Journal of Statistical Software

September 2009, Volume 31, Code Snippet 3. *http://www.jstatsoft.org/*

The Enhancement of Teaching Materials for Applied Statistics Courses by Combining Random Number Generation and Portable Document Format Files via LaTeX

Arthur Dryver
National Institute of Development Administration

Abstract

E-books, articles and other documents in portable document format (PDF) files are becoming more common, and they can incorporate videos, hyperlinks, and JavaScript. This article focuses on combining random number generation and PDF files in order to enhance the effectiveness of teaching materials. For example, a traditional textbook might have a few problems per chapter, but by utilizing JavaScript and embedding it within a PDF file, it is possible to create practically an unlimited number of problems with solutions. Another possible use of random number generation for an instructor is to create assignments and exams with unique numbers. The article contains two examples for the reader with their associated JavaScript and LaTeX code. Finally, the paper covers, in general, how to use JavaScript, random number generation, and LaTeX for enhancing teaching materials in terms of instruction and assessment.

Keywords: JavaScript, LaTeX, random numbers, PDF file, e-learning.

1. Introduction

It is becoming more and more common to read articles on the computer in a PDF file. An electronic document can be printed, but documents in a PDF file can offer a lot more than a hard copy. For example, PDF files can incorporate videos, hyperlinks, and JavaScript. A major benefit of teaching applied statistics from an electronic document concerns access to random number generation through JavaScript, which is the focus of this article. One example is that a printed version of material covering mathematical problems might offer a few problems, whereas utilizing JavaScript and embedding it within a PDF file makes it possible to create

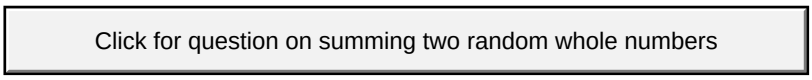

Figure 1: A simple example of using random number generation to create over ten thousand, 101×101, interactive problem-solution sets for the addition of two whole numbers ranging from 0 to 100.

practically an infinite number of problem-solution sets. In order to illustrate this concept, take one of the most simple of math problems, addition. For the following questions:

1. $12 + 4 = ?$
2. $3 + 7 = ?$
3. $6 + 5 = ?$
4. etc.,

the answers in the back of the textbook would be:

1. 16
2. 10
3. 11
4. etc.

Instead of writing a few practice problems for students on the topic of addition, in an electronic document the author could offer much more. Figure 1 is an example of a JavaScript pop-up application for creating practice problems on the topic of addition created using LaTeX, a document preparation system (Lamport 1994). A bonus feature not possible within a traditional document is that users can enter their answer and then compare this answer to the correct one for immediate feedback. The LaTeX with JavaScript code given below was used to generate the pop-up application in Figure 1.

```
\pushButton[\CA{Click for question on summing
two random whole numbers}\A{\JS{%
x1 = Math.round(Math.random()*100);
x2 = Math.round(Math.random()*100);
user = app.response(x1+"+"+x2+" = ???
Please enter the answer. ","Sum two numbers");
if ((x1+x2)==user) {
toshow = "Correct. The sum of "+x1+"+"+x2+" = " + (x1+x2);
app.popUpMenu(toshow);}
if ((x1+x2)!=user) {
toshow = "Incorrect. The sum of "+x1+"+"+x2+" = " +
(x1+x2) + " not " + user;
app.popUpMenu(toshow);}
}}]{jssumrnum}{300bp}{24bp}
```

In the next section, Section 2, some of the fundamentals required for embedding JavaScript using LaTeX in a PDF file are covered. Section 3 includes an example of the power of embedding JavaScript within a PDF file related to basic statistics. The example discusses probability calculations using binomial, hypergeometric, Poisson, and exponential distributions. The code which produces the examples in this paper can be downloaded along with the paper and are in the files `v31c03-examples.tex` and `jsbinhypv2.tex`. The file `v31c03-examples.tex` inputs the `jsbinhypv2.tex` which contains the JavaScript code for the example in Section 3. In Section 4, the author covers some of the various ways of enhancing teaching materials through the use of random number generation beyond the given examples. Finally, Section 5 consists of some general comments about embedding JavaScript into a PDF file using LaTeX.

2. Writing JavaScript to be compiled by LaTeX

Author(s) can use a built-in JavaScript function called `Math.random()` to generate a random number between 0 and 1. In order to generate numbers with the potential to be greater than 1, the author can multiply `Math.random()` by a number larger than one. To generate numbers between X and $X + 1$, the author could use `Math.random()+X`, etc. To turn a real number into an integer-valued number, the code is `Math.round(real number)`. By using these basic JavaScript functions and some additional programming, the author of an electronic document can go far in creating practically an infinite number of problem-solution sets.

Imagine a more complicated statistical problem, such as calculating the probability mass function of a random variable X that follows the binomial distribution, $X \sim Bin(n, p)$. Thus, $P(X = x) = \binom{n}{x} p^x (1-p)^{(n-x)}$, where n is the number of trials, p is the probability of success, and x the number of successes. The constraints when coding this problem in JavaScript are that x and n are integers, $0 \leq x \leq n$, and $0 \leq p \leq 1$. By using the functions `Math.random()`, `Math.round()`, and creating a function to calculate $\binom{n}{x}$, practically an infinite problem-solution set can be created.

The JavaScript code to be compiled by LaTeX requires including the LaTeX add-on package `insdljs` (Story 2001), where the acronym DLJS stands for document-level JavaScript. The JavaScript code must be kept within `\begin{insDLJS}{mydljsc}` and `\end{insDLJS}`. The following LaTeX code

```
\pushButton[\CA{Binomial}\RC{Click for the problem}
\A{\JS{params=thequestion(1);}}]{Button}{}{14bp}
```

will run the embedded JavaScript function `thequestion(1)` and pass the 1 into the function when the button is clicked. The variable `params` stores the values returned from the function for later use. Then code is needed to print the output that results from executing the JavaScript function `thequestion(1)`. The following LaTeX code performs that task:

```
\javas{\textField[\MaxLen{200}]{probabques}{6.1in}{108bp}}\\
```

where `javas` creates an empty shaded box in which to put the JavaScript output, and `probabques` is a string generated from the JavaScript code. Finally, in order for the code to run, the author has to incorporate the package `eforms` (Story 2009) with the option `pdftex`.

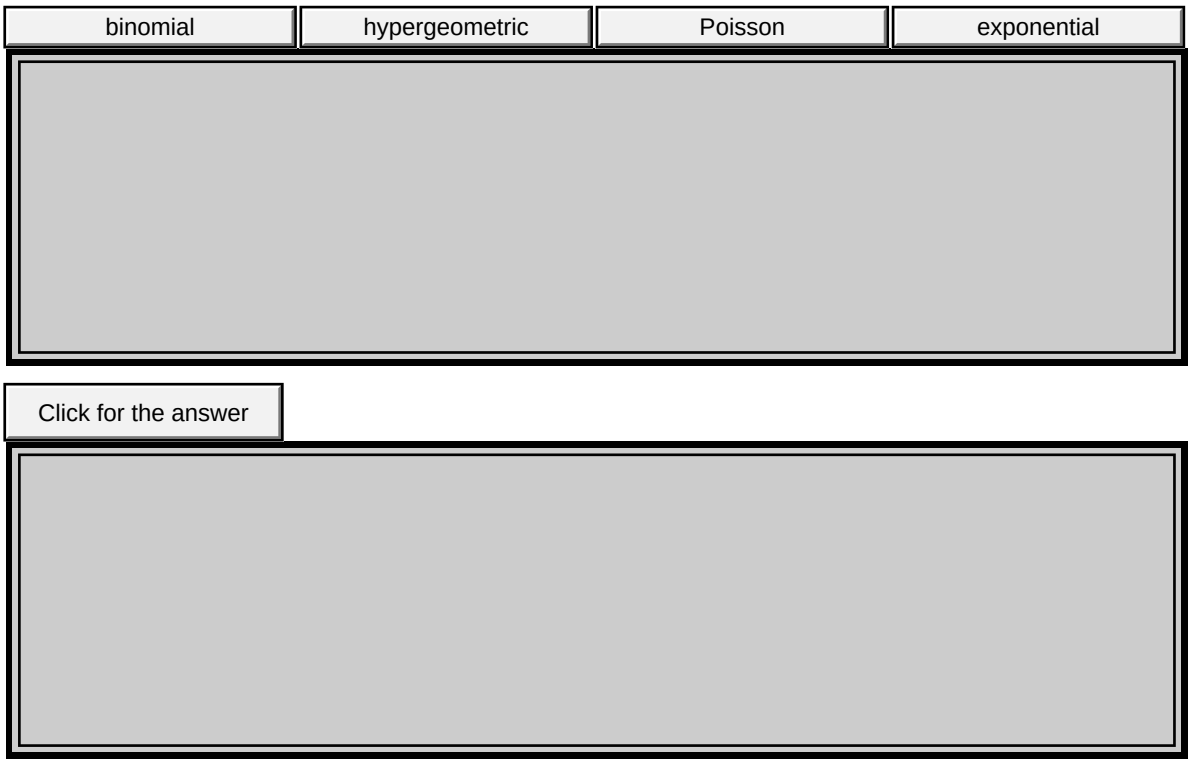

Figure 2: An example of creating practically infinite problem-solution sets using random number generation.

3. An illustrative example

This section, if read with Adobe Reader, is an example that demonstrates the power of using JavaScript and random number generation to create statistical exercises in a PDF file. The problem-solution sets in Figure 2 were created in LaTeX. The buttons above the first gray box create a question. The question outputted to the file depends on the button clicked. The button above the second gray box, when clicked, outputs the solution to the question in the question box. Note that some answers obtained from other software packages may differ from the PDF file output due to rounding.

4. Enhancing teaching materials

The examples given so far are simple examples of how random number generation can be embedded within a PDF file via LaTeX and used to supplement a PDF file in order to enhance the effectiveness of teaching materials. Random number generation in itself is not sufficient for teaching applied statistics, but can be a powerful aid in various ways. For example, numbers without information on what they represent, no "story" behind the data, are insufficient for teaching applied statistics. On the other hand, for teaching applied statistics, using only stories without any data to go with them would also be incomplete. For another example, consider teaching a data mining class with only a few small datasets for students; this too would be lacking.

4.1. Instructional material

Often numbers are made up by instructors to create data for teaching examples. Random number generation can be used for simulating data, ranging from as simple as animal populations, location and number of species, (e.g., Dryver and Thompson 2005; Dryver and Chao 2007) to credit information, the age of the customer's oldest credit card, a customer's payment history, debt to credit limit ratio, etc. (e.g., Dryver and Sukkasem 2009). Simulating data enables instructors to create data specifically for the lecture at hand without being limited by their access to real data. In a data mining class, it is often difficult to find extremely large datasets, let alone many large datasets, to cover all the topics in the class; via simulation this becomes a non-issue for the instructor.

For very small datasets, it is possible for the writer to generate randomly the data and have the data written within the PDF file. Of course, for large datasets one would not want to write it out within the PDF file, but it is possible for the writer to output the data generated from within the PDF file to a text file, such as a tab-delimited text file (Adobe Solutions Network 2005a).

4.2. Assessment

Random number generation can also assist instructors in the assessment of students. Unfortunately, copying does occur within some courses on homework as well as on exams, and statistics courses are not exempt from this problem. Unique sets of numbers for assignments and exams could possibly reduce but not eliminate cheating. There are two main points to think about before creating unique or even just multiple assignments and exams: first how to create them and second how to grade them. Creating an assignment/exam with the same questions for each student except for the numbers in a class is not too difficult, especially using random number generators. Needless to say, grading an assignment from n students with a different solution set for each student can be much more difficult than grading an assignment from n students with a single correct solution set.

An instructor giving homework in an applied statistics course where each student receives the same problem types but different numbers could encourage collaboration without the fear of straight copying. In terms of exams, straightforward copying would also not be possible. Embedding JavaScript in a PDF file could be used to accomplish this for both hard copy and electronic versions of exams and homework.

For hard copy assignments/exams, as in the example in Section 3, the instructor could write the code to calculate the answers as well as the numbers needed for the questions. Although, in this case the answers would be written at the end of the assignment/exam on a separate page(s) and would be withheld from the student. Each exam would require a unique "key" to match the questions with the answers after the exam for grading. Thus the instructor would click a button to generate the unique "key", the random numbers for the questions, and their answers. Finally, the instructor would print out the assignment/exam and separate the questions from the answers. In a normal situation, the assignment/exam would then be photocopied, but here this process would be done for each student in the class. For extremely large classes, it may be desirable to make only a few sets of different exams, for example, four different number sets as opposed to unique numbers for each student, and then seat the students accordingly.

For assignments/exams in a distance learning class, the issue of matching the numbers in

the questions to their answers is more difficult. Having an instructor or teaching assistant calculate the answers for each question on each assignment/exam can involve an extreme amount of work. This could be handled in a similar manner as the hard copy exam. Two PDF files could be written, one file with only the questions for the students and one file with both questions and answers for the instructor. Each student could be given a unique number to enter upon opening the PDF file for the initial seed for the pseudo random number generator. The JavaScript function `Math.random()` cannot take a seed. It uses the time, and thus a pseudo random number generator would have to be written that could take a seed. For grading the instructor assistant would enter the same number into the PDF file with both questions and answers in order to generate the associated answers.

Another possible solution for distance learning classes involves using a database. A PDF file can even connect to a database for the purposes of retrieving data, inserting data, etc. (Adobe Solutions Network 2005a). In the database there could be a different set of questions and answers for each student and the student could enter his or her student ID number in order to retrieve his or her set of numbers for the questions. In addition, the PDF file could be used to submit the student's answers and computer code could be written to check the answers reducing the grading workload. Another benefit of this methodology is that there is no restriction in the way in which the database could be populated. In short, the writer is no longer limited to JavaScript programming language for creating the data to be associated with the assignments/exams.

Random number generation can also aid in the implementation of the mastery learning teaching pedagogy (Block 1971). A concept behind mastery learning is that students should understand the i-th step before moving on to the $(i+1)$-th step. Some instructors might desire to employ mastery learning but this could require significantly more work on the instructor's part by having to give more individual attention to the students that are having a more difficult time with certain topics (Guskey 2007). Through the use of random number generation, students could be given assignments to do a problem type "X" number of times *correctly*, as opposed to typical assignments that require them to do assignments "X" number of times regardless whether they are correct or not. JavaScript code could be written to enable a student to know if his or her answer entered was correct or not, as seen in the addition example in Section 1. The instructor could ensure the number of times done correctly was equal to "X" by the PDF file exchanging information with a database.

If one only desires to increase the number problem-solution sets or create unique assignments/exams, there is an option other than embedding JavaScript that can be considered. The package **exams**, in conjunction with `Sweave` (Leisch 2002) in R, enables a person with knowledge of R and LaTeX to relatively easily create several problem-solution sets with unique numbers within a single PDF file or within multiple PDF files (Grün and Zeileis 2009). Each exercise type is written in what is called a `Sweave` file, which consists of basically R code for the data generation, and associated solution calculation along with LaTeX code for the problem and solution description. In this case the random number generation and solution calculation are carried out before the PDF file is generated, and the PDF file generated is a typical static PDF file. Thus, in order to have unique assignments for a class of 100 students with 10 assignments, for example, would require 2,000 PDF files, the number of students times the total number of assignments times two for questions and solutions in separate PDF files. This potentially large number of PDF files required may consequently be a deterrent from this approach. In addition, embedding JavaScript into PDF files offers more than multiple

static PDF files, such as the ability to connect to a database and access other features of the internet for distance learning courses.

5. Comments on coding

The author has found a few obstacles in terms of using the techniques presented for augmenting e-documents. In the author's opinion, JavaScript for web pages is much easier to code than JavaScript for PDF files via LaTeX in terms of debugging. Also, formating and generating the output are different for PDF files and for HTML files (Adobe Solutions Network 2005a,b; Goodman 2001). A PDF file with JavaScript embedded in it must be read using Adobe Reader with JavaScript enabled. For example, JavaScript will not be active on a stand alone Foxit version 2. Foxit does offer, however, a free add-on package that enables JavaScript support. Finally, a major difficulty encountered relates to limited documentation for incorporating JavaScript into a PDF file via LaTeX. Standard LaTeX books, such as "The LaTeX Companion (2nd Edition)" (Mittelbach, Goossens, Braams, Carlisle, and Rowley 2004), do not cover embedding JavaScript via LaTeX.

Ultimately there are more positives than negatives regarding embedding JavaScript via LaTeX. Such positives include:

1. practically unlimited problem-solution sets,
2. unique assignments and exams,
3. ability to connect to a database,
   (a) computerized grading,
   (b) mastery learning,
   (c) no longer restricted to JavaScript for simulating data as the database does not have to be populated using JavaScript,
4. once the JavaScript code is written for LaTeX:
   (a) it is easy to modify,
   (b) it is easy to leverage the code for additional problems or other appropriate work,
   (c) changes to the LaTeX code and recompiling is not an issue, as opposed to JavaScript code embedded via Adobe Acrobat, which will no longer be embedded upon LaTeX recompilation.

The author hopes that this article, by providing a full set of code for problem-solution sets and by adding to the sparse literature on incorporating JavaScript via LaTeX, will remove many of the difficulties for other LaTeX users desiring to incorporate JavaScript.

Acknowledgments

The author would like to thank Dr. Christopher Johnson of the National Institute of Development Administration for sharing his knowledge of JavaScript used in conjunction with Adobe

Acrobat. In addition, the author would like to thank the referees for their helpful comments and suggestions. Finally, the author would also like to thank the Research Center and Graduate School of Business Administration, National Institute of Development Administration, in Bangkok, Thailand, for their support of this work.

References

Adobe Solutions Network (2005a). *Adobe Acrobat 7.0.5 Acrobat JavaScript Scripting Guide*. Adobe Systems Inc. URL http://partners.adobe.com/public/developer/en/acrobat/sdk/pdf/javascript/AcroJSGuide.pdf.

Adobe Solutions Network (2005b). *Adobe Acrobat 7.0.5 Acrobat JavaScript Scripting Reference*. Adobe Systems Inc. URL http://www.adobe.com/devnet/acrobat/pdfs/AcroJS.pdf.

Block JH (ed.) (1971). *Mastery Learning: Theory and Practice*. Holt, Rinehart, & Winston, New York, N.Y.

Dryver AL, Chao CT (2007). "Ratio Estimators in Adaptive Cluster Sampling." *Environmetrics*, **18**(6), 607–620.

Dryver AL, Sukkasem J (2009). "Validating Risk Models With a Focus on Credit Scoring Models." *Journal of Statistical Computation and Simulation*, **79**(2), 181–193.

Dryver AL, Thompson SK (2005). "Improved Unbiased Estimators in Adaptive Cluster Sampling." *Journal of Royal Statistical Society B*, **67**(1), 157–166.

Goodman D (2001). *JavaScript Bible, Gold Edition*. Hungry Minds, Inc., New York, N.Y.

Grün B, Zeileis A (2009). "Automatic Generation of Exams in R." *Journal of Statistical Software*, **29**(10), 1–14. URL http://www.jstatsoft.org/v29/i10/.

Guskey TR (2007). "Closing Achievement Gaps: Revisiting Benjamin S. Bloom's 'Learning for Mastery'." *Journal of Advanced Academics*, **19**(1), 8–31.

Lamport L (1994). *LaTeX: A Document Preparation System*. 2nd edition. Addison-Wesley, Reading, Massachusetts.

Leisch F (2002). "Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *COMPSTAT 2002 – Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg.

Mittelbach F, Goossens M, Braams J, Carlisle D, Rowley C (2004). *The LaTeX Companion*. 2nd edition. Pearson Education, Inc., Boston, MA.

Story DP (2001). "Techniques of Introducing Document-Level JavaScript into a PDF File from a LaTeX Source." In *Proceedings of the 2001 Annual Meeting*, volume 22. URL http://www.tug.org/tug2001/authors/story/story.pdf.

Story DP (2009). *Support for AcroForms and Links, and for Document JavaScript and Open Page Events*. AcroTEX.Net, 2nd edition. URL http://www.math.uakron.edu/~dpstory/acrotex/eformman.pdf.

Affiliation:

Arthur Dryver Graduate School of Business Administration
National Institute of Development Administration
Bangkapi, Bangkok, Thailand 10240
E-mail: dryver@gmail.com
URL: http://www.LearnViaWeb.com/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 31, Code Snippet 3 *Submitted:* 2008-09-17
September 2009 *Accepted:* 2009-05-27