Using Simulations in Teaching Experimental Design

Russell V. Lenth
Department of Statistics & Actuarial Science
The University of Iowa
Iowa City, IA 52242

August 6, 2002

Abstract

All too often, a course on experimental design turns into a course that is almost entirely devoted to analysis of experimental data, with design issues in the background. The reason for this is that to learn to design experiments, you actually have to *do* experiments, and that is much harder to organize than lectures.

Design-conscious teachers often incorporate project work in a course. Such activities are extremely valuable to students; but there are problems too. Real experiments are often very time-consuming, and it is hard to handle more than one or two per semester. If students are given the freedom to choose their own project topics, the instructor has little opportunity to focus projects on specific instructional goals. An alternative approach is to develop scenarios and ask students to design and carry out experiments using software that simulates the data. There is a range of possibilities for how structured or unstructured is the activity. In this article, I illustrate my experiences in teaching design with simulation projects, and demonstrate some Web-based software that I have written to facilitate developing design scenarios.

Key words Simulation; Experimental design; Statistical education; Activity-based learning

1 Introduction

This article describes some of my experiences in using simulation activities to teach experimental design, the reasons that I think this is a good approach, and some web-based software in Java that facilitates these types of activities.

Section 2 discusses why “real data” don’t always satisfy the instructional needs of a design course. In Section 3, I argue that simulation activities often provide a better way to focus on the important aspects of experimental design.

To use simulations, we need suitable software, and Section 4 explains that Java is a good choice because it is free, runs on the web, and implements an object system whereby the common elements of a simulation activity can be programmed once and for all. Also shown there are the features of some Java software that I have developed. Section 5 describes the Java simulations that I actually used in teaching a recent design course. Finally, the source code for a particular plug-in is presented and explained in java-how.

2 Issues with using "real data"

The term "real data" always bothers me, because to me, "unreal data" would not be data at all. However, maybe it is a useful term for this paper for purposes of contrasting data that are collected in an experiment with data that are simulated. It might seem at first that real data are always better than fake data, but I will make the point that that is not always the case.

2.1 Emphasis on analysis, not design

Usually, real data are data that somebody else has collected. If you use data collected by somebody else, then you're not teaching design, you're teaching analysis. Certainly, analysis is a necessary part of just about any design course, and it is often (again, not always) preferable to use real data in examples and exercises.

2.2 Real experiments

To teach experimental design using real data, then students must do real experiments. This is definitely possible to do. For example, my standard first lecture in design involves an in-class experiment, where students are randomly assigned to "running" and "marching" groups. Students (and teacher) then do their assigned task (in place) for one minute, then each student counts her pulse over a 15-second period (I call out the start and end times), and the data are recorded. This is a completely randomized design (CRD). We can discuss the randomization, experimental units, measurements, responses, effects, and so forth.

Subsequently, I instruct everyone to switch treatments, and the experiment is repeated. This, combined with the first experiment, comprises a crossover design. We can now discuss what additional effects come into play. I usually deliberately lengthen or shorten the period of time over which the pulses are counted, which will tend to create a significant "period" effect when the data are analyzed later. I confess to this deliberate introduction of an effect so that students can think more concretely about how unintended effects can arise.

This real experiment can be done in one class period, plus it provides some real data for future lectures or exercises on analyzing CRDs and crossover experiments. It has worked well for me because there are subjects already available.[1]

Real experiments can be very rewarding, but they also can often be costly, time-consuming, or cumbersome—and they can be utter failures. I have had students do projects where they choose what to investigate. Often, these are kitchen experiments involving boiling water, cooking pasta, testing the strength of paper towels, etc.; and sometimes, students are amazingly ingenious in the way that they operationalize a research question. However, there is substantial variation in the quality of the projects, not just in how well thought-out they are but in what students can learn by doing them. They also vary considerably in the amount of time and effort required to complete them.

Even if you are very careful (by collecting and reviewing project proposals, etc.), it is hard to gauge the practicality of a project. An innocuous-sounding project where one measures the length of time it takes for ice cubes to melt under varying conditions is actually a nightmare for a busy student to carry out because of the time and tedium—almost none of it spent thinking about statistics or design issues. It is good for students to learn that real experiments are fraught with unanticipated complications, but the price of learning this experientially can be way beyond reasonable.

Having said all this, I am sure that many readers of this paper will be able to give me a multitude of counterexamples—simple, short, effective real experiments. I welcome hearing about them. I do know about paper helicopters, and the dozens of factors that can be varied on them; but even there, you cannot discount the difficulties associated with finding a place to drop them, various mechanical difficulties, and interference due to wind drafts or passers by.

2.3 It's a dirty world out there

Real data have built-in perils comparable to the complications encountered in real experiments. Just understanding the goals of the experiment can be difficult for a student, especially an international student. If there are too many complications, then the focus shifts to dealing with the complexities of the data, and away from the concepts or techniques that need to be illustrated. Of course, students should see examples of complicated data; but one usually needs to see straightforward analyses of clean data sets first so they can understand what makes the complicated ones complicated. If the textbook lacks them, it can be difficult to find a clean real data set that illustrates a particular experimental design. In that case, simulated or even made-up data are preferable for early illustrations of analyses and concepts.

[1]I wonder if I should run this idea by our university's Institutional Review Board, however, because it is indeed an experiment involving human subjects. So far, I have not had differently abled students in a design class, and if I did, this experiment could not be used.

3 Advantages of simulations

An obvious goal of this paper is to expose some of the advantages of using simulated data in certain parts of a design course. The main ones are now discussed.

3.1 Allows emphasis on design

To simulate data, we need a computer program. Such a program can be tedious to develop (more on this later); but once it is developed, it can be run just as easily tomorrow as yesterday. That opens up the possibility of involving students in the planning of the experiment—including pilot studies and sample-size determination. Even though the data are simulated, the *process* of developing a protocol and collecting data is real (though perhaps sanitized in important ways); so students will have ownership of their data just like in real experiments.

3.2 Practical in terms of time and effort

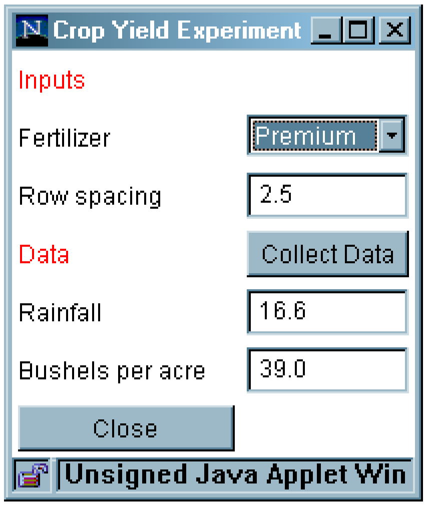
Again, once a program is written, it can be tested to make sure that it works and that a student can develop the protocol and collect the data in a reasonable period of time. This improves the ratio of student time spent thinking about the design to time spent doing the experiment. It also makes it possible to incorporate more student-designed experiments in the term of the course.

3.3 Ability to tailor to teaching goals

Simulated data are not real data, but they can be realistic data; and realistic data can have a real advantage over real data when one has specific instructional goals. Simulated data can be made as clean or as dirty as you like. You can simulate subjects dropping out or lying. You can make the data dirty in specific ways and later show how a different design choice will improve the precision of estimates. You can make the actual means or effects the same or different for each student. You can collect students' data sets and show the variation in the results obtained, or combine their results into a mega-experiment as a class illustration of a multi-center study.

4 Computational infrastructure

It is fairly simple to write computer code that simulates experimental data. What is more complicated and more tedious is providing a user interface to the code so that a student can vary factor levels and obtain response values. Moreover, user interfaces themselves are more complicated than in the past, when the norm was to interact with a computer via a terminal screen. Now, most human-computer interaction is graphical, via mouse clicks, selecting items from a list, and entering values in text fields. Programming with such graphical interfaces is



Data

Fertilizer	Premium
Row spacing	2.5
Rainfall	16.6
Bushels per acre	39.0

Figure 1: Data-collection dialog for a the crop-yield experiment.

very different from simple print and read statements, and possibly very foreign to the more senior among us.

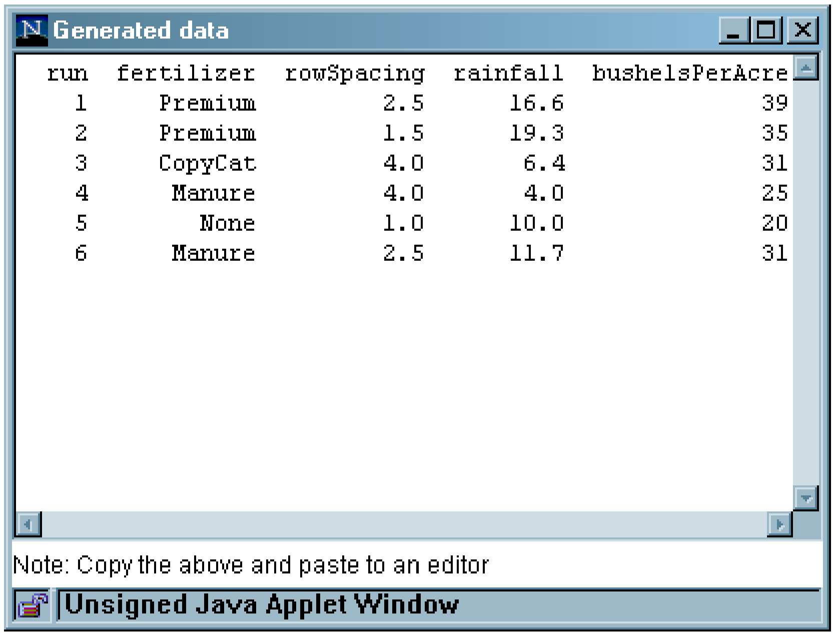
4.1 Java and object-oriented design

Java is a relatively new programming language. For the present purpose, its main advantages are

- Java applications can be run in an ordinary web browser, on any modern PC. Thus, Java applications are available to any student in any location where they can access a PC.
- Java provides for a graphical user interface.
- Java is object-oriented. This makes it possible to develop a common infrastructure for simulation experiments. Particular experiments may be built on this infrastructure with minimal programming.
- A Java compiler is free and available on any platform. This makes it possible for anyone to extend the simulation applications and object infrastructure that are described in this article.

4.2 Application features

Here is an illustration of a typical simulation-based activity that is implemented in Java. Figure 1 shows the data-collection dialog for a simulation experiment on crop yield. The scenario is that there are four fertilizers under test, and that



```
run  fertilizer  rowSpacing  rainfall  bushelsPerAcre
  1     Premium         2.5      16.6              39
  2     Premium         1.5      19.3              35
  3     CopyCat         4.0       6.4              31
  4      Manure         4.0       4.0              25
  5        None         1.0      10.0              20
  6      Manure         2.5      11.7              31
```

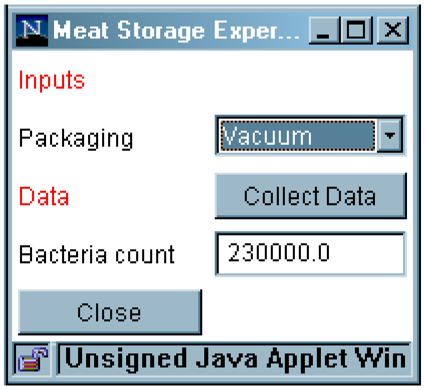
Note: Copy the above and paste to an editor

Figure 2: Data-output window. Contents can be copied and pasted into a text editor, then saved.

the spacing between rows of a crop can also be varied. The response variable is yield at harvest. In addition, the amount of rainfall is recorded. The experimental units are 24 farm fields at different locations around the state.

The student must design and randomize the experiment, then connect to the web page where the applet is available. The web page contains a description of the experimental objectives, and a button. The dialog in Figure 1 pops up when the button is clicked. Each run of the experiment consists of a combination of fertilizer and row spacing. In the dialog, the student selects one of the four fertilizers from a drop-down list, and enters the row spacing as text. The allowable row spacings are in the range from 1 to 4, inclusive. If a number outside that range is entered, or a non-number is entered, a message box pops up with an informative error message.

When the ”Collect Data” button is clicked, the simulated rainfall and yield appear in the fields towards the bottom of the dialog. In addition, the values or levels of all variables are displayed in an output window, shown in Figure 2. (The window appears when the first observation is generated.) For security reasons, a Java program running in a web browser does not allow saving of files; so instead, the student may select the data, copy it to the PC’s clipboard, paste it into a text editor, and save the data to a file. This is a bit cumbersome, but still much easier than manually entering data.


Meat Storage Exper...

Inputs

Packaging: Vacuum

Data: Collect Data

Bacteria count: 230000.0

Figure 3: Dialog for the meat-storage experiment.

The Java code for the crop-yield experiment is given in Section 6. First, some additional examples are presented, along with a discussion of their use in teaching.

5 Examples and experiences

This section describes the simulation activities I used in a graduate-level design course in the spring of 2002. It in fact describes *all* of them (the crop-yield example previously described was not used). My original plans were to use more, but this was the number that turned out to be practical, at least for those students with that textbook in that semester. The textbook was Oehlert (2000). The course topics included the chapters on completely randomized designs, contrasts, multiple comparisons, assumptions and diagnostics, sample size, factorials, random and mixed effects, complete-block designs, Latin squares, split plots, and designs with covariates. We did not discuss two-level factorials, incomplete blocks, fractional factorials, or response surfaces; those topics come in a different course.

5.1 Simple CRD

The scenario for the first experiment in my class is based on an example in another design text (Kuehl, 2000, Example 2.1, page 38). The instructions given to the students called for a completely randomized design with one factor at four levels—the type of gas treatment given to packaged meat. The response variable given in the textbook is the logarithm of the psychotrophic bacteria count after nine days of storage. In the user interface shown in Figure 3, the data are given in raw counts rather than on a log scale. The instructions specifically ask the student to log the data. (If the same exercise had been introduced a bit later,

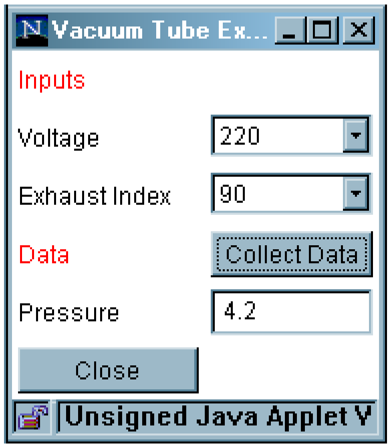


Figure 4: Dialog for the vacuum-tube experiment.

I probably would have expected students to discover on their own that it is desirable to transform the response.)

Students were asked to design an experiment in 20 runs. The data-output window (similar to that in Figure 2) includes sequence numbers, which helps to encourage students not to cheat on the randomization. Unknown to the student, the errors were not independent; instead, they were generated from a random walk (on the scale of the logged response). Under those conditions, failing to randomize will bias the treatment means and yield an unrealistically small mean squared error. After the papers were turned in, I confessed that the errors were not independent, and that laid the groundwork for some further discussion. For example, using my own results of the same experiment, I displayed a plot of residuals versus time order—reinforcing some of the material on diagnostics. It also gave me an opportunity to say, “aren’t we glad we randomized?”

I think this example illustrates one of the real advantages of simulation over real data. Simulation gives the teacher the chance to control the complexity of the data, and the nature and the extent to which the data deviate from standard assumptions. In general, I think it is usually a good idea to include *some* kind of irregularity, not just to illustrate diagnostics but to create a real penalty for not doing the experiment the right way. This simple simulation applet has rich enough possibilities that it can be used again—as described later.

5.2 Factorial experiment

Figure 4 shows the user interface for what is perhaps the most mundane of the simulations we did. It is for a two-factor experiment. Again, it is to be a completely randomized design. The scenario is from Hicks and Turner Jr. (1999), Table 5.1, page 138. Prior to collecting data from this applet, students needed to determine an appropriate sample size based on stated results of earlier studies with the same instrumentation (alternatively, I could have asked them to run a pilot study using this or some other applet). Unlike the original data in Hicks and Turner Jr. (1999), the generated data require a transformation—this time the students had to discover it for themselves.

5.3 Block design

Now we return to the meat-storage experiment with the dialog shown in Figure 3. We run the same experimental procedures but with a different protocol. Instead of using a complete randomization, view the 20 experimental runs as 5 sequences of 4 consecutive runs. In each sequence, run all four treatments in random order (separate randomization for each sequence). This is a randomized complete-block design with sequences as blocks. Due to the random-walk structure of the actual errors, there tends to be substantial variation among the block means, and you can basically count on the mean square error being much smaller for the blocked experiment than the original completely randomized one. Thus, there is a clear advantage to blocking.

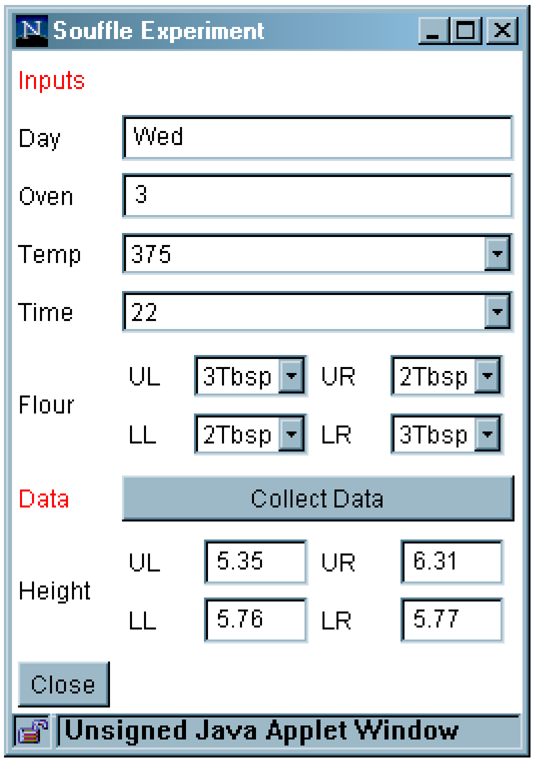
Re-using the same simulation exercise in these two different ways makes it possible to teach some powerful lessons. Three of the most important ones are

1. You have a choice about how you conduct an experiment.
2. Your choice affects the way the data are analyzed.
3. Your choice can make a big difference in the quality of the results.

5.4 Split-split-plot experiment

Near the end of the course, I asked the students to do a small project involving a fairly complex experiment. The scenario is cooking a soufflé, and the response variable is the height of the soufflé. Factors include the baking temperature (2 levels), baking time (2 levels), and amount of flour (2 levels). The procedure is to bake (or rather, to simulate baking) four soufflés at a time—two on the top shelf and two on the bottom shelf of the oven. The experiment is to be carried out in a five-day period, and it is possible to bake four oven loads per day. Students were asked to first write a proposal describing their basic procedures for randomizing and carrying out the experiment; then, after obtaining feedback from me, to develop the protocol, collect the data, explore and analyze, and write a report.

The user interface for this experiment is shown in Figure 5. The "Day" and "Oven" fields increment automatically and cannot be changed by the student; once day 5, oven 4 is completed, a message box pops up advising that the experiment is over, and no further data may be generated. After pondering this for awhile, students come to realize that the experimental units for Time and Temp are the ovens, while the experimental units for Flour are the individual soufflés. In addition, it seems apparent that the top shelf may be different


Souffle Experiment

Inputs

Day	Wed
Oven	3
Temp	375
Time	22

Flour

UL	3Tbsp	UR	2Tbsp
LL	2Tbsp	LR	3Tbsp

Data

Height

UL	5.35	UR	6.31
LL	5.76	LR	5.77

Figure 5: User interface for the souffle experiment.

from the bottom shelf, so it is probably a good idea to control for Shelf—i.e., randomize each shelf separately. These considerations suggest a split-split-plot design with days as blocks, ovens as whole plots (with whole-plot factors Time and Temp), shelves a split plots, and soufflés as split-split plots.

Most students were able to figure out the required design! All I can guess is that their success was due to a combination of necessity, discussion among students (not always such a bad thing), and the concreteness of the data-collection environment that forces them to collect data under certain constraints. In a couple of cases, more elaborate designs were considered that involved a Latin square structure on each oven (so as to control for both shelf effects and side effects (as in left and right). Students actually had an easier time identifying a reasonable design and corresponding model than randomizing it.

6 Developing a plug-in

In this section, I show the basics of developing a simulation experiment. Several Java objects provide the necessary infrastructure; these classes are kept together in a package named `rvl.datasim` and in a Java archive named `datasim.jar`. All that is needed for a particular simulation activity is a small Java class that extends an abstract class named `rvl.datasim.TestBed`. Such a class may be viewed as a plug-in for the `datasim` library, because all that is needed is for a handful of methods to be defined.

This section is not intended to be a complete reference; for that, the reader should refer to the `javadoc`-generated documentation available on my web site, `http://www.stat.uiowa.edu/~rlenth/datasim/`.

6.1 Java code

Figure 6 displays the complete source code for the plug-in that generates the user interface shown in Figure 1. The `import` statement brings-in references to the `rvl.datasim` classes. The plug-in is defined as the class `CropYield`, an extension of `TestBed`. The first three lines within the class define the factor, covariate, and responses required in the simulation. Then follows a constructor and four methods.

The constructor, `CropYield()`, sets up the required objects. First, a title is passed to its super class; this is what is displayed in the title bar in Figure 1. The factor `fert` is obtained by calling the constructor for a `Factor`, whose arguments include the name of the factor, the names of its levels, and the values of the factor effects at each level. In the user interface, it will be associated with a drop-down list. `spac` is constructed as a `Covariate` named “Row spacing” with an allowed range of [1,4] and a linear effect (slope) of –1.2. The responses `rain` and `yield` are constructed with the names shown, and specified to be displayed with 1 and 0 decimal places, respectively. All of the factors, covariates, and responses are automatically associated with suitable graphical components (drop-down lists, etc.); the programmer does not have to worry about that.

The `getInputs()` and `getOutputs()` methods are always pretty trivial; they need to return vectors of inputs to be controlled by the experimenter (factors and covariates), and responses to be observed.

The only other required method is `simulate()`. It is called when the user clicks on the “Collect Data” button, and is the only place that real programming is done. In this example, we need to simulate the amount of rainfall. Then the yield is calculated as the sum of a linear effect of rainfall, the spacing and fertilizer effects, and normally distributed error. Finally, we need to set the `Response` objects to their simulated values. The method returns zero, indicating no errors.

The last method, `main`, is not required at all, but is useful for development because when the class is compiled, it can be run and tested as a standalone Java application. A `DataSimGUI` object is an instance of the actual application that displays a window and implements the plug-in.

This example does not show some important features. For instance, another constructors for a `Factor` object creates a factor having random levels. There are `PredArray` and `RespArray` objects that allow for multiple instances of the same `Factor`, `Covariate`, or `Response`, such as in Figure 5.

```
import rvl.datasim.*;

public class CropYield extends TestBed
{
    Factor fert;
    Covariate spac;
    Response rain, yield;

    public CropYield () {
        super ("Crop Yield Experiment");
        fert = new Factor ("Fertilizer",
            new String[]{ "None","Manure","Premium","CopyCat" },
            new double[]{  -5,    -1,      4,        2      } );
        spac = new Covariate ("Row spacing", 1, 4, -1.2);
        rain = new Response ("Rainfall", 1);
        yield = new Response ("Bushels per acre", 0);
    }

    public Predictor[] getInputs() {
        return new Predictor[] { fert, spac };
    }

    public Response[] getOutputs() {
        return new Response[] { rain, yield };
    }

    public int simulate() {
        double rnfl = 2.4 + normal(0.5);
        double yld = 24.5 + .35 * rnfl
            + spac.getEffect()
            + fert.getEffect() + normal(2.3);
        rain.setValue (rnfl);
        yield.setValue (yld);
        return 0;
    }

    public static void main (String argv[]) {
        new DataSimGUI (new CropYield(), true);
    }
}
```

Figure 6: Source code for the crop-yield simulation dialog in Figure 1. This code should be saved in a file named `CropYield.java`—that is, its base file name should exactly match the class name.

6.2 Compiling and testing

The Java development kit is available for free download for most platforms from `http://java.sun.com/`. Before a plug-in can be compiled, the `rvl.datasim` objects must be made available to the Java compiler. To do this, download the file `http://www.stat.uiowa.edu/~rlenth/datasim/datasim.jar`. There is usually a standard place (on Windows, it is the subdirectory `lib\ext` of the Java installation) where the file may be stored, such that its classes are made available to the Java virtual machine. Either store `datasim.jar` there, or explicitly include its location in the CLASSPATH used by Java.

To compile the plug-in in Figure 6, use the command

```
javac CropYield.java
```

Assuming that it compiles with no errors, this will create a file named `CropYield.class` in the current directory. The plug-in may be tested using the command

```
java CropYield
```

6.3 Putting it on the web

Finally, once a plug-in is developed, we want to make it available to students. This requires three files to be made available: the `.jar` file, the `.class` file, and an HTML file. The latter presumably also contains instructions to the students. Create it using your favorite web-page editor. At the location where you want it in the web page, insert the following HTML code:

```
<APPLET
    code = "rvl.datasim.DataSimLoader.class"
    archive = "datasim.jar"
    width = 300 height = 30>
    <PARAM
        name = plugin
        value = "CropYield">
</APPLET>
```

`rvl.datasim.DataSimLoader` is an applet that requires one parameter, the name of the plug-in to be loaded. When the web page is loaded in a Java-capable browser, it displays as a button labeled “Press here to collect data.” The above assumes that files `datasim.jar`, `CropYield.class`, and the HTML file are all copied to the same web-accessible directory, and that they all be publicly readable.

7 Conclusions

I have tried to demonstrate that simulation experiments have a number of advantages over “real data” in teaching experimental design. Real data are usually collected by someone else, hence students never see the experiment. Real experiments take time, cost money, carry a real possibility of catastrophic failure, and tend to provide fairly uneven experiences among students. Simulation exercises can be tailored to particular instructional needs and made arbitrarily easy or complex. The same simulation scenario can be used with different experimental designs to teach their relative advantages and disadvantages. Simulations are inexpensive, less prone to catastrophe, provide more uniform benefits across a class, and can be used more often than real experiments.

The Java classes described here provide the infrastructure for quick development of simulation experiments. The user interface is implicit in the specifications of factors, covariates, and responses. It is only necessary to write code that defines these elements and that simulates the response value(s).

References

Hicks, C. R. and Turner Jr., K. V. (1999). *Fundamental Concepts in the Design of Experiments*. Oxford University Press, New York, fifth edn.

Kuehl, R. O. (2000). *Statistical Principles of Research Design and Analysis*. Duxbury Press, Belmont, CA, second edn.

Oehlert, G. W. (2000). *A First Course in Design and Analysis of Experiments*. Freeman, New York.