Monte Carlo simulation - using a random number generator to calculate statistical quantities such as power

Alan E. Berger 21 May 2021 v7

Summary of Monte Carlo simulation

Using a suitable random number generator to simulate statistical data and thereby obtain a good approximation for a statistical quantity is a versatile technique. This is often referred to as doing a *Monte Carlo simulation*. We will illustrate this approach by obtaining good approximate values for the power for the one group two-sided t-test, given values for the number of samples, the standard deviation for the population, the effect size, and the significance level.

This example will demonstrate the general approach and has the advantage that one can readily obtain an "exact" value to compare against. In situations where there is no readily available function to obtain some statistical quantity, use of simulated values (generated by an appropriate random number generator) can often provide an accurate, easily programmed method for obtaining (with sufficient accuracy) the desired statistical quantity, such as power.

One can estimate the accuracy by using an increasing sequence of the number **N** of simulations used to get the approximate value (N is also referred to as **the number of runs of the simulation**, or **the number of simulation runs**). If the simulation procedure has been done properly, the values should (with high probability) "settle down" to some value; meaning, as N increases the number of leading significant digits in the results that stay fixed should increase. In general, the Law of Large Numbers will be applicable since Monte Carlo simulations will generally have the form of taking an average of some random variable computed multiple times from values produced by the relevant random number generator (assuming the random number generator produces values that to a high degree resemble independent random samples from the specified probability distribution). However, we will also present an example where this is not entirely the case (due to the Monte Carlo procedure not coinciding exactly with what is represented by the analytical expression for the value).

This discussion is intended as an additional example for the section on Resampling in the Statistical Inference Course in the Johns Hopkins University Data Science Specialization on Coursera (see also the section on *Simulation* in the R Programming Course). Note the reader should not infer any endorsement or recommendation or approval for the material in this article from any of the sources or persons cited herein or from any other entities mentioned in this article.

Note the noncentral t-distribution is used below to calculate the exact value of the power for an example, however understanding the noncentral t-distribution is NOT necessary for the main point of this article, which is Monte Carlo simulation to calculate a statistical quantity.

Quick review of the one group t-test and power and setting up our notation

The one group (also called one sample) t-test assumes we have nx independent random samples from a population having a normal distribution Nx with mean mux and standard deviation sigmax. The null hypothesis is that the population mean is some constant mu0, and for calculating power, we assume that a specific alternative hypothesis is valid: the population mean is actually some value mua. The power will only depend on the *effect size* delta = mua - mu0, not on the individual values mu0 and mua, and, since the Student's t-distribution is symmetric about t = 0, we can without loss of generality assume delta is positive (or the "trivial case", delta = 0). Let meanx denote the sample mean, and sx be the sample standard deviation for a given set of nx independent random samples. The the T-score **TS** is defined by

TS <- (meanx - mu0) / (sx / sqrt(n)) equation (1)

If the null hypothesis were true (the population mean mux = mu0) then TS has the Student's t-distribution with (n-1) degrees of freedom. If the specific alternative hypothesis (mux = mua) is true, then TS has a noncentral t-distribution with (n-1) degrees of freedom and with non-centrality parameter ncp given by

ncp <- (mua - mu0) / (sigmax / sqrt(n)) equation (2)

The Student's t-distribution has ncp = 0. Equations (1) and (2) will be used to calculate the exact value of power in an example below. Equation (2) is given in the Details section of the R help on the t-distribution functions dt, pt, qt, rt; and use of the noncentral t-distribution for calculating power is described in terms of the notation in this article in:

[Ref 1] "Details_on_power_and_the_noncentral_t_distribution_why_it_is_needed_and_how_to_use_it_10_May_2021_v27.pdf"

https://github.com/AlanBerger/Statistical_Inference_GitHub_files/blob/master/Details_on_power_and_the_noncentral_t_distribution_why_it_is_needed_and_how_to_use_it_10_May_2021_v27.pdf

(reading these is NOT required for reading this article). Given a significance level alpha, for the two-sided t-test, the relevant quantile for the Student's t-distribution (ncp = 0) is:

t_alphad2 <- qt(alpha/2, df = n-1, lower.tail = FALSE) equation (3)

Note the dependence of t_alphad2 on the degrees of freedom df = (n-1) is to be understood. Then the two-sided p-value corresponding to a given T-score TS will be $\leq$ alpha if and only if |TS| $\geq$ t_alphad2, which for future reference is equivalent to:

TS $\geq$ t_alphad2 equation (4a)

or

TS $\leq$ -t_alphad2 equation (4b)

Note we are assuming delta > 0, so if delta is large enough to result in a "useful amount" of power, the probability that TS will be $\leq$ -t_alphad2 ("way out" in the "wrong

direction" on the left tail) will be quite small. Since this case is not really what one would usually want to consider as contributing to the power, this contribution (when (4b) occurs) is often not included in computed values of power (generally in R functions that compute power, this is called the **strict = FALSE option**; the **strict = TRUE option** directs that the probability that TS $\leq$ -t_alphad2 occurs be included in the caculated power). In the calculations below, strict will always be set to FALSE (don't include the contribution to power from values of TS "in the wrong direction", i.e., when equation (4b) holds).

The strict = FALSE power is then the probability that TS $\geq$ t_alphad2 when the alternative hypothesis mux = mua is valid. With our notation delta = (mua - mu0), the non-centrality parameter given in equation (2) becomes

ncp.value <- delta / (sigmax / sqrt(n)) equation (5)

and, under this alternative hypothesis, TS has a noncentral t-distribution with (n-1) degrees of freedom and non-centrality parameter given by ncp.value The one group two-sided t-test power (strict = FALSE) is then the probability that

TS $\geq$ t_alphad2

which is given by

power <- pt(t_alphad2, df = n-1, ncp = ncp.value, lower.tail = FALSE) equation (6)

This is in effect what power.t.test (and, for example, power_t_test from the MESS package authored by Claus Ekstrom) does to calculate the strict = FALSE power for a one group two-sided t-test.

A sample "exact" calculation

Let's set values for the relevant parameters and use equation (6), and power.t.test and power_t_test to calculate the "exact" strict = FALSE two-sided power. It is always a good idea to, whenever possible, compare results from severals ways of calculating something to check things were programmed correctly.

```
# R version 3.6.0 (2019-04-26) -- "Planting of a Tree"

n <- 16
delta <- 3
sigmax <- 4
alpha <- 0.05
# calculate strict = FALSE power for these parameters for a one group
# two-sided t-test

t_alphad2 <- qt(alpha/2, df = n-1, lower.tail = FALSE)
ncp.value <- delta / (sigmax / sqrt(n))
```

```
powerF <- pt(t_alphad2, df = n-1, ncp = ncp.value, lower.tail = FALSE)
powerF
```

```
## [1] 0.8005556
```

```
### [1] 0.8005556
# The F suffix is being used to indicate the strict = FALSE choice

############  now check using power.t.test

power.t.test(n = n, delta = delta, sd = sigmax, sig.level = alpha,
             power = NULL, type = "one.sample",
             alternative = "two.sided", strict = FALSE)
```

```
##
##      One-sample t test power calculation
##
##               n = 16
##           delta = 3
##              sd = 4
##       sig.level = 0.05
##           power = 0.8005556
##     alternative = two.sided
```

```
# which agrees, as it should

# now let's also use the power_t_test function from the
# MESS package authored by Claus Ekstrom

library(MESS)
```

```
## Warning: package 'MESS' was built under R version 3.6.3
```

```
# ?power_t_test
power_t_test(
  n = n, delta = delta, sd = sigmax, sig.level = alpha,
  power = NULL, ratio = 1, sd.ratio = 1,
  type = "one.sample",
  alternative = "two.sided",
  df.method = "welch",  strict = FALSE)
```

```
##
##      One-sample t test power calculation
##
##               n = 16
##           delta = 3
##              sd = 4
##       sig.level = 0.05
##           power = 0.8005556
##     alternative = two.sided
```

```
# the arguments
# ratio, sd.ratio, df.method only matter for two group t-tests
# again the result agrees (as it should)
```

Now let's use a Monte Carlo simulation to compute approximations to the power for the case above

The recipe for using random sampling (a random number generator) to approximate the power is pretty much given in the sentence from above: "The strict = FALSE power is then the probability that TS $\geq$ t_alphad2 when the alternative hypothesis mux = mua is valid." An outline for the code is:

N times do:

use the rnorm function with mean = delta and sd = sigmax to generate nx simulated independent random samples for the alternative hypothesis (denote the mean of these nx samples by meanx)

use R's t.test function to obtain the two-sided p-value (since we have sampled with mean = delta above, the null hypothesis here for t.test is mu = 0)

The calculated strict = FALSE power from this simulation is the fraction of the N simulation runs for which meanx $>$ 0 AND the computed p-value is $\leq$ alpha

R code for this is:

```
monte_carlo_power_function_two_sided_one_sample_ttest <-
      function(n, delta, sd, sig.level = 0.05, num.MC.runs = 10000L) {

# 12 May 2021  Alan E. Berger

# approximate the power (the probability of getting
# a p-value at or below sig.level from a two-sided
# one group t-test given n independent random samples
# from a normal distribution whose mean is mu_a > mu_0
# and whose standard deviation is sd,
```

```r
# where the null hypothesis is the mean is mu_0

# n is the number of independent random samples

# delta is   mu_a - mu_0 and here it should be non-negative

# for calculating the power,
# can take mu_0 = 0 and then mu_a = delta

if(delta < 0.) stop("delta should be non-negative")

# sig.level is the significance level for the two-sided
# t-test (often called alpha, often set to 0.05)

#### for strict = FALSE, require in addition to p-value <= sig.level,
#### that the sample mean be > 0

# num.MC.runs  is the number of Monte Carlo
# runs (simulations)

# before using a random number generator,
# should set the seed for the random number generation
# so can reproduce the calculation exactly:
# do
# set.seed(an.arbitrary.integer.value)
# before the computations using a random number generator(s)

# A note on the choice of the seed: R will take negative integers and 0 for
# the seed, but in the literature for random number generators,
# the seed has often been a positive integer
# less than 2^31 - 1 = 2,147,483,647
# (the maximum integer representable on older computers
# without invoking additional software).

# do the following  num.MC.runs  times:
# generate n independent random samples from a
# normal distribution with mean delta and
# standard deviation sd

# do the two-sided one-sample t-test on these n samples
# using R's t.test function
# comparing their mean with mu = 0, call the resulting p-value pv

# the approximate value of the strict = FALSE power from this calculation
```

```
# is the number of times
# the sample mean is > 0 AND pv is <= alpha
# divided by  num.MC.runs

# this is a standard type of statistical calculation,
# applicable in getting good approximations in
# many situations

# note one way to check on the accuracy of this calculation is to
# do several runs with the same value of N =  num.MC.runs

# the major way to check the calculation and
# "converge on" the exact value is to take successively larger
# values of  num.MC.runs  and compare the results

mu_0 <- 0.  # the null hypothesis here
mu_a <- delta  # the alternative hypothesis here

n.times.pv.le.alpha <- 0L
# the counter for the number of times the sample mean is > 0
# and the p-value is <= sig.level

num.MC.runs.int <- as.integer(num.MC.runs)
for(k in 1:num.MC.runs.int) {
    rsamples <- rnorm(n, mean = mu_a, sd = sd)
#   the n random samples for each run

    if(mean(rsamples) < 0.) next
#   since we are doing strict = FALSE, skip cases with sample mean < 0

    pv <- t.test(rsamples, alternative = "two.sided", mu = mu_0)$p.value
    if(pv <= sig.level) n.times.pv.le.alpha <- n.times.pv.le.alpha + 1L
}

# it would be more efficient to compare the T-score with the critical
# value of the T-score for the given sig.level and n,
# but using the t.test function conveys what the power is

pw <- n.times.pv.le.alpha / num.MC.runs.int
return(pw)

# to run this function:
# monte_carlo_power_function_two_sided_one_sample_ttest(n, delta, sd,
#   sig.level, num.MC.runs)
```

```
}
```

Now let's run monte_carlo_power_function_two_sided_one_sample_ttest for a succession of values of N (the number of simulation runs) for the parameters above, and also do several runs for each N to get an idea of the scatter of the values at a given N. First set the random number seed so could exactly reproduce the calculation.

```
# R version 3.6.0 (2019-04-26) -- "Planting of a Tree"

## "compile" the above function
## source("monte_carlo_power_function_two_sided_one_sample_ttest.R")
## or copy the above function into R

set.seed(112358L)

n <- 16
delta <- 3
sigmax <- 4
alpha <- 0.05

# set up a container for the results
mpower <- matrix(0, nrow = 15, ncol = 3)
colnames(mpower) <- c("two-sided power", "N", "difference")

# the power is the strict = FALSE power
# N is the number of simulations used to calculate the power
# difference is the calculated power - 0.8005556 (the "exact" value)
exactv <- 0.8005556

krow <- 0L # row index in mpower

# do a bunch of power calculations and place results in mpower

N <- 1000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)


N <- 1000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
```

```
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)

N <- 1000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)



N <- 10000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)

N <- 10000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)

N <- 10000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)



N <- 100000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)

N <- 100000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
```

```
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)

N <- 100000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)



N <- 1000000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)


N <- 1000000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)


N <- 1000000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)


N <- 10000000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)
```

```
N <- 10000000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)


N <- 10000000L
powerF <- monte_carlo_power_function_two_sided_one_sample_ttest(n, delta,
              sd = sigmax, sig.level = alpha, num.MC.runs = N)
diff.calc.exact <- powerF - exactv
krow <- krow + 1L
mpower[krow, ] <- c(powerF, N, diff.calc.exact)

# make a data frame of the results

resultsdf <- data.frame(mpower, check.names = FALSE)
resultsdf
   two-sided power     N difference
1        0.7960000 1e+03 -0.0045556
2        0.8030000 1e+03  0.0024444
3        0.8150000 1e+03  0.0144444
4        0.8028000 1e+04  0.0022444
5        0.8087000 1e+04  0.0081444
6        0.8011000 1e+04  0.0005444
7        0.7987300 1e+05 -0.0018256
8        0.8033500 1e+05  0.0027944
9        0.7984400 1e+05 -0.0021156
10       0.8004710 1e+06 -0.0000846
11       0.8014850 1e+06  0.0009294
12       0.8008170 1e+06  0.0002614
13       0.8006140 1e+07  0.0000584
14       0.8006179 1e+07  0.0000623
15       0.8004959 1e+07 -0.0000597
```

We did this calculation with some rather extreme values of N to display the apparent convergence to the exact value 0.8005556 for these parameters. For usual purposes (not involving exploring convergence) one would often only need to have the first 2 or 3 significant digits of the power value be estimated correctly or within perhaps 1 or 2 in the least significant digit being reported, which seems quite attainable for this example. Even with "only" 10,000 simulations, the three calculated power values are within 0.01 of the "exact" value (within 1 in the second significant digit). The calculation with N = 10,000 is quite fast.

A cautionary example - the Welch (unequal variance) t-test for two groups

If one "looks at the fine print", in general the Welch form of the T-score (with the Welch formula for the degrees of freedom) for a two group t-test when the null hypothesis is valid **only approximately has a t-distribution**. That this is approximate is noted for example in

[Ref 2] Brian Caffo, **Statistical inference for data science**, Leanpub, last updated on 2016-05-23, https://leanpub.com/LittleInferenceBook, page 83

[Ref 3] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers and Keying Ye, **Probability & Statistics for Engineers and Scientists**, Eighth Edition, Pearson Prentice Hall, Upper Saddle River NJ, 2007, page 347

Hence it might not be too surprising if power values for the Welch t-test from an "exact calculation" using a noncentral t-distribution might not coincide with limiting values from Monte Carlo simulations as N gets large. We will see an example of this below (but the results in the example are "not too bad"). The "take home message" is that one has to use due care in using simulation methods and be aware of potential issues.

Side note: in the special case that the sample sizes for both groups are the same AND the sample standard deviations from the two groups are equal; then the T-score from the classical Student's two group t-test and from the Welch t-test are equal, and also the degrees of freedom from the Welch formula equals that from the Student's t-test (the sum of the sample sizes for the two groups - 2). Hence if the standard deviations in the two populations are the same and the sample sizes are equal, then "exact values" for power from the Welch t-test and from the Student's t-test will be the same (given values for the effect size and significance level). In this case one can use power.t.test to calculate the power. For a two group t-test with unequal sample sizes and/or unequal population standard deviations, one can use power_t_test from the MESS package to calculate the power for the Welch t-test.

The next paragraph is from my article [Ref 1] noted above. For a two group t-test, one has the choice of using the equal variance form (the classical Student's t-test), or the form that explicitly deals with unequal variances in the two populations the samples are from (called the Welch t-test, also called the Smith-Satterthwaite t-test). Several references (I'm not saying this is a complete list of studies on this) suggest always using the unequal variance form (Welch) of the t-test (unless one has definitive information that the variances for the 2 populations are the same), because, in general, when the population variances are in fact equal, the Welch form of the t-test gives, on average, results pretty close to the equal variance form, and the Welch form is the proper choice when the variances for the 2 groups (the 2 populations that the samples are from) are not equal. (Also, it's not a coincidence that the default for R's t.test function is to use the unequal variance form.) The references also indicate using Welch is in general better than doing a test for equal variances and using that result to decide whether to use the equal variance (Student's t-test) form or the unequal variance form (Welch).

References for this: [Reference 2 above, page 83], and

[Ref 4] J.S. Milton and J.C. Arnold, Introduction to Probability and Statistics, 3rd Edition. McGraw-Hill, Boston, 1995, see page 353

[Ref 5] M. Delacre, D. Lakens and C. Leys, Why Psychologists Should by Default Use Welch's t-test Instead of Student's t-test, International Review of Social Psychology, (30)1, 92-101

[Ref 6] A. F. Hayes and L. Cai, Further evaluating the conditional decision rule for comparing two independent means, British Journal of Mathematical and Statistical Psychology (2007), 60, 217–244

Next we will present an example for the Welch t-test with unequal standard deviations and sample sizes. A function that explicitly uses the noncentral t-distribution to calculate power (strict = FALSE) for the two-sided Welch t-test is given in [Ref 1, page 23]. Here we will use the power_t_test function from the MESS package to calculate the "exact" power, and then compare that result with those obtained by corresponding Monte Carlo simulation.

Compare results from Monte Carlo simulation with the "exact" power for a Welch t-test example

For the Welch t-test one has nx independent random samples from a normal distribution Nx which has mean mux and standard deviation sigmax, and ny independent random samples from a normal distribution Ny which has mean muy and standard deviation sigmay, and one does not assume sigmax = sigmay. Introducing some more notation: for calculating the power for the Welch t-test, we can take the null hypothesis to be that the difference of the means (mux - muy) is a constant d0 which we can take to be 0; and the specific alternative hypothesis is that the difference of the means is da, a given non-negative constant equal to a non-negative effect size denoted by delta.

```
# R version 3.6.0 (2019-04-26) -- "Planting of a Tree"

# for this power calculation for a Welch t-test example,
# the null hypothesis is that the difference of the two population
# means mux - muy is 0
# and the alternative hypothesis is that mux - muy equals a given
# constant delta >= 0

# parameters for our example
delta <- 2.6
alpha <- 0.05
sigmax <- 4
sigmay <- 2
nx = 40
ny = 20
# nx = 40, ny = 20 gives the maximum power (strict = FALSE) for the two-sided
# Welch t-test for these values of delta, sigmax, sigmay, alpha,
```

```
# subject to the constraint that the integers nx and ny sum to 60
# (see [Ref 1])

# Calculate the "exact" power (strict = FALSE) for these parameters using power_t_test
library(MESS)
## Warning message:
## package 'MESS' was built under R version 3.6.3

packageVersion("MESS")
## [1] '0.5.7'

# here are comments that are an edited version of the R help for power_t_test
# on how to use it for two group t-tests, copied from [Ref 1]

####################

# for calculating power using power_t_test for a two group t-test:

# n is the smaller sample size if there are 2 groups (as will be the case here)
# delta is the effect size; (da - d0) in the notation we are using above
# sd is the standard deviation in the group with the smaller sample size
# sig.level is the significance level (alpha)
# power # set to NULL to calculate it given values for the other arguments
# ratio is the ratio of the larger sample size to the smaller sample size
#       (1 if equal)
# sd.ratio is the ratio of the standard deviations:
#          sd for the group with more samples / sd for the
#                              group with fewer samples
#          Need to be careful with sd.ratio if using
#                              power_t_test to solve for n when
#          have set ratio = 1 (so forcing equal sample sizes): check the two
#          output standard deviations to see if they are what was intended
# type is one of: c("two.sample", "one.sample", "paired")
# alternative is one of: c("two.sided", "one.sided")
# df.method = c("welch", "classical")  # method for calculating
#         the degrees of freedom for a two group (two.sample) t-test;
#         "classical" gives the equal variance (Student's) t-test degrees
#         of freedom (nx + ny - 2 in our notation),
#         "welch" gives dfW in our notation
# strict # FALSE to not include the "wrong side" contribution to the power
#          for a two-sided test.  NOTE the default in power_t_test is TRUE

####################
```

```
# for the data above, there are more samples for the "x group", whose
# standard deviation is 4,
# and the standard deviation for the group with the smaller number
# of samples (here y) has standard deviation equal 2

# calculate the "exact" power (strict = FALSE) for this example

power_t_test(n = min(c(nx, ny)), delta = delta, sd = 2,
   sig.level = alpha, power = NULL, ratio = max(c(nx, ny)) / min(c(nx, ny)),
   sd.ratio = 2,   type = "two.sample", alternative = "two.sided",
   df.method = "welch", strict = FALSE)


     Two-sample t test power calculation with unequal sample sizes and unequal variances

              n = 20, 40
          delta = 2.6
             sd = 2, 4
      sig.level = 0.05
          power = 0.9099642
    alternative = two.sided

NOTE: n is vector of number in each group

# get the exact value to compare with Monte Carlo simulation results

exact.value <- power_t_test(n = min(c(nx, ny)), delta = delta, sd = 2,
   sig.level = alpha, power = NULL, ratio = max(c(nx, ny)) / min(c(nx, ny)),
   sd.ratio = 2,   type = "two.sample", alternative = "two.sided",
   df.method = "welch", strict = FALSE)$power

exact.value
[1] 0.9099642
# this is the "exact" two-sided, strict = FALSE power for the Welch t-test
# for the parameters given above

# now also check this result
# using the function in [Ref 1] that explicitly calculates power for the Welch
# t-test using a noncentral t-distribution with the degrees of freedom
# given by the Welch formula

power_for_Welch_via_noncentral_t_distr(nx, ny, delta, sigmax, sigmay, alpha)

   power_two_sided_strict_FALSE power_one_sided wrong_side_power nx ny
1                     0.9099642       0.9527557      7.10435e-08 40 20
```

```
  delta sigmax sigmay alpha     dfW      ncp
1   2.6      4      2  0.05 57.9913 3.356586

# which indeed gives the same value two-sided strict = FALSE power value

# The "wrong_side_power" is the "left tail" contribution to the
# power that would be included if strict = TRUE; dfW is the Welch degrees of
# freedom for these values of nx, ny, sigmax, sigmay; note for this example
# dfW is close to the Student's t-test value (nx + ny - 2) for the degrees
# of freedom;
# and ncp is the appropriate value of the non-centrality parameter
# for calculating the power with these parameters
```

Review of the strict = FALSE option for power, for the Welch t-test

The T-score for the Welch t-test has the following form. For any one of the N simulation runs outlined above, let meanx be the mean of the nx simulated samples from Nx, and let meany be the mean of the ny simulated samples from Ny, and let sx and sy be the corresponding sample standard deviations, and vx = sx^2 and vy = sy^2 be the sample variances. Then the Welch T-score is

TS <- (meanx - meany) / sqrt(vx/nx + vy/ny)

The Welch form of the t-test uses the following formula to estimate the degrees of freedom, denoted here by dfW, (here nxm1 = nx - 1 and nym1 = ny - 1):

dfW <- (vx/nx + vy/ny)^2 / ((vx/nx)^2 / nxm1 + (vy/ny)^2 / nym1)

Given alpha and dfW, for a two-sided test, the critical value for the Welch T-score is the quantile for the t-distribution given by

t_alphad2 <- qt(alpha/2, df = dfW, lower.tail = FALSE)

Then the p-value from the Welch t-test will be $\leq$ alpha if and only if either

TS $\geq$ t_alphad2

or

TS $\leq$ -t_alphad2

We have taken delta to be positve, so for the purpose of computing "meaningful" power, it would make sense to require that the T-score not only satisfy |TS| $\geq$ t_alphad2 but that it be positive (i.e., that it be in the "correct" direction). For our calculations of power below we will require this (the **strict = FALSE** option). If delta is positive and "large enough" to have a meaningful amount of power, then the probability that the T-score will be $\leq$ -t_alphad2 will be quite small, and so excluding it will not have much effect.

Now do the corresponding Monte Carlo calculations

The outline of the code is:

N times do:

use the rnorm function with mean = delta and sd = sigmax to generate nx simulated random samples from Nx, denote their mean by meanx

use the rnorm function with mean = 0 and sd = sigmay to generate ny simulated random samples from Ny, denote their mean by meany

use R's t.test function to obtain the two-sided p-value from the Welch t-test

The calculated strict = FALSE power from this simulation is the fraction of the N simulation runs for which meanx > meany AND the computed p-value is $\leq$ alpha.

R code to do the Monte Carlo power calculation for the two-sided Welch t-test (strict = FALSE)

```
monte_carlo_powerF_function_two_sided_Welch_ttest <-
      function(n1, n2, delta, sd1, sd2, sig.level = 0.05,
      num.MC.runs = 10000L) {

# 14 May 2021  Alan E. Berger

# approximate the strict = FALSE power (the probability of getting
# a p-value at or below sig.level from a two-sided
# two group Welch (unequal variance) t-test
# having n1, n2 independent random samples
# from groups 1 and 2, respectively, AND
# having the difference of the means have the correct sign)

# the two populations are assumed to have normal
# distributions whose means are mu1 and  mu2
# and the null hypothesis is mu1 = mu2 (the two
# group means are the same)

# sd1 and sd2 are the standard deviations of the two
# normal distributions whose samples are being compared

# delta is  mu1 - mu2, the null hypothesis is mu1 - mu2 = 0
# and the alternative hypothesis is delta is a given constant >= 0

# for calculating the power,
# can take mu1 = delta and then mu2 = 0
```

```r
# sig.level is the significance level for the
# two-sided Welch t-test (often called alpha, often set to 0.05)

# num.MC.runs  is the number of Monte-Carlo simulation runs

# when doing a computation that uses
# a random number generator,
# should set the seed for the random number generation
# so can reproduce the calculation exactly:
# set.seed(an.arbitrary.integer.value)
# do this before the start of the computation

# do the following  num.MC.runs  times:

# generate n1 independent random samples from a
# normal distribution with mean mu1 = delta and
# standard deviation sd1, call the sample mean mean1

# generate n2 independent random samples from a
# normal distribution with mean mu2 = 0 and
# standard deviation sd2, call the sample mean mean2

# do the two-sided Welch t-test on these samples
# call the resulting p-value pv

# the approximate value of the strict = FALSE power from this calculation
# is the number of times (mean1 - mean2) > 0 AND pv is <= alpha
# divided by  num.MC.runs

# this is a standard type of statistical calculation,
# applicable in getting good approximations in
# many situations

# note one way to check on the accuracy of this calculation is to
# do several runs with a given value of num.MC.runs

# the major way to check the calculation and
# "converge on" a value is to take successively larger
# values of  num.MC.runs  and compare the results

if(delta < 0.) stop("delta should be non-negative")
mu1 <- delta
mu2 <- 0.
```

```
num.times.pv.below.alpha <- 0L

num.MC.runs.int <- as.integer(num.MC.runs)
for(k in 1:num.MC.runs.int) {

    rsamples1 <- rnorm(n1, mean = mu1, sd = sd1)
    mean1 <- mean(rsamples1)

    rsamples2 <- rnorm(n2, mean = mu2, sd = sd2)
    mean2 <- mean(rsamples2)

    if(mean1 < mean2) next    # skip to next run if (mean1 - mean2) has wrong sign
#                               this enforces the strict = FALSE option

    pv <- t.test(rsamples1, rsamples2, alternative = "two.sided", mu = 0)$p.value
#         the default when there are 2 samples in t.test is the Welch t-test
    if(pv <= sig.level) num.times.pv.below.alpha <- num.times.pv.below.alpha + 1L
}

pw <- num.times.pv.below.alpha / num.MC.runs.int
return(pw)
}

## to run this:
## on my computer:

## source("monte_carlo_power_function_two_sided_Welch_ttest.R")

## monte_carlo_powerF_function_two_sided_Welch_ttest(n1, n2, delta, sd1, sd2,
##                                   sig.level, num.MC.runs)
```

Now use the monte_carlo_powerF_function_two_sided_Welch_ttest function on the example parameters above

```
# R version 3.6.0 (2019-04-26) -- "Planting of a Tree"

# parameters for our example
delta <- 2.6
alpha <- 0.05  # sig.level
sigmax <- 4
sigmay <- 2
nx = 40
ny = 20
```

```
exact.value <- 0.9099642  # from calculation above

set.seed(314159L)

# set up a container for the results
mpower <- matrix(0, nrow = 15, ncol = 3)
colnames(mpower) <- c("two-sided Welch strict FALSE power", "N", "difference")

krow <- 0L # row index in mpower

# do a bunch of power calculations and place results in mpower


num.MC.runs <-  1000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  10000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
```

```
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  10000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  10000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  100000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  100000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  100000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)
```

```
num.MC.runs <-  1000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  10000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  10000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)
```

```
num.MC.runs <-  10000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)


# make a data frame of the results

resultsdf <- data.frame(mpower, check.names = FALSE)
resultsdf

   two-sided Welch strict FALSE power     N difference
1                           0.9130000 1e+03  0.0030358
2                           0.9040000 1e+03 -0.0059642
3                           0.9250000 1e+03  0.0150358
4                           0.9164000 1e+04  0.0064358
5                           0.9119000 1e+04  0.0019358
6                           0.9076000 1e+04 -0.0023642
7                           0.9099600 1e+05 -0.0000042
8                           0.9108800 1e+05  0.0009158
9                           0.9096100 1e+05 -0.0003542
10                          0.9098710 1e+06 -0.0000932
11                          0.9097450 1e+06 -0.0002192
12                          0.9095350 1e+06 -0.0004292
13                          0.9096638 1e+07 -0.0003004
14                          0.9097734 1e+07 -0.0001908
15                          0.9095937 1e+07 -0.0003705


### Notice how the errors level off with no clear systematic decrease after the
### number of simulation runs goes above 100,000
### The accuracy is still, for most practical purposes, quite sufficient

### This example had relatively large values of nx and ny; let's look at an example
### with relatively small sample sizes: nx = 10 and ny = 5 where there is
### more of a difference between the normal distribution
### and the t-distribution with these numbers of sample sizes


############### example with smaller sample sizes, rest of parameters as before
```

```
nx <- 10
ny <- 5

# other parameters as before
delta <- 2.6
alpha <- 0.05
sigmax <- 4
sigmay <- 2


# calculate the "exact" power (strict = FALSE) for this example
library(MESS)
Warning message:
package ‘MESS’ was built under R version 3.6.3

> packageVersion("MESS")
[1] ‘0.5.7’


power_t_test(n = min(c(nx, ny)), delta = delta, sd = 2,
   sig.level = alpha, power = NULL, ratio = max(c(nx, ny)) / min(c(nx, ny)),
   sd.ratio = 2,   type = "two.sample", alternative = "two.sided",
   df.method = "welch", strict = FALSE)

     Two-sample t test power calculation with unequal sample sizes and unequal variances

              n = 5, 10
          delta = 2.6
             sd = 2, 4
      sig.level = 0.05
          power = 0.3426068
    alternative = two.sided

NOTE: n is vector of number in each group

# The calculation in [Ref 1, page 34] directly using the noncentral
# t-distribution also gives the same value for the two-sided power (strict = FALSE)

power_for_Welch_via_noncentral_t_distr(nx, ny, delta, sigmax, sigmay, alpha)

#   power_two_sided_strict_FALSE power_one_sided wrong_side_power nx ny
# 1                    0.3426068       0.4781344     0.0002123278 10  5
#   delta sigmax sigmay alpha   dfW      ncp
# 1   2.6      4      2  0.05 12.96 1.678293
```

```
############################## now do Monte Carlo runs for these parameters

exact.value <- 0.3426068

set.seed(314159L)

# set up a container for the results
mpower <- matrix(0, nrow = 15, ncol = 3)
colnames(mpower) <- c("two-sided Welch strict FALSE power", "N", "difference")

krow <- 0L # row index in mpower

# do a bunch of power calculations and place results in mpower


num.MC.runs <-  1000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  10000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
```

```
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  10000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  10000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  100000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  100000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  100000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
```

```
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  1000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  1000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)



num.MC.runs <-  10000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)

num.MC.runs <-  10000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)
```

```
num.MC.runs <-  10000000
powerF <- monte_carlo_powerF_function_two_sided_Welch_ttest(nx, ny, delta,
    sigmax, sigmay, alpha, num.MC.runs)
powerF  # print the strict = FALSE Welch two-sided power result
diff.calc.exact <- powerF - exact.value
krow <- krow + 1L
mpower[krow, ] <- c(powerF, num.MC.runs, diff.calc.exact)


# make a data frame of the results

resultsdf <- data.frame(mpower, check.names = FALSE)
resultsdf

# N is the number of Monte Carlo simulation runs used to get the result

   two-sided Welch strict FALSE power     N difference
1                           0.3200000 1e+03 -0.0226068
2                           0.3490000 1e+03  0.0063932
3                           0.3300000 1e+03 -0.0126068
4                           0.3268000 1e+04 -0.0158068
5                           0.3426000 1e+04 -0.0000068
6                           0.3353000 1e+04 -0.0073068
7                           0.3333000 1e+05 -0.0093068
8                           0.3324100 1e+05 -0.0101968
9                           0.3344900 1e+05 -0.0081168
10                          0.3327880 1e+06 -0.0098188
11                          0.3330360 1e+06 -0.0095708
12                          0.3320190 1e+06 -0.0105878
13                          0.3328134 1e+07 -0.0097934
14                          0.3325870 1e+07 -0.0100198
15                          0.3326852 1e+07 -0.0099216


### Now the failure to converge to the result from the "exact" calculation
### is quite noticeable, again not surprising given the smaller sample sizes
### and the Welch t-test not being exactly governed by a t-distribution
```

Comments on the last example above

Even though the results are not "way off" (depending on one's point of view), this is a good example for showing one needs to be aware of the theoretical assumptions for doing a Monte Carlo calculation. Also, although not the issue here, one needs to have the simulation use the

appropriate random number generator(s) in an appropriate way (that models the statistical value being approximated). Indeed, I would suggest that the results from the Monte Carlo simulation of the Welch t-test power are more accurate than the "exact" calculation that uses a fixed value for the degrees of freedom (calculated from the Welch formula using nx, ny, sigmax and sigmay) since the Monte Carlo calculation is modeling how individual Welch t-tests are done. Note the Monte Carlo values do appear to converge (to 0.3327 +/- 0.0001 when reporting 4 significant digits) as the number of simulation runs gets very large (just not to the result from the "exact calculation").

Hope this discussion of Monte Carlo simulation is helpful.