A User's Guide to the Percentile Performance Index Software

Written by Ian Fillmore and Eric Nielsen

November 15, 2011

1 Overview

This Percentile Performance Index software (PPI) implements the method for evaluating teacher performance proposed by Barlevy and Neal [1]. See their paper for details on the theory underlying the methodology. The user supplies PPI with data on individual students' test scores and a variety of additional covariates (such as previous test scores, family income, etc.). PPI uses these covariates to transform each student's test score into a "percentile score." Intuitively, if Bobby gets a percentile score of 0.53, that means that he scored higher than 53% of students who look like him based on the covariates provided. Percentile scores are assigned to each student in a given grade in a given year. That is, they are student-grade-year specific. In short, you provide PPI with student level data and it calculates a percentile score for each student.

PPI is implemented in R, a widely used open source statistical program. For more information, visit http://www.r-project.org. PPI currently accepts STATA (.DTA) datasets and outputs its results as both an R object (.Rdata) and a STATA dataset. PPI comes with the following files in the same folder:

- Functions_PPI.r — This file contains all the functions used to perform the analysis. **Editing this file in any way is strongly discouraged.**
- PPI_Index_Create.r — This file is for users to edit. This is where you set parameters, supply the data, and tell PPI how to run the analysis.
- PPI_Index_Create(basic template).r — This file is merely a copy of PPI_Index_Create.r. It is here so that when you change PPI_Index_Create.r to suit your needs, you have a fresh version available to go back to.
- PPI_Index_Create(multiple template).r — This file is nearly the same as PPI_Index_Create(basic template).r. However, it has a simple demonstration of how to modify the code to allow you to run the analysis on multiple grades, years, and subjects (see Running PPI For Multiple Grades, Years, and Subjects).

As explained in detail below the basic steps to using PPI are

1. Install the packages `foreign`, `gap`, `SparseM`, and `quantreg` if they are not already installed. These packages only need to be installed the first time you use PPI.
2. Format your data in the way PPI is expecting it. The data should be saved as a STATA (.DTA) dataset.
3. Run the file PPI_Index_Create.r. You will need to modify this file to make it suitable for your needs.
4. Run diagnostics and do some basic error checking to insure that PPI ran correctly.

2 Supplying the Data

2.1 Data Format

PPI expects the user to supply the data in "wide" format. This means that each row of the dataset must correspond to a particular student, and there is only one row per student. It is essential that all the information for any one student be contained on the same row. Each column will contain a particular variable on that student. If the dataset has test scores for students over several years, then there should be a separate variable for each year (for example *math2007*, *read2007*, *math2008*, etc.). At a bare minimum, there must be at least one column containing student test scores for a particular year, say for the year 2008, and one column containing the student's grade level in that year. Usually the dataset will include some demographic information such as race, free/reduced lunch status, gender, etc. It is very helpful if there are also student test scores for the previous year or two. Remember that the data must be in wide format, so all of this information should be on a single row.

If you want to run the analysis for several grade-years, all of the relevant variables for each student can be included in one wide data set. For example, suppose that you want to analyze 4th grade math scores in 2007 and 2008. Suppose further that the model you want to use for the quantile regressions is

$$\text{mathScore}_{i,2008}(q) = \beta_0 + \beta_1 \text{mathScore}_{i,2007} + \beta_2 \text{gender}_i + \beta_3 \text{lunchStatus}_{i,2008}$$

where i indexes students and q refers to a particular quantile. To estimate this model for 2008 you must have the following variables in a wide format file where each row corresponds to one student: the student's grade in 2008 (4 in this example), math score in 2008, math score in 2007, gender, and free/reduced lunch status in 2008. The data needed to estimate the model for 2007 would be: grade in 2007, math score in 2007, math score in 2006,

gender, and free/reduced lunch status in 2007. You can use PPI to estimate both models from the same data set provided that the data set has all of the required variables for each model and that the format is wide (so all of these variables for a given student are on the same row).

Despite the fact that PPI is expecting the data in wide format, remember that percentile scores are student-grade-year specific. The names of the variables in the dataset are irrelevant with one exception. PPI expects the grade variable to be named "grade2008" if it's for 2008, "grade2007" if it's for 2007, etc. "grade2008" would contain the student's grade for the year 2008 (e.g. the variable would have the value 4 if the student was in the 4th grade in 2008). Grade should never be supplied as a covariate. Rather, PPI does the analysis separately for only one grade in one year at a time.

PPI expects to receive the data as a STATA dataset (.DTA format). If your data is not in .DTA format, there are several utilities which can convert most common file types to .DTA. StatTransfer is very easy to use while STATA itself has built in utilities as well. Please consult STATA documentation online for help in using these utilities.

2.2 Missing Data

It may be the case that some students are missing data on some of the variables you want PPI to use in calculating percentile scores. If you leave the data as missing, PPI will be unable to calculate percentile scores for those students–it will return missing values for them instead. One way around this is to create a dummy for whether a particular variable, say `lunchStatus`, has a missing value. Then set the missing values for `lunchStatus` to zero. You would need to make these changes to the dataset before reading it into PPI. Of course, this fix will only work for covariates. If a student is missing this year's score, `math2008` in our example, then that student cannot and will not receive a percentile score.

Note that some potential problems can creep up when creating missing dummies. Suppose that the only reason some students are missing data is that they are recent move-ins. Then these same students will also be missing data on last year's math test. If you include `lunchStatus` *and* last year's math score as covariates and create a separate missing dummy for each, these dummies will be identical and PPI will quit with an error about the design matrix being singular. If this happens, check to see if some of your missing dummies are redundant. You should also make sure that you are not missing data on a certain covariate for only one student (a dummy variable for a single student will result in strange percentile scores for that student).

3 Running the Program

All of the calculations are called from the file PPI_Index_Create.r which contains extensive documentation. This file organizes things in several steps. As you walk through these steps, you will be assigning/changing the following parameter values:

- `spacing`
- `pred_flag` and `obs_flag`
- `out.name`
- `path`
- `spec`
- `grade`
- `year`
- `outlist`–optional
- `test`–optional

In addition to changing these parameter values, you will also need to change all of the filepaths to the appropriate locations on your machine.

Preliminaries

You should have the files PPI_Index_Create.r and Functions_PPI.r in the same folder. If they are not already installed, you need to install the packages `foreign`, `gap`, `SparseM`, and `quantreg`. These preliminaries only need to be done once, when PPI is first used.

It is recommended that anyone using PPI take at least a short glance at the section A More Detailed Look Under the Hood. At a minimum, you should know the names of the functions defined in Functions_PPI.r to insure that you don't use those names for anything else you are doing. Also, note that PPI_Index_Create.r begins with `rm(list=ls(all=TRUE))` in an attempt to make sure it starts with a clean slate.

Step 1

Give PPI access to the functions defined in the file Functions_PPI.r using the command

`source("`*`filepath`*`/Functions_PPI.r")`

where *filepath* is the folder containing Functions_PPI.r on your machine. For example, if the R code for PPI were located on my machine in the folder /Users/ Administrator/ Documents/ppi_software, I would type

```
source("/Users/Administrator/Documents/ppi_software/Functions_PPI.r")
```
[1]

WARNING: Changing the code in Functions_PPI.r can cause PPI to calculate percentile scores incorrectly. The file PPI_Index_Create.r is intended as the interface for users. Functions_PPI.r should be left alone.

Step 2

Specify how dense of a grid you want to use for the quantile regressions by setting the parameter `spacing`. PPI uses a series of quantile regressions to estimate each student's individual test score distribution. The quantiles for these regressions are evenly spaced with their distance apart equal to `spacing`. For example, with `spacing` set to 0.03, PPI would run 33 quantile regressions at $q = 0.02, 0.05, 0.08, \ldots, 0.95, 0.98$. Choosing a smaller spacing improves the accuracy of the interpolation (see A More Detailed Look Under the Hood), but also increases computation time. Additionally, it increases the chances of having some quantile crossings, which is undesirable. Quantile crossings are explained in the section Error Checking. We found a spacing of 0.03 to be satisfactory for the data from Chicago Public Schools (CPS)–roughly 25,000 students per grade-year. If your dataset is much smaller, you may need to increase `spacing`.

Set the two error tolerance parameters, `pred_flag` and `obs_flag` (see Diagnostics, Error Checking).

Step 3

Save the name for the output files as a character string `out.name` and the location where the output should be written as a string `path`. For example, I might assign `out.name` and `path` something like

```
out.name <- "myOutputGr4Yr2008"
```

```
path <- "/Users/Administrator/ppi_software/output"
```

This will tell PPI to save the results in the folder "output" and to name those output files "myOutputGr4Yr2008". Notice that the folder where you save the output files does not need to be in the same place as the PPI code. You also have full flexibility with what you want to call the output files. However, if you run PPI for 4th graders in 2008 and then want to run the analysis for 5th graders, you will need to change `out.name`. Otherwise, PPI will overwrite the results for the 4th graders. Thus, it is advisable to include a grade

[1] All the examples in this document are for a Mac or Unix machine. On a Windows machine, the file path would look something like C:\Documents\ppi_software\Functions_PPI.r

and year indentifier in `out.name` if you will be running PPI for several different grades and/or years (see Running PPI For Multiple Grades, Years, and Subjects).

Step 4

Read in the data. It will be imported into an R data frame. The command looks like

```
data.all <- read.dta("filepath/myData.dta")
```

where *filepath* is the location of the Stata dataset on your machine and `myData.dta` is the STATA dataset you want to use. Notice that the data does not need to be in the same location as the PPI software. Thus, the command might look like

```
data.all <- read.dta("/Users/Administrator/Documents/datasets/myData.dta")
```

Step 5

Define your specification for the quantile regressions and give it a descriptive but short name. (This name will be used in the name of the percentile score variable in the final results. For compatibility with STATA, avoid using periods in the name of the specification.) The specification amounts to a list of strings stored as a data frame. The first element of the list will be the dependent variable (the test score which we will convert into a percentile score). The rest of the elements of the list will contain the covariates. These may be previous test scores, demographic variables, etc. The command looks like

```
spec <- as.data.frame(c("testscore","x1","x2","x3"))
```

where `testscore` is the test score we are using to construct the percentile score and `x1`, `x2`, ... are the covariates (of course there may be as many covariates as you'd like). The names you provide PPI in this command should correspond to the names of the variables in the dataset. It is essential that the names of each of these variables is surrounded by quotes. A second line of code

```
names(spec) <- "spec"
```

is just an important bit of housekeeping to make sure that everything is named appropriately. For example, suppose you ran

```
specMath2008 <- as.data.frame(c("math2008","math2007","gender","lunchStatus"))
```

This tells PPI that we will be calculating percentile scores for `math2008` using math scores for 2007, gender, and free/reduced lunch status as covariates. If you used dummies to deal with missing data in your covariates, then you should also include the names of these "missing dummies" in your specification. Lastly, we would need to follow this up with

```
names(specMath2008) <- "specMath2008"
```

Step 6

Provide the grade and year for which you want to calculate the percentile scores. For example

```
grade <- 4
year <- 2008
```

Step 7

This is where you perform the analysis. This is done by calling

```
do_all_analysis(data.all,grade,year,spacing,spec,pred_flag,obs_flag,path,out.name)
```

where `spec` is the specification you defined in Step 5. This function will calculate percentile scores using the data and other information you have provided. You may assign the output of this function a name, say `outlist`, if you want (this will be helpful for doing diagnostics immediately afterward, see Step 8). `outlist` will then be an R list containing the quantile regression results and predictions, information on crosses (see Error Checking), the original dataset with percentile scores added to it, and the specification used—`spec`. Whether or not you assign the results to a name, PPI will save the same R list as a separate file with the name and location you provided in Step 3. The list contained in this output file will be named `output.list`.

`do_all_analysis` will also save the dataset as a STATA data (.DTA) file, with the same name and location provided in Step 5. The new variable containing the percentile scores will be named

pctScore_*spec*_Gr*grade*Yr*year*

where *spec* is the name you gave in Step 5, and *grade* and *year* come from Step 6. This makes it easy to tell not just which grade-year these scores apply to, but also which regression specification was used to calculate them. However, if you make the name of your specification, *spec*, too long, R will have to truncate the name of the variable.

If `pred_flag` and/or `obs_flag` are violated, PPI will not produce percentile scores. Rather it will return a list containing the error flags and a matrix of the crosses that occurred (see Error Checking). It will also save this list in the folder you specified for output and with the name

*out.name*_errorFile.Rdata

where *out.name* is the string provided in Step 3. For more information, see Error Checking below.

Step 8

Perform some diagnostics to make sure that the resulting percentile scores are uniformly distributed between zero and one. This is done with the command

```
test <- diagnostics(outlist,qq.plot=TRUE,print=TRUE)
```

where `outlist` is the output from `do_all_analysis`. `qq.plot` and `print` are options that control whether a uniform quantile plot should be plotted and whether the output of diagnostics should be printed. For more information, see Diagnostics.

4 Diagnostics

The function `diagnostics` performs some basic checks on the percentile scores. Its only required argument is `outlist`, the output of `do_all_analysis`. It simply calculates some statistics on the percentile scores. These scores should be uniformly distributed by construction, so it prints out the actual and theoretical mean, variance, minimum, and maximum. These should be pretty close to each other. The function also provides a summary of the number of crosses that occurred (see, Error Checking).

`diagnostics` has an option to plot a qqplot of the data against a standard uniform distribution (the default is to omit the plot). To have it do the plot, you add `qq.plot=TRUE`. The values should lie almost identically on the diagonal. If you do not want to see the plot, leave `qq.plot=FALSE`. Similarly, you can turn off printing of the statistics and crossings summary by setting `print=FALSE`—the default is `print=TRUE`.

5 Error Checking

PPI uses quantile regression to estimate the percentile scores for each student. By its nature, quantile regression does not always have a unique solution in much the same way that the median of an even set of numbers is not unique. Every time PPI runs a quantile regression and the solution is not unique, R will produce a warning. After running `do_all_analysis`, you can see these with the command `warnings()`.

There is a second potential problem called crossing. PPI estimates a distribution of test scores for each student using a series of quantile regressions. For example, with `spacing` set

to 0.03, it would do this by estimating quantile regressions at $p = 0.02, 0.05, \ldots, 0.95, 0.98$. This comes out to be 33 quantile regressions. For each regression, Johnny–and every other student–has a fitted value. If you lined these fitted values up, they should be increasing. However, because the quantile regressions are estimated independently, there is nothing that guarantees they will be increasing. If, for example, Johnny's fitted value from the $p = 0.14$ regression were 25.2 but his fitted value from the $p = 0.17$ regression were 23.6, this would be a crossing. $\hat{y}_{.14} > \hat{y}_{.17}$ even though $0.14 < 0.17$. This is possible because the two quantile models are estimated independently of each other.

Two parameters, `pred_flag` and `obs_flag`, are set by the user to help catch crossings. While we can't guarantee that crossings will never happen, we want them to happen very rarely. Setting `pred_flag` to, say, 0.01 will cause the program to terminate and not produce any output if the total number of crossings is greater than one percent of the total number of predictions PPI made. Similarly, setting `obs_flag` to 0.01 will cause the program to terminate if more than one percent of the students in the dataset have at least one crossing in their estimated distributions. Using data from CPS, we set `pred_flag` and `obs_flag` to 0.005 without any problems. It is likely that if you set spacing too small relative to the size of your dataset, then crosses would become more of a problem.

Whether the program completes normally, or is terminated due to a crossing violation, among the results will be a matrix called `crosses`. This matrix will be of dimension $N \times (K-1)$ where N is the number of students and K is the number of quantile regressions (the length of `cuts`). `crosses` consists mainly of zeros with a one in position (i, j) indicating an occurrence of a crossing for student i between quantile j and $j + 1$.

6 Running PPI For Multiple Grades, Years, and Subjects

The function `do_all_analysis` will produce percentile scores for a single subject in one grade-year. Perhaps you want to calculate percentile scores for 4th graders in 2008 for both reading *and* math. You would first need to create two specifications, one for reading and the other for math. For example, you could type

```
specMath2008 <- as.data.frame(c("math2008","math2007","black","hisp"))

specRead2008 <- as.data.frame(c("read2008","read2007","black","hisp","ell"))
```

where `read2008` is the name of the variable containing reading scores for 2008 and `ell` is a dummy variable for English Language Learners. Notice that `ell` was included in the reading specification and not the math specification. This is completely alright. Now in Step 7, you would need to run `do_all_analysis` twice. When doing so, it is essential that you change `out.name` prior to running `do_all_analysis` the second time. If you don't,

PPI will replace the first results with the second set of results. Thus the command would look something like

```
out.name <- "myOutputMath"

do_all_analysis(data.all,4,2008,spacing,specMath2008,pred_flag,obs_flag,path,out.name)

out.name <- "myOutputRead"

do_all_analysis(data.all,4,2008,spacing,specRead2008,pred_flag,obs_flag,path,out.name)
```

Perhaps you want to create PPI scores for multiple grades or even multiple years. The simplest way to do this is to make multiple calls to `do_all_analysis`, changing the relevant parameters each time. For example, if you wanted to run PPI on math scores in 2008 for 4th, 5th, and 6th graders, you could simply call `do_all_analysis` three times, changing `grade` each time. An example of how to do this is found in the file `PPI_Index_Create(multiple template).r`.

Once you are comfortable with how `PPI_Index_Create.r` works, you can use the flexibility of R to automate much of this by using loops in Step 7.

7 A More Detailed Look Under the Hood

This section explains the functions that are called by `do_all_analysis`. All of these functions are located in the file Functions_PPI.r.

WARNING: Changing the code in Functions_PPI.r can cause PPI to calculate percentile scores incorrectly. The file PPI_Index_Create.r is intended as the interface for users. This section is provided so you have a clear idea of what PPI is doing. However, the code in Functions_PPI.r should be left alone.

`cuts.create(spacing)`

This function creates a vector `cuts` with elements evenly spaced between zero and one. The distance between the elements of `cuts` is determined by `spacing`. 0.5 is always included, zero and one never are, and the lowest and highest values of `cuts` are always more than `spacing`/2 away from zero and one respectively. For example, if `spacing` were 0.03, `cuts` would be

$$\texttt{cuts} = \{0.02, 0.05, \ldots, 0.47, 0.50, 0.53, \ldots, 0.95, 0.98\}$$

The elements of `cuts` are the quantiles at which PPI will run the quantile regressions. The function will not produce a vector `cuts` that has fewer than 7 elements.

`subset.grade(data.in,grade.cut,year)`

This function simply takes the main data file, and selects out those observations who were in grade `grade.cut` in year `year`. More specifically, it selects those observations for whom the variable "grade*year*" is equal to `grade.cut`. Thus, if `grade.cut` is 4 and `year` is 2008, then it selects out only those students who were in the 4th grade in the year 2008 (i.e. those for whom the variable "grade2008" is 4).

`run.qregressions(data.in,cuts,spec,rq.method="br")`

This function runs a series of quantile regressions at the quantiles defined in `cuts`. For example, with `spacing` set to 0.03, there will be 33 regressions run at quantiles 0.02, 0.05, ..., 0.95, 0.98. It uses `spec` to know what the variables in the quantile regression should be. `rq.method` is an option to change the algorithm for computing the regression, although for this application "br" (the default) should suffice. The function returns the results of all these regressions. For more information on how R performs quantile regression, see the R documentation for the function `rq()`.

`inverse.quantiles(data.in,qr.predictions,cuts,spec,year,grade)`

This function takes `qr.predictions`, the predictions obtained from applying the R command `predict()` to the output of `run.qregressions`, and computes an interpolated quantile for each student. This is really at the heart of PPI. The method we use is an adaptation of the methods presented in [3] and [2].

To illustrate, suppose Lisa's predicted quantiles were

Cuts	0.02	$\cdots$	0.17	0.20	0.23	$\cdots$	0.77
Predictions	13.5	$\cdots$	29.8	31.1	32.1	$\cdots$	85.0

The quantile regression for 0.17 resulted in a fitted value for Lisa of 29.8 based on her covariates. In other words, we are predicting that 17 percent of students who look like Lisa will have a test score at or below 29.8. If she had scored exactly 29.8, the function would give her a percentile score of 0.17. Now suppose that Lisa's actual score were 32. This lies in between the values of 31.1 and 32.1. In order to assign Lisa a percentile score, PPI locally approximates Lisa's true CDF with a normal distribution. It does this by defining $\lambda \equiv \frac{32-31.1}{32.1-31.1}$, $\underline{z} \equiv \Phi^{-1}(0.20) = -0.842$ and $\overline{z} \equiv \Phi^{-1}(0.23) = -0.739$ where $\Phi(\cdot)$ is the standard Normal CDF function. Then PPI calculates $\hat{z} = \underline{z} + \lambda(\overline{z} - \underline{z}) = -0.749$ and assigns Lisa a percentile score of $\Phi(\hat{z}) = 0.227$. Thus, PPI estimates that Lisa scored better than 22.7 percent of students like her.

This interpolation method runs into trouble if a student's score is below his or her lowest predicted quantile, or above his or her highest. Continuing with the example of Lisa, suppose her predicted quantile at cut 0.02 were 13.5, but that Lisa scored a 12. How should we do our interpolation?

The function solves this problem in the following way. It translates Lisa's score into standard deviation units, asks how far below 13.5 she scored *in standard deviation units*, and feeds that number into the Normal CDF. How does PPI translates Lisa's score of 12.0 into standard deviation units? It uses three different pairs of quantiles to estimate the standard deviation assuming normality. In the example with Lisa, the three pairs would be (0.11,0.89), (0.23,0.77), (0.35,0.65).[2] PPI always chooses pairs that are symmetric around the median. In our example, $\hat{\sigma}_2 = \frac{85.0-32.1}{\Phi^{-1}(0.77)-\Phi^{-1}(0.23)}$. $\hat{\sigma}_1$ and $\hat{\sigma}_3$ are calculated similarly. We call the average of these three $\hat{\sigma}$. Then Lisa's percentile score is given by

$$\Phi\left(\frac{12.0-13.5}{\hat{\sigma}}+\Phi^{-1}(0.02)\right).$$

In words, we calculate how far below the lowest cut Lisa scored in standard deviation units. Then we feed that back into a Normal CDF to get her percentile score.

If instead Lisa's actual score were higher than her estimated 0.98 quantile, the function would perform the same procedure to estimate $\hat{\sigma}$ and would calculate everything off of the 0.98 quantile instead of the 0.02 quantile. In effect, the function simply adds a Normal tail to the extremes of Lisa's estimated distribution.

`find_qr_crossings(predictions)`

This function takes `predictions`, the predictions obtained from applying the R command `predict()` to the output of `run.qregressions`, and finds any crossings that occur. It returns a matrix consisting mostly of zeros. A one in position (i, j) means that there was a crossing for student i between the quantile regressions corresponding to the jth and $j+1$st elements of `cuts`.

`get_qr_crossing_flags(crosses,pred_flag=.01,obs_flag=.01)`

This function uses `crossings`, the output of `find_qr_crossings`, to calculate the number of crossings that occur in total as well as the number of students who have at least one

[2]The actual rule for choosing the three pairs is to calculate the lower member of each pair using `cuts[floor(len*.25)]`, `cuts[floor(len*.50)]`, and `cuts[floor(len*.75)]` where `len` is the index of the 0.50 quantile. For example, with `spacing` = 0.03, `len` would be 17. The higher member of each pair is just one minus the lower member.

crossing. It returns a flag for whether the fraction of the elements of `crosses` that equal one is greater than `pred_flag`. It also returns a flag for whether the fraction of students with at least one crossing is greater than `obs_flag`.

```
do_all_analysis(data.in,grade,year,spacing,spec,pred_flag,obs_flag,path,out.name)
```

This function is the only one called directly from PPI_Index_Create.r. It calls all the previous functions in their proper order. If `pred_flag` and `obs_flag` are not violated, then it saves a STATA dataset named *`out.name`*.dta in the folder `path`. This dataset looks like `data.in` with one extra variable containing the percentile scores. Note, however, that this output dataset will only contain students who were in grade `grade` during year `year`. In addition to creating the STATA dataset, `do_all_analysis` also saves an R list in a file named *`out.name`*.Rdata in the folder `path`. This list is named `output.list` and contains the regression results, predictions, `crosses`, and the output dataset as an R dataframe. The regression results include the estimated coefficients from each quantile regression. The predictions are a matrix with a row for each student and a column for each quantile regression. Thus, element (i, j) of this matrix would be student i's predicted quantile for quantile regression j.

If `pred_flag` and/or `obs_flag` are violated, `do_all_analysis` does not produce a STATA dataset. Rather, an R list called `output.list` is in a file named *`out.name`*`_errorFile`.Rdata in the folder `path`. `output.list` will contain the crossing flags (the first one corresponds to `pred_flag` and the second to `obs_flag`) and `crosses`.

```
diagnostics(output.list,qq.plot=FALSE,print=TRUE)
```

This function performs some simple checks that the percentile scores are uniformly distributed. It calculates the mean, variance, minimum and maximum of the percentile scores and prints them along with the theoretical moments of the UNIF(0,1) distribution. Furthermore, it can plot a quantile-quantile plot of the percentile scores against the uniform distribution. Lastly, it will print the number of crossing violations. It returns a list containing the moment comparisons and the crossing summary.

References

[1] Gadi Barlevy and Derek Neal. Pay for percentile. Working Paper 17194, National Bureau of Economic Research, July 2011.

[2] D. Betebenner. Norm- and criterion-referenced student growth. *Educational Measurement: Issues and Practice*, 28(4):42–51, 2009.

[3] Ying Wei and Xuming He. Conditional growth charts. *The Annals of Statistics*, 34(5):pp. 2069–2097, 2006.