

PP01: Efficiently Simulating Normally Distributed Data For Animation



Ben White, GlaxoSmithKline
E-mail: ben.e.white@gsk.com

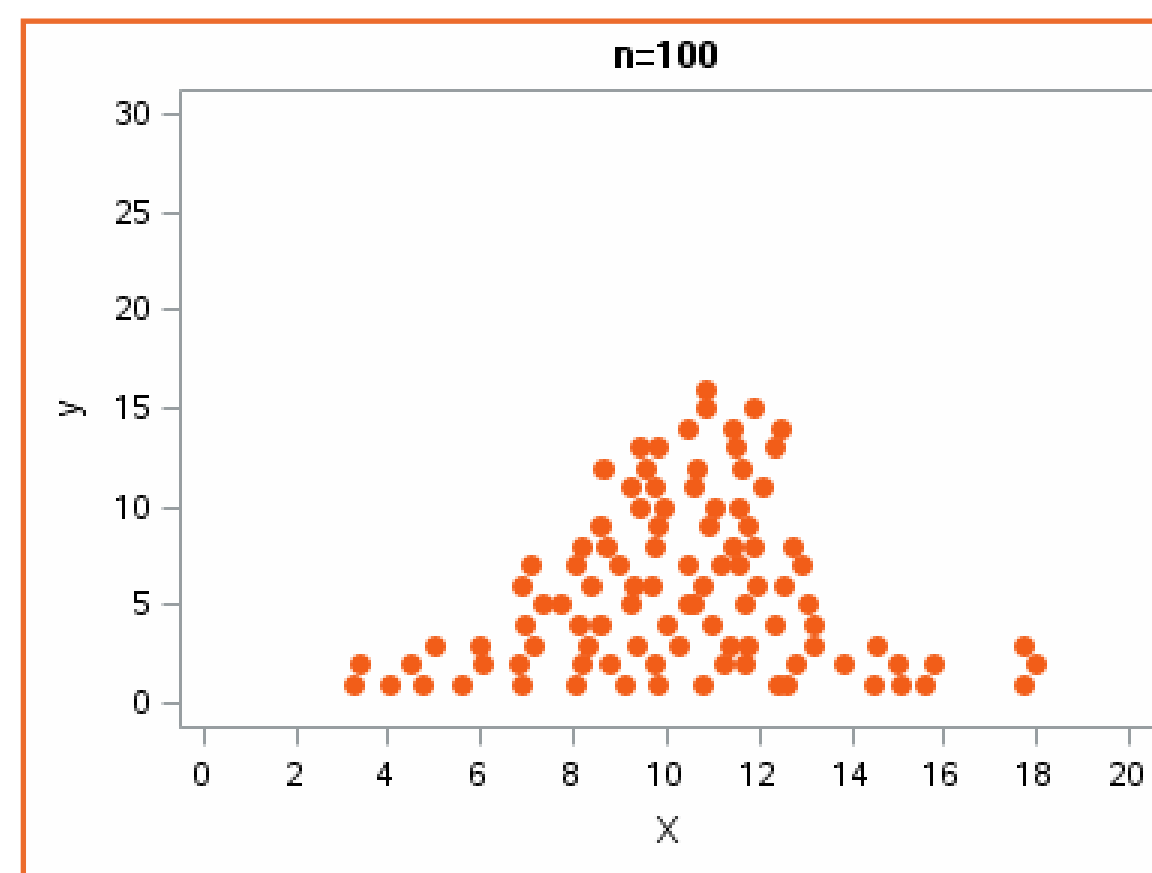
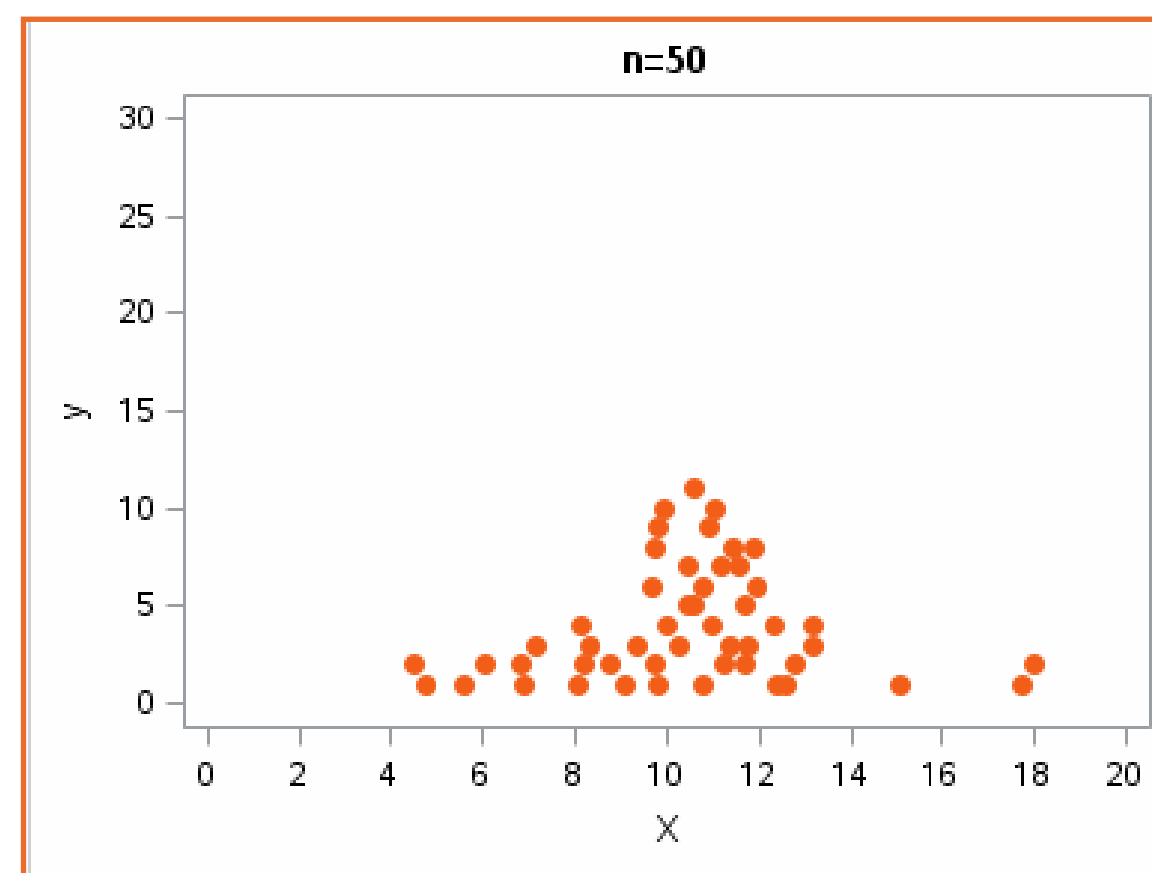
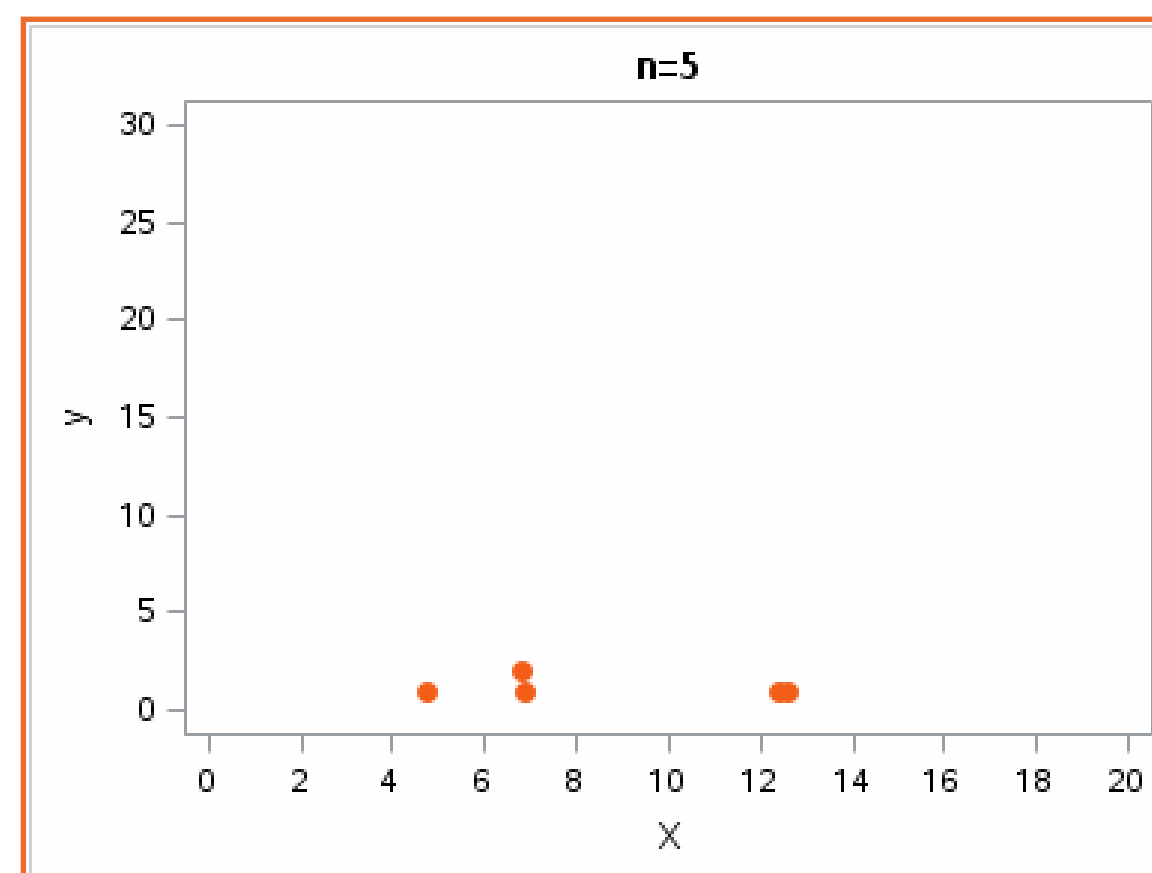
Introduction

I hope this poster will help other beginners approaching simulation for the first time not to fall into inefficient ways of simulating data. Creating animations was also a completely new process for me, and was the driving force behind my investigation of efficient simulation due to the unique requirements of simulated data for animation. The main requirement being the dataset must retain the values of all previous iterations for the purposes of the animation. This is the key requirement for animations and why simulating them efficiently is so important the final data will be used to create a time series of plots which will then be animated. I wanted to use this opportunity to explore languages beyond SAS. As a predominately SAS programmer I often hear about the efficiencies working with R brings and I wanted to put that premise to the test.

The Axioms of Efficient Simulation

- 1. Avoid Macro Do Loops**— If you are simulating using macro do loops stop! Macro loops call the procedure or data step for the number of iterations required so suffer from high overhead for minimal work.
- 2. Vectorise Your Program** — Anyone who uses R or has spent any time with a programmer who uses R.... will be aware that matrix and vector operations are more efficient than scalar operations.
- 3. Profile to Identify Trouble Spots** — Don't guess which parts of your program are causing your simulation to run slowly. Every programming language has ways of profiling the program to find which section is causing the slowdown.
- 4. Pre-allocating Arrays When Looping**— Allocate space prior to the loop that computes the results. Similarly if you need to simulate an array of random numbers do this in a single call.
- 5. Reduce The Number of Simulations**— How many simulations do you really need? Is 10,000 really required perhaps 1,000 is sufficient.
- 6. Estimating The Run Time** — Simulations tend to scale linearly with the number of iterations so if you perform a test run with 10 iterations and time how long it takes. When you run 1,000 iterations the final simulation should take 100 times longer than the test run.

Animation in SAS



Plotting the unrounded normally distributed values against the frequency of the rounded x values enables us to avoid the columns of dots that the rounded values or a dot plot would give us. The animation code itself uses ods graphics to produce a gif file containing our animation.

```
ods graphics / imagefmt=GIF width=4in height=3in; * each image is 4in x 3in GIF ;
options paperize=(4 in, 3 in);
nodate nonumber;
animationduration=0.5 animlog=yes noanimoverlay;
printerpath=gif animation=start; * start recording images to GIF ;
ods printer file="" File Directory""/Anim.gif; * images saved into animated GIF ;

proc sgplot data=plot;
  by n;
  scatter x=x y=y / markerattrs=(symbol=circlefilled color=#F25D18 size=8);
  xaxis values=(0 to 20 by 2); * set common vertical scale for all graphs ;
  yaxis values=(0 to 30 by 5);
run;

options printerpath=gif animation=stop; * stop recording images ;
ods printer close;
```

Working Towards an Efficient Simulation

Original inefficient code -
Note in the interest of space I removed the code but kept the comments to give you an idea of the structure of the program;
%let n=100; * n is the number of data points;
%macro loop(n); * Initialise a dataset with one row of data;
%do i=1 %to n;
* Compute random value with mean=10 and sd=2 to 1 dp;
* Give an initial y value of 1;
%do i=2 %to n;
* Iteratively creating each animation frame one by one;
* Starting with the values from the previous iteration;
* Insert an additional row of data growing our dataset by 1 observation;
* create a normally distributed value as a macro variable;
* Create a variable of our new norm. dist. value;
* Give the inserted row our new norm dist. value;
* Compare our new value to all previous values to increase y by 1;
* Append onto the previous iteration to retain the values for animation;
%end;
%mend;
%loop(n=&n);

- + Pre-allocate vector X1 and create a vector index from 1 to n.
- + Generate n random values producing all of the x-axis values we need with one call.
- + Create rounded values to group the x-axis values.
- + Append our vectors together creating a dataset containing all the details needed within one procedure.
- + Using the by/retain avoids doing a cell by cell comparison to create Y. It is a scalar operation, but it is only done on a relatively small dataset of size n.
- + We could use proc iml to do the data manipulation but from an efficiency perspective using a data step is better.
- + There is only one do loop in the simulation, outputting each frame into a single dataset using the 'n' variable as the ID for each frame.
- + Using a BY value of 5 also reduces the size of the resulting dataset even though the impact on the efficiency of the simulation is negligible it will speed up the creation of the animation.

- By using macro loops our simulation is going to suffer from a high overhead to work ratio.
- Creating our dataset from just one row of data may be intuitive but it is not efficient.
- For each iteration, 5 data steps/procedures are called within the loop. Each additional data step/procedure takes time, even if they only take 0.05 seconds over 1000 iterations that is 50 seconds.
- Currently the program computes one normally distributed value per iteration. Doing a row by row compare is resource intensive.
- A dataset is created for every iteration each increasing in size taking longer to read in and append onto.

```
%let n=100; * n is the number of data points;

proc iml;
  names=(index x Xround);
  i = T(1&n);
  X1 = j(1&n,1);
  call randseed(1234);
  call randgen(X1, 'normal', 10, 3);
  XR=round(X1,1);
  X2=insert(X1,0,2);
  X=insert(X2,XR,0,3);
  create sim1 from X [colname=names]; * Create sim1 dataset;
  append from X;
  close sim1;
quit;

proc sort data=sim1 out=sim2;
  by Xround index;
run;

data sim3;
  set sim2;
  by Xround;
  retain y;
  if first.Xround then Y=1;
  else Y=Y+1;
run;

data sim4;
  set sim3;
  do k=5 to &n by 5;
    n=k;
    if index=k then output;
  end;
  drop k;
run;

proc sort data=sim4;
  by n index;
run;
```



Simulation In R

```
## Required packages
library(tidyverse) library(tweenr) library(gganimate) library(ggfsk) library(ggplot2)
library(extraDistr)

obs_per_state <- seq(5,100, by=5) ## vector of observations per frame
max_obs <- seq(1,100) ## number of observations last frame
randomData1 <- rnorm(100, 10, 3) ## vector of simulated random values

## Creation of 2 lists: observations and per frame and id of each observation
tu_list <- list()
tu_list_2 <- list()

for (i in obs_per_state) {
  tu_list[[i]] <- randomData1[1:i]
  tu_list_2[[i]] <- max_obs[1:i]}

## Turning list into vectors to be used as columns in the animation dataset
tu_list_vec <- unlist(tu_list)
tu_list_vec_2 <- unlist(tu_list_2)

## Dataset for the animation
sim_Data1 <- tibble(
  ID = tu_list_vec_2,
  X = tu_list_vec,
  XROUND = round(tu_list_vec, digits = 0), ## rounds random values
  n = as.factor(rep(obs_per_state, obs_per_state)))

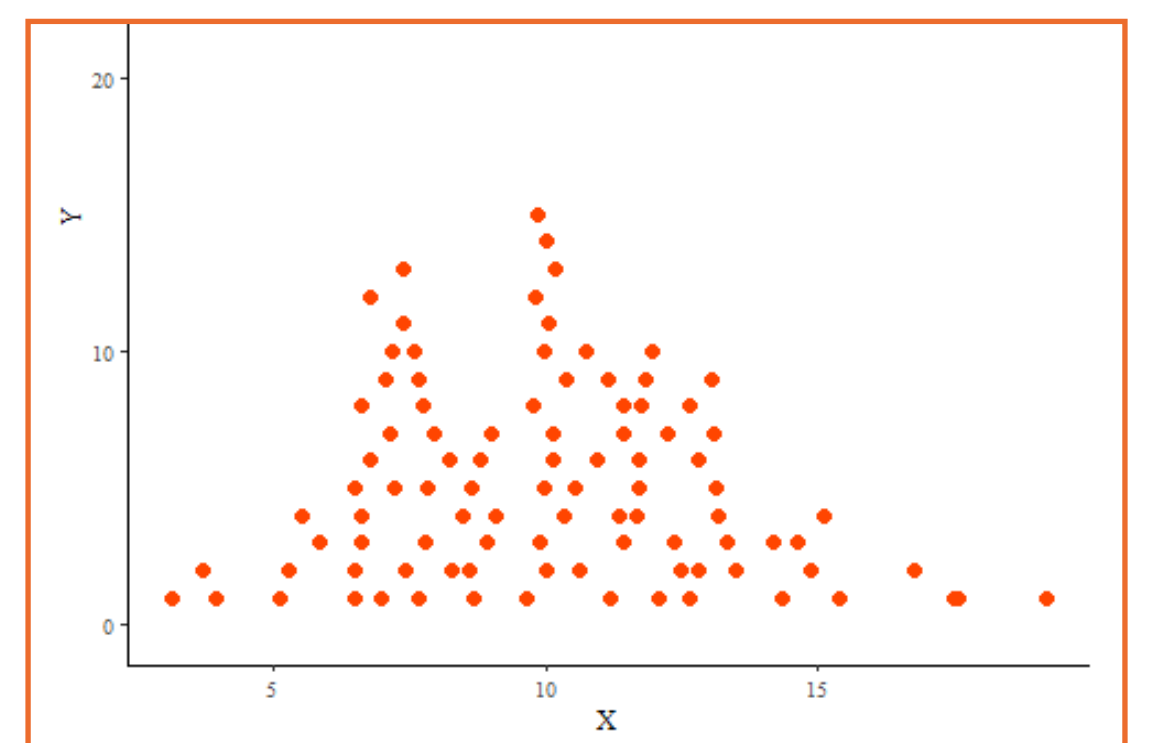
sim_Data2 <- select(sim_Data2, ID, X, XROUND, y_Frequency, n) # reordering vars

sim_Data3 <- select(sim_Data2, ID, X, XROUND, y_Frequency, n) # reordering vars

## This bit computes the cumulative frequency for each rounded value within each frame
sim_Data2 <- sim_Data1 %>%
  group_by(n) %>%
  mutate(y_Frequency = supply(1:length(XROUND),
    function(x) sum(XROUND[1:x] == XROUND[x])))
```

- The R script starts off by calling the required packages. This will induce additional load time with the first run.
- We start our simulation by creating three vectors containing: the number of observations per frame; a vector of ID values from 1 to n; our Normally distributed random values with mean 10 and standard deviation 3.
- Using the vector containing the number of observations per frame at the ith position in our list we populate two lists the first containing i random values and the second containing 1 to i values.
- We then unpack these lists into two vectors containing our random values and the corresponding ID value.
- We use tibble function, part of the tidyverse package, to create our simulated dataset. We also round our random values to use in the Y derivation and create our 'n' value which contains the number of observations per frame.
- Lastly we produce our Y value by grouping the XROUND column by n and then calculating the frequency of XROUND values within each group.

Animation In R



- Creating the Animation in R is relatively straightforward once you have the right packages installed.
- We can see the code efficiency of R when we do the comparison between code required in R and SAS, R only requires one line of code as opposed to 11 lines of code in SAS.

```
## animation
ggplot(sim_Data3, aes(x=X, y=y_Frequency)) +
  geom_point(size=3, color='orange') +
  transition_states(n, transition_length=1, state_length=5)
```

Runtime Comparison

```
%let timer_start = %sysfunc(datetime());
...
data _null_;
  dur = datetime() - &timer_start;
  put 30*"/ Total Duration : dur time13.2 / 30*";
run;
```

- Using the SAS code we can time both the simulation and also profile different sections of our code.
- Using this code I timed how long the my original simulation ran to compare it against the more efficient code to measure the improvement in time taken.
- I ran the efficient SAS code twice, once with a by value of 1 so that 1000 frames are produced; then again with a by value of 5 so that 200 frames are produced. I believe this gives a fairer comparison against the original SAS code.

- The table below shows the significant efficiency gained. The efficient SAS code runs over 180 times faster than the original code.
- Interestingly, increasing the by value to 5 has a minute impact on the efficiency, but note the resulting animation will be produced faster as it will have 1/5 of the frames.
- In fact when the n value is increased to 10,000 the simulation only took 5.06 seconds to run.
- I was slightly surprised by the time taken to run the R simulation which is 15 times slower for 1000 observations. But there could be mitigating factors both SAS and R are server based with similar capabilities but there is no way of knowing server load at time of run.
- Perhaps the R script is not the most efficient script for example the data manipulation section, in particular the derivation of the y_frequency variable, perhaps is an area where the SAS code has advantages over the R script.

n=1000	Original Inefficient Code	Efficient Code With By Value 1	Efficient Code With By Value 5	R Code with observations per frame 5
Time (seconds)	38.16	0.22	0.21	3.21

Conclusion

Simulating data is potentially not an every day task for clinical programmers, but the skills I have learned and developed have applications in all programming tasks where we are dealing with large amounts of data.

- One of the main skills is recognising what causes program to run slowly to be able to identify code which has a high overhead but is performing little work.
- Simulating data efficiently requires a change in mindset from dynamically creating your data observation-by-observation to instead building the vectors and matrices that you require and using them to create your required data with as few steps as possible.
- What I have also found is that SAS can perform efficiently, I think we need to be careful with the narrative that R is always more efficient than SAS. I believe it very much depends on the task at hand and it is about picking the right tool for the job.
- Working with R, one area where it is obviously more efficient is code efficiency. Only a relatively small amount of code is required to achieve the program aims.
- Animations are a great way to bring data to life and are a good option for training materials.

Acknowledgments

- Thank you to Gary Hearfield (Programming Director, GSK) for all of his support with the formatting and presentation of this poster.
- A special thank you to Sam Munoz Vicente (Statistician, GSK) for all of his assistance putting the R simulation together and answering my very basic questions on how R simulates data.

References

Wicklin, Rick. 2012. 'Eight tips to make your simulation run faster'. [ONLINE] Available at: <https://blogs.sas.com/content/iml/2012/06/06/tips-to-make-your-simulation-run-faster.html>. [Accessed 29 September 2019].
Wicklin, Rick. 2012. Simulation in SAS: The slow way or the BY way. [ONLINE] Available at: <https://blogs.sas.com/content/iml/2012/07/18/simulation-in-sas-the-slow-way-or-the-by-way.html>. [Accessed 29 September 2019].
Pratt, Jesse. 2016. Using Animation to Make Statistical Graphics Come to Life. [ONLINE] Available at: <https://support.sas.com/resources/papers/proceedings16/10920-2016.pdf>. [Accessed 29 September 2019].