**Running batch experiments** (and usingdatafarmingrubyscripts)

We have implemented tools and demos in the Ruby programming language.  Our choice is based on our experience that Ruby gives consistent performance across Windows, Macs, and Linux systems, without any need for tweaks or edits.

Step 0 only needs to be done once. A nice thing about this is that the datafarming scripts (and other installed gems) will be available from ANY directory on your computer, so you don’t need to worry about moving around multiple copies or limiting your experiments to a single directory.

**STEP 0: INSTALL SOFTWARE**

***Install the ruby programming language*** (you only need to do this a single time on your computer)

How? Watch the RubyInstall.mp4 video. Some useful links are <https://rubyinstaller.org> and <https://rubygems.org>

Ruby ships with Mac OS, instructions for downloading a nice package manager (“Homebrew”) are available at the end of the video, the link is <https://brew.sh>

Linux users can install Ruby using their favorite package manager, whatever that may be.

Windows users can get Ruby by downloading an installer from <https://rubyinstaller.org/downloads/>.  We recommend Ruby version 2.5.x, it’s substantially faster than 2.4.x and is compatible with the largest “gem” libraries.  Users will need to check if their system is 32 bit (x86) or 64 bit (x64), and select the appropriate version.  They can find this out in Windows 10 by going to Settings -> System -> About.  The installer comes with or without “devkit”, which includes a C compiler for supplemental modules implemented in C for speed.  Our tools are all pure Ruby, so installing with/without devkit is entirely up to you.  As the installation proceeds, just **make sure the checkboxes are checked for “Add Ruby executables to your PATH” and “Associate .rb and .rbw files with this Ruby installation”** (I believe both of these are checked by default).  There is a third checkbox which I also recommend, but it is optional:  “Use UTF-8 as default external encoding.”

In all cases, proper installation should be confirmed by opening a command-window (Windows) or Terminal window (Mac/Linux) and typing

ruby --version

(note: there’s a space before the two dashes, none after) followed by pressing Enter/Return.  You should see a response similar to

ruby 2.5.1p57 (2018-03-29 revision 63029) [x86\_64-darwin17]

although the details will vary based on OS and which particular version you installed.

After confirming that Ruby has been successfully installed, install the actual tools by going to a command/terminal window and typing

gem install datafarming

(Windows), or

sudo gem install datafarming

and entering your password to authenticate (Mac/Linux).

Source code and documentation are publicly available at <https://bitbucket.org/paul_j_sanchez/datafarmingrubyscripts>. A list of the scripts is also available in the README file at <https://gitlab.nps.edu/pjsanche/datafarmingrubyscripts>

We will be demonstrating data farming techniques with some simple simulation models, which use two additional “gem” packages.  These can be installed by typing:    
“gem install simplekit random\_variates” (Windows) or   
“sudo gem install simplekit random\_variates” (Mac/Linux).

Once you’ve done this, you can see what gems are installed by typing “gem list” – if you don’t see colorize, datafarming, fwt, quickstats, random\_variates, and simplekit then add the missing ones as above. (*Note from Susan: I don’t remember if colorize, fwt, and quickstats are automatically pulled in when you install the datafarming gem)*

**STEP 1: GET ORGANIZED**

1. ***Copy or download software you need to the desired directory***

You can select the files you wish to copy, and then **“Copy”** and **“Paste”** them into your folder. For this example, I’m putting

**StochasticLanchester.rb**

into my **Test** directory.

1. ***Open up a terminal window*** 
   1. **Mac users**: if you dragged the “Terminal” app to your dock at the bottom of your screen last week, double-click to open it up. If you don’t find it, you can go to **Applications -> Utilities -> Terminal**. **Windows users**: open either the **PowerShell** or the “**cmd**” window, whichever works best for you. *My limited experimentation with PowerShell shows that it doesn’t handle the file redirection, so I will stick with* **cmd***. Also, I had a couple of students for whom the “more” command didn’t work as it used to, and they had to type “more -e” to get a quick peak at the contents of a text file.*
   2. Change directory to your new folder. On either Mac or Windows, type “**cd** “   
      (don’t forget the space), then click over to your Finder/Explorer window and drag the **Test** folder onto the Terminal/cmd window. Make the Terminal/cmd window active, and hit “**return**” to finish.
2. ***Confirm that you can run your model***

To run the model interactively, type the first line and then follow the prompts.

ruby StochasticLanchester.rb

Enter initial number of Red forces: 100

Enter per-capita rate at which Reds kill Blues (b): 0.02

Enter initial number of Blue forces: 50

Enter per-capita rate at which Blues kill Reds (a): 0.045

Output similar to that below will appear on your screen, though the last three numbers will likely be different for this stochastic model.

InitialRed,RedKillBlueRate,InitialBlue,BlueKillRedRate,FinalRed,FinalBlue,BattleDuration

100,0.02000,50,0.04500,59,0,31.83418

This program also takes command-line input, so an alternative way of growing output is

ruby StochasticLanchester.rb 100 0.02 50 0.045

and you could send it to a file with redirection

ruby StochasticLanchester.rb 100 0.02 50 0.045 > SLout.csv

If you’re looking ahead to the ebola\_sim.model, you can try that one as well. This model does NOT prompt interactively, and the correct usage is

ruby ebola\_sim.rb <# replications> <population size> \

<initial # infected> <per capita avg transmission ratio> \

<intervention start day> \

<proportion reduction in transmission ratio>

Example:

ruby ebola\_sim.rb 4 100\_000 1 1.4 100 0.2

will print 4 replications of this single design point to the screen, something like

Population,Initial Infected,Transmission Ratio,Intervention Start Day,Transmission Ratio Reduction,Duration,Infected,Uninfected

100000,1,1.4,100,0.2,13.46,1,99999

100000,1,1.4,100,0.2,148.89,17,99983

100000,1,1.4,100,0.2,60.71,7,99993

100000,1,1.4,100,0.2,65.39,4,99996

***Note!* *Both of these models echo back the input settings, which saves you some steps later on and reduces chances of errors.***

**STEP 1: GET A DESIGN FILE** (you have several options).

1. ***Creating a full factorial design file using the cross.rb script****CAUTION: this is not advised for experiments with many factors or many factor levels! However, it may be helpful for crossing two separate designs, such as for robust design experiments.*

If you are using this program, your output is automatically sent to a tab-delimited file.

* 1. Use a text editor to create separate files for each factor, with one line for each factor level. For example, for the StochasticLanchester program we have four potential factors – InitRed, RkillB, InitBlue, BkillR – with different input levels, so we can have four different input files. Let’s suppose the files have the following content. I’ll use the “more” command command to show the contents. If “more” doesn’t work for you, you can open up the files in your text editor instead.

**more initRed**

100

110

120

**more RkillB**

0.02

0.022

0.024

**more initBlue**

50

65

80

**more BkillR**

0.045

0.0525

0.06

* 1. Cross the input files to get a full factorial design: in this case, a 3^4 factorial.

**cross.rb initRed RkillB initBlue BkillR >mySLdesign.csv**

*HINT: if several factors have the same factor levels, you can re-use those input files. Recall that for running TheOracle.rb with model1.rb (3 factors, all taking on levels between -5 and +5), we could create a single file f1.txt with a line for each level we want, and then type*

**cross.rb f1.txt f1.txt f1.txt > mydesign.csv**

1. ***Getting a design file from the stack\_nolhs.rb program***The results will automatically be sent to a tab-delimited file.
   1. Look at the options for generating designs by asking for help

**stack\_nolhs.rb -h**

* 1. To do a single stack (i.e., get the base design), enter the low and high levels of the factors and the number of decimal points (one set per line) from the screen, and send the design to a file, type

**stack\_nolhs.rb –s 1 >mydesign.txt**

Your cursor will be waiting for a response. Type in the values as prompted, then hit ctrl-d (Mac) or ctrl-z (Windows) to end the input. For the stochastic Lanchester example, you have four factors so just type in the min and max values and number of digits (decimals). Be sure to state “0” decimals for integer-valued factors, and multiple decimals for discrete-valued factors. Example (your inputs in **blue**):

To terminate input enter ctrl-d (Mac/Unix/Linux) or ctrl-z (Windows).

Enter ranges for each factor on a separate line.

MIN MAX #DIGITS

**100 120 0**

**0.02 0.024 4**

**50 80 0**

**0.045 0.06 4**

Alternatively, you can read all the values in using just three lines, corresponding to the three green rows at the top of one of the spreadsheets, by adding the “-e” option. Once again, your cursor will be waiting for a response. Type in the values as prompted, then hit ctrl-d (Mac) or ctrl-z (Windows) to end the input.

**stack\_nolhs.rb –s 1 -e >mydesign.txt**

and after the prompt, you can type

**1 1 1 1 1 1 1**

**17 17 17 17 17 17 17**

**0 0 0 0 0 0 0**

to get the basic design for a generic NOLH, or

**100 0.02 50 0.045**

**120 0.024 80 0.06**

**0 4 0 4**

for the stochastic Lanchester example.

There are other options for **stack\_nolhs.rb** that will let you choose how many stacks to make, or to choose a larger design, or to specify factor ranges in a different way. For example, even if we only have seven factors, we may want a base design with 33 (or more) design points, instead of only 17.

Unlike scaled\_rf\_cubed.rb, the stack\_nolhs.rb will read input from a file, either with or without the “-e” option. This is very helpful for large designs. If the factor low and high levels and number of decimals are in a file called fac\_in.txt, with one set per line, then type

**stack\_nolhs.rb <fac\_in.txt >mydesign.txt**

If the file contains three lines in the excel format for inputs, type

**stack\_nolhs.rb -e <fac\_in.txt >mydesign.txt**

For the stochastic Lanchester example, let’s create a design based on 2 stacks of a 33-dp NOLH as follows:

**stack\_nolhs.rb –s 2 -l 33 >testdesign.txt**

The “-l 33” is “dash-ell” (not “dash-1”) and is equivalent to “--levels 33”

1. ***Getting a design file from the scaled\_rf\_cubed.rb program***
   1. Look at the options for generating designs by typing

**scaled\_rf\_cubed.rb -h**

* 1. You need to give the low and high levels of each factor on the command line, because this script does not take input using the “<” file redirection.
     1. To create a resolution V fractional factorial (i.e., get the base design that allows you to estimate main effects and two-way interactions), you need to give the low and high levels of each factor on the command line. For the StochasticLanchester example, you could type

**scaled\_rf\_cubed.rb -d 100 120 0.02 0.024 50 80 0.045 0.06 >mydesign.txt**

The “-d” option makes sure you print the design only. This is all typed on one line even though it wraps around.

If you are just interested in the scaled design (all factor levels ranging from  
-1 to 1) then you can simply type the number of factors: see it on the screen

**scaled\_rf\_cubed.rb 4**

* If you have quantitative factors, then constructing a CCD or rotatable CCD will be more informative. In addition to the “-d” option, you can “-s” to get the star and center points for a CCD, or “-r” to create a rotatable CCD.

1. ***Getting an efficient design file from a design spreadsheet.***    
     
   If you are using one of the SEED design spreadsheets, such as NOLHdesigns\_v6.xlsx or NOB\_Mixed\_512dp.v3.xlsx, you can construct your design as follows:
   1. Set the low and high levels for each factor in the spreadsheet
   2. Select the block that contains all design points for your design but not the factor names, then **“Copy**” this block, open up a new Excel workbook, and **“Paste Special (values)”** into the new workbook.
   3. Save this file in either .csv format or tab-delimited text format. If you do tab delimited, let’s suppose you save it as **mydesign.txt**.
   4. **Mac users only!** Before you move on to the next step, you need to convert your design files to the correct line-ending format. Do this by typing  
       **convert\_line\_endings.rb mydesign.txt**  
      to rewrite the “mydesign.txt” file appropriately. (A backup copy will be saved as **mydesign.txt.orig**.)
   5. Note: if you have a design in a csv format, you can convert a csv file to a space-delimited file by typing **csv2blank.rb** <filename> without going through excel.
2. ***Getting a design file from JMP***JMP has a DOE module. You can use it to construct some that we’ve talked about, like fractional factorials (for smaller numbers of factors). It does have space-filling designs, but its LH designs are not necessarily nearly orthogonal. Our experience with the custom designer is that JMP is still geared toward constructing smaller designs for experiments involving a relatively small number of factors. If you use JMP’s **DOE** module to construct a design, then eliminate the additional descriptive and Y columns it adds, and **Export** the resulting table to a .csv file (**Mac users** will have to run the **convert\_line\_endings.rb** script) or to an Excel file.

**STEP 3a: GROW YOUR SIMULATION DATA FOR MODELS THAT USE COMMAND LINE INPUT**

**YOU CANNOT COPY AND PASTE THESE WITHOUT GETTING ERRORS, BECAUSE THE DASHES WILL SHOW UP DIFFERENTLY!**

We will do this with the **StochasticLanchester.rb** model in the Week5 folder. First, create a design. Suppose it’s the **mySLdesign.csv** file we created in Step 1.

The **StochasticLanchester.rb** model takes command line inputs separated by spaces. Our design is not quite ready, since it’s a comma-delimited file. So, we’ll run one more datafarmingscript before charging along.

**csv2blank mydesign.csv**

Now, mydesign.csv has been changed a space-delimited file instead of comma-delimited. First make a test run of the simulation. Here I’m doing this for 2 replications.   
**I recommend you always start with just a couple of replications to make sure that you’ve typed everything appropriately and that the program runs.** *Although it looks like this is being typed on two lines, it’s really just a long single line.*

**rundesign\_general.rb 'ruby StochasticLanchester.rb' mydesign.txt 2 output.csv**

That worked, so you can make your “real” run with more reps. Hit the “up” arrow on the keyboard to bring the command up again, then used the left arrow to go through and change the “2” (reps) to “100” (reps). I also added the “-d” option to write over (destructive write) my sample outfile. This may take a minute or two to run.

**ruby rundesign\_general.rb –d 'ruby StochasticLanchester.rb' mydesign.txt 100 slout\_ccd.csv**

Now I want to take a quick look at my output file. You might be able to type “more slout\_ccd.csv” to get a screen’s worth of output at a time. Type “q” or “ctrl-c” to quit. I’ll show the first few lines. Note that the lines will wrap around the screen if they’re too long.

> **more slout\_ccd.csv**

InitialRed,RedKillBlueRate,InitialBlue,BlueKillRedRate,FinalRed,FinalBlue,BattleDuration

10,0.01000,5,0.02000,3,0,103.76659

InitialRed,RedKillBlueRate,InitialBlue,BlueKillRedRate,FinalRed,FinalBlue,BattleDuration

8,0.01000,5,0.02000,5,0,99.59761

We don’t want the duplicate header rows, so go ahead and type

**ruby stripheaderdups.rb slout\_ccd.csv**

This will go ahead and create a backup file slout\_ccd.csv.orig and then rewrite the file. Hit the up arrow a few times, and you’ll find the first few lines look like

> **more slout\_ccd.csv**

InitialRed,RedKillBlueRate,InitialBlue,BlueKillRedRate,FinalRed,FinalBlue,BattleDuration

10,0.01000,5,0.02000,3,0,103.76659

8,0.01000,5,0.02000,5,0,99.59761

Now you’re ready to read things into JMP for analysis. You may want to define a new output column, such as ProportionBlueSurvivors, instead of using one of the raw outputs.

First, you should check and see that your design is what you thought it was. You can go to ***Analyze -> Distribution*** to check the low and high levels, and go to ***Graphs -> Scatterplot Matrix*** to see the design.



Regardless of your response, you can go to ***Analyze -> Fit Model*,** highlight all four factors, choose **Macros -> Factorial to degree** and **Macros -> Polynomial to degree** (with the default of 2) to see that you can fit all main effects and all two-way interactions simultaneously. This doesn’t mean that all are practically important (or even statistically significant), but it means that if there are two-way interactions or non-linear effects, you’ll be able to find them!