q1_plot1
dev.off()


q1_plot2 <- ggplot(data=means_df, aes(x=reorder(factor(puma), -trimmed_mean))) +
  geom_boxplot(aes(y=trimmed_mean, shape="DP means"), outlier.shape=NA, alpha=0.7) +
  geom_point(aes(y=true_mean, colour="True mean"), alpha=0.7) +
  scale_colour_manual(values=c("red")) +
  labs(x="PUMA region", y="Mean income", title="Trimmed exponential mechanism") +
  theme_bw() +
  theme(axis.text.x=element_blank(), axis.ticks.x=element_blank(), axis.ticks.y=element_blank(),
        axis.line = element_line(colour = "black"), panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(), panel.border = element_blank(),
        panel.background = element_blank(), legend.title = element_blank(),
        plot.title = element_text(hjust = 0.5))
pdf("plots/q1_plot2.pdf", width=8, height=8)
q1_plot2
dev.off()
```

Question 2

```
# parameters
global_epsilon = 1
global_delta = 10^(-9)
global_sens = 1
max_k = 100

# Laplace sd
laplaceSD <- function(epsilon) { return((sqrt(2)/epsilon)) }

# basic
basicComposition <- function(epsilon, k) { return(epsilon/k) }

# advanced
advancedComposition <- function(epsilon, k, delta) { return(epsilon/sqrt(2*k*log(1/delta))) }

# optimal
# use PSIlence:::update_parameters

# compute sds
sds <- matrix(NA, nrow=100, ncol=4)
for (k in 1:max_k) {
  epsilon_comp <- basicComposition(global_epsilon, k)
  epsilon_adv <- advancedComposition(global_epsilon, k, global_delta)

  init <- rep(c(1/k, 0), k)
  params <- matrix(init, nrow=k, ncol=2, byrow=TRUE)
  inverse <- PSIlence:::update_parameters(params, hold=0, eps=global_epsilon, del=global_delta)
  epsilon_opt <- max(inverse[,1])

  sds[k,1] <- laplaceSD(epsilon_comp)
  sds[k,2] <- laplaceSD(epsilon_adv)
  sds[k,3] <- laplaceSD(epsilon_opt)
  sds[k,4] <- k
}

sds_df <- data.frame(sds)
names(sds_df) <- c("basic", "advanced", "optimal", "k")

q2_plot <- ggplot(sds_df) + geom_line(aes(x=k, y=basic, colour="Basic")) +
  geom_line(aes(x=k, y=advanced, colour="Advanced")) +
  geom_line(aes(x=k, y=optimal, colour="Optimal")) +
  labs(x="k", y="Standard deviation") +
  scale_colour_manual(values=c("#FF6347", "#0000FF", "#228B22")) +
  theme_bw() +
  theme(legend.title = element_blank())
pdf("plots/q2_plot.pdf", width=8, height=8)
q2_plot
dev.off()

# when do advanced/optimal beat basic?
paste0("Advanced: ", min(sds_df$k[sds_df$advanced < sds_df$basic]))
paste0("Optimal: ", min(sds_df$k[sds_df$optimal < sds_df$basic]))
```

Question 3

```
sgn <- function(x) {     # borrowed from class
  return(ifelse(x < 0, -1, 1))
}

rlap = function(mu=0, b=1, size=1) {     # borrowed from class
  p <- runif(size) - 0.5
  draws <- mu - b * sgn(p) * log(1 - 2 * abs(p))
  return(draws)
}

clip <- function(x, lower, upper){     # borrowed from class
  x.clipped <- x
  x.clipped[x.clipped<lower] <- lower
  x.clipped[x.clipped>upper] <- upper
  return(x.clipped)
}

bootstrap <- function(x, y=NULL, n){     # borrowed from class
  index <- sample(x=1:length(x), size=n, replace=TRUE)

  if(is.null(y)){
    return(x[index])
  }else{
    return(list(x=x[index], y=y[index]))
  }
}

normalize <- function(x){     # borrowed from class
  x[x<0] <- 0
  x <- x/sum(x)
  return(x)
}

mse <- function(pred, true) {
  mse = mean((pred-true)^2)
  return(mse)
}
```

```
xyzHistogramRelease <- function(x, y, z, xlower, xupper, ylower, yupper, zlower, zupper, xnbins=0, ynbi

  if(xnbins==0){
    xlower <- floor(xlower)
    xupper <- ceiling(xupper)
    xbins <- xlower:(xupper+1)
    xnbins <- length(xbins)-1
    xgranularity <- 1
    xcodebook <- xbins[1:xnbins]
  } else {
    xbins <- seq(from=xlower, to=xupper, length=xnbins+1)
    xgranularity <- (xupper-xlower)/xnbins
    xbins[xnbins+1] <-  xbins[xnbins+1] + xgranularity
    xcodebook <- xbins[1:xnbins] + 0.5*xgranularity
  }

  if(ynbins==0){
    ylower <- floor(ylower)
    yupper <- ceiling(yupper)
    ybins <- ylower:(yupper+1)
    ynbins <- length(ybins)-1
    ygranularity <- 1
    ycodebook <- ybins[1:ynbins]
  } else {
    ybins <- seq(from=ylower, to=yupper, length=ynbins+1)
    ygranularity <- (yupper-ylower)/ynbins
    ybins[ynbins+1] <-  ybins[ynbins+1] + ygranularity
    ycodebook <- ybins[1:ynbins] + 0.5*ygranularity
  }

  if(znbins==0){
    zlower <- floor(zlower)
    zupper <- ceiling(zupper)
    zbins <- zlower:(zupper+1)
    znbins <- length(zbins)-1
    zgranularity <- 1
    zcodebook <- zbins[1:znbins]
  } else {
    zbins <- seq(from=zlower, to=zupper, length=znbins+1)
    zgranularity <- (zupper-zlower)/znbins
    zbins[znbins+1] <-  zbins[znbins+1] + zgranularity
    zcodebook <- zbins[1:znbins] + 0.5*zgranularity
  }

  x.clipped <- clip(x=x, lower=xlower, upper=xupper)
  y.clipped <- clip(x=y, lower=ylower, upper=yupper)
  z.clipped <- clip(x=z, lower=zlower, upper=zupper)

  sensitivity <- 2
  scale <- sensitivity / (epsilon)

  sensitiveValue <- DPrelease <- matrix(NA, nrow=xnbins*ynbins*znbins, ncol=4)

  row_ind <- 1
  for(i in 1:xnbins){
    for(j in 1:ynbins){
      for(k in 1:znbins){
        bin_count <- sum(x.clipped >= xbins[i] & x.clipped < xbins[i+1] & y.clipped >= ybins[j] & y.cli
        release_count <- bin_count + rlap(mu=0, b=scale, size=1)
        sensitiveValue[row_ind,] <- c(i,j,k,bin_count)
        DPrelease[row_ind,] <- c(i,j,k,release_count)
        row_ind = row_ind + 1
      }
    }
  }

  return(list(release=DPrelease, true=sensitiveValue, xcodebook=xcodebook, ycodebook=ycodebook, zcodebo
}

# clip and log income
pums$log_income_clipped <- log(clip(x=pums$income, lower=1, upper=1000000))
log_income_low <- floor(log(1))
log_income_high <- ceiling(log(1000000))

# run histogram release
start = Sys.time()
res = xyzHistogramRelease(x=pums$educ, y=pums$age, z=pums$log_income_clipped,
                          xlower=1, xupper=16, ylower=18, yupper=100, zlower=log_income_low, zupper=log
                          xnbins=0, ynbins=0, znbins=10, epsilon=1)
end = Sys.time()
end-start

# private results

synthetic_n <- 10000
synthetic_bin_probs <- normalize(res$release[,4])
synthetic_inds <- sample(x=1:nrow(res$release), size=synthetic_n, replace=TRUE, prob=synthetic_bin_prob
synthetic_data <- res$release[synthetic_inds,1:3]
synthetic_data_df <- data.frame(synthetic_data)
names(synthetic_data_df) <- c("educ", "age", "log_income")

synthetic_data_df$educ <- plyr::mapvalues(synthetic_data_df$educ,
                                          from=sort(unique(synthetic_data_df$educ)),
                                          to=res$xcodebook)
synthetic_data_df$age <- plyr::mapvalues(synthetic_data_df$age,
                                          from=sort(unique(synthetic_data_df$age)),
                                          to=res$ycodebook)
synthetic_data_df$log_income <- plyr::mapvalues(synthetic_data_df$log_income,
                                         from=sort(unique(synthetic_data_df$log_income)),
                                         to=res$zcodebook)

synthetic_reg <- lm(log_income ~ age + educ, data = synthetic_data_df)
synthetic_slopes <- coef(synthetic_reg)[1:3]

# sensitive results
true_reg <- lm(log_income_clipped ~ age + educ, data = pums)
true_slopes <- coef(true_reg)[1:3]
```

```
# error
paste0("MSE for intercept: ", mse(synthetic_slopes[1], true_slopes[1]))
paste0("MSE for age coefficient: ", mse(synthetic_slopes[2], true_slopes[2]))
paste0("MSE for education coefficient: ", mse(synthetic_slopes[3], true_slopes[3]))

# simulations to examine contribution to MSE of bias and variance
n_sims <- 10
sim_history <- matrix(NA, nrow=n_sims, ncol=3)

if(run_sims_flag){
  for(i in 1:n_sims){
    print(i)
    res_sim = xyzHistogramRelease(x=pums$educ, y=pums$age, z=pums$log_income_clipped,
                                  xlower=1, xupper=16, ylower=18, yupper=100, zlower=log_income_low, zu
                                  xnbins=0, ynbins=0, znbins=10, epsilon=1)
    synthetic_bin_probs_sim <- normalize(res_sim$release[,4])
    synthetic_inds_sim <- sample(x=1:nrow(res_sim$release), size=synthetic_n, replace=TRUE, prob=synthe
    synthetic_data_sim <- res_sim$release[synthetic_inds_sim,1:3]
    synthetic_data_sim_df <- data.frame(synthetic_data_sim)
    names(synthetic_data_sim_df) <- c("educ", "age", "log_income")

    synthetic_data_sim_df$educ <- plyr::mapvalues(synthetic_data_sim_df$educ,
                                                  from=sort(unique(synthetic_data_sim_df$educ)),
                                                  to=res_sim$xcodebook)
    synthetic_data_sim_df$age <- plyr::mapvalues(synthetic_data_sim_df$age,
                                                 from=sort(unique(synthetic_data_sim_df$age)),
                                                 to=res_sim$ycodebook)
    synthetic_data_sim_df$log_income <- plyr::mapvalues(synthetic_data_sim_df$log_income,
                                                        from=sort(unique(synthetic_data_sim_df$log_income))
                                                        to=res_sim$zcodebook)

    synthetic_reg_sim <- lm(log_income ~ age + educ, data = synthetic_data_sim_df)
    synthetic_slopes_sim <- coef(synthetic_reg_sim)[1:3]

    sim_history[i,1] <- synthetic_slopes_sim[1]
    sim_history[i,2] <- synthetic_slopes_sim[2]
    sim_history[i,3] <- synthetic_slopes_sim[3]
  }
  write.csv(sim_history, "sim_history.csv")
}

sim_history <- read.csv("sim_history.csv")


# bootstrap to examine sampling error
n_boots <- 1000
boot_size <- 1000
boot_history <- matrix(NA, nrow=n_boots, ncol=3)
for(i in 1:n_boots){
  boot_inds <- bootstrap(x=1:nrow(pums), n=boot_size)
  boot_data <- pums[boot_inds,]
  boot_reg <- lm(log_income_clipped ~ age + educ, data=boot_data)
  boot_slopes <- coef(boot_reg)[1:3]
  boot_history[i,1] <- boot_slopes[1]
  boot_history[i,2] <- boot_slopes[2]
  boot_history[i,3] <- boot_slopes[3]
}

# print results
mse_int <- mse(pred=sim_history[,2], true=true_slopes[1])
var_int <- var(sim_history[,2])
bias_sq_int <- mse_int - var_int
sampling_mse_int <- mse(boot_history[,1], true_slopes[1])

mse_age <- mse(pred=sim_history[,3], true=true_slopes[2])
var_age <- var(sim_history[,3])
bias_sq_age <- mse_age - var_age
sampling_mse_age <- mse(boot_history[,2], true_slopes[2])

mse_educ <- mse(pred=sim_history[,4], true=true_slopes[3])
var_educ <- var(sim_history[,4])
bias_sq_educ <- mse_educ-var_educ
sampling_mse_educ <- mse(boot_history[,3], true_slopes[3])

cat("Intercept","\nDP MSE: ", mse_int, "\nDP Variance: ", var_int, "\nDP Bias^2: ", bias_sq_int, "\nSamp
cat("Age","\nDP MSE: ", mse_age, "\nDP Variance: ", var_age, "\nDP Bias^2: ", bias_sq_age, "\nSampling M
cat("Educ","\nDP MSE: ", mse_educ, "\nDP Variance: ", var_educ, "\nDP Bias^2: ", bias_sq_educ, "\nSampli

# save results
q3_results <- data.frame("Metric"=c("DP MSE","DP Variance","DP Bias^2","Sampling MSE"),
                         "Intercept"=c(mse_int,var_int,bias_sq_int,sampling_mse_int),
                         "Education"=c(mse_age,var_age,bias_sq_age,sampling_mse_age),
                         "Age"=c(mse_educ,var_educ,bias_sq_educ,sampling_mse_educ))
write.csv(q3_results, "q3_results.csv")
```