PRIVACY TOOLS FOR SHARING RESEARCH DATA
Releasing a Differentially Private Variance-covariance Matrix

Connor Bain*, Vito D'Orazzio, Or Sheffet, Salil Vadhan
Harvard University

Abstract

With online research data repositories like Harvard's **Dataverse Network** quickly gaining prominence, considering the security issues of sharing sensitive research data is of utmost importance. The question becomes: *how do we allow access to research data without compromising the privacy of the individual's in the dataset?* As part of the **Privacy Tools for Sharing Research Data** project, we propose using *differential privacy* to protect sensitive data. By integrating differential privacy into the statistical analysis software **Zelig**, we can release summary statistics to users without compromising privacy. In this paper, we discuss methods of releasing a differentially private variance-covariance matrix. In addition to implementing and analyzing two different differentially private mechanisms, we also use the covariance matrix in order to perform ordinary least regressions. We end with an implementation of the Gaussian mechanism to release a differentially private covariance matrix.

CONTENTS

*Work done as part of the School of Engineering and Applied Sciences REU program.

1 Introduction

One of the most basic tools in a statistician's toolbox is ordinary least squares regression. By minimizing the sum of squared vertical distances between the observed responses in the dataset and the responses predicted by the linear approximation. The resulting estimator can be easily transformed into a easy-to-read linear equation. Being able to perform these simple linear regressions in a differentially private manner would be a huge boon to data privacy.

1.1 Covariance

There are many ways to measure the relationship between variables in a dataset but one of the most prolific is the *covariance*. Informally, covariance is a measure of how much two random variables change together. If the greater values of one variable mainly correspond with the greater values of the other variable, and the same holds for the smaller values, i.e., the variables tend to show similar behavior, the covariance is positive. In the opposite case, when the greater values of one variable mainly correspond to the smaller values of the other, i.e., the variables tend to show opposite behavior, the covariance is negative. The sign of the covariance therefore shows the tendency in the linear relationship between the variables. The magnitude of the covariance is not easy to interpret. The normalized version of the covariance, the correlation coefficient, however, shows by its magnitude the strength of the linear relation.

Definition 1. *The* Pearson product-moment correlation coefficient *is a measure of the linear correlation (dependence) between two variables X and Y, giving a value between +1 and -1 inclusive, where 1 is total positive correlation, 0 is no correlation, and -1 is total negative correlation.*

A distinction must be made between (1) the covariance of two random variables, which is a populxation parameter that can be seen as a property of the joint probability distribution, and (2) the sample covariance, which serves as an estimated value of the parameter.

Definition 2. *The* covariance *between two jointly distributed real-valued random variables x and y with finite second moments is defined as:* $\sigma(x, y) = E[(x - E[x])(y - E[y])] = E[xy] - E[x]E[y]$ *where* $E[x]$ *is the expected value of* x.

Fact 1. (symmetry) $\sigma(x, y) = E[(x - E[x])(y - E[y])] = E[(y - E[y])(x - E[x])] = \sigma(y, x)$

Fact 2. (variance) $\sigma(x, x) = E[x^2] - E[x]E[x] = \sigma^2$

Now we can generalize this idea of covariance given the d different attributes of a dataset. Consider $\mathbf{X}$ to be a column vector $\begin{bmatrix} X_1 \\ \vdots \\ X_d \end{bmatrix}$ where each entry is a random variable with finite variance.

Definition 3. *The* covariance matrix (Σ), *is the matrix whose* (i, j) *entry is the covariance* $\Sigma_{ij} = cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$.

$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1-\mu_1)(X_1-\mu_1)] & \mathrm{E}[(X_1-\mu_1)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_1-\mu_1)(X_n-\mu_n)] \\ \mathrm{E}[(X_2-\mu_2)(X_1-\mu_1)] & \mathrm{E}[(X_2-\mu_2)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_2-\mu_2)(X_n-\mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n-\mu_n)(X_1-\mu_1)] & \mathrm{E}[(X_n-\mu_n)(X_2-\mu_2)] & \cdots & \mathrm{E}[(X_n-\mu_n)(X_n-\mu_n)] \end{bmatrix}$$

1.2 Ordinary Least Squares

One of the main motivators behind calculating a variance-covariance matrix is that it contains all of the necessary information to perform ordinary least squares regressions. *Ordinary least squares* (OLS) or linear least squares is a method for estimating the unknown parameters in a linear regression model. OLS minimizes the sum of squared vertical distances between the observed responses in the dataset and the responses predicted by the linear approximation. The resulting estimator can be expressed by a simple formula, especially in the case of a single regressor on the right-hand side.

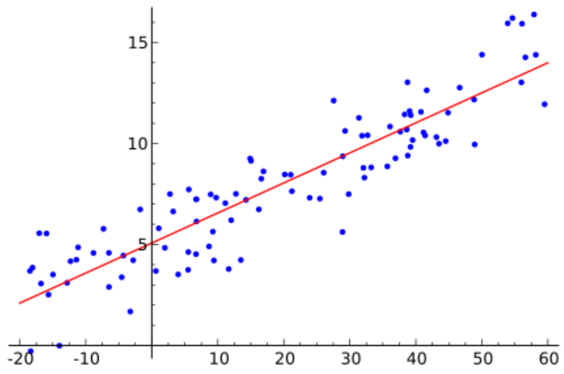


Figure 1: In the above example, the data are the blue points and the red line is the OLS estimator

There are many assumptions that the ordinary least squares estimator operates under including correct specification, strict exogeneity, lack of linear dependence, and normality of errors. For a discussion of these assumptions, we recommend the reader visit a statistics reference. The pertitent assumption for our purposes deals with the fact that our data is not random. Our datasets have fixed rows and attributes on which the algorithms operate. We only analyze the algorithms performance assuming the data are i.i.d. samples from some random variables.

Suppose our data consists of n observations $\{y_i, x_i\}_{i=1}^n$. Every observation has a scalar response y_i and a vector of p regressors x_i. In a linear regression model the response variable is a linear function of the regressors: $y = x_i^T \beta + \varepsilon_i$, where β is a $p \times 1$ vector of unknown parameters; ε_i's are unobserved scalar random variables (errors) which account for the discrepancy between the actually observed reopens y_i and the "predicted outcomes" $x_j^T \beta$; and ^{T} denotes matrix transpose, so that $x^T \beta$ is the dot product between the vectors x and β. This model can also be written in matrix notation as $y = X\beta + \varepsilon$, where y and ε are $nx1$ vectors, and X is an nxp matrix of regressors.

Definition 4. *The* residual sum of squares *(RSS) is a measure of the overall model fit:*

$$S(b) = \sum_{i=1}^{n} (y_i - x_i' b)^2 = (y - Xb)^T (y - Xb).$$

In the above definition, ^{T} represents matrix transpose and the value of b which minimizes the sum is called the *OLS estimator* for β. The function $S(b)$ is a quadratic function with positive-definite Hessian. This means it possesses a unique global minimum at $b = \hat{\beta}$ which is given by an explicit formula:

$$\hat{\beta} = \arg\min_{b \in \mathbb{R}^p} S(b) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i x_i' \right)^{-1} \cdot \frac{1}{n} \sum_{i=1}^{n} x_i y_i$$

After we estimate β, the *fitted values* from the regression will be $\hat{y} = X\hat{\beta}$. Using these fitted values, we can create a matrix of *residuals*, $\hat{\varepsilon} = y - X\hat{\beta}$. Then, using the residuals, we can estimate the value of σ^2.

$$s^2 = \frac{S(\hat{\beta})}{n-p}$$

The numerator, $n - p$, represents the *statistical degrees of freedom.* The quantity, s^2 is the OLS estimate for σ^2.

Below is sample linear model output similar to most popular statistical packages:

Table 1: Sample Model Results

	Estimate	Std. Error	t value	Pr[$> t$]	
(int)	-23054.102	154.580	-149.1	<2e-16	***
X_1	275.253	2.218	124.1	<2e-16	***
X_2	4332.877	11.560	374.8	<2e-16	***

The *Estimate* column gives the least squares estimates of coefficients β_j. The *Std. Error* column shows standard errors of each coefficient estimate $\widehat{\text{s.e}}(\hat{\beta}_i) = \sqrt{s^2(X'X)^{-1}_{i,i}}$. The *t-statistic* and *p-value* (*Pr*) are testing if any coefficients might be equal to zero. The t-statistic is calculated in the usual manner:

$$t_i = \hat{\beta}_i / \widehat{\text{s.e}}(\hat{\beta}_i)$$

If the errors ε follow a normal distribution, t follows a Student-t distribution. Under weaker conditions, t is asymptotically normal. Large values of t indicate that the null hypothesis can be rejected and that the corresponding coefficient. The p-value column represents the results of the hypothesis test as a significance level. Conventionally, p-values smaller than 0.05 are taken as evidence that the population coefficient is nonzero.

Although OLS is not always the best way of performing linear regression, because the variance-covariance matrix contains all the necessary information, it is perfect for our purposes.

1.3 Regressions from the Variance-covariance Matrix

From your d random variables, pick some size k subset $\mathbf{X}$ such that $d \geq k+1$. Then pick a random variable $X_Y \notin \mathbf{X}$. In order to perform a linear regression of the form, $X_Y = \beta_1 * X_1 + \beta_2 * X_2 + \cdots + \beta_k * X_k + \beta_{k+1} * 1$ using only the variance-covariance matrix we have to perform what is called a *sweep operation.*

When performing a sweep operation, one must operate only on the variables that are being used on the regression. That is, when performing a regression, we take a subset of the overall variance-covariance matrix. With our added intercept column and row, the overall variance-covariance matrix is now a $(d+1) \times (d+1)$ matrix. If we plan to run a regression using k x-variables to predict a y-variable, then we only need the k rows and columns of those x-variables from $D^T D$ matrix in addition to the single row and column representing our intercept value. Call this new sub-matrix, $X^T X$. We will also need to set aside the column of the $D^T D$ matrix representing the y-variable, call it X_Y.

Now we are ready to perform an ordinary linear regression using our variance-covariance matrix. First consider the prediction of the true β values. Using ordinary linear regression, there is a simple analytical solution for the coefficient predictions.

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

We can easily calculate $(X^T X)^{-1}$ since this is just the inverse of the covariance matrix shifted toward the means. $X^T y$ on the other hand is a slightly different story. But notice this is simply the going to be a $(k+1)x1$ column vector where the i-th value is inner product $< X_i, X_Y >$. This corresponds to the X_Y column in our original VCV matrix minus the Y-th element that represents $< X_Y, X_Y >$. Let this k length column vector be Y Thus, to calculate $\hat{\beta}$, in this case a $(k+1)x1$ matrix, we use the formula $\hat{\beta} = (X^T X)^{-1} Y$.

The next step is to calculate the residual sum of squares from our $\hat{\beta}$ predictions. The usual formula is $RSS(b) = \sum_{i=1}^{n} (y_i - x_i' b)^2 = (y - Xb)^T (y - Xb)$, but it turns out all of this information is already calculated in the VCV matrix.

$$\begin{aligned} S(\beta) &= \|y - \hat{\beta} X\|^2 \\ &= \|y\|^2 - 2\langle \beta X, y\rangle + \langle \hat{\beta} X, \hat{\beta} X\rangle \\ &= \|y\|^2 - 2\langle \textstyle\sum_{i=1}^{k} \hat{\beta}_i X^{(i)}, y\rangle + \langle \textstyle\sum_{i=1}^{k} \hat{\beta}_i X^{(i)}, \textstyle\sum_{j=1}^{k} \hat{\beta}_j X^{(j)}\rangle \end{aligned}$$

$$= \|y\|^2 - 2\sum_{i=1}^{k} \hat{\beta}_i \langle X^{(i)}, y\rangle + \sum_{i=1}^{k}\sum_{j=1}^{k} \hat{\beta}_i\hat{\beta}_j \langle X^{(i)}, X^{(j)}\rangle$$

$$= \langle X_Y, X_Y\rangle - 2\langle \hat{\beta}, Y\rangle + \hat{\beta}^T(D^T D)\hat{\beta}$$

This allows us to calculate the OLS estimator for σ^2 just as before: $s^2 = \frac{S(\hat{\beta})}{n-(k+1)}$. This is all of the necessary information to perform an OLS estimation.

1.4 Zelig

Zelig is a general purpose statistical package built on R, with a large and diverse array of methods from all theories of inference. All documentation follows the same style and mathematical notation so that if you understand one method, you'll be able to learn any other one easily. Zelig commands for all models involve the same simple three commands with the same syntax. Zelig marshals the power of R by harnessing the cacophony R's different styles, examples, syntax, documentation, and programming logic, all with a simple, command, and easy-to-use interface. It also adds value to existing models by providing tools to make them easy to interpret and present. Developers can add R packages to be run from Zelig by writing a few bridge functions.

Unfortunately, data analyzed on Zelig is totally open to attacks by malicious users. The overarching goal of the Privacy Tools for Sharing Research Data project is to create a version of Zelig that preserves overall (ε, δ)-differential privacy. In order to do this, every summary statistic, graphic, or regression generated by Zelig must be computed using a differentially private algorithm. This report focuses on releasing the variance-covariance matrix of attributes in a dataset in a differentially private manner.

2 Implementation

We decided to implement two different algorithms to release a differentially private variance-covariance matrix. The first is method based off of the Gaussian Mechanism proposed by Dwork, Talwar, Thakurta, and Zhang. The second utilizes the Johnson-Lindenstrauss Transform and was proposed by Blocki, Blum, Datta, and Sheffet. The algorithms are in fact quite different from each other, each not actually producing the variance-covariance matrix per se. This affects the way we test each mechanism, but we will discuss this later. In this section, we use A to represent the matrix representation of a dataset where each row corresponds to a sample and each column represents a different data attribute.

2.1 Analyze Gauss

This method, proposed by Dwork et. al in 2014, utilizes the Gaussian Mechanism to perturb the true variance-covariance matrix using random noise drawn from a normal distribution. Rather than producing a variance-covariance matrix, it produces a *s*econd-moment matrix of sorts. However, we can treat this as equivalent to a variance-covariance matrix in the regression process.

Let $f : A \to \mathbb{R}^p$ be a vector-valued function operating on databases. The l_2-*senstitivity* of f, denoted Δf, is the maximum over all pairs A, A' of neighboring datasets of $\|f(A) - f(A')\|_2$. The Gaussian mechanism adds independent noise drawn from a Gaussian with mean zero and standard deviation slightly greater than $(\Delta f)\log(1/\delta)/\varepsilon$ to each element of its output.

Theorem 1. (Gaussian mechanism) *Let $f : A \to \mathbf{R}^p$ be a vector-valued function. Let $\tau = \Delta f\sqrt{2\log(1.25/\delta}/\varepsilon$. The Gaussian mechanism, which adds independently drawn random noise distributed as $\mathcal{N}(0, \tau^2)$ to each output of $f(A)$, ensures (ε, δ)-differential privacy.* [3]

We are interested in the function $f(A) = A^T A$, which may be viewed as n^2-dimensional vector. This method assumes that $\|a_i\|_2 \le 1$ which means that the sensitivity of f is at most one.

The algorithm itself is quite simple (see algorithm 1). The fact that Algorithm 1 provides (ε, δ)-differential privacy is immediate from Theorem 1, using the fact that the l_2-sensitivity of $f(A) = A^T A$, when viewed as an n^2 dimensional vector, is 1.

Algorithm 1 The Gaussian Mechanism: releasing the covariance matrix privately

```
Input: matrix $A \in \mathbf{R}^{nxd}$ with $\|a_i\|_2 \leq 1$ and privacy parameters $\varepsilon, \delta > 0$.
1: $E \in \mathbf{R}^{dxd}$ be a symmetric matrix where the upper triangle (including the diagonal) is i.i.d. samples from $\mathcal{N}(0, \Delta^2_{\varepsilon,\delta})$ and each lower triangle entry is copied from its upper triangle counterpart.
2: Output $\hat{C} \leftarrow A^T A + E$
```

In order to release a differentially private variance-covariance matrix for a real world dataset, we must be given the following metadata for a dataset in a differentially private manner: the number of attributes; the min, mean, and max of each attribute; the number of samples; and our privacy parameters. Rather than ensuring a global sensitivity of 1, we change the algorithm to ensure a global sensitivity of $\sqrt{d}$ by adding a factor of $\sqrt{d}$ to the standard deviation of the error distribution we draw from. But in order to ensure this new global sensitivity, we must first scale the data. We scale each attribute to a $[0, 1]$ range in the usual way: $\forall i \in \{1, 2, \dots, d\}, x' = \frac{x - min_i}{min_i - max_i}$. This means that that given two neighboring databases, the l_2-sensitivity of a row is at most $\sqrt{d}$.

Now we run the algorithm on our database D, first computing $D^T D$ and then perturbing it with a symmetric matrix of i.i.d draws from a the previously discussed Gaussian distribution. The last step is to "upscale" the released covariance matrix. Let $\hat{C}_{i,j}$ represent the i, jth element of the released variance covariance matrix. We scale each element by the following formula:

$$\hat{C}'_{i,j} = (\hat{C}_{i,j} * range_i * range_j) - (min_i * min_j) + (min_j * mean_i) + (min_i * mean_j)$$

The result is a $\hat{C}'$, an (ε, δ)-differentially private approximation to the variance-covariance matrix that the sweep operator can be applied to in order to perform any of the 2^d possible ordinary linear regressions.

2.2 The Johnson Lindenstrauss Transform

The JLT mechanism does not produce a variance-covariance matrix, rather, it produces a so-called $t$est-matrix. The authors do not claim that the released matrix and $D^T D$ have comparable eigenvectors or singular values, but rather that if you believe A has high directional variance along some direction x, then you can test your hypothesis on the released matrix and (w.h.p.) get the good approximated answer.

For now, we have decided to forgo work on the JLT.

3 Experimentation

In alignment with regression results from most statistical packages, we assign an easy to read significance code to each $\hat{\beta}$. The codes are assigned in the following manner:

If the $Pr[\hat{\beta}_i] < 0.001$, the significance code is '***'
Else if $0.001 < Pr[\hat{\beta}_i] < 0.01$, '**'
Else if $0.01 < Pr[\hat{\beta}_i] < 0.05$, '*'
Else if $0.05 < Pr[\hat{\beta}_i] < 0.1$, '.'
Else, the $\hat{\beta}$ receives a code of ' '

This means that when we perform a regression, we release the results in the following format:

Table 2: Linear Model Results

	Estimate	Std. Error	t value	Pr[$> t$]	
(int)	-0.4498	3.1555	-0.1426	0.8866	
X_1	2.000	3e-04	6336.4951	<2e-16	***
X_2	1e-04	3e-04	0.2146	0.8301	

3.1 Data Generation

In order to evaluate the utility of the Analyze Gauss algorithm, we ran it on two different types of data: simulated and real. The simulated data was generated in the following manner.

1. Draw $d-1$ random means from a uniform distribution over the interval [100, 10000].
2. Draw $d-1$ random standard deviations from a uniform distribution over the interval [0, 100].
3. Generate $d-1$ columns of data, each having n rows, from $\mathcal{N}(\mu_i, \sigma_i)$.
4. Generate the dth column of data by creating a linear relationship with the 1st column (e.g. $X_d = 3X_1$).
5. Add random noise to the dth column from a uniform distribution over $\{-1, 1\}$.

One of the problems with this data generation method is that by directly manipulating the linear relationship, we are artificially inflating the range of the Y variable. For example, if we force $C = 1000$ and $1000X_1 = Y$, the range of Y is now 1000 times the range of X_1. In future experiments, we plan to take care of this by directly modifying the covariance between the two variables rather than the linear relationship between the two.

The "real" data was taken from the Public Use Microdata Sample (PUMS) released in 2014 by the US Census Bureau. [2] In order to utilize this data we removed all non-numeric data attributes and also disregarded any numerical columns where computing the variance or covariance does not make sense (e.g. zip code).

3.2 Calculating a Differentially Private Variance-covariance Matrix

In our first experiments, we focused on sanity tests for the algorithm. Using the PUMS data, we estimate the variance of the first variable and the covariance between the first and second attributes. The actual interpretation of these calculations is not important for this first round of tests, we are simply trying to verify that the algorithm worked as it claimed. In these tests, we use $\varepsilon = 1/100$ and $\delta = 2^{-16}$. For reference, the number of attributes was $d = 3$ and the number of samples was $n = 1,223,992$. (Note: remember that if $d = 3$, our covariance matrix will be a 4×4 matrix since we are also estimating the intercept term of the linear model)

$$\begin{bmatrix} 30848265 & 30876220 & -45843185 & 10544042 & 22921658 \\ 30876220 & 30904300 & -45884730 & 10553598 & 22942431 \\ -45843185 & -45884730 & 68128866 & -15669409 & -34064482 \\ 10544042 & 10553598 & -15669409 & 3604102 & 7834727 \\ 22921658 & 22942431 & -34064482 & 7834727 & 17032390 \end{bmatrix}$$

Figure 2: True variance-covariance matrix

In figures 4 and 5, the green vertical line represents the true value of the statistic, calculated in a non-differentially private manner, from the original data. Each histogram contains 1000 counts. We do this by running the Gaussian mechanism 1000 separate times to generate 1000 different trials. As expected, the distribution of noisy statistics is centered around the true value of the statistic. Having verified the sanity of our algorithm, we moved on to analyzing the noise added by the algorithm. We focused on two types error measurement, the spectral norm and the Frobenius norm. In both cases, we apply the norm to the matrix of noise values the algorithm draws from the appropriate Normal distribution.

$$\begin{bmatrix} 30848261 & 30876213 & -45843176 & 10544045 & 22921660 \\ 30876213 & 30904289 & -45884733 & 10553592 & 22942446 \\ -45843176 & -45884733 & 68128911 & -15669387 & -34064507 \\ 10544045 & 10553592 & -15669387 & 3604103 & 7834730 \\ 22921660 & 22942446 & -34064507 & 7834730 & 17032375 \end{bmatrix}$$

Figure 3: Noisy variance-covariance matrix

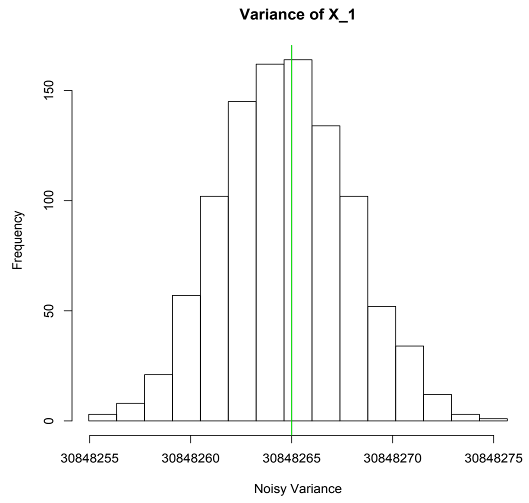


Figure 4: Noisy variance of attribute 1

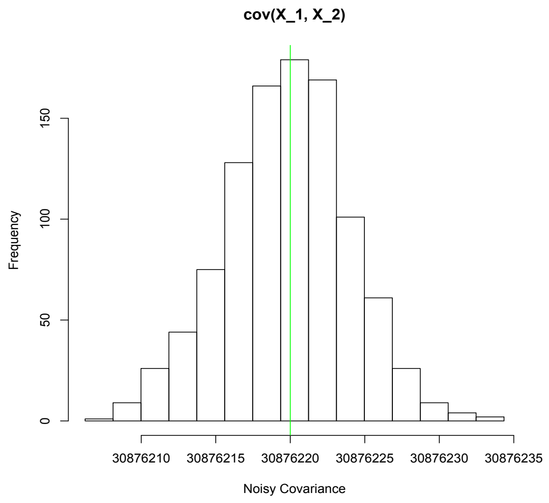


Figure 5: Noisy covariance of attributes 1 and 2

Definition 5. *The* spectral norm *is the natural norm induced by the l^2-norm. Let A^H be the conjugate transpose of the square matrix A, then the spectral norm is defined as the square root of the maximum eigenvalue of $A^H A$, i.e. $\|A\|_2 = (maximum\ eigenvalue\ of\ A^H A)^{1/2}$.*

Notice that since the variance-covariance matrix is positive semi-definite, $D^H D = D^T D$.

Definition 6. *The* Frobenius norm *is a matrix norm of an $m \times n$ matrix A defined as the square root of the sum of the absolute squares of its elements,*

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}.$$

In both figure 6 and 7, we have 1000 different norm calculations from 1000 independently generated noise matrices (same experiments as above).

Then, using the spectral norm as a base measurement of error, we performed three more sanity experiments. Figures 8, 9, and 10 show the results of these experiments with each point representing the mean spectral norm of the noise matrix for that value of d or n over 1000 runs of the algorithm. The error bars represent the minimum and maximum spectral norm of the error matrix over those 1000 runs. In the first experiment, we gradually increased the value of ε while holding all other values static. The results, shown in figure 8, are as expected: as ε increases, the overall noise added decreases. This makes intuitive sense. The less privacy we guarantee, the less we information we have to obscure.

In the second two experiments, we gradually increase the values of n and d, and measure the spectral norm of the resulting noise matrix.

After verifying the sanity of the algorithm, we moved on to using the noisy covariance matrix to perform

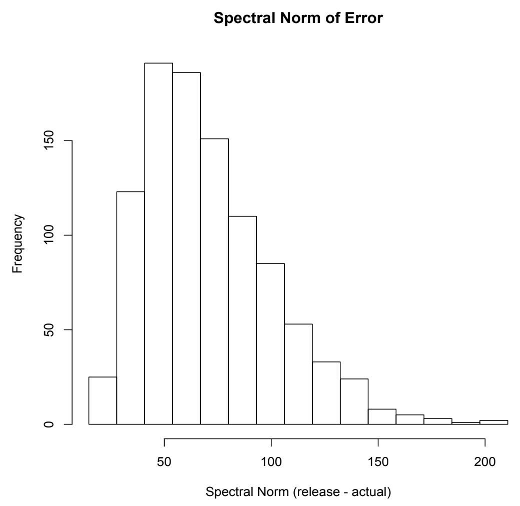


Figure 6: Spectral norm of the noise matrix

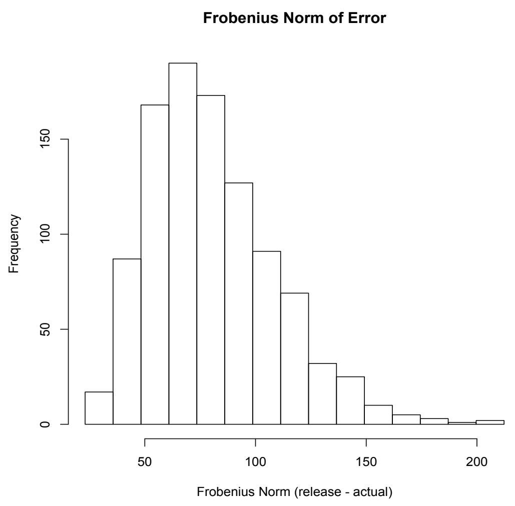


Figure 7: Frobenius norm of the noise matrix

ordinary least squares regression. Using the sweep operator previously discussed, we operate on the covariance matrix to estimate the coefficients of the linear model (see algorithm 2). Through this method we can calculate any of the 2^d possible linear regressions from D^TD.

Algorithm 2 The sweep operator

Input: matrix $D^TD \in \mathbf{R}^{d+1\times d+1}$, X the list of regressors, Y the attribute to predict.

```
1: Y.DTD ← the column in the $D^TD$ matrix that corresponds to the variable specified in $Y$.
2: Y.Y ← the variance from $D^TD$ of the variable specified in $Y$.
3: Y.DTD ← Y.DTD with the $Y$-th element removed
4: $D^TD \leftarrow$ sub matrix of $D^TD$ that corresponds to the regressors in $X$
5: $\hat{\beta} \leftarrow (D^TD)^{-1} \times$ Y.DTD
6: $S(\hat{\beta}) \leftarrow \text{Y.Y} - 2 * (\hat{\beta}^T * \text{Y.DTD}) + \hat{\beta}^T * D^TD * \hat{\beta}$
7: $s^2 \leftarrow S(\hat{\beta})/(n - \|X\|)$
8: std. error$_i \leftarrow \sqrt{(D^TD)^{-1}[i,i] * s^2}$
9: $t_i \leftarrow \hat{\beta}_i/$std. error$_i$
10: $\Pr[t_i] \leftarrow$ probability of that t-value from a student-t distribution
11: Output $\hat{\beta}, S(\hat{\beta}), s^2,$ std. error$, t,$ and Pr
```

The usual way of measuring the "accuracy" of a regression is the R^2 value or coefficient of determination. Unfortunately, there is currently no mechanism for calculating this in a differentially private manner. For now, we approach this problem in two different manners. In the first method, we select one coefficient in particular and find the distance between the noisy, differentially private coefficient and the true coefficient as estimated by the `lm` package in R (we call it the coefficient distance). In the second method, we compare all coefficients simultaneously through the l^2-norm (we call this beta distance): $\|\beta - \hat{\beta}\|_2$.

This sweep operation presents a problem in that, when we add noise to our D^TD matrix, we might make it non-invertible. In order to fix this we follow a method proposed by Xi et al. [1] We loop through the eigenvalues of the D^TD matrix and if any of the eigenvalues are less than 0, we set them to be equal to the least positive eigenvalue. We then use the idea of eigenvalue decomposition to augment the D^TD matrix in order to force all positive eigenvalues.

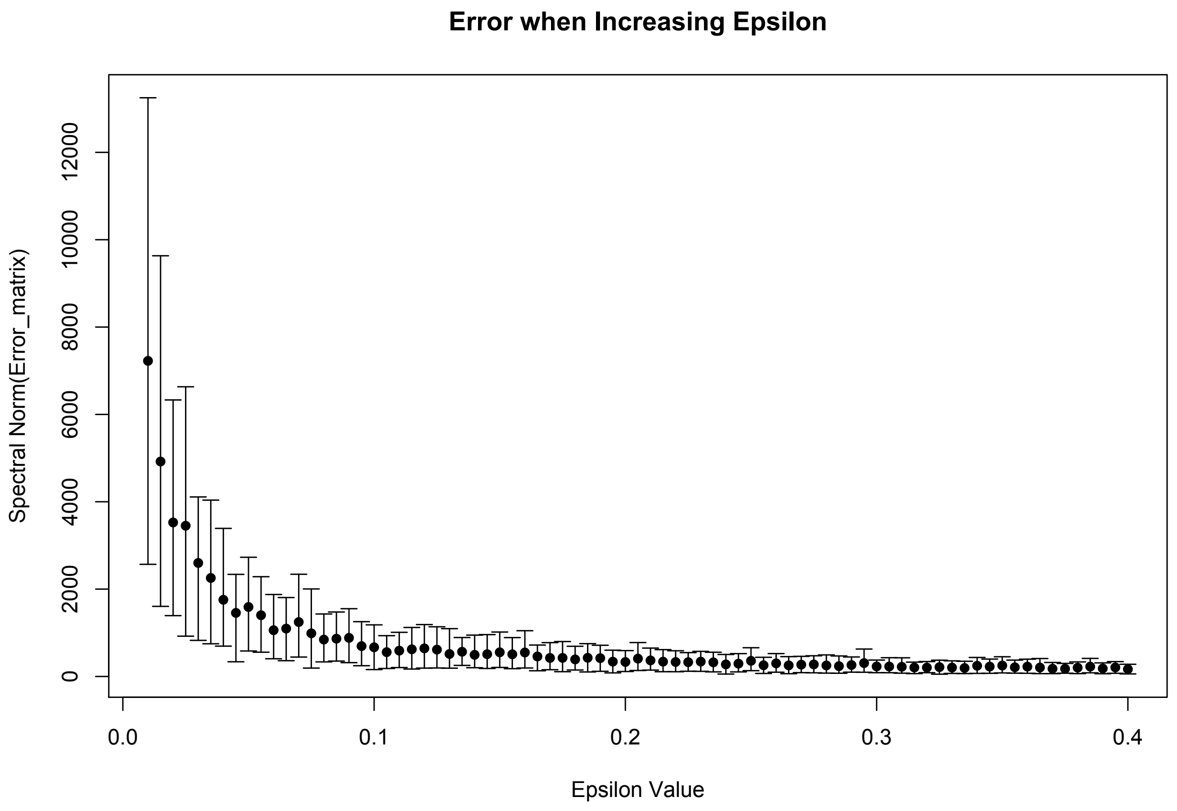
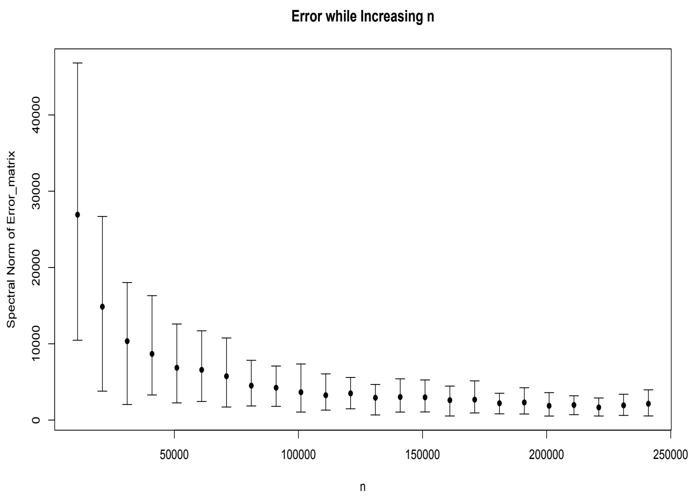


Figure 8: Spectral norm of noise matrix as epsilon increases



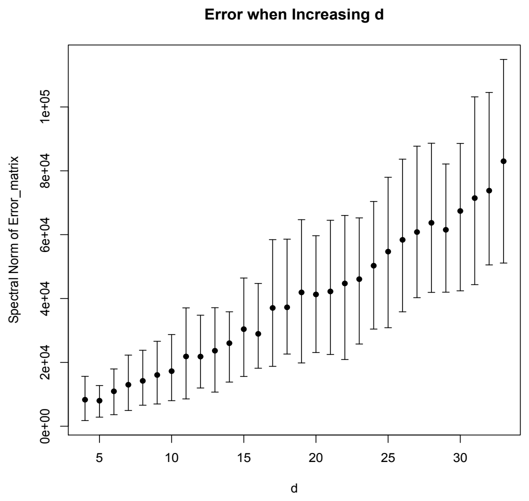


Figure 9: Spectral norm of the noise matrix

Figure 10: Frobenius norm of the noise matrix

3.3 Simulated Data Regressions

In our first tests, we utilized simulated data, produced by the previously discussed method. In this test, we generated the data with a linear model in mind: $Y = C * X_1 + 0$. We set $\varepsilon = 1/100$, $\delta = 2^{-16}$, $d = 5$, and $n = 50000$. Each column is independently generated from a normal distribution with random standard deviation and mean. We force a relationship between Y and X_1 by a factor of C and add noise from $\mathcal{N}(0, 1)$. Each experiment is repeated 1000 times.

In figure 11 we display the average beta distance over 1000 trials for each value of C. Figure 12 displays the 95th-percentile of the beta distance over the 1000 trials.

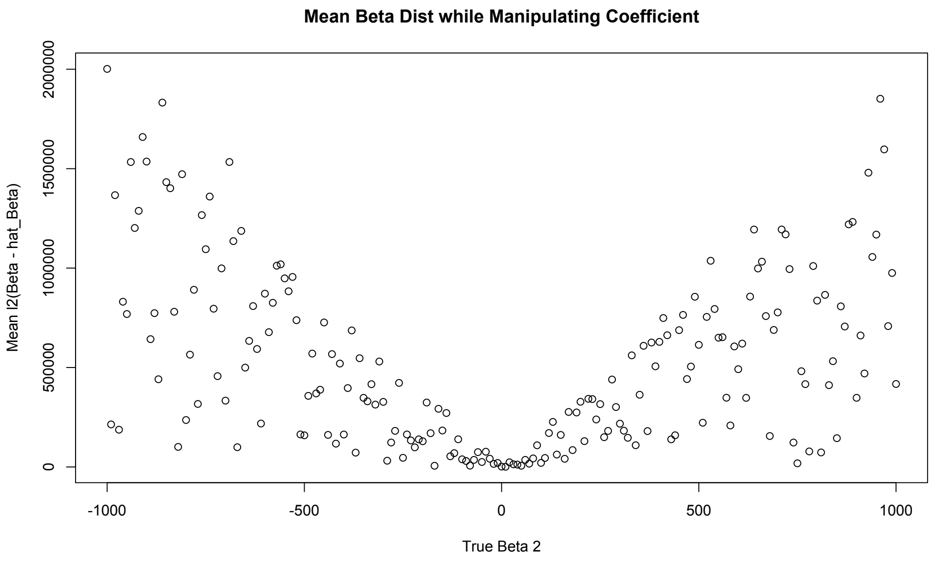


Figure 11: Mean beta distance while manipulating C

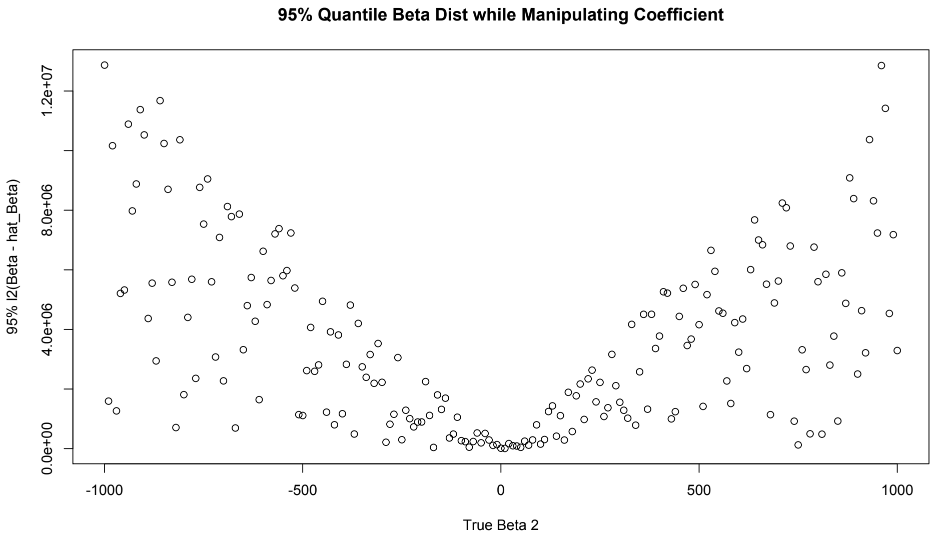


Figure 12: 95th-percentile beta distance while manipulating C

In figure 13, we display the OLS estimator (s^2) while manipulating C.

In figures 14 - 17, we show an interesting trend as we decrease our "confidence" in estimating C. At the 70% level, nearly all of the noisy coefficients are very close to the true coefficient, C.

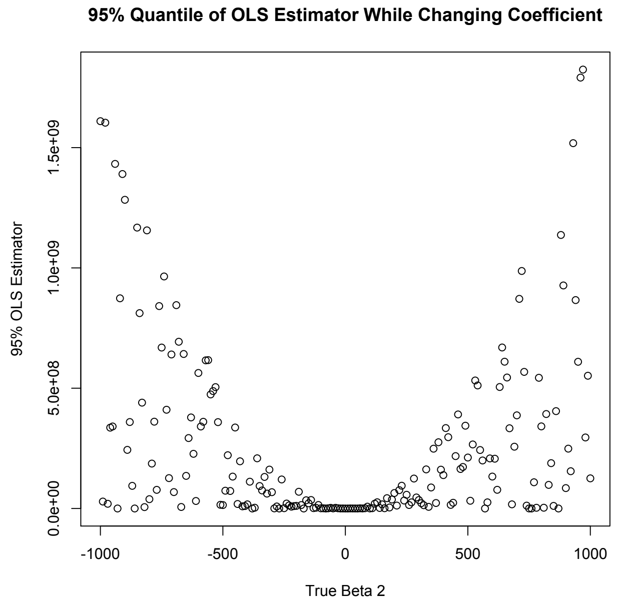


Figure 13: 95th-percentile of OLS estimator while manipulating C

3.4 Real Data Regressions

Now instead of using simulated data, we use the PUMS data sample. As such, our parameters change a little: $\varepsilon = 1/100$, $\delta = 2^{-16}$, $n = 1223992$, $d = 3$. We only use three data attributes from the PUMS data since they are the only attributes where numerical analysis makes sense: `age`, `educ`, and `income`. We decided to focus on the following regression:

$$\texttt{income} \sim \texttt{age} + \texttt{educ} \Rightarrow Y = \hat{\beta}_2 * \texttt{age} + \hat{\beta}_3 * \texttt{educ} + \text{int} * 1$$

Like before each experiment is run 1000 times. In tables 3 and 4, we show the results of this ordinary linear regression. Table 3 represents the noisy regression (one randomly selected trial) while table 4 represents the regression performed by the `lm` function.

Table 3: Noisy Model Results

	Estimate	Std. Error	t value	Pr[$> t$]	
(int)	-19580.2866	3.1555	-114.923	<2e-16	***
X_2	206.9308	2.4531	84.3545	<2e-16	***
X_3	4286.8762	12.4743	343.658	<2e-16	***

Table 4: True Model Results

	Estimate	Std. Error	t value	Pr[$> t$]	
(int)	-23054.102	154.580	-149.1	<2e-16	***
X_2	275.253	2.218	124.1	<2e-16	***
X_3	4332.877	11.560	374.8	<2e-16	***

At a glance, these results seem reasonable. The algorithm still declares each of the three coefficients, $\hat{\beta}_2$, and $\hat{\beta}_3$, and the intercept, to be statistically significant, with reasonably close predictors for each of those coefficients. Figures 18 - 20 are histograms of the 1000 trials with results for each coefficient. The vertical bars represent the true coefficient value as calculated by `lm`.

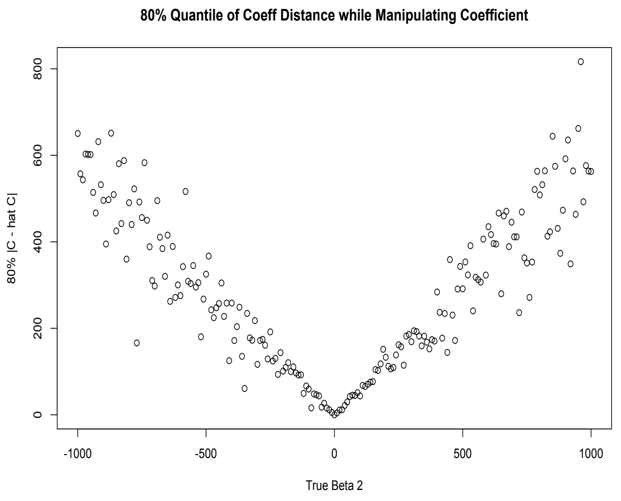


Figure 14: 80th-percentile of coefficient distance

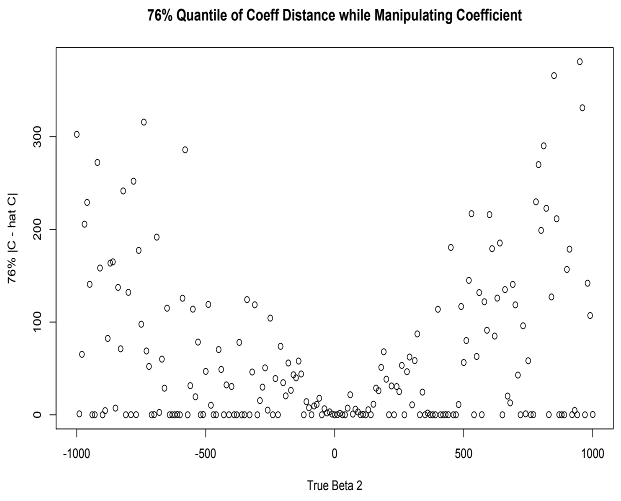


Figure 15: 76th-percentile of coefficient distance

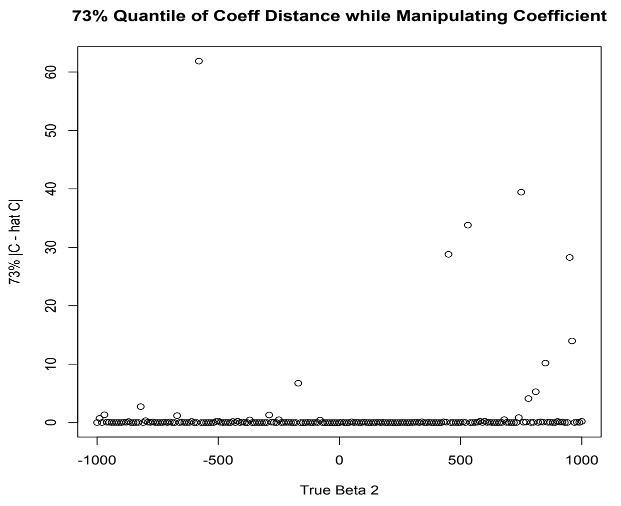


Figure 16: 73rd-percentile of coefficient distance

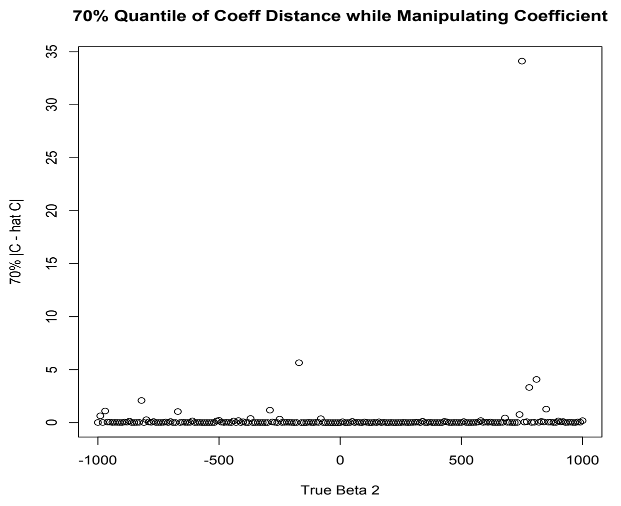


Figure 17: 70th-percentile of coefficient distance

As you can see, when we do release a "badly" noisy coefficient, it is possible it will be extremely bad. Of particular note is the intercept coefficient which has an incredibly wide spread. OLS operates under the assumption that any relationship between the regressors and the and predicted values that isn't explained by the regressors themselves needs to be made up in the intercept term. In other words, the intercept term becomes a sort of "trash" variable in our case. Since we are adding random noise to the covariance matrix, we are altering the sweep operation for regression in a random way. If we artificially decrease the covariance between Y and X_1, in order to minimize the residuals, OLS increases or decreases the intercept term.

Overall the algorithm seems to do what we were expecting, but we still do not have any way of concretizing the usefulness of the regression nor a way of telling the user how much faith to place in the regression in a differentially private manner.

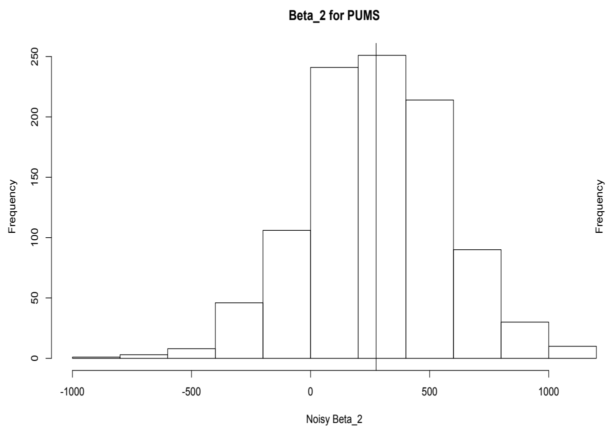


Figure 18: $\hat{\beta}_2$ over 1000 trials

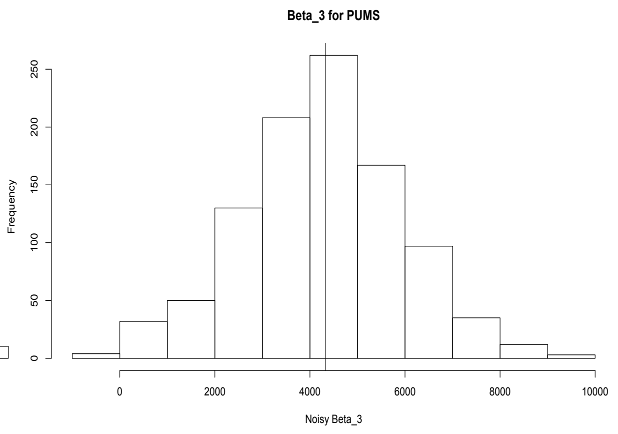


Figure 19: $\hat{\beta}_3$ over 1000 trials

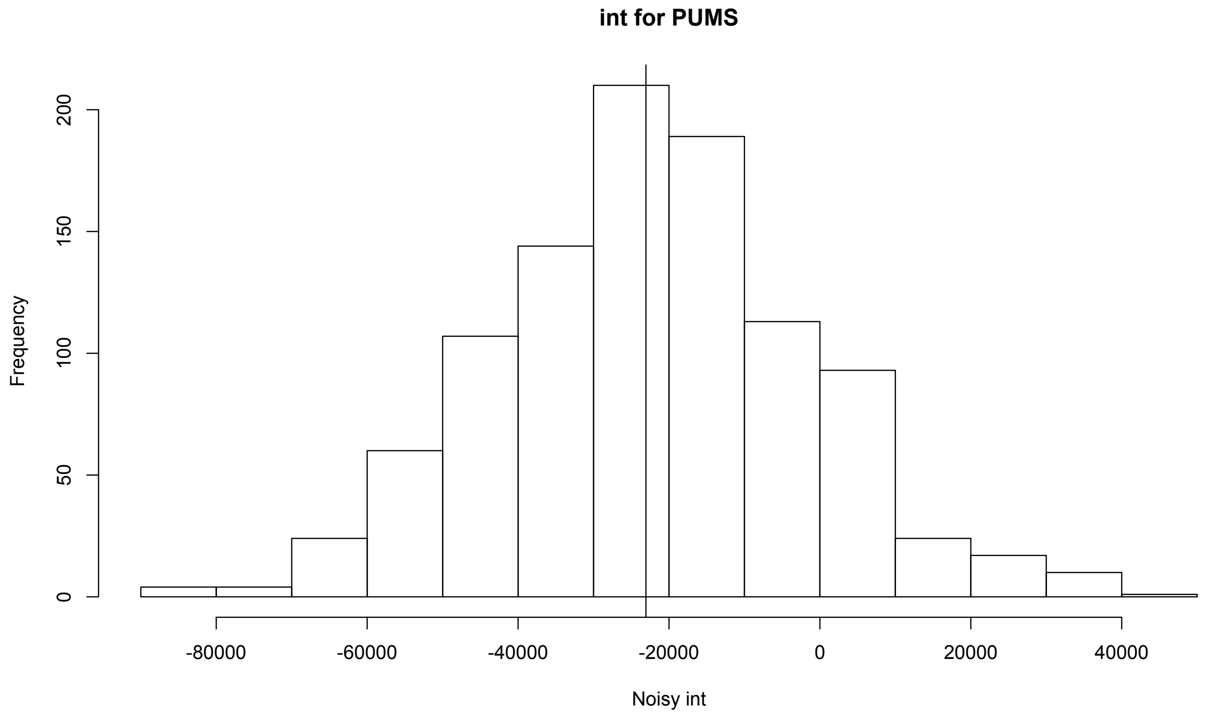


Figure 20: Intercept over 1000 trials

4 Integrating with Zelig

Due to the structure of Zelig, the interface we provide is rather simple. In fact, we only offer three public methods: `VCV.getAccuracy`, `VCV.getParameters`, and `VCV.computeVCV`. The first two methods allow the overall privacy controller to specify an ε, δ, a β probability, the usual dataset metadata (n, d, mins, maxes) and we provide the size of a $1-\beta$ confidence interval around the true variance-covariance matrix and vice versa. This allows the privacy controller and data uploader to experiment with different levels of privacy and what the repercussions on accuracy are.

The `VCV.computeVCV` function is the workhorse of our API. It is responsible for releasing the differentially private variance-covariance matrix. This function will be the one called by the privacy controller and given a dataset, differentially private metadata (n, d, mins, maxes), and the appropriate privacy parameters (ε, δ), and then return a differentially private variance-covariance matrix that can be used to perform "differentially private" regressions.

For now, the `VCV.getAccuracy` and `VCV.getParameters` methods only return estimates based on the amount of gaussian noise added to the variance-covariance matrix. Until we decide on a better measure of accuracy for this particular differentially private statistic, this error estimate will have to suffice.

5 Conclusion

Using the Gaussian mechanism, we were able to implement an algorithm that releases a differentially private covariance matrix. In addition, we implemented a statistical sweep operation in order to perform ordinary linear regressions using the d.p. covariance matrix. Unfortunately, we were not able to identify a successful measurement of utility which relates the d.p. parameters (ε, δ) to the accuracy of a linear regression. However we did make useful observations on the degradation of utility as the number of regressors increases and as the range of the variables in question increase. We hope that using regularization will help rectify this issue. Future work will need to focus on how to measure the utility of the algorithms in a more direct manner. In addition, we will continue to investigate alternative algorithms like the Johnson Lindenstrauss Algorithm.

In conclusion, we successfully implemented an (ε, δ)-differentially private algorithm to release a private covariance matrix that can be used with the Zelig statistical package and TwoRavens graphics user interface.

References

[1] M. Kantarioglu B. Xi and A. Inan. Mixture of gaussian models and bayes error under differential privacy. In *CODASPY*, February 2011.

[2] US Census Bureau. Public use microdata sample, 2014.

[3] A. Thakurta C. Dwork, K. Talwar and L. Zhang. Analyze gauss: optimal bounds for privacy-preserving pca. In *STOC*, June 2014.