2019 ■ Vol. 15 ■ no. 1

GRD for R: An intuitive tool for generating random data in R

Matias Calderini [a, ✉] and Bradley Harding [b]

[a] Université d'Ottawa
[b] Memorial University of Newfoundland - Grenfell Campus

Abstract ■ Statistics courses prove to be a common difficulty to many social sciences students. To address this problem, we developed a tool in the R programming language (R Core Team, 2018) that can be used to easily and quickly generate data to be analysed. The Generator of Random Data or GRD, allows the user to build datasets according to any design type, both within and between, with or without statistical effects and/or correlations. By default, GRD creates normally distributed data, but any type of distribution defined in R can be specified. With GRD, it is possible to generate samples of any size and see the benefit of larger sample sizes on the precision of statistical measures. The students of statistics can acquire better skills in analyzing custom-made datasets, skipping long data-acquisition processes. They can also experience first-hand concepts such as statistical power and type-I and type-II errors. Each sample being different, students can appreciate randomness at the tips of their fingers.

Keywords ■ Statistics Teaching; Data Simulations; Sampling; R.

✉ mcald052@uottawa.ca

MC: na; *BH*: 0000-0003-1222-8769

10.20982/tqmp.15.1.p001

Acting Editor ■ Denis Cousineau (Université d'Ottawa)

Introduction

The study of statistics is often dreaded or at the very least undesired by many students, particularly those in the social sciences (Onwuegbuzie & Seaman, 1995). This is reflected both in lower grade averages relative to other courses (Cantinotti, Lalande, Ferlatte, & Cousineau, 2017), and in the reported levels of anxiety it causes to students (Vigil-Colet, Lorenzo-Seva, & Condon, 2008). In addition, the learning and teaching of applied statistics can sometimes be hampered by a complete reliance on abstract mathematical concepts (Cousineau & Harding, 2017). Often, students lack simple, approachable datasets on which they can apply and practice their newly acquired knowledge. Other times, students are provided trivially simple datasets from which they manually compute their measures of interest. This can be doubly inefficient because most statistics outside of the classroom are performed using software packages (*i. e.*, SPSS, SAS) or programming languages (*i. e.*, R, Python, Matlab) which are rarely exploited except in advanced or specialized courses. In short, the teaching of statistics at the undergraduate level –for most, the entry and only point of contact with statistics– is hardly benefiting both in content and methods from the technological advancements and trends of our era.

Herein, we present a generator of random data (GRD) for the programming language R (R Core Team, 2018), inspired from a previously published extension to SPSS (Harding & Cousineau, 2014, 2015). The function GRD and this accompanying document allows users with no prior programming experience to generate customizable random datasets so that they can practice statistical procedures and statistical testing. The created datasets can be specified to reflect any combination of experimental designs and probability distributions. It can contain underlying effects between groups and/or across repeated measures, both as main effects and as interactions. Most importantly, the dataset is in dataframe format, which can be readily analyzed using common R commands. Thus, the learner of statistics can have new and different datasets in just a matter of seconds.

Basic Use

Loading* GRD *into memory

Before using GRD, the commands shown below must be executed to load GRD into memory.[1]

```
setwd('c:\\users\\me\\GRD{}_Directory')
source('GRD_20.R')
```

The first command sets the current working directory to the directory specified within quotes; you must change it to the folder\\subfolders containing the source file GRD_20.R (the current version is 2.0). The second command tells R to import the GRD source file by its name. Once these instructions have been executed, GRD is active in memory. This step is done only once per session.

Generating a dataset with default specifications

The command

```
dta <- GRD()
```

creates the variable dta which contains a randomly generated data frame. The name of the variable dta is arbitrary and can be replaced by any name of your choice. For simplicity, throughout the text, we will assume that the variable name is dta. Note that R is case-sensitive.

The variable dta contains random data. By default, GRD creates a random sample of size 100, following the standard normal distribution (i. e., *z* scores) for a design with no between-subject factor and only one variable measured once (i. e., no within-subject factor). To print only the first or last few rows of the dataset, use the commands head(dta) or tail(dta) respectively, as seen below. With head(dta,n) or tail(dta,n), the number of lines n to be shown can be specified. In what follows, we show the output produced by R as comments begining with "#"

```
head(dta$Data,2)
#   id         DV
# 1  1 -0.4135888
# 2  2 -0.4638726

tail(dta$Data,2)
#       id         DV
# 99    99  0.7616850
# 100 100  0.4953989
```

On this simple dataset, there are only few things to mention. The first column, id, contains the "subject" number and the second, DV, contains the score obtained for that "subject". As between-subject factors and repeated measures will be added, more columns and rows will appear in the data frame. For an illustration of the data, we can plot a frequency distribution plot of the data (a.k.a. a *histogram*) with hist(dta$DV). The resulting plot should look like that of Figure 1, first panel.

At this point, the shape of the distribution is worth a brief discussion. While the distribution from which GRD randomly samples its data is by default the standard normal distribution, the shape of the actual data will not necessarily appear normally distributed. This random sampling effect arises from the limited number of data points that compose the dataset (by default, $N = 100$). The bigger the sample size, the closer the distributions and statistical measures will be to the true population distribution. Once the argument to change the sizes of experimental groups will be introduced in the next section, the student is encouraged to explore the effect of random sampling on the distribution of the data.

As described next, the command can be given arguments to change the default specifications. The seven arguments (plus two to display informations) are presented in Table 1. The rest of this article describes how to use these arguments to tailor the desired random dataset.

By default, and as was seen in the above outputs, the column containing the simulated scores is called DV, standing for *dependant variable*. To change the name of the dependent variable from 'DV' to any desired name, use the RenameDV argument. The new name must be given within quotes. Note that because ':' and ',' are used as separators in the subsequent arguments, it cannot be used as part of a variable or level name.

The following changes the name of the dependant variable from 'DV' to 'score':

```
dta <- GRD( RenameDV = "score" )
```

Note that spaces are optional anywhere in GRD and that the command can span multiple lines, as always in R. The order of the arguments is also irrelevant.

Changing the experimental design and the sample size

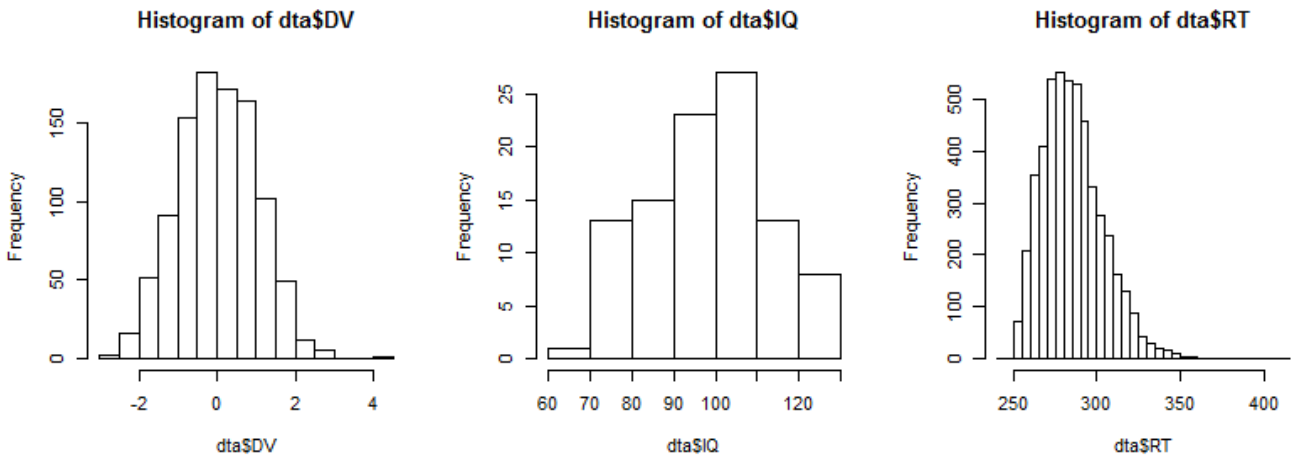
The two arguments to customize the experimental design are BSFactors and WSFactors whereas SubjectsPerGroup is used to changed the sample size.

Adding groups by specifying between-subject factors

We start tailoring the data set defining between-subject factors with the BSFactors argument. It requires one obligatory input which is a string following a specific format that indicates the between-subject factors of the experimental design as well as their levels. For a fully tailored dataset, one can specify the name of independent variables

[1] When starting a new project, it is good practice to remove all previously used variables. This helps in avoiding errors due to inappropriate use of old data, data structures and functions. In R, this can be done by the command rm(list = ls()) which clears from memory all variables in the workspace.

Figure 1 ■ Three examples of data generated with GRD.



(or leave them unspecified, in which case generic names are generated) and their respective levels within parenthesis. GRD does not impose any restrictions on the number of between-subject (BS) factors or their number of levels. However, one must be cautious in adding factors because the complexity of the dataset increases rapidly when the design is expanded. The design specification can be inputted using four increasingly detailed forms. In all four cases, different BS factors are separated by ":". Spaces are optional.

In its easiest form, specify the number of levels for a single factor with:

```
dta <- GRD( BSFactors = '3')
```

The head of the data frame generated is

```
head(dta,2)
#   id BS1         DV
#1   1   1  0.7587990
#2   2   1 -0.8917315
```

As seen, a new column is added, whose default name is BSi where i is 1 for the first between-subject factor created.

For illustrative purposes, let's assume that we wish to create a dataset simulating the following experimental design: A ficticious study which recruits participants attending different types of therapy divided by whether they had eye surgery in the past or not. Their performance on a visual task is evaluated under different contrast levels. In formal terms, this is a study with two between-subject factors (*Therapy* and *Surgery*) and one within-subject factor (*Contrast*). Table 2 shows all the possible combinations of between-subjects and within-subjects factors from this design.

From the above example, the between-subject factors could be specified using the number of levels with:

```
dta <- GRD(BSFactors ='2 : 3' )
```

Instead of the number of levels, the names of each of the levels could be provided, still leaving the factors unnamed:

```
dta <- GRD( BSFactors = '(yes, no) :
    (CBT, Control, Exercise)')
```

Inversely, one could name the between-subject factors and provide the number of levels it contains, without naming them, in which case successive integers are used:

```
dta <- GRD( BSFactors = 'Surgery(2) :
    Therapy(3)' )
```

Finally, one can specify both the factor names and their respective levels:

```
dta <- GRD(
  BSFactors ='Surgery(yes, no) : Therapy
    (CBT,Control,Exercise)'
)
```

All of these formats can be mixed together, for example:

```
dta <- GRD(BSFactors = '(yes, no) :
    Therapy(3)')
```

It is possible to have a factor with a single level. Note that a numerical name cannot be assigned to levels in this

Table 1 ■ Arguments to GRD

Argument name	Description
SubjectsPerGroup	Changes the sizes of the groups [a]
RenameDV	Renames the dependent variable [b]
BSFactors	Defines between-subject factors and their respective levels [c]
WSFactors	Defines within-subject factors and their respective levels [c]
Effects	Sets the effects [d]
Population	Adjusts the characteristics of the population [e]
Contaminant	Adjusts the characteristics of a source of contaminants [e]
Summary	Prints an overview of the design requested [f]
Debug	Prints debugging information (for programming purposes only) [f]

Note. [a]: Default is 100 subjects per group;
[b]: Default is DV;
[c]: Default is "" which represents no factor;
[d]: Default is an empty list of effects, list();
[e]: Default is a standard normal distribution, and the proportion of contaminants is 0.00;
[f]: Default is FALSE.

case, as it will be confused with the number of levels instead of its name. *A contrario*, if the numerical level is not the only level listed (numerical or otherwise), it will be considered as a level label. In the case of Therapy(1, 3) and Therapy(3, CBT), the "3" in both commands would be considered as a label name. When label names are provided, they are treated as character string by GRD whereas when the number of levels is provided, the levels generated (from 1 to the number given) are numerical integers.

Adding repeated measures by using within-subject factors

The WSFactors argument allows one to create experimental design with repeated measures. The argument WSFactors receives the same type of input as the BSFactors: a string where the within-subject factors are specified separated by ':'. As shown before, the input can be increasingly more detailed. The format remains the same as before, so we only show examples of the least and most detailed formats:

```
dta <- GRD( WSFactors='3' )
dta <- GRD( WSFactors=' Contrast(Low,
    Medium,High)' )
```

For each combination of within-subject factors, one column is added to the dataset with the format DV.*leveloffactor1.leveloffactor2.etc*. All of these combinations can be seen when head(dta,2) is used:

```
# id     DV.Low       DV.Medium   DV.High
# 1     0.05163949  -0.57360397  -0.2073698
# 2    -0.55032289  -0.05957762   0.5362954
```

Mixed designs

Between-subject and within-subject factors can be intermixed in the same instructions to create mixed designs datasets. For example,

```
dta <- GRD(
  BSFactors = 'Surgery(yes,no) : Therapy
    (CBT, Control,Exercise)',
  WSFactors = 'Contrast(C1,C2,C3)',
)
```

Changing the number of participants per groups

By default, GRD creates 100 data points (*e.g.* participants) per groups. To change the number of data in each group, use the SubjectsPerGroup argument. As an example, the following

```
dta <- GRD( SubjectsPerGroup = 1000 )
```

increases the number of participants to 1000, as can be verified with dim(dta).

When multiple groups are simulated, they all have the same size. Groups of different sizes can be specified by providing their respective sizes as a sequence of integer numbers as input, e. g., with

```
dta <- GRD(
  BSFactors = "3",
  SubjectsPerGroup = c(20,25,50)
)
```

The length of the sequence of sample sizes must match the number of groups.

Table 2 ■ Illustration of an experimental design in a $2 \times 3 \times (3)$ design with factors Surgery × Therapy × Contrast.

Surgery	Therapy	DV.1	DV.2	DV.3
yes	CBT			
no	CBT			
yes	Exercise			
no	Exercise			
yes	Control			
no	Control			

Changing the population characteristics

Data following a Gaussian distribution

Data is sampled from a population with a given distribution. By default, the population distribution is a standardized normal distribution, that is, a normal distribution with mean (μ) equal to 0 and standard deviation (σ) equal to 1. While the normal distribution is ubiquitous in statistical analysis, rarely does one observe a mean of 0 and a variance of 1. In GRD, we can set the parameters of the Gaussian distribution to arbitrary mean and variance using the Population argument. To do so, it is necessary to provide the mean and standard deviation within a list with the attributes mean and stddev respectively.

As an example, to simulate IQ scores –whose population mean is 100 and population standard deviation is 15–, the following can be used:

```
dta <- GRD(
  RenameDV = "IQ",
  Population=list(mean=100,stddev=15)
)
hist(dta$IQ)
```

Figure 1, second panel, shows the resulting dataset; notice how sample mean is close to $\mu = 100$ and standard deviation is close to $\sigma = 15$.

Changing the distribution directly

Internally, GRD uses a random number generator based on rnorm. The full specification is rnorm(1, mean = GM, sd = STDDEV) where GM and STDDEV are initialized by the attributes seen in the above section. It is possible to change the full specification, with the Population argument, by specifying the attribute scores. For example, you could have a sample of constants with

```
dta <- GRD(
  BSFactors = "Group(2)",
  Population = list( scores = "1" )
)
```

If two groups were defined, the group number could be used as the constant

```
dta <- GRD(
  BSFactors = "Group(2)",
  Population = list( scores = "Group" )
)
```

As a last example, the following command creates two groups, and the standard deviation in each group is proportional to the group number:

```
dta <- GRD(
  BSFactors = "Group(2)",
  Population = list(
    mean = 100,       # will set GM to 100
    stddev = 15,      # will set STDDEV to 15
    scores = "rnorm(1, mean = GM, sd =
    STDDEV*Group)"
  )
)
```

Any factor created in the BSFactors or WSFactors can be used anywhere in the scores attribute. Note that it would have been simpler to just use constants in the scores attribute with

```
  ...
  Population = list(
    scores = "rnorm(1, mean = 100, sd =
    15*Group)"
  ...
```

The example creates data that violate the homogeneity of variances assumption. This can be tested using the Levene test which is available, for example, in the package lawstat (Gastwirth et al., 2017) with

```
library(lawstat)
levene.test(dta$DV, dta$Group, location=
    "mean")
# data:  dta$DV
# Test Statistic = 32.567, p-value = 4.157e-08
```

so that with the present dataset, homogeneity of variance is rejected (Levene's $F(2, 198) = 32.6$, $p < .001$). Make sure that the first argument to rnorm (the number of data

sampled from the distribution) is 1 as GRD proceeds one line at a time.

(Advanced) Switching to an arbitrary theoretical distribution

For most uses, Gaussian distributed data is an adequate approximation of measured data. But some phenomena might be better represented by other distributions. GRD can accommodate this need with the scores attribute in which a different theoretical distribution is specified.

The distribution must be available in the R language. For example, if one wanted data that are sampled from a Weibull distribution, the scores attribute of Population is changed. The Weibull distribution is a good approach to simulating response times (RT) data (Cousineau, Brown, & Heathcote, 2004):

```
dta <- GRD(SubjectsPerGroup = 5000,
  RenameDV = "RT",
  Population=list(
    scores = "rweibull(1, shape=2, scale
    =40)+250"
  )
)
hist(dta$RT,breaks=seq(250,425,by=5))
```

The resulting data are illustrated in Figure 1, third panel. Again, whatever the distribution used, it is necessary that the first argument (the number of data sampled from the distribution) be equal to 1.

For reference, Table 3 contains some common theoretical distributions. For help on how to use them, one can simply type help(<function>).

Adding statistical effects

Once the between- and within- subject designs are specified with the desired number of participants per group, one can start tailoring the difference between them. The argument Effects allows modifying the effect size of different factors by shifting their means. GRD can specify effects for multiple factors independently or from interactions of factor levels. We first define the between and within-subject factors described previously and use larger group sizes with:

```
dta <- GRD(
  BSFactors = 'Therapy(CBT, Control,
    Exercise)',
  WSFactors = 'Contrast(3)',
  SubjectsPerGroup = 1000,
  <<insert one or many effects as
    described below>>
)
```

The input to Effects is a list of effects, each based on a selection of factors to be modified and the effect to be applied. The type of effect must be one of the four effect types offered by GRD, namely range, slope, custom and Rexpression. The range and slope effect types require only one number. For range, this number represents the difference between the highest mean and the lowest mean of all groups of interest. The mean of the means across all levels will remain at its original value, but the mean of the individual levels will be uniformly distributed on either side within the range specified by the magnitude. For the slope type, the magnitude represents the difference between two consecutive means. In other words, the mean of means will again be centred at its original value, but this time, the mean of the individual levels will be linearly distributed with slope given by the magnitude. Finally, the custom attribute, allows the user to specify the specific shifts for all subsets of data separately.

As an example, We can define a slope effect of 2 points for the 'Therapy' factor, displacing each of its groups ('CBT', 'Control' and 'Exercise') respectively by -2, 0 and +2.

```
  ...
  Effects = list('Therapy' = slope(2))
  ...
```

The result for one of the repeated measure variable is seen on Figure 2, top row, using the histogram command from the lattice library (Sarkar, 2008), with the commands:

```
library(lattice)
histogram(~ DV.1 | Therapy, data = dta,
    breaks=seq(-7,7,by=0.5),layout = c
    (3,1) )
```

In that figure, we did not change the base standard deviation from its default value of 1. A difference in means of 2 consequently corresponds to a Cohen's d of 2, a rather large effect that is easily seen on the plot (Goulet-Pelletier & Cousineau, 2018).

Effects can be applied on within-subject factors as well. For example:

```
  ...
  Effects = list('Contrast'= range(4) )
  ...
```

Visualizing effects that are located among repeated measure variables can be more difficult. The approach suggested here is to transform the data frame from wide format to long format. This can be done in multitude of ways. Here, we use the wideToLong command from the lsr library (Navarro, 2015) which match factors with the labels found after "DV." in the dependant variable names:

```
library(lsr)
```

Table 3 ■ Examples of random number generator functions for common distributions that can be used in GRD

Distribution	R function	Distribution	R function
Constant	1	Poisson	rpois()
Uniform	runif()	t distribution	rt()
Bernouilli	rbern()	Weibull	rweibull()
Binomial	rbinom()	Exponential	rexp()
Geometric	rgeom()	Chi-Squared	rchisq()

```
dta2 <- wideToLong(dta, within = c("
    Contrast"),sep=".")
histogram(~ DV | Contrast, data = dta2,
    breaks=seq(-7,7,by=0.5),layout = c
    (3,1), aspect =1)
```

The results are seen in Figure 2, second row.

Both slope and range are linear effects: their impact is gradual on every level of the factor(s) named. For non-linear effects, it is possible to specify custom effects, one per cell of the specified factor(s).

```
  ...
  Effects = list( "Therapy"=range(4) )
  ...
```

There must be as many effect sizes listed in the custom effect specification as there are levels in the factor named (here Therapy). The result is seen in Figure 2, bottom panel.

More generally, GRD allows the specification of interaction effects using '*' to separate different factors. This effectively creates sub-groups from every combination of levels of each factor specified. Interactions can be specified for as many factors as needed. For example, interactions between the within and between factors 'Therapy' and 'Contrast' is obtained with:

```
  ...
  Effects = list('Therapy * Contrast' =
    custom(XXXXXx))
  ...
```

It is also possible to provide any R expressions that can be used to compute an effect. In this case, there is no need to provide a factor name, but a string must be provided in all case. Hence, arbitrary strings such as 'code1' can be used. One example is

```
dta <- GRD(
  BSFactors = 'Therapy(CBT,Control,
    Exercise)',
  WSFactors = 'Contrast(3) ',
  SubjectsPerGroup = 1000,
  Effects = list(
    "code1"=Rexpression("if (Therapy ==
      'CBT'){-50} else {0}"),
     "code2"=Rexpression("if (Contrat ==
     3)    {+50} else {0}")
  )
)
library(lsr)
dta2 <- wideToLong(dta, within = c("
    Contrast"), sep = ".")
histogram(~ DV | Contrast + Therapy,
    data=dta2,
  breaks = seq(min(dta2$DV)-5,max(dta2$
    DV)+5,by=2.5)
)
```

Note the use of single quotes within double-quote specifications.

Rexpression is any expression which can be applied to the subject "id", and the factor(s) names. For example, one could print the subject number as lines are processed by GRD:

```
  ...
  Effects = list( "code1" = Rexpression(
    "print(id);0") )
  ...
```

in which ;0 is added to the expression so that the data are not affected by this line of code (the effect is zero).

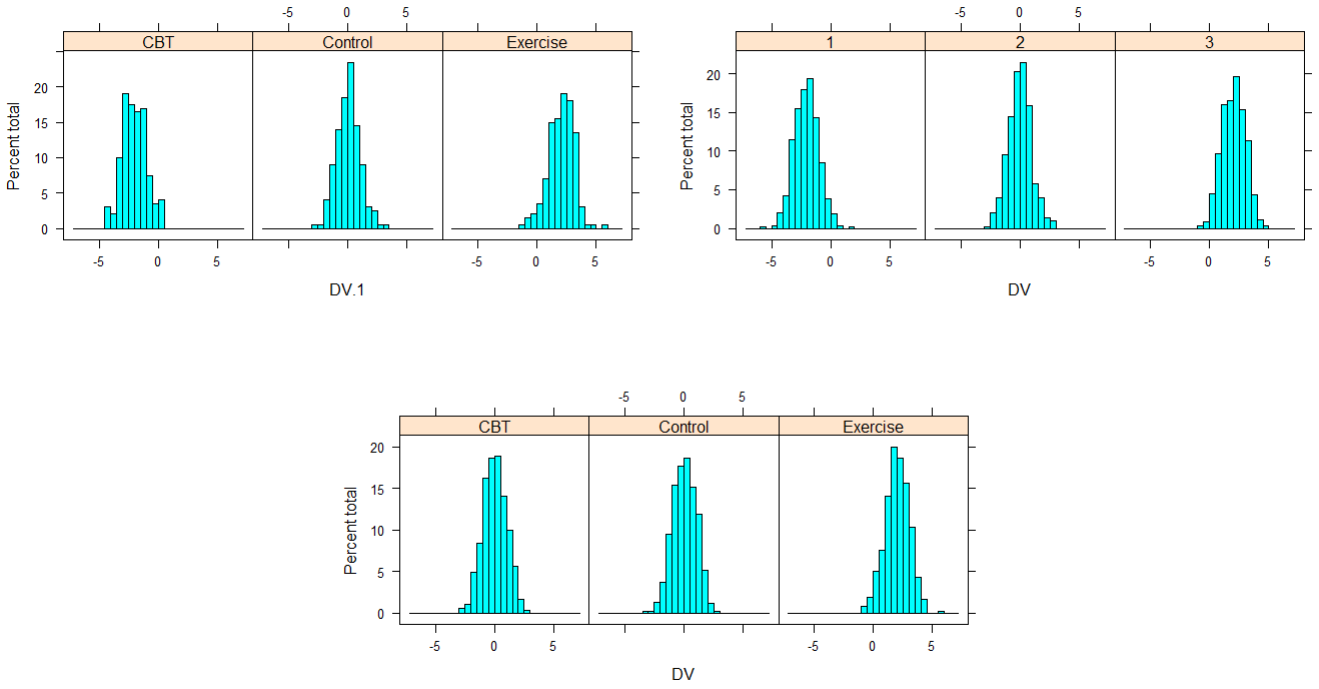
(Advanced) Working with multivariate distributions

When exploring a dataset containing repeated measures, a frequent measure of interest is the correlation between different experimental groups. GRD allows to create multivariate correlated data with the Population argument rho. By default, a multivariate normal distribution is generated but this default can be changed.

As an example, we generate two repeated measures with the within-subject factor Difficulty. We request a correlation of $\rho = 0.5$:

```
dta <- GRD(
  WSFactors = 'Difficulty(1, 2)',
  SubjectsPerGroup = 1000,
  Population=list(mean = 0,stddev = 20,
    rho = 0.5)
```

Figure 2 ■ Three examples of effects generated with GRD. The experimental design is a $2 \times 3 \times (3 \times 2)$ with factors *Reply*, *Therapy*, *Contrast* and *Size*. Top left panel shows a slope effect of 2 on Therapy, the top right, a range effect of 4 on Surgery, and the bottom panel, a custom effect of 0, 0 and 2 for the levels *CBT*, *Control* and *Exercice* respectively.




```
)
plot(dta$DV.1, dta$DV.2)
```

In this example, whose result is seen in Figure 3, first panel, the standard deviations are all constant to 20 and the covariances are all identical. This corresponds to *compound symmetry*, a much less stringent requirement than sphericity in order to run linear model analysis such as ANOVAs (Cousineau, submitted).

We can apply a correlation of -0.85 to the 'Difficulty' factor, and we can specify distinct means and standard deviations to each of the variables with:

```
dta <- GRD(
  WSFactors = 'Difficulty(1, 2)',
  SubjectsPerGroup = 1000,
  Population=list(mean = c(10,2),stddev
    = c(1,0.2),rho =-0.85)
)
plot(dta$DV.1, dta$DV.2)
```

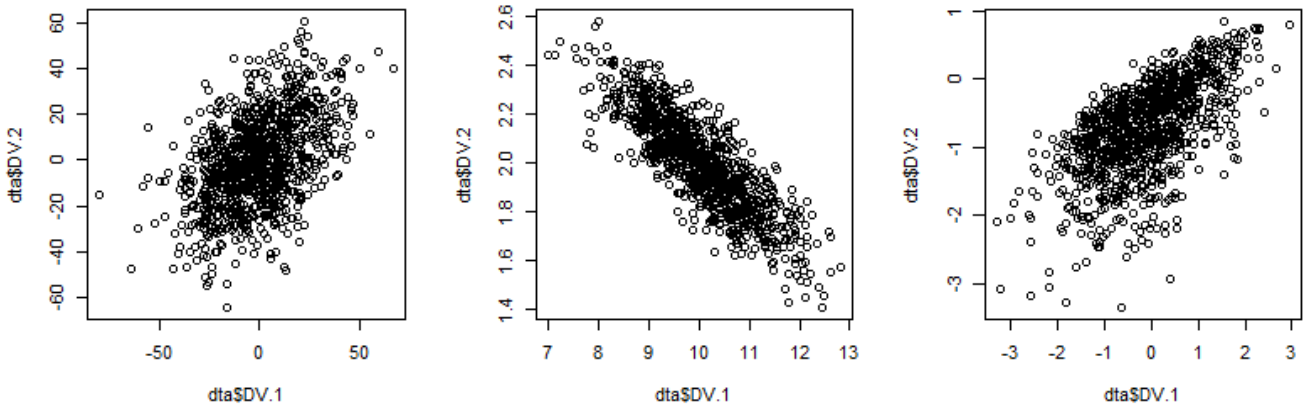
The result is seen in the central panel of Figure 3. Note that with negative ρ's, the covariance matrix sometimes become non-positive definite and consequently, no data are generted. In that case use a less extreme ρ.

Arbitrary multivariate function

As soon as ρ is set to a non-null value, GRD is switched to a multivariate mode. From that moment, any multivariate distribution can be specified in the scores option of the Population argument. Here, we show an example with the *multivariate skewed normal distribution*, an extension of the multivariate normal distribution which includes an asymmetry parameter α (Wuertz, Setz, & Chalabi, 2017). To generate random data following this distribution, we first set the parameters of that distribution, ξ (a measure of central tendency), Ω (a matrix of covariance) and α that characterizes the asymmetry:

```
library(fMultivar)
xi    <- c(0,0)
Omega <- diag(2)
Omega[1,2] <- Omega[2,1] <- 0.5
```

Figure 3 ■ Three examples of bivariate data generated with GRD. The first plot shows a bivariate normal distribution with a correlation of 0.85, the second with a negative correlation of 0.3 and the last shows a multinormal skewed distribution with parameters $\xi = \{0, 0\}, \Omega = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$ and $\alpha = \{2, -6\}$.




```
alpha <- c(2,-6)
```

The function GRD is then called with the scores option set with the function sn::rmns followed by the parameters of that function. Note that if you defined mean or stddev, these will not be used by the function.

```
dta <- GRD(
  WSFactors = "Difficulty(2)",
  SubjectsPerGroup = 1000,
  Population=list(
    rho = 99,     # any non-zero value will do
    scores = "sn::rmsn(1,xi,Omega,alpha)
   ")
)
plot(dta$DV.1, dta$DV.2)
```

The result is seen in Figure 3, third panel.

Adding contaminants

It is possible to define a second population. That second population can be used to "contaminate" the scores obtained from the population of interest. The argument Contaminant works in the same way than Population except that an additional parameter is provided, proportion. By default, the proportion of contaminant is zero. However, if you set that proportion to be different from zero, then some of the data from the population will be replaced by scores from the contaminating population. The substitution is random but on average the proportion of scores replaced should match that parameter.

With this example,

```
dta <- GRD(SubjectsPerGroup = 5000,
  Population= list( mean=100, stddev =
    15 ),
  Contaminant=list( mean=200, stddev =
    15, proportion = 0.10 )
)
hist(dta$DV,breaks=seq(0,265,by=2.5))
```

approximately 10% of contaminants are added having a mean of 200. The result is seen in Figure 4, first panel.

Any population can be defined for the contaminant as well, such as

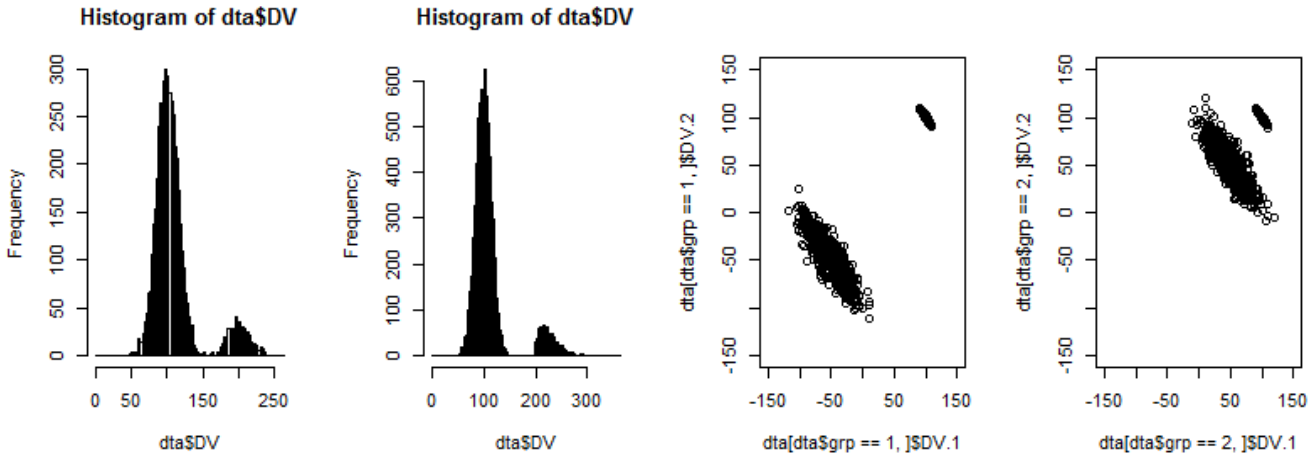
```
dta <- GRD(SubjectsPerGroup = 10000,
  Population=list( mean=100, stddev = 15
    ),
  Contaminant=list( proportion = 0.10,
    scores="rweibull(1,shape=1.5, scale
   =30)+1.5*GM"
  )
)
hist(dta$DV,breaks=seq(0,365,by=2.5))
```

GM and STDDEV are set by the Population argument only. The result is seen in Figure 4, second panel.

Also, the contaminants can be multivariate, as for example:

```
dta <- GRD( BSFactors="grp(2)",
    WSFactors = "Moment (2)",
```

Figure 4 ■ Three examples of samples with contamination. The first panel represents a univariate population along with univariate contaminants; the second has contaminants following a different theoretical distribution; the last two panels illustrate multivariate contaminants for two groups. The effect affects the population data only, not the contaminants.




```
    SubjectsPerGroup = 1000,
    Effects = list("grp" = slope(100) ),
    Population=list(mean=0,stddev=20,rho=
    -0.85),
    Contaminant=list(mean=100,stddev=4,
    rho=-0.99,proportion=0.2)
)
par(mfrow=c(1,2))
plot(dta[dta$grp == 1,]$DV.1,dta[dta$grp
    ==1,]$DV.2,
     ylim=c(-150,150), xlim=c(-150,150))
plot(dta[dta$grp == 2,]$DV.1,dta[dta$grp
    ==2,]$DV.2,
     ylim=c(-150,150), xlim=c(-150,150))
```

As seen with this example plotting the results separately for both groups, any effect defined will affect only the true population, not the contaminants which are unaffected by conditions and effects. Figure 4, last two panels, shows the two plots.

Finally, contaminants can be used to insert missing data in a random fashion with

```
dta <- GRD(
  SubjectsPerGroup = 5000,
  Population = list(mean = 100, stddev =
    15 ),
  Contaminant = list(scores = 'NA',
   proportion = 0.1)
)
```

Conclusion

This function is based on GRD for SPSS (Harding & Cousineau, 2014, 2015). Most of the functions herein are identical to those in SPSS. The present however extends previous work by giving access to any multivariate distribution whereas GRD for SPSS was limited to the multinormal distribution. Also, the present offers the possibility to enter effects using arbitrary R expressions. However, note that for very large samples, GRD for SPSS runs faster. More broadly, GRD improves on its previous SPSS version in that it is based on a free software environment widely used in the academic world. Since the research community is gradually but decisively phasing out from pre-built packages such as SPSS or SAS, it is necessary for students to be trained so that their skill-set matches the requirements of their future careers.

GRD provides students with an approachable tool to create custom-tailored data sets for statistical exploration. It is built on intuitive syntax and supported by powerful random number generators. It aims to facilitate the practice and learning of concepts such as central tendency measures, the law of big numbers, type-I and type-II errors, probability distributions and experimental designs.

Furthermore, it introduces the user to basic commands in R language. In that aspect, GRD helps students take their first steps into data-science tools and methods that are both ubiquitous in our modern era and rapidly growing. It is the authors' hope that GRD will be a useful tool in the teaching of statistics and one of the first stepping-

stones for students to dive into the world of statistics and data-science.

References

Cantinotti, M., Lalande, D., Ferlatte, M.-A., & Cousineau, D. (2017). Validation de la version francophone du questionnaire d'anxiété statistique (sas-f-24). *Canadian Journal of Behavioural Science*, *49*, 133–142. doi:10.1037/cbs0000074

Cousineau, D. (submitted). Correlation-adjusted standard errors and confidence intervals for within-subject designs: A (much) simpler solution. *The Quantitative Methods for Psychology*, *na*, ss–ss.

Cousineau, D., Brown, S., & Heathcote, A. (2004). Fitting distributions using maximum likelihood: Methods and packages. *Behavior Research Methods, Instruments, & Computers*, *36*(4), 742–756. doi:10.3758/BF03206555

Cousineau, D., & Harding, B. (2017). Why statistics is difficult to teach: A few considerations. *Mesure et évaluation*, *46*, 397–419.

Gastwirth, J. L., Gel, Y. R., Hui, W. L. W., Lyubchich, V., Miao, W., & Noguchi, K. (2017). *Lawstat: Tools for biostatistics, public policy, and law*. R package version 3.2. Retrieved from https://CRAN.R-project.org/package=lawstat

Goulet-Pelletier, J.-C., & Cousineau, D. (2018). A review of effect sizes and their confidence intervals, part i: The cohen's d family. *The Quantitative Methods for Psychology*, *14*, 242–265. doi:10.20982/tqmp.14.4.p242

Harding, B., & Cousineau, D. (2014). GRD: An SPSS extension command for generating random data. *The Quantitative Methods for Psychology*, *10*, 80–94. doi:10.20982/tqmp.10.2.p080

Harding, B., & Cousineau, D. (2015). GRD 2.0: An extended SPSS extension command for generating random data. *The Quantitative Methods for Psychology*, *11*, 127–138. doi:10.20982/tqmp.11.3.p127

Navarro, D. (2015). *Learning statistics with R: A tutorial for psychology students and other beginners. (version 0.5)*. R package version 0.5. University of Adelaide. Adelaide, Australia. Retrieved from http://ua.edu.au/ccs/teaching/lsr

Onwuegbuzie, A. J., & Seaman, M. A. (1995). The effect of time constraints and statistics test anxiety on test performance in a statistics course. *Journal of Experimental Education*, *63*, 115–124.

R Core Team. (2018). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria. Retrieved from https://www.R-project.org/

Sarkar, D. (2008). *Lattice: Multivariate data visualization with R*. ISBN 978-0-387-75968-5. New York: Springer. Retrieved from http://lmdvr.r-forge.r-project.org

Vigil-Colet, A., Lorenzo-Seva, U., & Condon, L. (2008). Development and validation of the statistical anxiety scale. *Psicothema*, *20*, 174–180.

Wuertz, D., Setz, T., & Chalabi, Y. (2017). *Fmultivar: Rmetrics - analysing and modeling multivariate financial return distributions*. R package version 3042.80. Retrieved from https://CRAN.R-project.org/package=fMultivar

Open practices

The *Open Material* badge was earned because supplementary material(s) are available on the journal's web site.

Citation

Calderini, M., & Harding, B. (2019). GRD for R: An intuitive tool for generating random data in R. *The Quantitative Methods for Psychology*, *15*(1), 1–11. doi:10.20982/tqmp.15.1.p001

Received: 18/01/2019 ~ Accepted: 28/01/2019