Based on "SimulatingPower.pdf"
by Jeffrey Hughes
Simulating Power with the paramtest Package

```
> library(paramtest)
Warning message:
package 'paramtest' was built under R version 3.6.3
> library(pwr)
> library(ggplot2)
Warning message:
package 'ggplot2' was built under R version 3.6.3
> library(knitr)
Warning message:
package 'knitr' was built under R version 3.6.3
> library(nlme)
> library(dplyr)

Attaching package: 'dplyr'

The following object is masked from 'package:nlme':

    collapse

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

Warning message:
package 'dplyr' was built under R version 3.6.3
>
> windows(7,7)
> #save graph(s) in pdf
> windows(7,7)
>  pdf(file="C:/Users/jmard/OneDrive/Desktop/Computing and Graphics in Applied
Statistics2020/Output/PowerSampleSizeExamplesSimulation_Figure.pdf")
>
> # varying N and Cohen's d
> # create user-defined function to generate and analyze data
> set.seed(3214)
>
```

In real-world data, statistical assumptions when using canned sample size programs may not hold, therefore estimates of power when these assumptions are assumed will likely be inflated.

Power by simulation is another way to compute power estimates and offers significant flexibility to the user to explore the impact of various statistical assumption violations may have on power.

two-sample t-test simulation

Simulations are commonly performed to further support statements based on statistical theory.

```
> t_func <- function(simNum, N, d) {
+ x1 <- rnorm(N, 0, 1)
+ x2 <- rnorm(N, d, 1)
+ t <- t.test(x1, x2, var.equal=TRUE)
+ # run t-test on generated data
+ stat <- t$statistic
+ p <- t$p.value
+ return(c(t=stat, p=p, sig=(p < .05)))
+ # return a named vector with the results we want to keep
+ }
>
```

Generate two independent samples of size N from a normal distribution with mean =0 and mean=d and std deviation =1

```
> power_ttest_vary2 <- grid_search(t_func, params=list(N=c(25, 50, 100), d=c(.2, .5)),
+ n.iter=5000, output='data.frame')
Running 30,000 tests...
> power <- results(power_ttest_vary2) %>%
+ group_by(N.test, d.test) %>%
+ summarise(power=mean(sig))
> print(power)
# A tibble: 6 x 3
# Groups:   N.test [3]
  N.test d.test power

1     25    0.2 0.108
2     25    0.5 0.412
3     50    0.2 0.167
4     50    0.5 0.686
5    100    0.2 0.289
6    100    0.5 0.937
>
> ggplot(power, aes(x=N.test, y=power, group=factor(d.test), colour=factor(d.test))) +
+ geom_point() +
+ geom_line() +
+ ylim(c(0, 1)) +
+ labs(x='Sample Size', y='Power', colour="Cohen's d") +
+ theme_minimal()
>
>
> #using simulation, we can determine the power for more complex models,
> #including interactions and simple effects.
>
```

grid_search function: run a function iteratively using a grid search approach for parameter values, with options for parallel processing.

Tibbles are a modern take on data frames. A tibble() is a nice way to create data frames. It encapsulates best practices for data frames.

Simulation in a regression model with 2 predictors and interaction

```
> set.seed(2134)
> lm_test_interaction <- function(simNum, N, b1, b2, b3, b0=0, x1m=0, x1sd=1,
+ x2m=0, x2sd=1) {
+
+ x1 <- rnorm(N, x1m, x1sd)
+ x2 <- rnorm(N, x2m, x2sd)
+
+ yvar <- sqrt(1 - b1^2 - b2^2 - b3^2) # residual variance
+ y <- rnorm(N, b0 + b1*x1 + b2*x2 + b3*x1*x2, yvar)
+
+ model <- lm(y ~ x1 * x2)
+
+ # pull output from model (two main effects and interaction)
+
+ est_x1 <- coef(summary(model))['x1', 'Estimate']
+ p_x1 <- coef(summary(model))['x1', 'Pr(>|t|)']
+ sig_x1 <- p_x1 < .05
+ est_x2 <- coef(summary(model))['x2', 'Estimate']
+ p_x2 <- coef(summary(model))['x2', 'Pr(>|t|)']
+ sig_x2 <- p_x2 < .05
+ est_int <- coef(summary(model))['x1:x2', 'Estimate']
+ p_int <- coef(summary(model))['x1:x2', 'Pr(>|t|)']
+ sig_int <- p_int < .05
+ return(c(est_x1=est_x1, p_x1=p_x1, sig_x1=sig_x1, est_x2=est_x2, p_x2=p_x2,
+ sig_x2=sig_x2, est_int=est_int, p_int=p_int, sig_int=sig_int))
+ }
>
> #varying N at 200 and 300; setting coefficient of x1 = .15, coefficient of
> # x2 = 0, and coefficient of interaction = .3
> >
> power_lm_int <- grid_search(lm_test_interaction, params=list(N=c(200, 300)),
+ n.iter=5000, output='data.frame', b1=.15, b2=0, b3=.3, parallel='snow', ncpus=4)
Running 10,000 tests...
> results(power_lm_int) %>%
+ group_by(N.test) %>%
+ summarise(
+ power_x1=mean(sig_x1),
+ power_x2=mean(sig_x2),
+ power_int=mean(sig_int))
# A tibble: 2 x 4
```

from regression theory

Shorter way of writing the model y ~ bo + b1x1 + b2x2 + b3x1x2

```
  N.test power_x1 power_x2 power_int
1    200    0.593   0.0564      0.989
2    300    0.777   0.0562      1
> dev.off()
windows
      2
>
```

Power to reject H0:B1=0 when in fact B1 = .15 when N=200 is 0.593

H0: b2=0 b2 was set at 0 so the power should be close to the Type I error rate used in the simulation.