15-316: Software Foundations of Security and Privacy **Spring 2017**

Lecture 21: Provable Privacy

Lecturer: Matt Fredrikson

21.1 Introduction

In the previous lecture, we talked about various notions of data privacy. In the first case, we considered **exact disclosure** of statistical functions over sensitive data. This approach is appealing because it provides users with the best possible accuracy, but we concluded that it is fairly easy for curious users to devise sequences of queries that end up disclosing too much specific information about individuals. In the second case, we went the other direction and considered **absolute privacy**, which requires that no more about an individual is learned after interacting with the database than was known before. Unfortunately, we found that a fairly general formalization of this model quickly leads to impossibility results when users require utility from their interactions. In short, both models are non-starters along different but equally-important dimensions.

In today's lecture, we'll look at another model of privacy that seeks a middle ground between the two we've already discussed. In particular, this model is a *relativized relaxation* of absolute privacy, where instead of insisting that users learn nothing about individuals in the database, we instead require that anything a user can learn from interacting with the database should be "almost" as learnable regardless of whether any individual is actually in the database. As for utility, we'll back away from releasing exact statistics about the database, and instead be content with approximate correctness. Indeed, the degrees of *"almost" as learnable* and *approximate correctness* are closely related, and are governed by a **privacy budget**, which is a parameter that the data processor can use to fine-tune the tradeoff between privacy and utility. The particular notion that we will discuss is called **differential privacy**.

21.2 Differential Privacy

Recall first our definitions from the statistical database model. We assume an $N \times M$ table-structured database X (i.e., X has N rows each corresponding to an individual described by M features). The data processor San is the entity that accepts queries from users, described by symbol A, and computes their results over the database. From now on, we will assume that San only computes one type of query (e.g., sum, average, count, etc.) and returns it to A. With these definitions in mind, we can now define differential privacy (or more specifically, ϵ-differential privacy).

Definition 21.1 ϵ-Differential Privacy. *Let $\epsilon > 0$. A randomized function San satisfies ϵ-differential privacy if for all possible databases X_1 and X_2 that* differ in exactly one row*, and all $s \in \mathsf{Range}(\mathsf{San})$, the following inequality holds:*

$$\Pr[\mathsf{San}(X_1) = s] \leq e^{\epsilon} \times \Pr[\mathsf{San}(X_2) = s] \tag{21.1}$$

The probabilities in this expression are taken over the randomness of San's outputs.

First, notice that Definition 21.1 is a property of the function San, and *not* of the data being computed on or any particular output of San. In other words, when we speak of something as being differentially private,

we are always referring to a process used to produce query responses from sensitive databases. You may at times hear people refer to a piece of data as "differentially private", but do not get confused; when used correctly, this language means that the data was computed by a function that satisfies ϵ-differential privacy.

As you might have guessed, the ϵ in Definition 21.1 corresponds to the privacy budget we discussed above. We'll get into some high-level intuitive interpretations of this definition in a little while, but first let's think about its various components and how they relate to San's behavior directly.

Privacy budget ϵ. We hinted earlier that the privacy budget has an influence on both the degree of privacy established by the model, as well as the degree of approximation in the results. ϵ is our privacy budget, and it is a numeric real-valued quantity. To understand what it means, let's look at the behavior of an ϵ-differentially private San for extremal values of ϵ.

Suppose that we make $\epsilon = 0$. Then Definition 21.1 requires that for any X_1, X_2 that differ in one row, Equation 21.1 holds. However, notice that the definition is symmetric in the values that X_1, X_2 take; there is nothing that distinguishes them from eachother, so San must also satisfy:

$$\Pr[\mathsf{San}(X_2) = s] \leq \Pr[\mathsf{San}(X_1) = s] \tag{21.2}$$

Combining equations 21.1 and 21.2, it must be that $\Pr[\mathsf{San}(X_1) = s] = \Pr[\mathsf{San}(X_2) = s]$ for all X_1, X_2 that differ in one row. What does this mean for the privacy of individuals in X_1 and X_2, and the utility of San?

- When it comes to privacy, we can conclude that $\epsilon = 0$ implies **no leakage** of information about the contents of *any* individual row. Why does this hold for any row? Recall that Definition 21.1 needs to hold for *all* pairs X_1, X_2. So, if our actual database is X_1, then all databases X_2 that we obtain by changing a row in X_1 must produce the same distribution of outputs in San.
- As for utility, you probably guessed that $\epsilon = 0$ isn't great. In fact, because San's output distribution needs to remain the same for all adjacent databases, we can observe that by transitivity San's output distribution needs to remain the same for *all* databases. In other words, the results can't contain any information about the input database, which clearly means no utility is possible.

Question. *What functions satisfy 0-differential privacy? Suppose that* $\mathsf{Uniform}(r)$ *represents a random varible for the uniform distribution in the range* $[-r, r]$*, and* X *is a list of real-valued numbers. Does* $\mathsf{Sum}(X) + \mathsf{Uniform}(r)$ *satisfy Definition 21.1 when* $\epsilon = 0$*? Prove your answer.*

Clearly, $\epsilon = 0$ is good in terms of privacy but terrible for utility. We might expect that when ϵ is large, say 10. Note that $e^{10} \approx 22,000$. Probabilities range in $[0, 1]$, so in order for Equation 21.1 to place any meaningful limits on San's behavior, e.g. a limit on $\Pr[\mathsf{San}(X) = s]$ for some s, the probability of returning s on the neighboring database would need to be quite small ($\sim 4.5 \times 10^{-5}$). Conversely, because this large ϵ gives San quite a bit of freedom in its behavior, utility is not a problem.

So as ϵ grows, the privacy offered by ϵ-differential privacy drops off very quickly, and the utility begins to approach what we could achieve from exact disclosure.

Neighboring databases. Definition 21.1 quantifies universally over pairs of databases X_1, X_2 that differ in one row. Such pairs are called **neighbors**. What exactly do we mean by "differ in one row"? First of all, it's important to mention that we aren't concerned with the order of the rows in X_1, X_2, so that if they were permutations of each other, we would consider them to be the same[1]. There are two reasonable interpretations.

[1] Usually the queries performed in this model are associative with respect to rows, so from the user's perspective, permuted databases are the same

1. X_1 and X_2 are identical, except that X_1 has an additional row that X_2 doesn't. So if we view databases as sets (assuming all rows are unique within each database), then $|X_2| = |X_1| - 1$ and $X_2 \subseteq X_1$, $X_2 = X_1 \cup \{x_m\}$ for some x_m.
2. X_1 and X_2 have the same number of rows, but the value of one row is different. In other words, we could find a permutation of X_1 such that $X_1[1 \dots N-1] = X_2[1 \dots N-1]$, and $X_1[N] \neq X_2[N]$.

We will use by convention the latter definition of neighboring databases.

Randomized San. Is it essential that San be a random function? First of all, the Definition 21.1 is an inequality over probabilities, and the only source of randomness comes from San. So in a technical sense, we are required to assign probabilities to San's responses. However, we can interpret deterministic functions as a special case of randomized functions, so let's think about which deterministic functions might satisfy the definition.

Suppose that $X_1 = [0, 0, 0]$ and $X_2 = [0, 0, 1]$, and $\mathsf{San}(X_1) = s$. Any deterministic San whose value depends on the last element, so that $\mathsf{San}(X_2) = s'$ where $s \neq s'$ will give us:

$$\Pr[\mathsf{San}(X_1) = s] = 1, \Pr[\mathsf{San}(X_2) = s] = 0$$

so that for any ϵ Equation 21.1 fails to hold. Because the order of rows in X_1, X_2 doesn't matter, this means that San's response can't depend on *any* row in X. Thus, the only deterministic functions that satisfy Definition 21.1 are constant.

21.2.1 Interpreting the Definition

Now that we've thought about the definition and some of its technical implications, let's think about what it means for privacy. Earlier, we described it as something of a "relativized" relaxation of absolute privacy from last lecture, so we'll pick up from there.

Inference and protection from harm. One view of privacy is that it is about protecting individuals from harm that may arise from the release of their data. Absolute privacy was appealing because if nothing more about an individual can be inferred, or learned, from the release than could have been without the release, then it follows that no harm can come to the individuals whose data contributed to that release. Put differently, we assume that the only way that an individual can be harmed by an information release is if someone learns about that individual from the release; if nothing new about the individual can be learned from the release, then no harm can follow.

Unfortunately, we saw that this is generally impossible to achieve, but differential privacy is aimed at protecting individuals from such harm to the greatest extent possible. The key to this is the *relative* nature of the definition. Rather than trying to prevent users from learning *anything* about an individual, we can think of the definition as trying to prevent users from learning new things about an individual relative to what they *could have* learned had the individual not shared their data. This is where the idea of neighboring databases comes from: a neighboring database is one in which a particular individual's data takes a different value, which we can view as being a database where everyone *except* that individual shared (i.e., some other individual took their place). Differential privacy requires that any output of San be approximately as likely in both cases: one where the individual shared their data, and one where they did not.

For example, suppose that you are given the opportunity to share your medical records with a researcher who will use them in a study intended to improve treatments. You may rightly be concerned that if the researcher

publishes results based on your data, a data-savvy insurance provider might be able to infer something about your health status from these results in the future, and decide to raise your premiums or deny coverage. However, if the researcher applied differential privacy with an appropriately-chosen ϵ, then you might be reassured that no results that could come of the study would be much more or less likely because of your decision to share. It follows that if an insurer were to base their decision on those differentially-private results, then it is similarly not much more or less likely that you would be denied coverage.

Plausible deniability. Another good way of looking at the protection given by differential privacy is in terms of **plausible deniability**, or the individual's ability to make a believable claim that their data takes some value of their choosing, i.e., to "deny" a claim that their data took the value it did. Because Definition 21.1 requires that the likelihood of San responding with any value s is nearly identical regardless of what value the individual's data took, it would indeed be reasonable for the individual to claim that their data took another value; the probability of producing s would be about the same no matter what value they chose.

Indistinguishability and influence. Another way of viewing the definition, which brings us closer to the semantics of the computation done by San, is in terms of how much individuals' data can influence, or cause changes to, San's response. We've talked about influence before in the context of noninterference, which required that the H-typed initial state have no influence on the L-typed final state:

$$\forall \sigma_1, \sigma_2. \sigma_1 \approx_{\mathsf{L}} \sigma_2 \implies \mathsf{Ev}(\sigma_1, c) \approx_{\mathsf{L}} \mathsf{Ev}(\sigma_2, c) \tag{21.3}$$

We might rewrite Definition 21.1 more concisely as follows.

$$\forall X_1, X_2. \mathsf{Neighbor}(X_1, X_2) \implies \forall s. \Pr[\mathsf{San}(X_1) = s] \leq e^{\epsilon} \times \Pr[\mathsf{San}(X_2) = s] \tag{21.4}$$

Notice the similarities between Equations 21.3 and 21.4.

- In both cases, the definitions quantify over all pairs of inputs (i.e., initial states) that are related in a way that reflects that we are trying to protect. For noninterference, the relation does this by only constraining the L variables, so that the final state is indistinguishable regardless of the initial H variables. For differential privacy, the neighbor relation works similarly by letting each individual's data take an arbitrary value, and fixing the rest of the database.
- The right-hand side of the implication in each case describes the sort of changes that inputs, and more precisely inputs described by the left-hand side, are allowed to cause. Noninterference rules out any changes to L variables, whereas differential privacy places limits on the probability of variation in the response.

Viewed this way, differential privacy is a property which states that the influence of individual rows on San's response should remain low, so that responses computed under neighboring databases are "almost" indistinguishable. This is the essential property that allows for plausible deniability and protection from harm, and the core of differential privacy's strong guarantees.

Recall also that we were able to prove that programs satisfy noninterference, even to the point of designing type systems that simplify the task of writing noninterferent programs, and can be checked efficiently. Given the similarity between Equations 21.3 and 21.4, it should not be too surprising that we can also prove program's adherence to differential privacy. This is part of the appeal of using the definition in practice: it provides a crisp mathematical formulation of what it means to be private, that can be proven on real computations.

21.3 Building Differentially-Private Computations

Now that we're comfortable with the definition and what it means, we'll turn to strategies for implementing programs that satisfy ϵ-differential privacy. At first glance, the definition might seem limited from a practical point of view, as it only applies to random functions. However, this is not a real issue because it turns out we can systematically convert deterministic functions into random ones that satisfy differential privacy using a few general-purpose techniques.

21.3.1 Adding Noise

The first approach we'll discuss is based on adding "noise", or carefully-chosen random values, to the result of a computation. In particular, we'll begin by computing the exact result, and then add noise before releasing it. Recall that a differentially-private San needs to produce outputs with similar distributions for all pairs of neighboring databases. This implies that the noise added to the output will need to be large enough to mask the differences in San's outputs on neighboring pairs. This quantity is formalized by San's **global sensitivity**, shown in Definition 21.2.

Definition 21.2 Global Sensitivity. *Assume that* San *is a function* $\mathsf{San} : \mathbf{X} \mapsto \mathbb{R}$*, where* $\mathbf{X}$ *is the set of all databases up to a particular size accepted by* San*. Then the* global sensitivity *of* San*, written* $\Delta\mathsf{San}$*, is defined as:*

$$\Delta\mathsf{San} = \max_{X_1, X_2} |\mathsf{San}(X_1) - \mathsf{San}(X_2)|$$

where X_1, X_2 *are neighboring databases.*

Example 21.3 *Recall the* $\mathsf{count}(X, e)$ *function from the previous lecture, which takes a database* X *and a Boolean expression* e *over the column names in* X*.* $\mathsf{count}(X, e)$ *returns the number of rows in* X *that satisfy (i.e., evaluate to* true*)* e*.* $\Delta\mathsf{count} = 1$ *because in the "worst" case, changing the value of a row in* X *will either cause* e *to be satisfied on another row, or cause it to be satisfied on one fewer rows.*

Example 21.4 *Let* $\mathsf{sum}(X)$ *be the function that sums the values in database* X*, and* $\mathbf{X} = \mathbb{Z}^n$ *for some* n*. In this case* sum *operates over databases whose rows consist of unbounded values, so* $\Delta\mathsf{sum}$ *is undefined because changing any single row in* X *could cause* $\mathsf{sum}(X)$ *to change by an unbounded amount.*

Note that if we assumed bounds on the values in databases, for example by setting $\mathbf{X} = \mathbb{Z}^n \cap [0, M]^n$ *for some fixed* M*, then* $\Delta\mathsf{sum}$ *is no longer undefined. More precisely,* $\Delta\mathsf{sum} = M$*.*

Global sensitivity bounds the magnitude by which San's response can change on pairs of neighboring databases. Our goal in designing a differentially-private version of San is to hide such changes, so that the function's response on neighboring pairs is approximately the same. Thus, $\Delta\mathsf{San}$ tells us how much noise we need to add to the result to do so.

Laplace noise. The key to finding the right noise to add is in selecting an appropriate distribution to sample from. We'll use the **Laplace distribution** with parameter b, denoted $\mathsf{Lap}(b)$, which has density function:

$$\mathbf{Pr}[z] = \frac{1}{2b} \exp\left(\frac{-|z|}{b}\right) \tag{21.5}$$

Alternatively, the density function for $\mathsf{Lap}(b)$ can be given as:

$$\mathbf{Pr}[z] = \frac{1}{2b} \begin{cases} \exp\left(\frac{z}{b}\right) & \text{if } x < 0 \\ \exp\left(\frac{-z}{b}\right) & \text{if } x \geq 0 \end{cases} \tag{21.6}$$

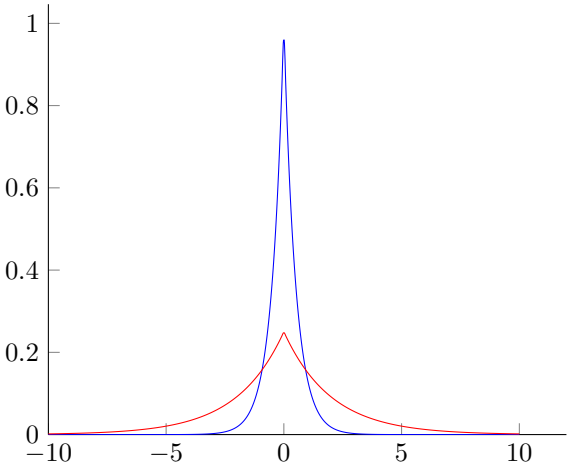


Figure 21.1: Laplace distribution with $b = \frac{1}{2}$ (blue) and $b = 2$ (red).

The Laplace distribution is shown in Figure 21.3.1. There are several important points to note about its shape.

- The distribution is centered (i.e., has its mean) at 0 and is symmetric. Because we're going to add samples from this distribution to our response, both of these properties are important. If the distribution were biased (i.e., its mean were something other than 0), then on average we would unnecessarily skew the results of San by offsetting them by the distribution's mean. The symmetric property of $\mathsf{Lap}(b)$ means that the noisy result is no more likey to be greater (or smaller) than the true result. If the distribution only had support in the positive half-space of $\mathbb{R}$, for example, then the only databases for which $\mathsf{count}(X, e)$ could return 0 would be those with no rows satisfying e. A user who is aware of this fact could leverage it to learn more about X than we want to provide in our response.
- Most of the mass in this distribution is close to 0. In fact, looking at Equation 21.5, the probability of sampling a value z drops off exponentially as z moves further from 0, so drawing a very large or small value is exceedingly unlikely. This is good for accuracy, because it means that the noise we add will have small magnitude most of the time.
- Notice in Figure 21.3.1 that the shape of the distribution for $b = 2$ is quite a bit "flatter" than the one for $b = 0.5$. If we were to add noise from $\mathsf{Lap}(2)$, then our response will on average diverge more from the true answer than they would for $\mathsf{Lap}(0.5)$. While $\mathsf{Lap}(2)$ is clearly worse in terms of utility, it offers stronger privacy. We will exploit this fact by relating the parameter b to the privacy budget ϵ when we generate noise.

Now that we have some intuition for the Laplace distribution, we can show how it is used to make San differentially-private.

Theorem 21.5 *(Dwork et al., 2006) Assume that* San *is a function* $\mathsf{San} : \mathbf{X} \mapsto \mathbb{R}$. *Then the function*

$$\widehat{\mathsf{San}}(X) = \mathsf{San}(X) + \mathsf{Lap}\left(\frac{\Delta\mathsf{San}}{\epsilon}\right)$$

satisfies ϵ-differential privacy.

Theorem 21.5 describes the **Laplace mechanism**, a general strategy for obtaining differentially-private functions by the addition of noise.

Inputs: $\mathsf{San} : \mathbf{X} \mapsto \mathbb{R}$, ϵ, $\Delta\mathsf{San}$

1. Let $b \leftarrow \Delta\mathsf{San}/\epsilon$
2. Sample u from the uniform distribution on $[0, 1]$
3. Let $u' \leftarrow u - \frac{1}{2}$
4. Let $r \leftarrow -b \times \mathrm{sgn}(u') \times \ln(1 - 2|u'|)$
5. Return $\mathsf{San}(X) + r$

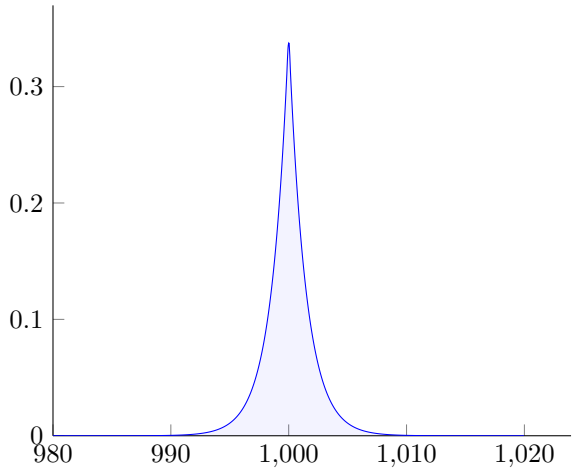
Figure 21.2: Differentially-private San via the Laplace mechanism, when San is real-valued and has finite global sensitivity.

Implementing the approach. This gives us a straightforward general-purpose strategy for converting a real-valued function into a differentially-private approximate version. Of course, the original function needs to have bounded global sensitivity, and the accuracy of the approximation depends on the sensitivity being small. The main difficulty in utilizing Theorem 21.5 lies in drawing samples from $\mathsf{Lap}\left(\frac{\Delta\mathsf{San}}{\epsilon}\right)$. However, assuming that we can draw random numbers from the uniform distribution over $[0, 1]$, then an approach called *inverse transform sampling* is an efficient solution in this case. We won't cover inverse transform sampling in any greater detail, but the procedure for $\mathsf{Lap}(b)$ works out to the following:

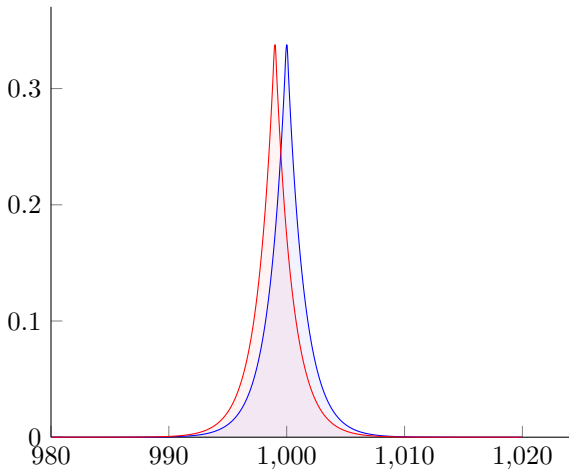
1. Sample u from the uniform distribution on $[0, 1]$
2. Let $u' \leftarrow u - \frac{1}{2}$
3. Return $-b \times \mathrm{sgn}(u') \times \ln(1 - 2|u'|)$

Putting everything together, the procedure to convert a real-valued San into an approximate differentially-private version is shown in Figure 21.3.1.

Example 21.6 *Lets apply the procedure from Figure 21.3.1 to* count. *Suppose that the database X has* $10,000$ *elements, and we select an ϵ such that* $1,000$ *of them return* true. *So the true response* $\mathsf{San}(x) = 1,000$. *When we apply the Laplace mechanism with* $\epsilon = \ln(2)$, *the output distribution of* $\widehat{\mathsf{San}}(x)$ *will be* $1,000 + \mathsf{Lap}(1/\ln(2))$, *as shown below. In this graph, the x-axis corresponds to the response value of* $\widehat{\mathsf{San}}$, *and the y-axis the probability of each response value.*

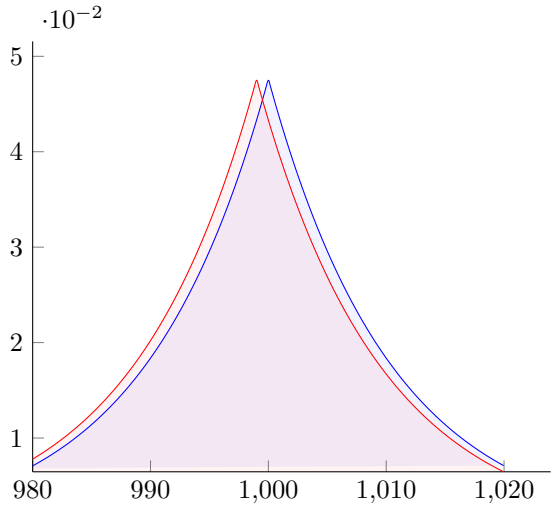


Now suppose that X contained 999 rows satisfying the count query e. This hypothetical corresponds to a neighboring database of the original, and the output distribution of $\widehat{\mathsf{San}}$ looks like the following curve in red.

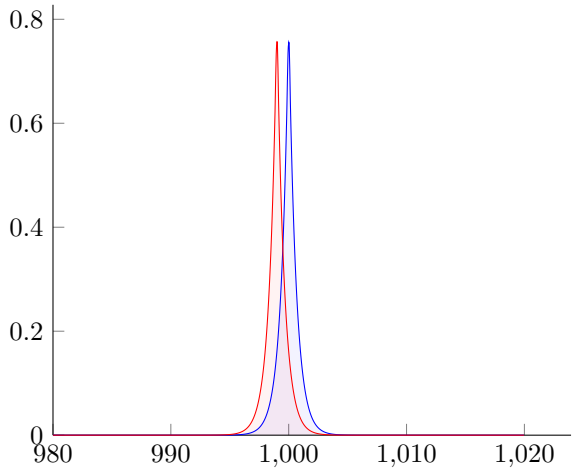


Notice that the two response distributions are very similar. Although they have means that differ by one, the probability of observing any particular response is quite close, so it will be difficult for an observer to tell whether the true response is 1,000 or 999.

Let's consider what happens when we make the privacy budget ϵ smaller, thus strengthening the privacy guarantee. Setting $\epsilon = \ln(1.1)$, we see that the curves are quite a bit more spread out, and remain closer together.



Finally, when we make ϵ quite a bit larger, say $\ln(5)$, we see that nearly all of the mass is centered on the true response in either case, and the curves are further apart and thus easier to distinguish.


0.8
0.6
0.4
0.2
0
980
990
1,000
1,010
1,020