Utility Test Development and Utility Testing for Differentially Private CDFs and PDFs

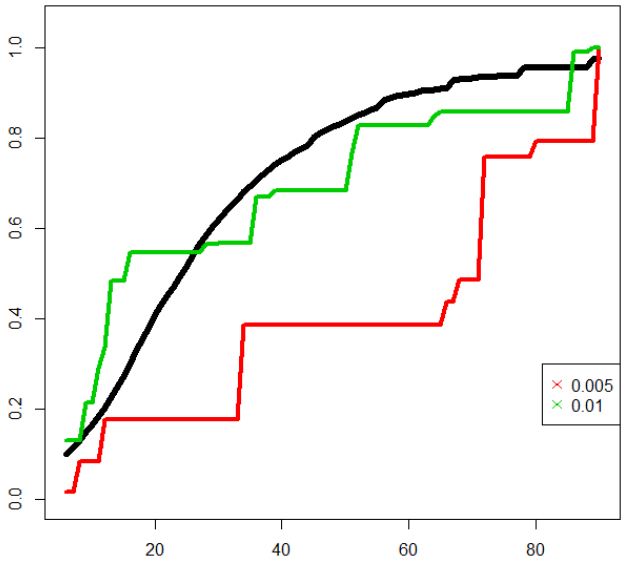
Daniel Muise*

Mentors: Victor Balcer Mark Bun Kobbi Nissim

Harvard University, Privacy Tools Project: Differential Privacy Group

1. Task and Context

Prior to this summer, the Privacy Tools team developed several differentially private methods of computing probability density functions (PDFs) and cumulative density functions (CDFs). However, the absolute and contextual utility of these methods had not been empirically tested. The overarching goal of this work was to design a system to quantify and clarify the usefulness of various methods of computing CDFs and PDFs. The systems' output ranks methods of differentially private data release to be incorporated into Dataverse, TwoRavens, and future projects.



1. A CDF and two D.P. representations. The red CDF, with Ɛ= .005, is more private than the green. Y-Axis is cumulative probability, X-axis is a set of ordinal categories. The median of this dataset is the x-value where the black CDF's Y-value = 0.5

2. Main Principles and Goals

2.1 Principles

Foremost we designed all of our tasks for maximum abstraction and generalizability. Second, we worked strictly with our end-users in mind, putting communication of results to the forefront.

2.2 Goals

1. The creation of a system of comparing the utility of CDF/PDF computation algorithms to each other and to non-private computation methods. We used the R programing language to create this system. It allows for the input of data, functions, and privacy limits, and returns statistical analysis of algorithms' performance in various contexts. The results of this system, known as the testing suite, inform differential privacy research, and its design can be replicated to make similar systems.
2. Clarification of differential privacy's effects in CDFs. To do this, we initialized a reference manual for social scientists and end users to refer to. It includes practical explanations of differential privacy, contextual descriptions of differentially private error, and how to properly read differentially private CDFs. Ideally, this clarification

endeavor will ensure best practices and responsible use of differential privacy by social science researchers.

We also collaborated in coding, testing, and developing newer CDF/PDF computation algorithms. For example, this involved the implementation of algorithms such as 'efficient tree' variance-cancelling methods.

3. Contributions

- Development of diagnostic choices to best determine utility of differentially private CDFs, PDFs, and a system of doing so for other D.P. statistics;
- Construction of a user-friendly CDF & PDF 'testing suite' program with graphic and statistical output;
- A set of statistical and graphical descriptions of differentially private CDF/PDF computations methods' absolute and contextual utility;
- Implementing methods of automatically finding differentially private quantiles and base-level statistics from CDFs, thereby conserving privacy budget;
- Integration of efficient-tree methods into applied settings;
- Initialization of a reference manual for end users to interpret differentially private statistical output and utility measurements.

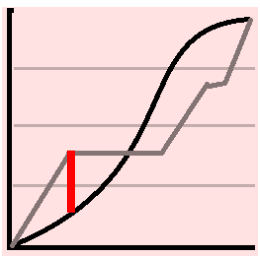
4. The CDF Testing Suite

4.1 The CDF test overview

The CDF testing suite is a set of wrapped functions written entirely in one R script, and can be used in various ways by entering only one command with various arguments. The system's two main functions are "CDFtestTrack" and "CDFtest".

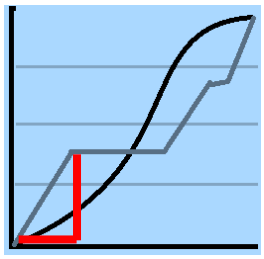
CDFtestTrack takes a single data vector, a single CDF function, and a single epsilon value and runs that combination through a variety of tests, which are internal functions defined in the same script. These internal diagnostic functions determine various sorts of differences (errors) between the differentially private CDF and a real CDF of the data. Specifically, these diagnostic functions are:

- The maximum vertical error,[1] which is measured as the largest difference between the real CDF and the D.P.-CDF at any element of the domain. This value offers a data-dependent boundary within which all other parallel differences exist. In an applied sense, when determining the percentile measurement of a given value, the result would not be more than 'Maximum Vertical Error' away from the true percentile measurement.



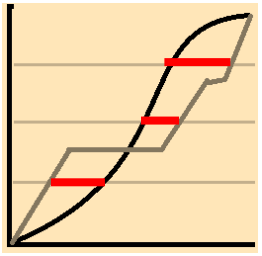
[1] This diagnostic, along with "location of max error" and L1norm, are the only three that are performed on PDFs.

- The location of the maximum vertical error[2], which is useful in determining skewness and heteroscedasticity. If the maximum vertical error consistently arises in one region of the random variable's domain, then researchers could conclude something about the variance introduced by the dp-CDF function.



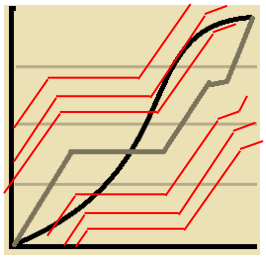
- Vertical distance at median, which is akin to maximum error. The median is an often-sought statistic, so it's useful to check performance at this key point.
- Vertical distance at the 25th & 75th percentiles, which follow the same logic as the median measurement above.

- Horizontal distance at the median, which is the difference in values corresponding to the 50th percentile measure. Possibly the most commonly used quantile measurement is this median value. In the case that a researcher seeks the value associated with the 50th percentile, this diagnostic will express the distance between the median value returned by the non-private CDF and the differentially private CDF.
- Horizontal distance at the 25th & 75th percentiles, which is follow the same logic as the horizontal median measurement above.

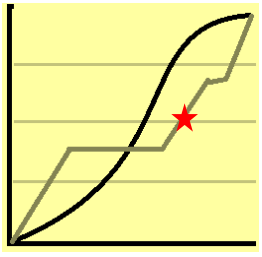


- The L1 norm[3], which computes the area between the non-private CDF and a differentially-private CDF. Which this measurement is naturally normalized and generalizable across CDFs of different data, its utility and purpose are as yet unclear. More work will be done on this diagnostic, as it's the closest we've gotten to a true overall score of a CDF, and is expected to be useful in measuring PDFs.

- Accuracy bounds at the 90%,95%,99% confidence levels, which are computed strictly via mathematical theory, based on user-defined epsilon values and the structure of the CDF function in question. However, these bounds do not hold after post-processing. Since differentially-private CDF computation functions often gain in utility with post-processing, there is a tradeoff between the empirical accuracy of the CDF and the precision of this accuracy measurement.
- An accuracy bound at a user-defined confidence level. See above.



- The median given by the differentially private CDF. Over many iterations and comparison with the true median of a dataset, this helps reveal bias and error in a differentially-private CDF function.



[2] See footnote (2).
[3] See footnote (1)

4.2 The CDFtestTrack[4]

The "CDFtestTrack" function performs each diagnostic on a user-defined amount of iterations of the differentially private CDF/PDF output of the function it is fed. Because differentially private computations have randomness, each one of these curve-iterations is unique, but restricted to bounds determined by the epsilon value. The seed that defines each curve's random error is recorded and reported, so any test can be reproduced by user specification.

4.3 CDFtest[5]

The "CDFtest" function takes in multiple epsilon values, functions, and datasets and feeds them into "CDFtestTrack". It then takes "CDFtestTrack"'s several outputs and organizes them into matrices, statistical analyses, and graphic outputs. Specifically, the "CDFtest" function outputs:

- A time-stamped index and commented description of the function's arguments in a comma-separated values (.csv) log.
- An excel sheet containing mean and median diagnostic scores from the test, categorized by argument, and a list of seeds necessary to reproduce specific results.
- A portable document format (.pdf) file with many CDF and PDF graphs, each showing a separate combination of dataset, epsilon, and function, with two iterations of the same differentially private CDF or PDF function overlaid on a non-differentially private version of that same CDF or PDF. Also in the file is a boxplot for each combination of epsilon value and diagnostic function, juxtaposing all datasets/function combinations. Notably, the median plots depict both error and bias, as they're overlaid with the true median.

We use the CDF testing suite to visualize and empirically validate the performance of differentially private algorithms in typical use-cases. Specifically, the suite has helped to show the benefits of tree-based computations through error evaluation, the effects of enforced monotonicity on various differentially private CDFs, and the interaction of domain size, data-length, and epsilon choice.

4.4 The CDF Testing Suite's design principles

The CDF Testing Suite was designed under the following principles, which can be applied to future testing suite designs for other differentially private statistics. These are as follows:

Modular: We designed the system in two major components: the executer ("CDFtestTrack") and the wrapper ("CDFtest"). Users wishing to reproduce individual results or test on single parameter combinations can bypass the wrapper to use the more streamlined executor function. In theory, this also implies that the wrapper function could give useful output if supplied with any function set outputting results similar to what the wrapper expects.

[4] API in Appendix 1
[5] API in Appendix 1

Customizable multiple simultaneous input: We designed the "CDFtest" function as a wrapper to the internal CDFtestTrack code. Users can supply several datasets, functions, and epsilon values, and does the work of applying every possible combination of these parameters to the CDFtestTrack. This also allows for testing output on these different parameter combinations to be printed on a single pdf file and a single csv file for direct comparison.

Created per social science usage: All error-measurements taken and reported by the testing suite are designed to be directly relevant to social science usage, so testing results imply expected performance in application.

Always based on true CDFs/PDFs: The diagnostics outputted by the testing suite are generally based on deviations from a dataset's true CDF/PDF, as opposed to being measurements taken solely from the CDF/PDF being tested. This means that output does not just describe the output of a CDF/PDF function, but describes that output relative to an ideally useful case. This allows output to be considered in context of its own dataset and in general across output gained from tests on other data.

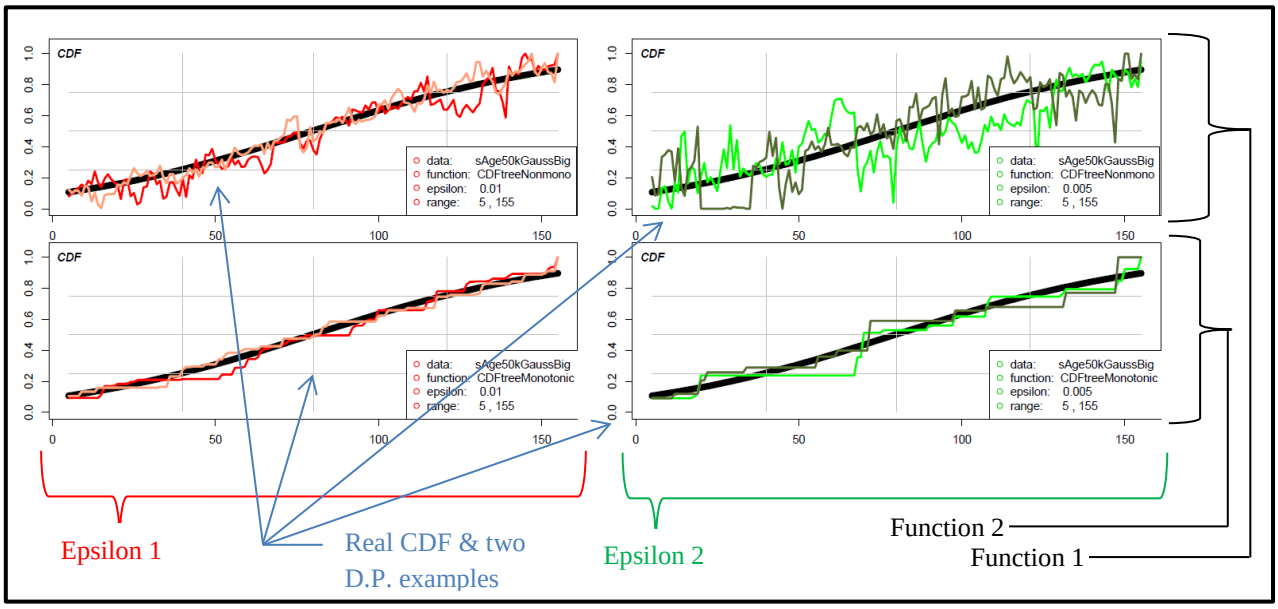
Chronological recording of graphic and statistical information/returns all data interactions. Every execution of the testing suite is timestamped and saved as a set of testing parameters. Using the timestamp as an index causes all execution records to be uniquely identified and automatically sorted chronologically (by most file systems.)

Includes synthetic datasets. While the testing suite is made foremost for usage of real datasets, it also includes several synthetic datasets that can be called upon by name. Datasets chosen are automatically factored into testing. This allows functions to be examined in highly controlled situations, and removes the burden of needing external data to test functions upon. For a list and description of all synthetic datasets currently callable, refer to appendix 1.

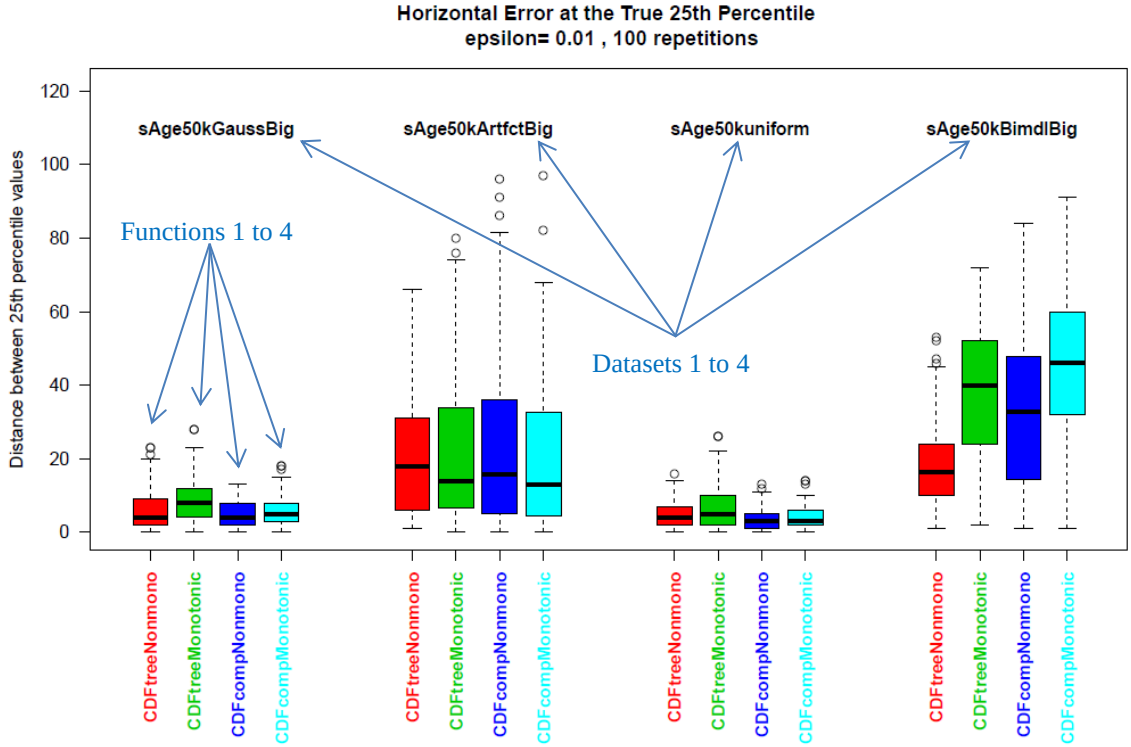
4.5 The CDF Testing Suite's output

We designed the testing suite to output statistical and graphical results. Per execution of the testing suite, all graphic elements are printed to one portable document format (pdf) file. As shown in figure 2, the first section is comprised of graphical depictions of the differentially private CDF functions compared against corresponding true CDFs. Each dataset is given its own page with a grid of CDF images. Each row in this grid depicts CDFs made with a unique CDF-computing function, and the epsilon value varies across columns. Each graph shows two arbitrary iterations of the D.P.-CDF algorithm, at that epsilon, on that dataset. The same is done for corollary PDFs.

As shown in figure 3, the second section of the testing suite's graphical output is a series of boxplots depicting the distributions of the various diagnostics inside CDF test track. Each page represents one combination of epsilon and diagnostic. Within the graph, each function's distribution of results for that diagnostic is represented in its own color, and then clustered by dataset.



2. Graphical output: CDF diagrams



3. Graphical Output: diagnostic result distributions

5. The D.P.-CDF Reference Manual

Conveying the information gained by this research is of paramount importance. The reference manual is designed mainly for social scientists and end users of differentially private systems, and includes practical explanations of differential privacy, of contextual

descriptions of differentially private error, and of the testing suite's diagnostic outputs. In this way, reasonably anyone interested can make use of the project's work. To convey the information most clearly, the three main components of this manual are as follows.

First is an example-laden walkthrough of differentially-private error in CDFs as analogous to statistical error (i.e., sampling error and standard error.) Assuming that social science researchers are familiar with statistical methods, this is the clearest connection to build from. It's important to start with this overview because responsible use of differentially private CDFs/PDFs (and all statistics) relies on knowledge of their transformation.

The next section is on proper interpretation of individual CDFs/PDFs, versus reading multiple iterations of a single CDF/PDF. For example, artificial artifacts can be misleading if a single iteration of a dp-CDF is read alone, but they quickly lose their apparent meaning when two iterations are overlaid. Proper reading of diagnostic outputs from the CDF testing suite may somehow be incorporated, if it's decided that some form of the suite will make its way to end-users.

The remaining section is intended to discuss extreme cases and discuss technical aspects. Certain datasets (i.e., Bernoulli, small n, miniscule or enormous domains, ,etc) could necessitate context-specific interpretation. If it's decided that end users can choose between different CDF-computation functions, this section could include the utility of each method, and which method works best per situation.

The reference manual is foremost designed for continual updating and expansion. While it is currently centered on CDFs, the same styling and logic used thus far can be appropriated to clarify usage of other differentially private statistics.

6. Results, Summary, and Points for Future Research

The main purpose and result of our work was the creation of the testing suite. As such, the use of this testing suite to gain empirical results is mainly a point for future work. That said, we used the testing suite to begin to determine the usefulness of a few CDF-computation algorithms. Early on, this was limited to testing variations on binary-tree based algorithms, such as fixing CDF endpoints to 1, and truncating all CDF y-values to being between 0 and 1. Truncation and endpoint-fixing offered small improvements. Larger questions included comparison of tree-based and naïve CDF algorithms, and applying a monotonicity-enforcing algorithm designed last year.[6] Results generally found that enforcing monotonicity increased utility in D.P.-CDFs, while the utility of binary-tree based CDFs relative to naïvely computed CDFs is more ambiguous. Details and descriptions of these tests are in Appendix 3.

Some specific future questions to be answered with the testing suite include:

- What is the overall relative utility of using any given CDF-computation method versus using another? This can be measured by running simulations (using the different synthetic datasets provided) to determine which functions generate less errors in all categories for the same level of privacy, all things equal.

[6] See Nathan Manohar's DP_Quantiles.R code on Github, function: "smoothVector"

    - We're currently interested in seeing this performance comparison between the 'efficient tree' methods versus the standard 'binary tree' methods. This inquiry can be extrapolated to each level of 'K-ary' trees, not just binary.
- Which functions perform best for finding particular statistics? This can be found in the same way as above, but with specific examination of the errors in just one type of measurement at a time (i.e., which function performs best at locating medians?)
- Under what circumstances is it more efficient to compute CDFs using 'naïve' methods that don't include a tree at all? Thus far, theory and preliminary empirical tests indicate that this occurs with small domains (<100) when examining statistics graphically rightward on a CDF (such at the 75^{th} percentile), but otherwise there isn't empirical evidence that tree-based methods are more useful for finding other quantiles.[7]

Another general interest is measurement and control of range-truncation bias, which is currently user-determined. Potentially, it may be more useful to let the function select its range using a small privacy budget. For this, identical simulations could be run using several different ranges, and their results compared.

Deeper theoretical questions also remain, such as the utility of so-called K-ary trees over existing CDF computation methods. Efficient variance-cancellation in post-processing, and right-to-left tree-node search are also viable alternatives. As each of these theories is coded into an applicable function, each will be testable on the CDFtestingSuite.

In an applied sense, one avenue is to automatically select bins-addition or tree-based CDF algorithms based on metadata. For example, through several experiments, we found that binary-tree based CDF computation methods are consistently less accurate than naïve CDF computation methods for domain sizes roughly less than 150[8]. We conceive that domain-size threshold can be pinned down and used to automatically direct a CDF function on which method to use.

To a small extent, surveys have been conducted on social scientists to determine the variance in decisions made when looking at various differentially private constructions of CDFs on identical data. If this endeavor were expanded, it would directly provide information on the problem of pro per CDF interpretation.

Lastly, we suggest that a controlled and isolated version of CDF testing suite be somehow incorporated into TwoRavens as an educational tool for end users to familiarize themselves with differential privacy.

[7] See Appendix 3 for examples of this line of thinking.
[8] See Appendix 3.

Appendix 1: APIs

A1.1

CDFtestTrack

This function takes in a CDF-computing function and a dataset and measures various types of error in the special CDF generated. It's especially useful as the main component of the CDF test wrapper function defined below.

Arguments:

funct	The differentially-private CDF-generating function to be tested.
eps	Epsilon value for Differential privacy control
cdfstep	The step sized used in outputting the approximate CDF; the values output are [min, min + cdfstep], [min, min + 2 * cdfstep], etc.
setbeta	At a given point in the CDF, the true value is within the accuracy range with probability 1-setbeta
data	A vector of the data
range	The range of the universe as a vector (min, max)
gran	The granularity of the universe between the min and max. It is assumed that the data input is rounded to the granularity
reps	The number of times the combination of CDFfunction, dataset, and epsilon will be tested

Returns: A data frame of with one row and seven columns holding the average and median of reps iterations of each diagnostics' output, using different iterations random errors (from a set distribution defined by epsilon) using a random seed

A1.2

CDFtest

This function is used to loop the CDFtestTrack over several datasets, epsilons, and functions, and to plot the output. This checks performance of CDF generators at different privacy levels.

Arguments:

funclist	A list of CDF-computing functions to be tested on the CDFtestTrack (defined above).
epslist	A vector of epsilon values for differential privacy
cdfstep	The step sized used in outputting the approximate CDF; the values output

	are [min, min + cdfstep], [min, min + 2 * cdfstep], etc.
setbeta	At a given point in the CDF, the true value is within the accuracy range with probability 1-setbeta
datalist	A list containing vectors of data, each to be used in a test.
range	The range of the universe as a vector (min, max). Defined based on user intuition. Setting the minimum too high will bias output upward. The same applies in reverse for a low maximum. However, setting min too low and max too high could reveal the true limits of your data, compromising some privacy.
gran	The granularity of the universe between the min and max. It is assumed that the data input is rounded to the granularity
reps	The number of times the combination of CDFfunction, dataset, and epsilon will be tested
Fnameslist	A list of function names corresponding to the functions being tested; used for labelling the output.
Dnameslist	A list of dataset names corresponding to the functions being tested; used for labelling the output.
setseed	In the function, each combination of data, epsilon, and function is executed with a separate seed, which by default is randomly generated and reported. Users interested in replicating specific results can locate the reported seed and parameter combination to replicate tests.
synthsets	This script generates pre-defined synthetic datasets upon request, and fulling incorporates them into testing. To call them, users should input a character vector containing the names of the sets they desire. For example, synthsets = c("dataset1", "dataset2"). Preloaded datasets are listed in Appendix 2. There are no limits on the amounts of datasets included.

Returns:

A very large list containing:

$**allscores**: a list holding the output of each repetition on each eps, data, funct combination

$**seed**: a vector of the seeds used in the test

$**means**: a data frame holding the mean result (across reps) of each diagnostic on each combination of data, eps, funct

$**medians**: a data frame holding the median result (across reps) of each diagnostic on each combination of data, eps, funct

$**permetric**: a dataframe (ordered by parameter combinations) useful for plotting

A 'pdf' file with several boxplots on the distribution of diagnostic outputs, as well as the distribution of differentially private medians and categorized plots of dp-CDF function output. Each such graph with show two arbitrary CDF iterations.

A 'csv' file containing the mean and median scores of each diagnostic on each combination of data, eps, function and the seedlist for reproduction.

Both the PDF and CSV components are named by/include a time stamp index, in the form of YearMonthDayHourMinute. To locate particular tests, look at the CDFtestindexchart.csv, which automatically records the parameters and index of each test.

A1.3

getMedian

This function returns the median value of a dataset by locating it within a CDF. In the context of the testing suite, this returns medians from D.P. CDFs for comparison against datasets' true medians. It first creates a values vector of possible values (same length as the CDF) by counting from the user-defined minimum to the user-defined maximum by the gran. It then locates the value 0.5 in the CDF vector, and returns the corresponding value in from the values vector.

Arguments:

est	A vector of CDF values (possible values from 0 to 1)
range	The range of the universe as a vector (min, max). Defined based on user intuition. Setting the minimum too high will bias output upward. The same applies in reverse for a low maximum. However, setting min too low and max too high could reveal the true limits of your data, compromising some privacy.
gran	The granularity of the universe between the min and max. It is assumed that the data input is rounded to the granularity

Returns:

A single median value.

A1.4

diffatMed

This function differences the Y-value of the dataset's true median value (the point with a Y-value closest to .5) and the Y-value of the D.P. CDF immediately above or below it.

Arguments:

Y	A vector of CDF values (possible values from 0 to 1)
est	A vector of CDF values (possible values from 0 to 1)
Returns:	A single difference value; possible values are between 0 and 1.

A1.5
horzdiffatMed

This function differences the estimated (D.P.) median found through the getMedian function above to the true median of its original dataset. That difference is returned.

Arguments:

est	A vector of CDF values (possible values from 0 to 1)
Y	A vector of CDF values (possible values from 0 to 1)
range	The range of the universe as a vector (min, max). Defined based on user intuition. Setting the minimum too high will bias output upward. The same applies in reverse for a low maximum. However, setting min too low and max too high could reveal the true limits of your data, compromising some privacy.
gran	The granularity of the universe between the min and max. It is assumed that the data input is rounded to the granularity
Returns:	A single value.

Appendix 2: Synthetic Datasets

Below is a list of pre-made synthetic datasets inside the CDFtest.R code. The datasets can be called by typing the name as written, with quotes, into the "synthsets" argument of the CDFtest function.

Dataset Name	n	Approx. range	gran	Description	Type
"Age2kGaussian"	2000	0,100	1	N(38,25)	normal
"Age10kGaussian"	10000	0,100	1	N(38,25)	normal
"Age50kGaussian"	50000	0,100	1	N(38,25)	normal
"Age50kGaussSmall"	50000	25,80	1	N(45,20)	normal
"Age50kGaussBig"	50000	0,170	1	N(45,20)	normal
"Age2kArtifact"	2000	0,100	1	9 bins filled + Pois(38) as 8% of n	sparse
"Age10kArtifact"	10000	0,100	1	9 bins filled + Pois(30) as 8% of n	sparse

"Age50kArtifact"	50000	0,100	1	9 bins filled + Pois(30) as 8% of n	sparse
"Age50kArtfctSmall"	50000	25,80	1	9 bins filled + Pois(30) as 8% of n	sparse
"Age50kArtfctBig"	50000	0,170	1	9 bins filled + Pois(55) as 8% of n	sparse
"Age2kuniform"	2000	(truncation)	1	Uniform in domain	uniform
"Age10kuniform"	10000	(truncation)	1	Uniform in domain	uniform
"Age50kuniform"	50000	(truncation)	1	Uniform in domain	uniform
"Age2kBimodal"	2000	0,100	1	50% N(20,5) 50% N(60,10)	bimodal
"Age10kBimodal"	10000	0,100	1	50% N(20,5) 50% N(60,10)	bimodal
"Age50kBimodal"	50000	0,100	1	50% N(20,5) 50% N(60,10)	bimodal
"Age50kBmdlSmall"	50000	25,80	1	50% N(35,3)	bimodal
"Age50kBmdlBig"	50000	0,160	1	50% N(60,12) 50% N(120,12)	bimodal
"Wage2kGaussian"	2000	0,300000	1k	Abs(N(120k,70k)) rounded to 1000s	normal
"Wage10kGaussian"	10000	0,300000	1k	Abs(N(120k,70k)) rounded to 1000s	normal
"Wage50kGaussian"	50000	0,300000	1k	Abs(N(120k,70k)) rounded to 1000s	normal
"Wage2kArtifact"	2000	0,500000	1k	64 bins filled + Pois(160k) as 5% of n	sparse
"Wage10kArtifact"	10000	0,500000	1k	64 bins filled + Pois(160k) as 5% of n	sparse
"Wage50kArtifact"	50000	0,500000	1k	64 bins filled + Pois(160k) as 5% of n	sparse
"Wage2kuniform"	2000	(truncation)	1k	Uniform in domain	uniform
"Wage10kuniform"	10000	(truncation)	1k	Uniform in domain	uniform
"Wage50kuniform"	50000	(truncation)	1k	Uniform in domain	uniform
"Wage2kBimodal"	2000	0,100	1k	50% N(20k,15k) 50% N(80k,15k)	bimodal
"Wage10kBimodal"	10000	0,100	1k	50% N(20k,15k) 50% N(80k,15k)	bimodal
"Wage50kBimodal"	50000	0,100	1k	50% N(20k,15k) 50% N(80k,15k)	bimodal

Appendix 3: Preliminary Results from the Testing Suite

We used the Testing Suite to find some initial empirical results. The following treatments can be used as rubrics for future testing, both in their structure and their findings.

Monoticity-enforcing post-processing[9]

Enforcing monotonicity on CDFs makes intuitive sense, since a downward-sloping segment of a CDF indicates a negative probability. Graphically, differentially-private CDFs can be seen to match their corresponding non-private CDFs better when they are 'monotonized', and this improvement can be measured numerically with the testing suite. We ran a simulation on monotonized and non-monotonized versions of two functions on 4 differently-shaped datasets[10] 100 times each with all else equal (eps=.005, n=50k), and measured the vertical[11] errors found at the 25th/50th/75th percentile. In the tables below, we divided the errors from monotonized CDF approximations by errors found from non-monotonized CDF approximations and averaged this over the 100 iterations. In most cases, the resulting statistic is less than 1, indicating the monotonicity-enforced CDF algorithms perform better. The instances where monotonicity gave worse results are highlighted in red.

Binary-Tree

All scores are relative error	"Age50kuniform" (small)	"Age50kuniform" (med)	"Age50kuniform" (big)
25th percentile	0.750	0.968	0.652
50th percentile	0.931	0.903	0.716
75th percentile	0.250	1.063	0.831

All scores are relative error	"Age50kGaussSmall"	"Age50kGaussian"	"Age50kGaussBig"
25th percentile	0.803	0.964	0.600
50th percentile	0.755	1.130	0.615
75th percentile	0.978	0.984	1.492

[9] See Nathan Manohar's DP_Quantiles.R code on Github, function: "smoothVector"

[10] Dataset descriptions can be found in Appendix 2, function APIs can be found in Appendix 2.

[11] Refer to page 3 for an explanation of vertical and horizontal errors.

All scores are relative error	"Age50kBmdlSmall"	"Age50kBimodal"	"Age50kBmdlBig"
25th percentile	0.888	0.982	0.735
50th percentile	0.661	1.032	0.871
75th percentile	1.085	0.951	0.794

All scores are relative error	"Age50kArtfctSmall"	"Age50kArtifact"	"Age50ArtfctBig"
25th percentile	0.825	0.762	1.460
50th percentile	1.212	0.714	1.260
75th percentile	0.897	0.918	1.152

Histogram-bin composition

All scores are relative error	"Age50kuniform" (small)	"Age50kuniform" (med)	"Age50kuniform" (big)
25th percentile	1.000	1.000	1.000
50th percentile	0.931	1.100	1.000
75th percentile	1.000	1.429	1.229

All scores are relative error	"Age50kGaussSmall"	"Age50kGaussian"	"Age50kGaussBig"
25th percentile	0.833	0.857	1.026
50th percentile	1.333	1.000	1.030
75th percentile	1.889	1.000	0.971

All scores are relative error	"Age50kBmdlSmall"	"Age50kBimodal"	"Age50kBmdlBig"
25th percentile	0.833	1.200	1.167
50th percentile	1.000	1.000	1.000
75th percentile	0.895	0.533	3.143

All scores are relative error	"Age50kArtfctSmall"	"Age50kArtifact"	"Age50ArtfctBig"
25th percentile	1.100	0.906	0.700
50th percentile	1.061	1.280	1.250
75th percentile	1.000	0.556	1.000

The utility improvements of post-processing have yet to be determined analytically, so support for their continued use comes solely in the form of empirical research. Tests such as the one above should be done on all new CDF computation algorithms, such that we can know for certain whether or not we should be applying that post-processing step.

Binary-Tree versus Histogram-Bin Composition

We haven't found binary tree-based CDF-computation algorithms (Method A) consistently performing better than the histogram-based CDF computation algorithms (Method B). Our most robust comparison of these methods measured distance at the 25th, 50th, and 75th percentiles between true CDFs and their dp-CDF approximations. Method A and Method B were both applied to nine synthetic datasets 100 times each, using the same epsilon and n values (.005, n=50k). All nine datasets had identical granularity. Each had one of three shapes (uniform, bimodal, Gaussian) and one of three domain sizes (narrow, medium, and broad). The domain expansion can be thought of as an increase in the number of bins into which the data is sorted, and theory shows that Method A should out-perform Method B by an increasing margin as domain size increases. We checked this by averaging the 100 vertical[12] distances between the 25th/50th/75th percentile value given by the true CDF and the 100 corresponding percentile values given by dp-CDF approximations, for each function on each dataset. We then divided the average error from Method A by the average error from Method B.[13] We should expect that when approximating CDFs of datasets with increasingly large domain sizes, this statistic should decrease. This effect should occur most markedly at the 75th percentile, when errors compound. Low scores imply that the binary tree offers higher utility, while a score of 1 implies that both methods offer equal utility for that statistic. The results can be seen in the following table; dataset descriptions can be found in Appendix 2.

As seen here, there is no clear trend of the Method A (binary trees) performing comparatively better in a datasets with larger domains. One instance of performance

[12] Refer to section 4 for an explanation of vertical and horizontal errors.

[13] Method A and B both have post-processing monotonicity enforced. Without monotonicity enforced, these relative scores don't change much, errors slightly increase universally, on average.

trends supporting the theory is highlighted in yellow. There is only one instance here where Method A strictly outperformed Method B, and that's highlighted in green.

'Vertical'

All scores are relative error	"Age50kuniform" (small)	"Age50kuniform" (med)	"Age50kuniform" (big)
25th percentile	5.250	10.167	6.142857
50th percentile	4.154	5.091	8.00
75th percentile	4.000	2.500	1.140

All scores are relative error	"Age50kGaussSmall"	"Age50kGaussian"	"Age50kGaussBig"
25th percentile	4.900	8.833	5.571429
50th percentile	4.625	8.714	4.000
75th percentile	2.650	4.428	0.874

All scores are relative error	"Age50kBmdlSmall"	"Age50kBimodal"	"Age50kBmdlBig"
25th percentile	3.200	4.583	7.143
50th percentile	3.000	5.818	6.750
75th percentile	3.000	7.250	2.455

From the simulation, we also found *horizontal* errors created by each function, and can compare these in the same way as above. While it's less clear how these results relate to theoretical expectations, we can assume that horizontal errors should generally decrease as vertical errors decrease. Also, measuring horizontal error is useful in application, as it expresses how far off an estimated quantile value is from the truth.[14] As seen below, the binary tree method performs relatively better when measured in this way (compared to vertical errors), but still doesn't show clear trends, nor support theoretical expectations very well.

[14] See section 4 for details.

'Horizontal'

All scores are relative error	"Age50kuniform" (small)	"Age50kuniform" (med)	"Age50kuniform" (big)
25th percentile	1.375	0.981	1.231
50th percentile	1.727	1.24	1.465
75th percentile	1.073	0.692	0.701

All scores are relative error	"Age50kGaussSmall"	"Age50kGaussian"	"Age50kGaussBig"
25th percentile	1.555	0.455	1.158
50th percentile	3.200	2.072	1.158
75th percentile	1.121	7.600	0.715

All scores are relative error	"Age50kBmdlSmall"	"Age50kBimodal"	"Age50kBmdlBig"
25th percentile	0.974	0.640	1.100
50th percentile	5.100	3.640	1.433
75th percentile	2.159	1.584	1.071

The next step in this research question is to run simulations on datasets with increasingly large domains. If it's unclear that binary-tree based algorithms perform consistently better than histogram-bin based CDF algorithms, we should focus more on newer types of trees, or newer types of CDF algorithms altogether.

Appendix 4: File Locations

All R codes are stored on Github as follows:

IQSS/PrivateZelig/summer2015/cdfs

CDFfunctions
This file contains two main functions, yourCDFfunction and composeCDF.
yourCDFfunction computes a CDF vector from a binary tree, and has flags for monotonicity, truncation and fixing.
composeCDF computes a CDF vector by composing the bins of a DP histogram.

CDFtester
This file is a UI for utilizing the CDFtestingSuite. The functions are ready to use, with some arguments pre-set. Change the arguments and set the directory to test your CDF.

CDFtestingSuite
A system for determining the utility and privacy of DP-CDF algorithm functions. This file contains two main functions, CDFtest and CDFtestTrack. CDFtest is a wrapper to CDFtestTrack, which runs several error tests on the output of a differentially private function.

PostCDFstats
This file defines a set of functions to be applied to CDF vectors. In differential privacy, this allows the computation of several differentially private statistics with just the privacy budget used to make the CDF. These functions can be called by using the "CDFtestTrack" function inside the CDFtestingSuite code.

IQSS/PrivateZelig/DP_modules
Histogram
This file defines functions needed for computing differentially private histograms, and is sourced by the CDFfunctions file.

For all other related files, please contact Kimia Mavon at kmavon.g.harvard.edu
This document will be updated with a directory once all files have been cleared for publicizing.