CS 208 - Applied Privacy for Data Science Homework 2

Jason Huang
Spring 2019 - Harvard University

The public Github repo containing all work is at https://github.com/TurboFreeze/cs208hw. All code has also been included in the appendix of this PDF as specified.

Problem 1

(i) (a) The clamping function is effectively applying a post-processing function to the noisy query result. In other words, Laplace noise is added to the true mean $\bar{x}$, which must be $(\epsilon, 0)$-DP. The following clamping function does not change the privacy characteristics guaranteed by differential privacy, meaning that this mechanism **meets the definition** of $(\epsilon, 0)$-DP (following directly by privacy under post-processing and the proof of Laplace DP).

Note that the scale factor parameter of the Laplace distribution should be set to $s = GS_q/\epsilon$ for differential privacy, meaning that $\epsilon = GS_q/s$. In this case, the global sensitivity GS_q is the maximum change that can be affected to the statistic by a single entry's change, which in this case would be $1/n$ for the mean. Furthermore $s = 2/n$. The $\epsilon = (1/n)/(2/n) \implies \boxed{\epsilon = 0.5}$.

(ii) Constant ratios of Laplace mechanisms

(iii)

$$\frac{P[M(x', q) = r]}{P[M(x, q) = r]} =$$

(iv)

$$\begin{aligned} P[M(x, q) = r] &= P[[\bar{x} + Z]_0^1 = r] \\ &= \\ \frac{P[M(x, q) = r]}{P[M(x', q) = r]} &= \\ P[M(x, q) = r] &= P[\bar{x} + [Z]_{-1}^1 = r] \\ &= \end{aligned}$$

(d)

Problem 2

(a) The DGP is the following likelihood of some data vector $k \in \mathbb{N}^n$:

$$P(\mathbf{x} = \mathbf{k}) = \prod_{i=1}^{n} \frac{10^{\mathbf{k}_i} e^{-10}}{\mathbf{k}_i!}$$

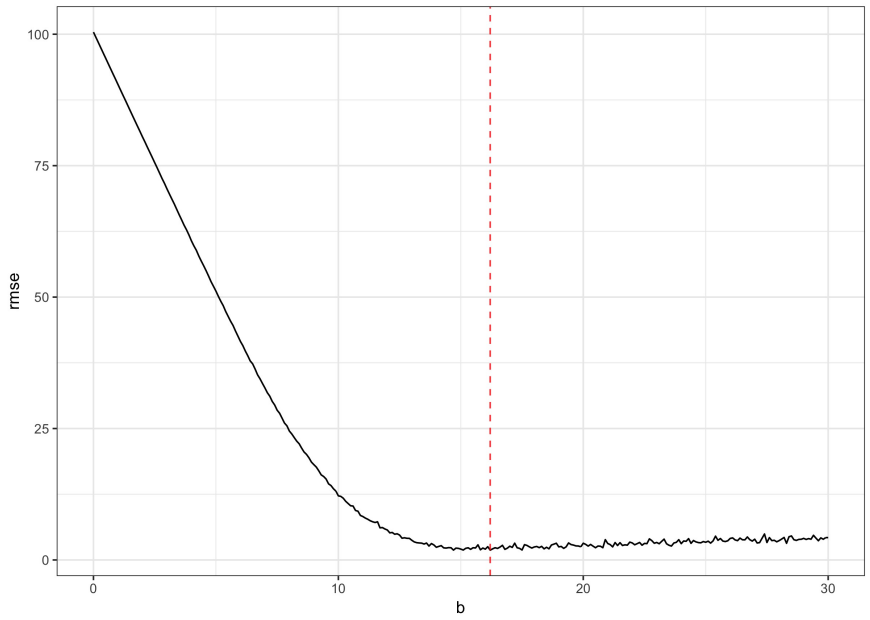
The DGP function was implemented using a Poisson random draw.

See the attached R script `q2.R` *for the implementation.*

(b) The first mechanism was chosen, involving clamping after Laplace noise has been added.

See the attached R script `q2.R` *for the implementation.*

(c) The optimal value b^* for b is $\boxed{b^* \approx 15}$. As expected, root mean squared error is indeed high with small clamping regions and decreases as it becomes more appropriate, with large clamping regions yielding high RMSE again.



See the attached R script `q2.R` *for the implementation.*

(d) This approach is not safe and might violate differential privacy because it implicitly incorporates information about the dataset that is not included in the data. More specifically, bootstrapping is powerful because of its very ability to simulate the data generating process and the distribution of the data. Therefore, bootstrapping to find an estimate for the optimal value b^* is effectively the same thing as calculating the true optimal b^*, only numerically instead of analytically. Now, knowing the optimal b^* violates privacy because it also provides insight into the distribution (i.e. one of the ways this can be intuitively described is as an approximate maximum of everything that is not an outlier). This is similar to the idea of why local sensitivity, which is a characteristic of the data set, is not

(ϵ, δ)-DP but global sensitivity is. The optimal b^* is a specific data-dependent attribute of the data set that can leak privacy if not controlled for, as is the case with bootstrapping here.

(e) Use an external reference for determining an approximation for b^*. Generally, a similar public data set (i.e. past iterations of the census or different geographic tracts that have been made public) can help provide a rough estimate of the upper bound for the individual. Often times, a reasonable heuristic can be used based on common knowledge, such as the rough range of human ages or other metadata provided for the data set (i.e. in a codebook or schema).

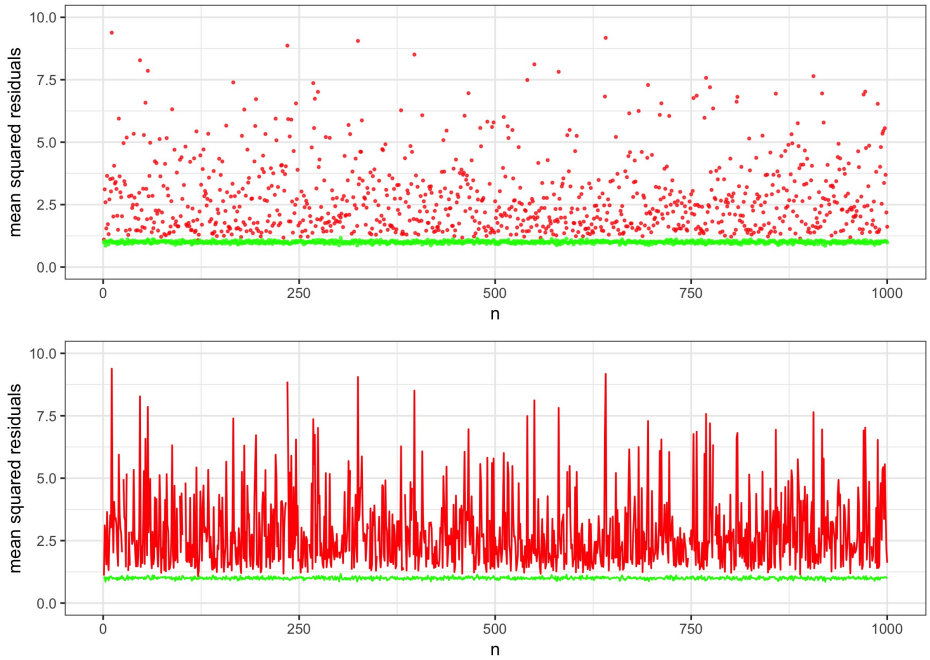
Problem 3

(a) There are differentially private techniques to release the means $\bar{y}$ and $\bar{x}$ as well as the slope $\hat{\beta}$. However, given the careful considerations needed for the slope $\hat{\beta}$, it may be challenging to come up with a single differentially private mechanism to derive the intercept estimate. However $\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$ lends itself nicely to privacy preservation under composition and post-processing. Since there are three differentially private statistics needed here of $\bar{x}, \bar{y}, \hat{\beta}$, for a total epsilon budget of ϵ_t, then calculate differentially private releases of each statistic with $\epsilon = \epsilon_t/4$ (note that the slope $\hat{\beta}$ is actually two statistics of covariance and variance in the numerator and denominator respectively) to yield $(\epsilon_t/4, 0)$-DP statistics. Since post-processing is allowed without affecting privacy, then this three differentially private statistics will lead to $\epsilon_t/4 + \epsilon_t/4 + \epsilon_t/2 = \epsilon_t$ differential privacy for $\hat{\alpha}$ by composition and $\epsilon_t/4 + \epsilon_t/4 = \epsilon_t/2$ differential privacy for $\hat{\beta}$ (which is used in $\hat{\alpha}$ and does not require separate consumption of the budget). Therefore, the overall method for computing both $\hat{\alpha}$ and $\hat{\beta}$ would be ϵ_t-DP, as desired.

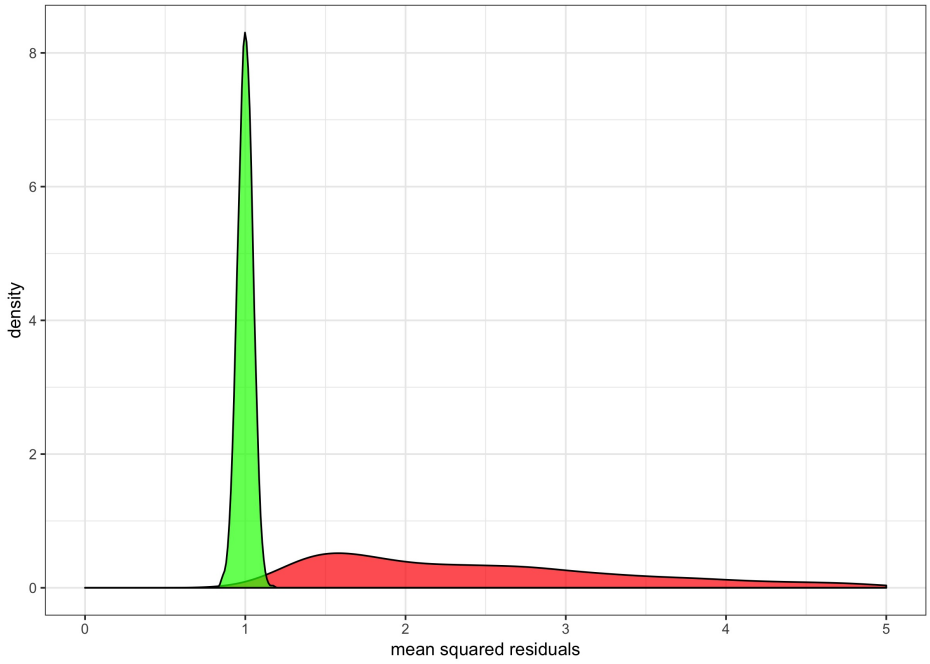
Since the data x_i is generated by a Poisson process according to the previous problem, it can be clamped using the optimal value of $b^* \approx 10$ found before. Since there is a linear relationship between x_i and y_i here (and it is known in the following part that the slope is simply 1), then similarly clamp y_i by $b^* \approx 15$.

See the attached R script `q3.R` *for the implementation.*

(b) A Monte Carlo simulation yields the following plots. The differentially private regression is in red while the non-private simple linear regression results are in green. Here is the traceplot and scatter plot of mean squared residuals against the number of simulations n. Note also that the residual error is clipped to (0, 10), with a very small number of differentially private errors (around 25 out of 1000) jumping well beyond the upper bound of 10.



When examining the distribution of mean-squared residuals as desired, then following density curves were calculated.



Recall that mean squared residuals is a measure for residuals, or the addition of noise on top of everything that is not already explained by the model. It can be seen from the above graphic that the non-private residuals in green peak tightly around 1, likely due to the fact that the noise added is Gaussian with standard deviation of 1. In fact, the distribution of the noise itself is symmetrical around 1 and *appears Gaussian in distribution*. Similarly, the differentially private residuals in red have a mode in density between 1 and 2, and has a substantially longer right tail. In fact, the entire distribution is reminiscent of the *Poisson distribution*, even specifically Poisson process with $\lambda = 10$, speaking to the ability of this differentially private mechanism to inject noise while accurately reflecting the behavior of the true data generating process.

See the attached R script `q3.R` *for the implementation.*

(c) Grid search was implemented here by setting a hyperparameter domain of $x_1, x_2, x_3, x_4 \in \mathcal{D} = \{0.1, 0.2, \ldots, 1\}$, recursively creating every possible combination of 4 values from this domain and most importantly, normalizing these to create a distribution; in other words, such that the normalized vector $\mathbf{x}' = (x'_1, x'_2, x'_3, x'_4)$ has $\sum_{i=1}^{4} x'_i = 1$. This is the process by which the $|\mathcal{D}^4| = 10000$ combinations within the parameter space are generated and calculated.

The differentially private regression was run on each of these parameter spaces, with the mean squared residuals calculated. With equal partitioning of ϵ, the mean squared residuals is ≈ 6.070752. With the 10000 combinations tested by the grid search, the top 10 with lowest mean squared residual error were found, with the corresponding hyperparameter configuration included all in the table below. The mean and median of these best 10 configurations is also included as a rough approximation of the optimal configuration for distribution ϵ. Note

that these do have substantially lower mean squared residuals than an equal partition.

	ϵ for $\text{Cov}(x, y)$	ϵ for $\text{Var}(x)$	ϵ for $\bar{x}$	ϵ for $\bar{y}$	Error
Even partition	0.25	0.25	0.25	0.25	6.070752
	0.18750000	0.56250000	0.06250000	0.18750000	0.9590050
	0.40000000	0.40000000	0.06666667	0.13333333	0.9595579
	0.05555556	0.33333333	0.55555556	0.05555556	0.9906817
	0.15384615	0.07692308	0.23076923	0.53846154	1.0065410
	0.30434783	0.34782609	0.04347826	0.30434783	1.0109240
	0.33333333	0.26666667	0.26666667	0.13333333	1.0116808
	0.28571429	0.42857143	0.14285714	0.14285714	1.0236383
	0.40000000	0.24000000	0.04000000	0.32000000	1.0285891
	0.34782609	0.21739130	0.34782609	0.08695652	1.0321955
	0.37500000	0.37500000	0.08333333	0.16666667	1.0404765
Mean	0.2843123	0.3248212	0.1839653	0.2069012	
Median	0.3188406	0.3405797	0.1130952	0.1547619	

There is a quite some variation among these best trials simply due to noise, hence taking the mean and median. There are nonetheless some patterns that are present, which the mean and median are particularly helpful in capturing, particularly the higher emphasis on $\text{Var}(x)$ and $\text{Cov}(x, y)$, and then the remaining $25\% - 40\%$ for $\bar{y}$ and $\bar{x}$, in that order of decreasing weight.

See the attached R script `q3.R` *for the implementation.*

Problem 4

Use linearity of expectations and fundamental bridge to convert between probabilities and expectation of indicators.

$$\begin{aligned}
\mathbb{E}[\#\{i \in [n] : A(M(X))_i = X_i\}/n] &= \mathbb{E}[\mathbb{1}\{i \in [n] : A(M(X))_i = X_i\}/n] \\
&= \mathbb{E}[\sum_{i=1}^{n} \mathbb{1}(A(M(X))_i = X_i)/n] \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[\mathbb{1}(A(M(X))_i = X_i)] \\
&= \frac{1}{n}\sum_{i=1}^{n} P(A(M(X))_i = X_i) \\
&= P(A(M(X))_i = X_i)
\end{aligned}$$

Where the last simplification is made by noting the independence of the X_i data draws. Let there be a dataset X^{i0} or X^{i1} such that the i^{th} row is changed to 0 or 1 respectively, with corresponding probabilities of p and $1-p$ by the nature of the given Bernoulli data generating process. Now by the definition of (ϵ, δ)-DP:

$$\begin{aligned}
\mathbb{E}[\#\{i \in [n] : A(M(X))_i = X_i\}/n] &= P(A(M(X))_i = X_i) \\
&\leq e^{\epsilon} \cdot \max\{P(A(M(X^{i0}))_i = X_i), P(A(M(X^{i1}))_i = X_i)\} + \delta \\
&\leq e^{\epsilon} \cdot \max\{p, 1-p\} + \delta
\end{aligned}$$

The desired result has been shown.

$$\therefore \boxed{\mathbb{E}[\#\{i \in [n] : A(M(X))_i = X_i\}/n] \leq e^{\epsilon} \cdot \max\{p, 1-p\} + \delta} \text{ as desired}$$

Appendix

Code for Problem 1

Code for Problem 2

```
##
## q2.r
##
## Evaluating DP algorithms with synthetic data
##
## JH 2019/03/10
##



##3 PART (A)
# data generating process
dgp <- function (n, lambda=10) {
  rpois(n, lambda)
}



### PART (B)

# sign function
sgn <- function(x) {
  return(ifelse(x < 0, -1, 1))
}

# laplace random draws
rlap <- function(mu=0, s=1, size=1) {
  p <- runif(size) - 0.5
  draws <- mu - s * sgn(p) * log(1 - 2 * abs(p))
  draws
}

# clamping helper function
clamp <- function (data, a, b) {
  data.clamped <- data
  data.clamped[data < a] <- a
  data.clamped[data > b] <- b
  data.clamped
}

# differentially private mechanism (clamping)
dpclamping <- function (data, epsilon, a=0, b) {
  mean.actual <- mean(clamp(data, a, b))
```

```
  # generate noise by Laplace mechanism
  data.len <- length(data)
  sensitivity <- (b - a) / data.len
  laplace.shift <- sensitivity / epsilon
  noise <- rlap(s=laplace.shift, size=1)

  # inject noise
  mean.noisy <- mean.actual + noise

  # apply clamping
  mean.clamped <- clamp(mean.noisy, a, b)
}



### PART (C)

# parameters
n <- 200
epsilon <- 0.5
b.seq <- seq(0, 30, by=0.1)
b.num <- length(b.seq)

# generate data
data.poisson <- dgp(n, lambda=10)

# calculate RMSE
rmse <- function (data, m) {
  sqrt(sum((data - m)^2))
}

n.trials <- 100
rmses <- vector("numeric", b.num)
for (i in 1:b.num) {

  dpmeans <- vector("numeric", n.trials)
  for (j in 1:n.trials) {
    # calculate dp query of mean
    dpmeans[j] <- dpclamping(data=data.poisson, epsilon=epsilon, b=b.seq[i])
  }
  # calculate and store RMSE
  rmses[i] <- rmse(dpmeans, m=mean(data.poisson))
}
```

```
# find index of minimum RMSE (optimal value of b)
b.optimal <- b.seq[which.min(rmses)]; b.optimal

#plot(b.seq, rmses, type='l') # uncomment if no ggplot

# visualize
rmsedata <- data.frame(b=b.seq, rmse=rmses)
library(ggplot2)
clamping.plot <- ggplot(rmsedata, aes(x=b, y=rmse)) +
  geom_line() +
  geom_vline(xintercept=b.optimal, color="red", alpha=0.8, linetype="dashed")
  theme_bw()
clamping.plot
ggsave("clamping.jpg", clamping.plot)
```

Code for Problem 3

```
##
## q3regression.r
##
## Differentially private regression
##
## JH 2019/03/10
##



### PART (A)

# optimal upper bound found in previous part
b.optimal <- 15

# data generating process
dgp <- function (n, lambda=10) {
  rpois(n, lambda)
}

# sign function
sgn <- function(x) {
  return(ifelse(x < 0, -1, 1))
}

# clamping helper function
clamp <- function (data, a, b) {
  data.clamped <- data
  data.clamped[data < a] <- a
  data.clamped[data > b] <- b
  data.clamped
}

# laplace random draws
rlap <- function(mu=0, s=1, size=1) {
  p <- runif(size) - 0.5
  draws <- mu - s * sgn(p) * log(1 - 2 * abs(p))
  draws
}

# differentially private mean release
dpmean <- function (data, epsilon, a=0, b) {
  mean.actual <- mean(clamp(data, a, b))
```

```
  # generate noise by Laplace mechanism
  data.len <- length(data)
  sensitivity <- (b - a) / data.len
  laplace.shift <- sensitivity / epsilon
  noise <- rlap(s=laplace.shift, size=1)

  # inject noise
  mean.noisy <- mean.actual + noise

  # apply clamping
  mean.clamped <- clamp(mean.noisy, a, b)
}


# Differentially private regression slope release (from provided code)
dpslope <- function (y, x, ylower=0, yupper=b.optimal, xlower=0, xupper=b.opt
  x <- clamp(x, xlower, xupper)
  y <- clamp(y, ylower, yupper)

  n <- length(x)
  sens.sxy <- (xupper - xlower) * (yupper - ylower)
  sens.sxx  <- (xupper - xlower)^2

  scale.sxy <- sens.sxy / eps1
  scale.sxx <- sens.sxx / eps2

  sensitive.value <- sum((x - mean(x)) * (y - mean(y))) / sum((x - mean(x))^2

  release.sxy <- sum((x - mean(x)) * (y - mean(y))) + rlap(mu=0, s=scale.sxy,
  release.sxx <- sum((x - mean(x))^2) + rlap(mu=0, s=scale.sxx, size=1)

  postprocess.beta <- release.sxy / release.sxx
  postprocess.beta
}

# Differentially private linear regression release of intercept and slope
dpregression <- function (y, x, ylower=0, yupper=b.optimal, xlower=0, xupper=
                          epsilon, epsilon.partition=c(0.25, 0.25, 0.25, 0.25
  x <- clamp(x, xlower, xupper)
  y <- clamp(y, ylower, yupper)

  # calculate dp slope
  dpbeta <- dpslope(y, x, ylower, yupper, xlower, xupper,
                    epsilon * epsilon.partition[1], epsilon * epsilon.partiti
```

```
  # calculate dp intercept
  dpmean.x <- dpmean(x, epsilon * epsilon.partition[3], b=xupper)
  dpmean.y <- dpmean(y, epsilon * epsilon.partition[4], b=yupper)
  dpalpha <- dpmean.y - dpbeta * dpmean.x

  list(dpalpha=dpalpha, dpbeta=dpbeta)
}



### PART (B)

n <- 1000
alpha <- 1
beta <- 1
sigma <- 1
epsilon <- 1

# mean squared residuals for quantifying utility
msr <- function (y, x, beta.hat, alpha.hat) {
  sum((y - beta.hat * x - alpha.hat)^2) / length(y)
}

simulate <- function(n.trials=100) {
  msrs.dp <- vector("numeric", n.trials)
  msrs.non <- vector("numeric", n.trials)
  for (i in 1:n.trials) {
    # generate data
    x <- dgp(n, lambda=10)
    y <- beta * x + alpha + rnorm(n, 0, sigma)

    # calculate differentially private release
    results.dp <- dpregression(y, x, epsilon=epsilon)
    msrs.dp[i] <- msr(y, x, results.dp$dpbeta, results.dp$dpalpha)

    # calculate non-private release
    results.non <- lm(y ~ x)
    msrs.non[i] <- msr(y, x, coef(results.non)[[2]], coef(results.non)[[1]])
  }
  data.frame(n=(1:n.trials), dp=msrs.dp, non=msrs.non)
}

msrs.data <- simulate(1000)
```

```
# visualize
library(ggplot2)
y.window <- c(0, 10)

# monte carlo scatterplot
msrs.scatterplot <- ggplot(msrs.data, aes(x=n)) +
  geom_point(aes(y=dp), color="red", alpha=0.7, size=0.5) +
  geom_point(aes(y=non), color="green", alpha=0.7, size=0.5) +
  ylim(y.window) +
  ylab("mean squared residuals") +
  theme_bw(); msrs.scatterplot

# monte carlo traceplot
msrs.traceplot <- ggplot(msrs.data, aes(x=n)) +
  geom_line(aes(y=dp), color="red") +
  geom_line(aes(y=non), color="green") +
  ylim(y.window) +
  ylab("mean squared residuals") +
  theme_bw(); msrs.traceplot

# distributions
msrs.distribution <- ggplot(msrs.data) +
  xlim(c(0, 5)) +
  xlab("mean squared residuals") +
  geom_density(aes(dp), fill="red", alpha=0.7) +
  geom_density(aes(non), fill="green", alpha=0.7) +
  theme_bw(); msrs.distribution
# distribution histograms
#hist(msrs.data$dp[msrs.data$dp < 10]) # uncomment if no ggplot

library(gridExtra)
gridplots <- grid.arrange(msrs.scatterplot, msrs.traceplot)

ggsave("msrs.jpg", gridplots)
ggsave("msrsdist.jpg", msrs.distribution)



### PART (C)
# helper function to normalize to distribution of weights
normalize <- function(vec) {
  vec / sum(vec)
}

# hyperparameter space
```

```
param.domain <- seq(0.1, 1, by=0.1)
# recursive function to generate all hyperparamater combinations
param.combs <- function (hypers) {
  if (length(hypers) == 4) {
    normalize(hypers)
  } else {
    res <- c()
    for (i in 1:length(param.domain)) {
      res <- rbind(res, param.combs(c(hypers, param.domain[i])))
    }
    res
  }
}
# generate hyperparameter combinations
param.space <- param.combs(c())
param.msr <- vector("numeric", nrow(param.space))

# calculate mean squared residuals
for (i in 1:nrow(param.space)) {
  # generate data
  x <- dgp(n, lambda=10)
  y <- beta * x + alpha + rnorm(n, 0, sigma)

  # calculate differentially private release
  results.dp <- dpregression(y, x, epsilon=epsilon, epsilon.partition=param.s
  param.msr[i] <- msr(y, x, results.dp$dpbeta, results.dp$dpalpha)
}

# equal partition of epsilon
results.dp <- dpregression(y, x, epsilon=epsilon, epsilon.partition=param.spa
equal.msr <- msr(y, x, results.dp$dpbeta, results.dp$dpalpha); equal.msr

# get minimum mean squared residuals
min.index <- c()
min.space <- c()
min.msr <- c()
param.msr.working <- param.msr
param.space.working <- param.space
# get top 10 hyperparameter configurations
for (i in 1:10) {
  m <- which.min(param.msr.working)
  min.index <- c(min.index, m)
  min.msr <- c(min.msr, param.msr.working[m])
  min.space <- rbind(min.space, param.space.working[m,])
  param.msr.working <- param.msr.working[-1 * m]
```

```
  param.space.working <- param.space.working[-1 * m,]
}
# find mean and median of hyperparameters
min.msr
mea <- apply(min.space, mean, MARGIN = 2); mea
med <- apply(min.space, median, MARGIN = 2); med
```