HW2

CS208

Lipika Ramaswamy

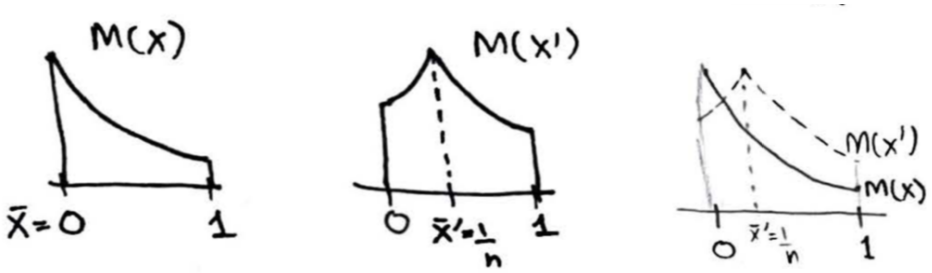
Problem 1:

Mechanisms: Consider the following mechanisms M that takes a dataset $x \in [0, 1]^n$ and returns an estimate of the mean $\bar{x} = (\frac{1}{n} \sum_{i=1}^{n} x_i)$.

(i) $M(x) = [\bar{x} + Z]_0^1$ for $Z \sim Lap(2/n)$.

The global sensitivity of the mean with $x \in [0,1]^n$ is $GS_q = \frac{1-0}{n} = \frac{1}{n}$.

Consider the worst case corresponding to this mechanism.
Let $x = \{0, 0, \ldots 0\}$ and and $x' = \{1, 0, 0, \ldots, 0\}$. Then $\bar{x} = 0$ and $\bar{x}' = \frac{1}{n}$. M(x) and M(x') are bounded in $[0, 1]$. The plots below show the distribution of $M(x)$, $M(x')$ and the two overlaid.



To determine whether a mechanism is ϵ-DP, we need to determine whether we can scale up the probability mass of $M(x')$ to cover the probability mass of $M(x)$. Visually, we can see that this is possible here.

According to the definition of privacy loss, we have:

$$\begin{aligned}
\frac{Pr[M(x) = r]}{Pr[M(x') = r]} &= \frac{\frac{n}{4} exp(-|r - \bar{x}|\frac{n}{2})}{\frac{n}{4} exp(-|r - \bar{x}'|\frac{n}{2})} \\
&= exp\left(\frac{n}{2}\right) + exp\left(-|r - \bar{x}| + |r - \bar{x}'|\right) \\
&\leq exp\left(\frac{n}{2}\right) + exp\left(|-r + \bar{x} + r - \bar{x}'|\right) \\
&= exp\left(\frac{n}{2}\right) + exp\left(|\bar{x} - \bar{x}'|\right) \\
&= exp\left(\frac{n}{2}\right) + exp\left(|0 - \frac{1}{n}|\right) \\
&= exp\left(\frac{n}{2} \times \frac{1}{n}\right) \\
&= exp\left(\frac{1}{2}\right)
\end{aligned}$$

We know that $\frac{Pr[M(x)=r]}{Pr[M(x')=r]} \leq e^{\epsilon}$. Hence we get that $\frac{1}{2}$ is the smallest ϵ that satisfies the ϵ-0 definition of differential privacy.

Thus, this mechanism is ϵ-0 DP.

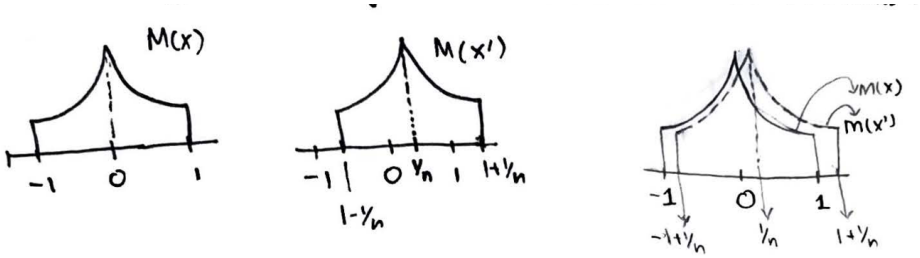
For the given range of data [0,1], we know that the scale of the Laplace noise added is $\frac{GS_q}{\epsilon} = \frac{\frac{1-0}{n}}{\epsilon}$. We used this information above to calculate the minimum ϵ for this mechanism. We could tune this ϵ to be larger than 0.5, but the larger the value of ϵ, the lower the privacy as the scale of the Laplace noise would be lower.

If the data domain changes to $[a, b]$, the global sensitivity scales by a factor of (b-a), and the minimum value of ϵ changes proportional to (b-a). This follows from a new worst case where we have a dataset with all 'a's, and then flip one to a 'b'.

(ii) $M(x) = \bar{x} + [Z]^{1}_{-1}$ for $Z \sim Lap(2/n)$

The global sensitivity of the mean with $x \in [0,1]^n$ is $GS_q = \frac{1-0}{n} = \frac{1}{n}$.

Consider the worst case corresponding to this mechanism.
Let $x = \{0, 0, \ldots 0\}$ and and $x' = \{1, 0, 0, \ldots, 0\}$. Then $\bar{x} = 0$ and $\bar{x}' = \frac{1}{n}$. The noise is bounded in $[-1, 1]$.
The plots below show the distribution of $M(x)$, $M(x')$ and the two overlaid.



To determine whether a mechanism is ϵ-DP, we need to determine whether we can scale up the probability mass of $M(x')$ to cover the probability mass of $M(x)$. Visually, we can see that this is not possible since the probability mass of $M(x')$ is 0 for $x \leq -1 + \frac{1}{n}$. **Thus, this mechanism is not ϵ-0 DP.**

$\delta*$, the minimum δ required to make the mechanism ϵ-δ DP, can be determined using the formula:

$$\delta^* \geq \max_{x \sim x'} \left[\sum_y max\Big[\Pr[M(x) = y] - e^{\epsilon} \cdot \Pr[M(x') = y], 0 \Big] \right]$$

$$= \left[max\Big[\int_{-\infty}^{-1+\frac{1}{n}} \frac{\exp(\frac{|y-\bar{x}|}{2/n})}{4/n} \partial y - e^{\epsilon} \cdot \int_{-1}^{-1+\frac{1}{n}} \frac{\exp(\frac{|y-\bar{x}'|}{2/n})}{4/n} \partial y, 0 \Big] \right]$$

$$= \left[max\Big[\int_{-1}^{-1+\frac{1}{n}} \frac{\exp(\frac{|y-\bar{x}|}{2/n})}{4/n} \partial y - e^{\epsilon} \cdot 0, 0 \Big] \right]$$

$$= \int_{-1}^{-1+\frac{1}{n}} \frac{\exp(\frac{|y-0|}{2/n})}{4/n} \partial y$$

$$= \int_{-1}^{-1+\frac{1}{n}} \frac{\exp(\frac{-yn}{2})}{4/n} \partial y$$

$$= \frac{n}{4} \Big[\frac{2}{n} e^{yn/2} \Big]_{-\infty}^{-1+\frac{1}{n}}$$

$$= \frac{n}{4} \cdot \frac{2}{n} \Big[e^{(-n+1)/2} - e^{-\infty} \Big]$$

$$= \frac{1}{2} e^{(-n+1)/2}$$

The minimum value of δ is $\frac{1}{2} e^{(-n+1)/2}$. **Thus, this mechanism is ϵ-$\frac{1}{2} e^{(-n+1)/2}$ DP.**

The value of ϵ has no minimum in this case and can be freely tuned. Note that the value of delta will be smaller than 0.6 (= $e^{(-2+1)/2}$ for a dataset of size 2). With a probability of δ, the output of the mechanism reveals whether the database used was x or x'. Thus, δ can never be greater than 1. The smaller the value of delta, the less the privacy loss.

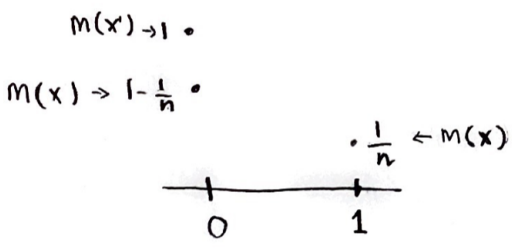
If the data domain changes to $[a, b]$, the value of δ would scale by $e^{\frac{b-a}{2}}$, increasing the value of delta. Thus larger ranges of (b-a) will provide higher probability that the mechanism reveals whether the database used was x or x', i.e. the amount of information revealed increases and there's a loss of privacy.

(iii)

$$M(x) = \begin{cases} 1, & \text{w.p. } \bar{x} \\ 0, & \text{w.p. } 1 - \bar{x} \end{cases}$$

The global sensitivity of the mean with $x \in [0,1]^n$ is $GS_q = \frac{1-0}{n} = \frac{1}{n}$.

Consider the worst case corresponding to this mechanism.
Let $x = \{1, 0, \ldots 0\}$ and and $x' = \{0, 0, \ldots, 0\}$. Then $\bar{x} = \frac{1}{n}$ and $\bar{x}' = 0$. The mechanism corresponds a Bernoulli trial, with the probability of success set to $\bar{x}$. The figure below shows the probability masses for the two mechanisms, and it's clear that the entire mass of $M(x')$ lies at 0, so it can never be scaled up to cover $M(x)$ at 1. **Thus, this mechanism is not ϵ-0 DP.**



$\delta*$, the minimum δ required to make the mechanism ϵ-δ DP, can be determined using the formula:

$$\begin{aligned}
\delta^* &\geq \max_{x \sim x'} \left[\sum_y max\Big[\Pr[M(x) = y] - e^{\epsilon} \cdot \Pr[M(x') = y], 0 \Big] \right] \\
&= \max_{x \sim x'} \left[max\Big[\Pr[M(x) = 0] - e^{\epsilon} \cdot \Pr[M(x') = 0], 0 \Big] + max\Big[\Pr[M(x) = 1] - e^{\epsilon} \cdot \Pr[M(x') = 1], 0 \Big] \right] \\
&= \left[max\Big[1 - \frac{1}{n} - e^{\epsilon} \cdot 1, 0 \Big] + max\Big[\frac{1}{n} - e^{\epsilon} \cdot 0, 0 \Big] \right] \\
&= \left[max\Big[\frac{1}{n} - e^{\epsilon} \cdot 0, 0 \Big] \right] \\
&= \frac{1}{n}
\end{aligned}$$

The minimum value of δ is $\frac{1}{2}$. **Thus, this mechanism is ϵ-$\frac{1}{n}$ DP.**

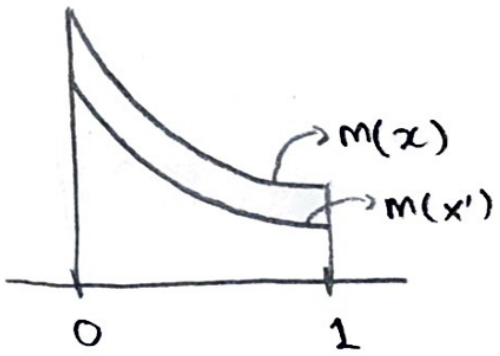
The value of ϵ has no minimum in this case and can be freely tuned. Note that the value of delta will be smaller than 0.5 (= $1/2$ for dataset of size 2). With a probability of δ, the output of the mechanism reveals whether the database used was x or x'. Thus, δ can never be greater than 1. The smaller the value of delta, the less the privacy loss.

If the data domain changes to $[a, b]$, the value of δ would scale by $b - a$, increasing the value of delta. Thus larger ranges of (b-a) will provide higher probability that the mechanism reveals whether the database used was x or x', i.e. the amount of information revealed increases and there's a loss of privacy.

(iv) $M(x) = Y$ where Y has probability density function f_Y given as follows:

$$f_Y(y) = \begin{cases} \frac{e^{-n|y-\bar{x}|/10}}{\int_0^1 e^{-n|y-\bar{x}|/10} dy} & \text{if } y \in [0, 1] \\ 0 & \text{if } y \notin [0, 1] \end{cases}$$

Consider the worst case corresponding to this mechanism.
Let $x = \{0, 0, \ldots 0\}$ and and $x' = \{1, 0, 0, \ldots, 0\}$. Then $\bar{x} = 0$ and $\bar{x}' = \frac{1}{n}$. M(x) and M(x') are bounded in $[0, 1]$. The figure below show the distribution of $M(x)$ and $M(x')$.



To determine whether a mechanism is ϵ-DP, we need to determine whether we can scale up the probability mass of $M(x')$ to cover the probability mass of $M(x)$. Visually, we can see that this is possible here. The ratio of $M(x)$ and $M(x')$ is computed below:

$$\frac{P[M(x) = y]}{P[M(x') = y]} = \frac{\frac{e^{-n|y-\bar{x}|/10}}{\int_0^1 e^{-n|y-\bar{x}|/10} dy}}{\frac{e^{-n|y-\bar{x}'|/10}}{\int_0^1 e^{-n|y-\bar{x}'|/10} dy}}$$

$$= \frac{e^{-n|y-\bar{x}|/10}}{e^{-n|y-\bar{x}'|/10}} \cdot \frac{\int_0^1 e^{-n|y-\bar{x}'|/10} dy}{\int_0^1 e^{-n|y-\bar{x}|/10} dy}$$

Treating the left term, we get:

$$\frac{e^{-n|y-\bar{x}|/10}}{e^{-n|y-\bar{x}'|/10}} = e^{\frac{1}{10}(-n|y-\bar{x}|+n|y-\bar{x}'|)}$$

$$= e^{\frac{n}{10}(|y-\bar{x}'|-|y-\bar{x}|)}$$

$$\leq e^{\frac{n}{10}|\bar{x}-\bar{x}'|} \quad \text{using the triangle inequality}$$

$$\leq e^{\frac{n}{10}|0-\frac{1}{n}|}$$

$$\leq e^{\frac{n}{10}\frac{1}{n}}$$

$$\leq e^{\frac{1}{10}}$$

Treating the right term, we get:

$$
\begin{aligned}
\frac{\int_0^1 e^{-n|y-\vec{x}'|/10}dy}{\int_0^1 e^{-n|y-\bar{x}|/10}dy} &= \frac{\int_0^1 e^{-n|y-\frac{1}{n}|/10}dy}{\int_0^1 e^{-n|y-0|/10}dy} \\
&= \frac{\int_0^{1/n} e^{-\frac{n}{10}|y-\frac{1}{n}|}dy + \int_{\frac{1}{n}}^1 e^{-\frac{n}{10}|y-\frac{1}{n}|}dy}{\int_0^{1/n} e^{-\frac{n}{10}y}dy + \int_{\frac{1}{n}}^1 e^{-\frac{n}{10}y}dy} \\
&= \frac{\int_0^{1/n} e^{-\frac{n}{10}y+\frac{1}{10}}dy + \int_{\frac{1}{n}}^1 e^{-\frac{n}{10}y+\frac{1}{10}}dy}{\int_0^{1/n} e^{-\frac{n}{10}y}dy + \int_{\frac{1}{n}}^1 e^{-\frac{n}{10}y}dy} \\
&= \frac{e^{\left(\frac{1}{10}\right)}\left(\int_0^{1/n} e^{-\frac{n}{10}y}dy + \int_{\frac{1}{n}}^1 e^{-\frac{n}{10}y}dy\right)}{\int_0^{1/n} e^{-\frac{n}{10}y}dy + \int_{\frac{1}{n}}^1 e^{-\frac{n}{10}y}dy} \\
&= e^{\frac{1}{10}}
\end{aligned}
$$

Combining the two, we get:

$$
\frac{e^{-n|y-\bar{x}|/10}}{e^{-n|y-\vec{x}'|/10}} \cdot \frac{\int_0^1 e^{-n|y-\vec{x}'|/10}dy}{\int_0^1 e^{-n|y-\bar{x}|/10}dy} \leq e^{\frac{1}{10}} e^{\frac{1}{10}}
$$

1

For the given range of data [0,1], we know that the scale of the Laplace noise added is $\frac{GS_q}{\epsilon} = \frac{\frac{1-0}{n}}{\epsilon}$. We used this information above to calculate the minimum ϵ for this mechanism. We could tune this ϵ to be larger than $1/5 = 0.2$, but the larger the value of ϵ, the lower the privacy as the scale of the Laplace noise would be lower.

If the data domain changes to $[a, b]$, the global sensitivity scales by a factor of $(b - a)$, and the minimum value of ϵ changes proportional to $e^{(b-a)}$. This follows from a new worst case where we have a dataset with all 'a's, and then flip one to a 'b'. The output of the mechanism would have less privacy but higher utility.

(d) Which of these algorithms do you consider to be “best” for releasing a mean and why? (There is not a single “right” answer for this problem.)

I would prefer the $\epsilon - 0$ mechanisms to $\epsilon - \delta$ mechanisms, since we essentially have 0 probability that the output of these mechanisms reveal which of two neighboring datasets was used. I think the first mechanism (i) is easier to implement and interpret than mechanism (iv). Also, given that epsilon of the first mechanism is smaller, the scale of Laplace noise added is less and we have more utility from the released statistic.

R Setup

In [142]:
```
## Setup
rm(list=ls())
library(caret)
library(repr)
library(ggplot2)
# install.packages('matrixStats')
library(matrixStats)
```

In [143]:

```
### Using James's functions

# Random draw from Laplace distribution
#
# mu numeric, center of the distribution
# b numeric, spread
# size integer, number of draws
#
# return Random draws from Laplace distribution
# example:
#
# rlap(size=1000)

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


# Probability density for Laplace distribution
#
# x numeric, value
# mu numeric, center of the distribution
# b numeric, spread
#
# return Density for elements of x
# example:
#
# x <- seq(-3, 3, length.out=61)
# dlap(x)

dlap <- function(x, mu=0, b=1) {
    dens <- 0.5 * b * exp(-1 * abs(x - mu) / b)
    return(dens)
}


# Laplace Cumulative Distribution Function
#
# Determines the probability a draw from a LaPlace distribution is less
 than
#    or equal to the specified value.
#
# x Numeric, the value(s) at which the user wants to know the CDF heigh
t.
# mu Numeric, the center of the LaPlace distribution, defaults to 0.
# b Numeric, the spread of the LaPlace distribution, defaults to 1.
#
# return Probability the LaPlace draw is less than or equal to \code{x}.
# example:
#
# x <- 0
# plap(x)

plap <- function(x, mu=0, b=1) {
```

```
    cdf <- 0.5 + 0.5 * sgn(x - mu) * (1 - exp(-1 * (abs(x - mu) / b)))
    return(cdf)
}


# Quantile function for Laplace distribution
#
# p Numeric, vector of probabilities
# mu numeric, center of the distribution
# b numeric, spread
# return Quantile function
# example:
#
# probs <- c(0.05, 0.50, 0.95)
# qlap(probs)

qlap <- function(p, mu=0, b=1) {
    q <- ifelse(p < 0.5, mu + b * log(2 * p), mu - b * log(2 - 2 * p))
    return(q)
}

# Sign function
#
# Function to determine what the sign of the passed values should be.
#
# x numeric, value or vector or values
# return The sign of passed values
# example:
#
# sgn(rnorm(10))


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

Problem 2:

Evaluating DP Algorithms with Synthetic Data:
Consider a dataset $x \in N^n$ drawn from a Poisson process, which has probability distribution $\Pr[x_i = k] = \frac{10^k e^{-10}}{k!}$ for natural numbers k (where we consider k = 0 to be a natural number and define 0! = 1).

(a) Write a data generating process (DGP) function that generates a dataset $x \in N^n$ according the above Poisson process.

The Poisson process described above has mean and standard deviation, $\lambda = 10$.

In [144]:
```
dgp <- function(n, lambda = 10){
    data = rpois(n, 10)
    return(data)
}
```

(b) Pick one of your differentially private mechanisms from question 1 (generalized to allow for arbitrary choices of ϵ and data range $[a, b]$ as parameters) that releases an estimate of $\bar{x}$. Implement this mechanism as a function which is given a vector of values $x \in [a, b]^n$ and an ϵ that makes a differentially private release. You can assume the sample size n is public knowledge. To apply your mechanism to unbounded data $x \in R^n$, you will have to clamp x to a chosen range $[a, b]$. For simplicity, we will fix $a = 0$ and only consider the effect of varying b.

The mechanism chosen is (i) $M(x) = [\bar{x} + Z]_0^1$ for $Z \sim Lap(2/n)$.

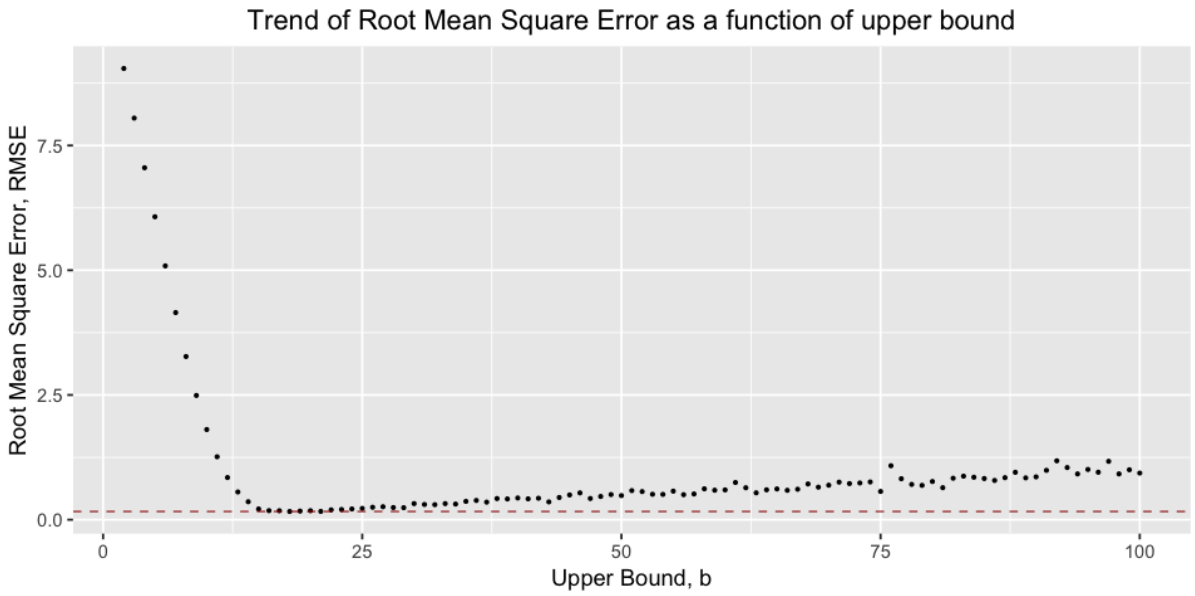
In [145]:
```
## clip variable to a range
clip <- function(x, lower, upper){
    x.clipped <- x
    x.clipped[x.clipped<lower] <- lower
    x.clipped[x.clipped>upper] <- upper
    return(x.clipped)
}

## Differentially private mean release
meanRelease <- function(x, lower, upper, epsilon){
    n <- length(x)

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

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

    return(list(release=DPreleaseclip, true=sensitiveValue))
}
```

c) Recall the discussion on clamping from class; if the range is large, the sensitivity increases, so noise increases and utility drops. However, if you clip the values too aggressively the answer will be biased, and again utility will drop. For $n = 200$ and $\epsilon = 0.5$, plot the root mean squared error as a function of the upper bound b. Identify the approximate optimal value b^* of b for this data distribution.

In [146]:
```
set.seed(208)
n.iter = 100

# set options for upper bound, b
b.options <- seq(from = 2, to = 100, length=99)

# store data
history <- matrix(NA, nrow=length(b.options), ncol=n.iter)  # matrix to
 store results

## run 100 simulations for each value of b
counter = 1
for(iter in 1:n.iter){
    data = dgp(n = 200)
    true.mean = mean(data)
    counter = 1
    for(b in 1:length(b.options)){
        mean.release = meanRelease(x = data, lower = 0, upper = b, epsil
on = 0.5)
        RMSE = postResample(mean.release$release, true.mean)[1]
        history[counter, iter] = RMSE
        counter = counter + 1
    }
}

# calculate average RMSE across all iterations
RMSE.calc = rowMeans(history, na.rm = FALSE, dims = 1)
```

In [147]:
```
RMSE.for.plot = data.frame(b.options , RMSE.calc)
p1 <- ggplot(RMSE.for.plot, aes(x = b.options, y = RMSE.calc)) +
    geom_point(size = 0.5) +
    labs(x = "Upper Bound, b", y = 'Root Mean Square Error, RMSE', title
=
         'Trend of Root Mean Square Error as a function of upper bound')
+
    theme(plot.title = element_text(hjust = 0.5)) +
    geom_hline(aes(yintercept = min(RMSE.calc)), colour="#990000", linet
ype="dashed", size = 0.3 )
```

In [148]:
```
options(repr.plot.width=8, repr.plot.height=4)
p1
```



In [149]:
```
b_star = b.options[match(min(RMSE.calc),RMSE.calc)]
b_star
```

18

From the plot and calculation above, the approximate true value of the upper bound of the data is $b^* = 18$.

(d) Suppose we have an actual (not synthetic) dataset $x \in N^n$ for which we want to release a differentially private mean, and we don't know the underlying distribution of x. Again, we need to select the parameter b and want to do so in a way that minimizes the error. A natural idea is to use a (nonparametric) bootstrap to generate many datasets that are "similar" to x in place of the data-generating process above, and optimize the choice of b as above. Once we find an optimal value b^*, we then do our differentially private release on the dataset x itself. Explain why this approach is not safe in general and may violate differential privacy.

The approach of bootstrapping to get datasets that are similar to x involves some leakage of information from the dataset at hand and might violate differential privacy.

Consider the case where outliers exist in x, and we have a bootstrap that selects these outliers. As a concrete example, consider when x is mostly in the range of 0-20, but there are some outliers out in 40-50 range. Suppose some bootstraps are generated with predominantly outliers. Then, for each of these bootstraps, when we optimize for the choice of b, we will find a dip in RMSE when the upper bound is selected in the 40-50 range as the mean will be closer to the true mean of the data. If we were to select an optimal value of b from just this process, we may end up with one that is too large and reveals information about the dataset x. A differentially private mechanism ensures that releases from neighboring datasets should be within e^{ϵ} of each other, i.e. indistinguishable. The trouble here would be that that releases from neigboring datasets might be wildly different if an outlier varies between the neighbors.

Across all bootstraps, we'll end up with a choice of b that lowers the average MSR, and in effect reveals information about the optimal bound for this specific dataset. So we've overfit our choice of b to this dataset, i.e. it's dataset specific, and revealed information about it.

(e) Propose some alternative methods for determining a good upper bound b for a given sensitive dataset x, while continuing to satisfying the definition of differential privacy.

If we have some prior knowledge about the prior distribution of the data, one way to get a good upper bound would be to create some synthetic data and follow an approach similar to what we did in part (c). Note that this is a parametric approach. A non parametric approach might be to generate synthetic data using DP histograms as we went over in class. If we use external information about the number of bins, construct a DP histogram, and then use the DP histogram to construct synthetic datasets to perform what we did in part (c) to find an upper bound. Note that these approaches don't leak information about the sample at hand.

Problem 3

Regression:
Consider a dataset where each of its n rows is a pair of real numbers (x_i, y_i), where x_i is drawn from a Poisson distribution as in Problem 2, and y_i is then a noisy linear function of x_i, specifically:

$$y_i = \beta x_i + \alpha + \nu_i$$

where $\nu_i \sim N(0, \sigma^2)$ for unknown parameters α, β, σ.

One simple way to estimate the parameters α and β without privacy is by a simple linear regression, using the following estimators:

$$\hat{\beta} = \frac{S_{xy}}{S_{xx}} = \frac{\sum_{i=1}^{2}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{2}(x_i - \bar{x})^2}$$

$$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$$

(a) In class and section, we saw an algorithm and implementation to produce a differentially private version of $\hat{\beta}$. Augment this implementation to also produce a differentially private version of $\hat{\alpha}$, so that the overall method for computing both $\hat{\alpha}$ and $\hat{\beta}$ is ϵ-DP, for an input parameter ϵ. If your algorithms use several basic differentially private algorithms as subroutines, divide the overall privacy budget of ϵ among them evenly. You can clamp both the x_i's and y_i's using reasonable values of your choosing (perhaps following your choice from Problem 2 for x). Describe the reasoning behind your choice in your write up.

In [150]:
```
epsilon.partition <- c(0.25, 0.25, 0.25, 0.25)
names(epsilon.partition) <- c("sxy", "sxx", "xbar", "ybar")
epsilon.partition <- as.data.frame(t(epsilon.partition))
```

In [151]:

```
meanRelease <- function(x, lower, upper, epsilon){
    n <- length(x)

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

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

    return(list(release=DPreleaseclip, true=sensitiveValue))
}

## Differentially private regression slope release
regressionRelease <- function(y, x, ylower, yupper, xlower, xupper,
                              epsilon.sxx, epsilon.sxy, epsilon.xbar, ep
silon.ybar){

    # clip x and y
    x <- clip(x, xlower, xupper)
    y <- clip(y, ylower, yupper)

    # get length of data
    n <- length(x)

    ###  Calculate BETA

    # calc sensitivity of var and cov
    sens.Sxy <- ((xupper-xlower)*(yupper-ylower))
    sens.Sxx  <- ((xupper-xlower)^2)

    # get scale for laplace draws
    scale.Sxy <- sens.Sxy / epsilon.sxy
    scale.Sxx <- sens.Sxx / epsilon.sxx

    # calculate true beta
    beta.true <- sum((x - mean(x))*(y - mean(y))) / sum((x - mean(x))^2)

    # calculate noisy var and cov
    release.Sxy <- sum((x - mean(x))*(y - mean(y)))  + rlap(mu=0, b=scal
e.Sxy, size=1)
    release.Sxx <- sum((x - mean(x))^2)              + rlap(mu=0, b=scal
e.Sxx, size=1)

    # calculate noisy beta
    release.beta <- release.Sxy/release.Sxx

    ### Calculate ALPHA

    # calculate means
    x.bar = meanRelease(x = x, lower = xlower, upper = xupper, epsilon =
epsilon.xbar)
    y.bar = meanRelease(x = y, lower = ylower, upper = yupper, epsilon =
epsilon.ybar)
```

```
    # calculate alpha
    alpha.dp = y.bar$release - x.bar$release * release.beta
    alpha.sensitive =  y.bar$true - x.bar$true * beta.true

    return(list(release.beta = release.beta, true.beta = beta.true,
                release.alpha = alpha.dp, true.alpha = alpha.sensitive))
}
```

(b) Evaluate the performance of your algorithm using a Monte Carlo simulation with synthetic data as in Problem 2 Parts (a)–(c), for the parameters $\alpha = \beta = \sigma = \epsilon = 1$ and $n = 1000$. Measure utility by the mean-squared residuals:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\beta}x_i - \hat{\alpha})^2$$

(The non-private method for simple linear regression described above is chosen to minimize this quantity, hence the term “least-squares regression”.) Plot and compare the distributions of mean-squared residuals you get with your differentially private simple linear regression and with a non-private simple linear regression.

In [152]:
```
### find optimal upper bound for mean of x and mean of y
set.seed(208)
n.iter = 100

# set options for upper bound, b
b.options <- seq(from = 2, to = 100, length=99)

# store data
history.x <- matrix(NA, nrow=length(b.options), ncol=n.iter)
history.y <- matrix(NA, nrow=length(b.options), ncol=n.iter)

## run 100 simulations for each value of b
counter = 1
for(iter in 1:n.iter){

    # draw sample of size 1,000
    x = dgp(n = 1000)
    y = x + rep(1,length(x)) + rnorm(length(x), 0,1)
    # get true means
    true.x = mean(x)
    true.y = mean(y)

    # get RMSE for each option of upper bound, b
    counter = 1
    for(b in 1:length(b.options)){

        # calc and store RMSE for x
        mean.release = meanRelease(x = x, lower = 0, upper = b, epsilon
= epsilon.partition$xbar)
        RMSE = postResample(mean.release$release, true.x)[1]
        history.x[counter, iter] = RMSE

        #calc and store RMSE for y
        mean.release = meanRelease(x = y, lower = 0, upper = b, epsilon
= epsilon.partition$ybar)
        RMSE = postResample(mean.release$release, true.y)[1]
        history.y[counter, iter] = RMSE

        # add one to counter
        counter = counter + 1
    }
}

# calculate average RMSE across all iterations
RMSE.calc.x = rowMeans(history.x, na.rm = FALSE, dims = 1)
RMSE.calc.y = rowMeans(history.y, na.rm = FALSE, dims = 1)
```

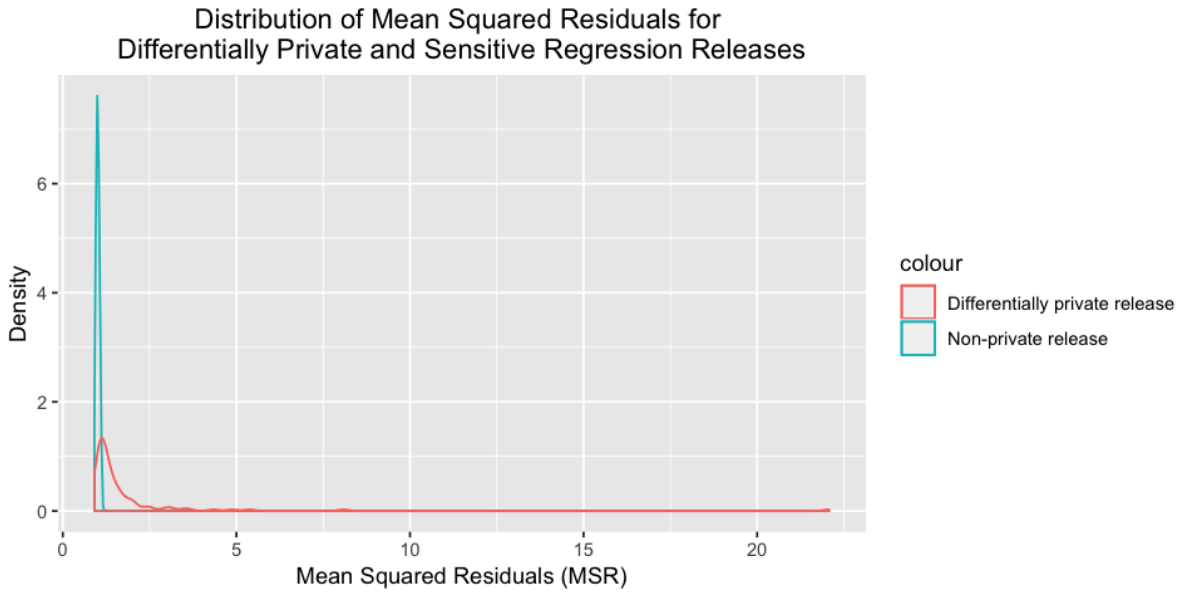
In [153]:
```
b.star.x = b.options[match(min(RMSE.calc.x),RMSE.calc.x)]
b.star.y = b.options[match(min(RMSE.calc.y),RMSE.calc.y)]
cat("optimal upper bound for the mean of x: b star x:" ,
    b.star.x, "\noptimal upper bound for the mean of y: b star y", b.sta
r.y)
```

```
optimal upper bound for the mean of x: b star x: 19
optimal upper bound for the mean of y: b star y 21
```

In [154]:

```
### Monte Carlo Simulations

set.seed(24)
n.iter = 100
len.data = 1000

# store MSR
MSR.dp <- rep(NA, nrow=n.iter)
MSR.sensitive <- rep(NA, nrow=n.iter)

## run 100 simulations for each value of b
counter = 1

for(iter in 1:n.iter){

    # make 1000 data points
    x = dgp(n = len.data)
    y = 1 + x + rnorm(len.data, 0, 1)

    # true values for alpha and beta
    true.beta.full = 1
    true.alpha.full = 1

    # calculate regression release
    reg = regressionRelease(y = y, x = x,
                            ylower = 0, yupper = b.star.y,
                            xlower = 0 , xupper = b.star.x,
                            epsilon.sxx = 0.35,
                            epsilon.sxy = 0.35,
                            epsilon.xbar = 0.15,
                            epsilon.ybar = 0.15)

    release.beta = reg$release.beta
    true.beta = reg$true.beta
    release.alpha = reg$release.alpha
    true.alpha = reg$true.alpha

    # calculate MSR and store
    MSR.dp. = (1/len.data)*(sum((y - release.beta * x - release.alpha)^2
))
    MSR.sensitive. = (1/len.data)*(sum((y - true.beta.full * x - true.al
pha.full)^2))

    # store MSR values
    MSR.dp[iter] = MSR.dp.
    MSR.sensitive[iter] = MSR.sensitive.
}
```

In [155]:
```
## plot MSR for DP release and sensitive release

MSR.to.plot <- data.frame(MSR.dp, MSR.sensitive)

p2 <- ggplot(MSR.to.plot) +
    geom_density(aes(x=MSR.to.plot$MSR.sensitive, color = 'Non-private r
elease'))+
    geom_density(aes(x=MSR.to.plot$MSR.dp, color = 'Differentially priva
te release'))+
    labs(x = 'Mean Squared Residuals (MSR)', y = 'Density',
         title = 'Distribution of Mean Squared Residuals for \nDifferent
ially Private and Sensitive Regression Releases') +
    theme(plot.title = element_text(hjust = 0.5))
p2
```



The distribution of mean-squared residuals (MSR) for the differentially private simple linear regression is more spread out than that of the non-private simple linear regression. This makes sense as we expect the differentially private release with noisy estimates to predict y's that deviate more from the true y values, so it's reasonable to expect that for some iterations, due to the randomness of the noise, we ended up with very noisy predictions. In contrast, the distribution MSRs of the non-private release is much tighter and centered around 1 (mean 0.9986). This makes sense as we introduced the the error term in the regression with standard deviation of 1.

(d) Now, run experiments to see if there is a different partition of your privacy budget ϵ in your algorithm that yields better utility. Use a grid search to explore different partitions of ϵ and see if you find one that is convincingly better (in terms of mean-squared residuals) than an equal partition under the given data distribution. Show and explain your results.

In [156]:

```
epsilon.options <- seq(0.1,0.9,0.05)

# df with all combinations of epsilon
epsilon.comb <- as.data.frame(expand.grid(list(sxy=epsilon.options,
                                               sxx=epsilon.options,
                                               xbar=epsilon.options,
                                               ybar=epsilon.options) ))
epsilon.comb['sum'] <- rowSums(epsilon.comb)
unique.epsilon.comb <- epsilon.comb[epsilon.comb$sum == 1,]
number.of.epsilons.to.try <- nrow(unique.epsilon.comb)
rownames(unique.epsilon.comb) <- seq(length=nrow(unique.epsilon.comb))
```

In [157]:
```
### Experiment to choose best partition of privacy budget

# true values for alpha and beta
true.beta = 1
true.alpha = 1
len.data = 1000
n.iter = 100

# store MSR
MSR.epsilon <- matrix(NA, nrow=nrow(unique.epsilon.comb), ncol = n.iter)


for (iter in 1:n.iter){
#     set.seed(iter)
    x = dgp(n = len.data)
    y = 1 + x + rnorm(len.data, 0, 1)

    ## run through the combinations of epsilon
    counter = 1
    for(row in 1:nrow(unique.epsilon.comb)){

        use.this.epsilon <- unique.epsilon.comb[row,]

        # calculate regression release
        reg = regressionRelease(y = y, x = x,
                                ylower = 0, yupper = b.star.y,
                                xlower = 0 , xupper = b.star.x,
                                epsilon.sxx = use.this.epsilon$sxx,
                                epsilon.sxy = use.this.epsilon$sxy,
                                epsilon.xbar = use.this.epsilon$xbar,
                                epsilon.ybar = use.this.epsilon$ybar)

        release.beta = reg$release.beta
        true.beta = reg$true.beta
        release.alpha = reg$release.alpha
        true.alpha = reg$true.alpha

        # calculate MSR and store
        MSR.dp. = (1/len.data)*(sum((y - release.beta * x - release.alph
a)^2))
        MSR.sensitive. = (1/len.data)*(sum((y - true.beta * x - true.alp
ha)^2))
        # store MSR values
        MSR.epsilon[row, iter] = MSR.dp.
    }
}
```

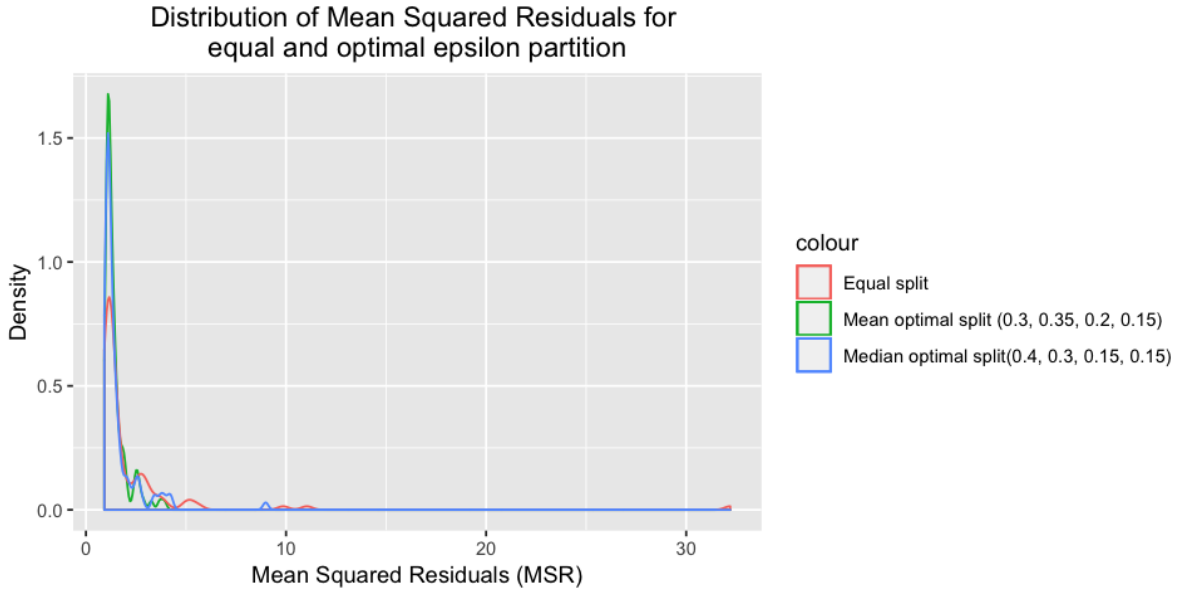
In [158]:
```
unique.epsilon.comb['mean.MSR']<- rowMeans(MSR.epsilon)
unique.epsilon.comb['median.MSR']<- rowMedians(MSR.epsilon)
# unique.epsilon.comb['median.MSR']<- rowMedians(MSR.epsilon)
```

In [160]:
```
unique.epsilon.comb[which.min(unique.epsilon.comb$mean.MSR),]
unique.epsilon.comb[which.min(unique.epsilon.comb$median.MSR),]
```

	sxy	sxx	xbar	ybar	sum	mean.MSR	median.MSR
100	0.35	0.35	0.15	0.15	1	1.437777	1.205588

	sxy	sxx	xbar	ybar	sum	mean.MSR	median.MSR
99	0.4	0.3	0.15	0.15	1	1.597191	1.173492

In [166]:
```
## plot MSR for DP release and sensitive release
min.eps.dist = MSR.epsilon[as.numeric(rownames(unique.epsilon.comb[whic
h.min(unique.epsilon.comb$mean.MSR),])),]
equal.eps.dist = MSR.epsilon[as.numeric(rownames(unique.epsilon.comb[(un
ique.epsilon.comb$sxx == 0.25) & (unique.epsilon.comb$sxy == 0.25) & (un
ique.epsilon.comb$xbar == 0.25),]
)),]
median.min.eps.dist = MSR.epsilon[as.numeric(rownames(unique.epsilon.com
b[which.min(unique.epsilon.comb$median.MSR),])),]

MSR.for.eps <- data.frame(min.eps.dist, equal.eps.dist, median.min.eps.d
ist)

p3 <- ggplot(MSR.for.eps) +
    geom_density(aes(x=MSR.for.eps$min.eps.dist, color = 'Mean optimal s
plit (0.3, 0.35, 0.2, 0.15)'))+
    geom_density(aes(x=MSR.for.eps$equal.eps.dist, color = 'Equal split'
)) +
    geom_density(aes(x=MSR.for.eps$median.min.eps.dist, color = 'Median
 optimal split(0.4, 0.3, 0.15, 0.15)')) +
    labs(x = 'Mean Squared Residuals (MSR)', y = 'Density',
         title = 'Distribution of Mean Squared Residuals for \nequal and
optimal epsilon partition') +
    theme(plot.title = element_text(hjust = 0.5))
p3
```



The search space was defined with 427 options for the partition of epsilon. 100 simulations were performed and the median and mean of MSR were used to find the optimal split (taking the min mean/median MSR in each case).

After performing 100 simulations to check which partition of epsilon would yield the lowest mean MSE, the optimal split was found at partitioning $\epsilon = 1$ as $\epsilon_{S_{xy}} = 0.3$, $\epsilon_{S_{xx}} = 0.35$, $\epsilon_{\bar{x}} = 0.25$, $\epsilon_{\bar{y}} = 0.15$. The optimal split based on median MSR was found at partitioning $\epsilon = 1$ as $\epsilon_{S_{xy}} = 0.4$, $\epsilon_{S_{xx}} = 0.3$, $\epsilon_{\bar{x}} = 0.15$, $\epsilon_{\bar{y}} = 0.15$.

The distribution above of the optimal "mean" split regression release MSR clearly has higher density for lower values of MSR as compared to the distribution of the optimal "median" split regression release MSR and equal split regression release MSR. I would use the values associated with the mean release as a split for epsilon. Clearly, these are not evenly split, and we can see that distributing more of the epsilon budget to β yields lower MSR values.

Problem 4

DP vs. Reconstruction Attacks:

Suppose $M : \{0,1\}^n \to \mathscr{Y}$ is an (ϵ, δ)-DP mechanism and $A : \mathscr{Y} \to \{0,1\}^n$ is an adversary that is trying to reconstruct the sensitive bits in the dataset $x \in \{0,1\}^n$ from the output $M(x)$. Suppose the dataset is a random variable $X = (X_1, X_2, \ldots, X_n)$ consisting of n iid draws from a Bernoulli(p) distribution, for a known value of p. Prove that the expected fraction of bits that the adversary successfully reconstructs is not much larger than the trivial bound of $\max\{p, 1-p\}$, which can be achieved by guessing the all-zeroes or all-ones dataset). Specifically,

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

For an arbitrary adversary, we have

Expanding expectation as average of indicator r.v.'s,

$$E\Big[\#\{i \in [n] : A(M(X))_i = X_i\}/n] = \sum_{i=1}^{n} Pr[A(M(X_i, X_{-i})=X_i]\frac{1}{n}$$

By ϵ-δ definition (replace i^{th} row with 0), post processing of M with A,

$$\leq \sum_{i=1}^{n} \frac{e^{\epsilon}}{n} Pr[A(M(0, X_{-i})=X_i] + \delta$$

Given best attack strategy,

$$= \frac{e^{\epsilon}}{n} max\Big(\sum_{i=1}^{n} Pr[A(M(0, X_{-i})=1$$

$$= max\Big(\frac{e^{\epsilon}}{n} \sum_{i=1}^{n} p, \frac{e^{\epsilon}}{n} \sum_{i=1}^{n} (1-p)\Big)$$

$$= e^{\epsilon} max\{p, 1-p\} + \delta$$

Hence, the expected fraction of bits that the adversary successfully reconstructs is not much larger (within e^{ϵ}) than the trivial bound of $\max\{p, 1-p\}$.

5. Final Project Next Step:

Karina and I will get this in tomorrow so we can give it more thought. Thanks! :)