Power Analysis

Terrence Jorgensen, Ben Kite, Paul Johnson[1]

[1] Center for Research Methods and Data Analysis

2018

Outline

Recall Hypothesis Testing?

- This is mostly about using R (R Core Team, 2017) for power analysis

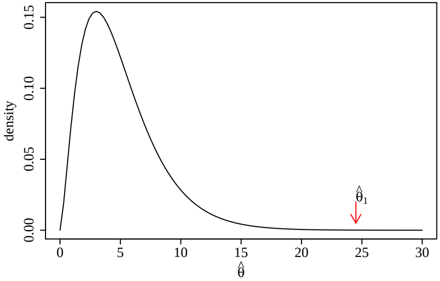
Outline

Recall Hypothesis Testing?

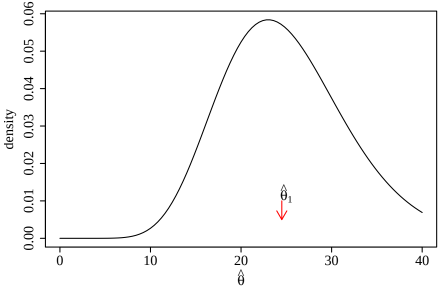
- Null Hypothesis Significance Testing (NHST) is a common application in social science
- Frame research hypothesis as
  - a “null” hypothesis (H_0) that is assumed true, and is to be rejected, in favor of
  - the “alternative” hypothesis (H_1)
- Design study (collect data) to test H_0
- Logic: Reject H_0 if data results are unexpected if H_0 were true
- If you fail to reject H_0, that means H_0 is a plausible explanation for the observed data

Estimate of θ is way out there. Or not

Exciting! New! Different



Just what we expected all along



Examples of H_0

- Theory
  $ElectricityDemand = \beta_0 + \beta_1 Wealth + \varepsilon$
- Hypothesis testing
  - Null Hypothesis: Effect of wealth on electricity demand is 7

$$H_0 : \beta_1 = 7$$

  - The estimate from data is $\hat{\beta}_1 = 10$
- Question: Is 10 far enough from 7 for H_0 to be rejected?

''nil'' versus ''null'' hypothesis tests

- Previous example had meaningful null based on experience
- Often we assert simply the null value is 0, as if to say "variable X does not matter"
- That "nil" hypothesis test is useful when comparing groups
- Example: we build a model in which the expected value of depression in humans is μ.
  - Another person says our model is incorrect because it ignored gender differences. They suggest instead there should be two parameters, μ_{men} and μ_{women}.
  - To decide, we create a new parameter, $\mu_{diff} = \mu_{men} - \mu_{women}$ and try to estimate it.
  - Set the null, $H_0 : \mu_{diff} = 0$
  - Suppose the estimate is $\hat{\mu}_{diff} = -5$
  - Is the observed difference big enough to convince us that H_0 is untenable?

Type I and Type II error

- Type I error: the null is true, but our procedure rejects it
- Type II error: the null is false, but our procedure does not reject it
- Many statistical procedures are based on the idea that we accept a certain level of risk–α– in making a Type I error, we will incorrectly reject the null hypothesis
- The acceptable risk, α, depends on field of research and context.
  - Social science, often 0.05
  - Medical science, sometimes 0.01 or 0.001

What is Statistical Power?

- The chance of making a Type II error is often called β. Unlike α, it is not a parameter we set, so much as problem we incur.

  Power The probability of rejecting the null, if it is FALSE
  - power is $1 - \beta$ AKA (1 – chance of Type II error)

- Power concept only makes sense in the context of NHST
- Should power analysis conduct before data collection (avoid post hoc)
- Power is affected by 4 factors
  - Rejection criterion (α level)
  - Sample size (N)
  - Variability anticipated from one sample to another
  - Effect size (the degree to which H_0 is false)

Rejection Rates

Power and Type I concepts are based on the idea of a sampling distribution "under the null hypothesis".

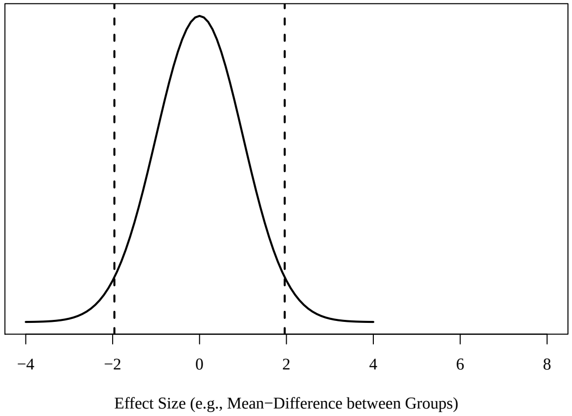
- Type I error rates refer to the probability of rejecting a null hypothesis.
- When the null is FALSE, you'd like to reject it as often as possible (have high power).
- The following R code will show how you can visualize that.
  adjust N, SD, alpha, and ES (one at a time) to see how they affect power

Rejection Rates

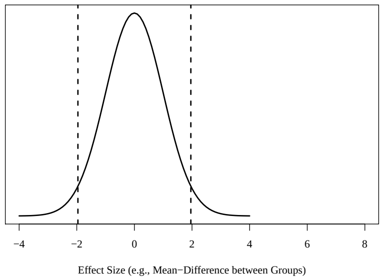
- Plot the sampling distribution under the null hypothesis
- This is based on assumption we've scaled the estimator so that its true standard deviation is 1.0 and true center point is 0

```
x.null <- seq(-4, 4, .1)
dx.null <- dnorm(x.null, m = 0, s = 1)
plot(x.null, dx.null, type = "l", lwd = 2, xlim =
    c(-4, 8), yaxt = "n",
 xlab = "Effect Size (e.g., Mean-Difference
     between Groups)", ylab = "")
## If abs(z) > 1.96, reject the null at alpha =
    .05
abline(v = qnorm(c(.025, .975)), lwd = 2, lty =
    "dashed")
## Type I errors occur for observations drawn
    outside the dashed lines
```

Rejection Rates ...



Discussion

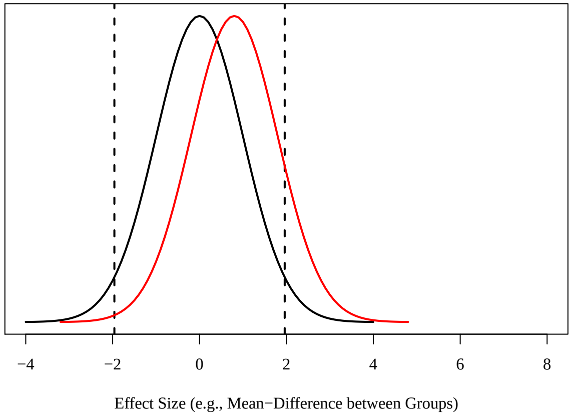


- Type I errors occur for estimates that are outside the dashed lines
- Power is not a meaningful concept when discussing the sampling distribution under the null

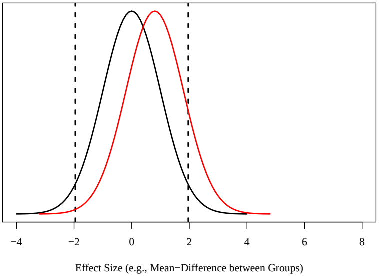
Imagine an Alternate Reality

```
x.null <- seq(-4, 4, .1)
dx.null <- dnorm(x.null, m = 0, s = 1)
plot(x.null, dx.null, type = "l", lwd = 2, xlim =
    c(-4, 8), yaxt = "n",
 xlab = "Effect Size (e.g., Mean-Difference
     between Groups)", ylab = "")
## If abs(z) > 1.96, reject the null at alpha =
    .05
abline(v = qnorm(c(.025, .975)), lwd = 2, lty =
    "dashed")
## Type I errors occur for observations drawn
    outside the dashed lines
x.8 <- x.null + 0.80
dx.8 <- dnorm(x.8, m = 0.80)
lines(x.8, dx.8, lwd = 2, col = "red")
```

Imagine an Alternate Reality ...



Discussion



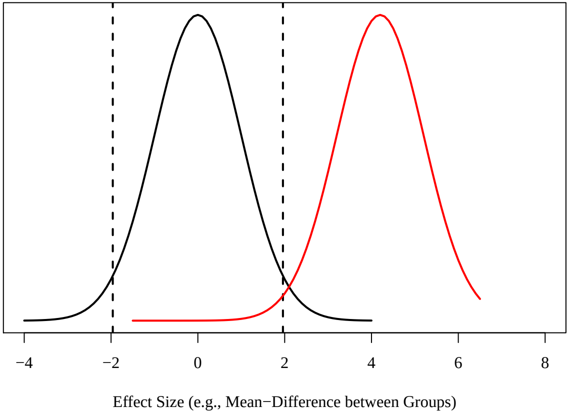
- The red line is the “true” sampling distribution, as it occurs under a hypothesized alternative
- The red sampling distribution overlaps with the black (null) distribution to a considerable extent. Under the red, the null is rejected more often, but it is not rejected with extremely high probability.
- Most would say this is an “under-powered study”.

In a World Where . . .

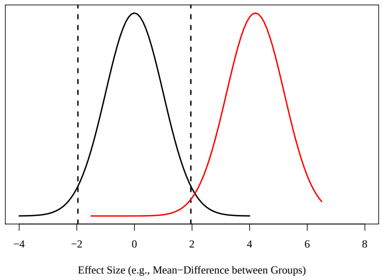
Effect sizes are much larger (Gigantic by Cohen's standards)

```
x.null <- seq(-4, 4, .1)
dx.null <- dnorm(x.null, m = 0, s = 1)
plot(x.null, dx.null, type = "l", lwd = 2, xlim =
    c(-4, 8), yaxt = "n",
 xlab = "Effect Size (e.g., Mean-Difference
     between Groups)", ylab = "")
## If abs(z) > 1.96, reject the null at alpha =
    .05
abline(v = qnorm(c(.025, .975)), lwd = 2, lty =
    "dashed")
## Type I errors occur for observations drawn
    outside the dashed lines
x.25 <- c(x.null + 2.5)
dx.25 <- dnorm(x.8, m = 2.5, s = 1)
lines(x.25, dx.25, lwd = 2, col = "red")
```

In a World Where . . . ...



Discussion



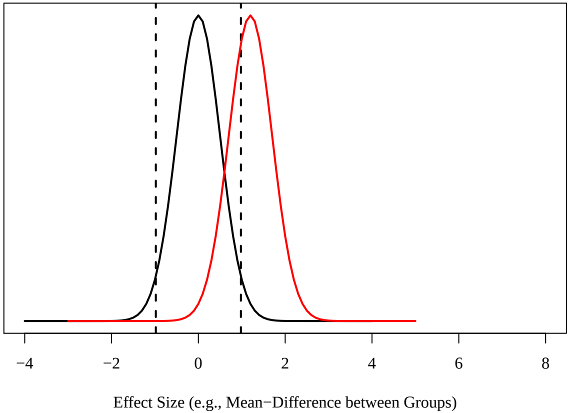
- If we assume the true effect is massive, then the power analysis will say we have great power.
- Critics will say we are proposing a ridiculously huge difference between groups

In a World Where . . .

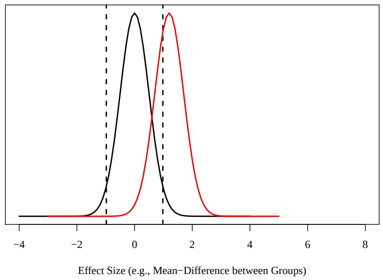
Standard error of sampling distribution is smaller

```
x.null <- seq(-4, 4, .1)
dx.null <- dnorm(x.null, m = 0, s = 0.5)
plot(x.null, dx.null, type = "l", lwd = 2, xlim =
    c(-4, 8), yaxt = "n",
 xlab = "Effect Size (e.g., Mean-Difference
     between Groups)", ylab = "")
## If abs(z) > 1.96, reject the null at alpha =
    .05
abline(v = qnorm(c(.025, .975), m = 0, s = 0.5),
    lwd = 2, lty = "dashed")
## Type I errors occur for observations drawn
    outside the dashed lin
x.1 <- c(x.null + 1)
dx.1 <- dnorm(x.8, m = 1, s = 0.5)
lines(x.1, dx.1, lwd = 2, col = "red")
```

In a World Where . . . ...



Discussion



- Rather than supposing that the effect size gets bigger and bigger (which is frowned upon)
- Best idea is to suppose the standard error can be “shrunken” by using larger and larger sample sizes.

Motivation for Power Analysis

- Required by funding agencies that award research proposals
  - How many cases are required to reject your H_0?
  - Funding agencies (and dissertation advisors) want to make sure we aren't wasting time and money
- Think backwards
  - Imagine a completed study, with data
  - MUST write down the actual model to be estimated
  - With "made up data" of size N, using carefully chosen population parameters, how often is a "significant" effect detected?
  - If not, how large must N be to detect the effect at least as often as a minimum threshold?

Real-Life Research Example

Researcher collects data on N = 10 people to find out whether tobacco causes cancer

- Statistical procedure says there's no relationship, so we can't reject H_0 of no relationship
- Suppose the effect of tobacco on cancer risk is actually present, but we missed it by not collecting enough data (Type II error)
- 80% is a customary threshold for "enough" power
- We should design experiments so $power \geq 0.8$
- You wish
  - error variance would be small
  - Effect must be "large"
- You may need to dial up the sample size otherwise.
  - A bigger sample almost always increases chances of finding a "significant" result (i.e., of rejecting H_0)

Effect Sizes

- "Effect Size" is a term coming from education and psychological research. It is motivated by the desire to reduce limitations of "apples and oranges" comparisons
- "Raw Effect Sizes" are the parameter estimate minus the null hypothesized value
  - Regression slopes ($\hat{\beta}$ - β_{null})
  - Mean-differences between groups ($\hat{u}_{group\,1} - \hat{\mu}_{group\,2}$)
- Attempts to "standardize" effect sizes across studies usually rely on standard errors, e.g.,
  - Divide difference by SE for a t statistic

Effect Sizes

- Effect Size = magnitude of difference between a parameter estimate and its H_0 value, eg $\hat{\mu} - \mu$
- APA and some funding agencies suggest/require "standardized" effect sizes
  - Seeking a number that is generic across contexts
  - Supposed to represent "practical" significance, but effects in units of SD or proportions are not always intuitive or useful
- Cohen (1988) pioneered the most frequently used criteria for describing effect sizes and estimating power among social scientists

How do Effect Sizes Matter in Power Analysis?

- Researchers are pressured to change the way they think about the eventual analysis
- Rather than saying "the difference between people from the North and South is 7 units"
    - they are expected to say "in standardized effect size units, the difference between people from the North and South is 0.4 units"
    - The power calculation has to be scaled into the standardized effect sizes
- Presumably, by putting expected differences into terms of standardized effect sizes, a project reviewer can look at the anticipated difference and say "that is unrealistically large".

Outline

G Power

G*Power (http://www.gpower.hhu.de/en.html)

- Cookbook works with regression, correlation, t test, ANOVA, ANCOVA, MANOVA, MANCOVA
- Some generalized linear models (Poisson or logistic regression)
- Contingency tables (χ^2, McNemar's test)
- Proportion tests
- The user's manual on the website is easy to read (pictures and easy instructions)
- But... G*Power only covers fairly simple cases.
  - "Standardized" effect sizes aren't intuitive.
  - When you need to know the power simultaneously for several tests/parameters in a single model, then Monte Carlo methods become necessary where analytical methods break down.

Multilevel Models

- PINT (`http://www.stats.ox.ac.uk/~snijders/multilevel.htm#progPINT`)
  - Uses analytical approximation, 2-level models only
  - Faster than a simulation, perhaps more analytically meaningful
- MLPowSim (`http://www.bristol.ac.uk/cmm/software/mlpowsim`)
  - Writes an R file with which one can do an MC Power simulation, after
  - User runs a program-builder program that quizzes the user about
    - design, predictors, parameters
  - Has been "beta software" for 7 years, will probably never be "done"
  - Only available for Windows

WebPower

WebPower (http://webpower.psychstat.org/wiki/)

- Correlation, regression
- Proportion/Mean differences
- Mediation
- Multilevel and Longitudinal modeling
- Structural equation modeling
- Fairly new, may have bugs

Zhang, Z., & Yuan, K.-H. (2018). *Practical Statistical Power Analysis Using Webpower and R* (Eds). Granger, IN: ISDSA Press.

Mplus Software Suite

For SEMs (and more), see CRMDA Guide 12: *Monte Carlo Simulation in Mplus* *Monte Carlo Simulation in Mplus* (other guides `http://crmda.ku.edu/guides-index`)

- Mplus is primarily SEM software (not free), but it can also be used for anything that can be framed as a
  - Linear model (t test, ANOVA, regression)
  - Generalized linear model (Poisson or logistic regression)
  - Multilevel / mixed-effects model
- Just need to know how to write model in Mplus syntax

R package "pwr"

- As usual,
  - if you don't have pwr, install with `install.packages("pwr")` .
  - Review the help page with `help(package = pwr)`

```
library(pwr)
```

Normative standards in pwr

- Cohen offered opinions about realistic norms for small, medium and large effects in various kinds of statistical models.

```
cohen.ES(test = "t", size = "large")     # Cohen's D
```

```
     Conventional effect size from Cohen (1982)

           test = t
           size = large
    effect.size = 0.8
```

```
cohen.ES(test = "t", size = "medium")
```

```
     Conventional effect size from Cohen (1982)

           test = t
           size = medium
    effect.size = 0.5
```

```
cohen.ES(test = "t", size = "small")
```

Normative standards in pwr ...

```
     Conventional effect size from Cohen (1982)

           test = t
           size = small
    effect.size = 0.2
```

```
cohen.ES(test = "r", size = "large")    # Pearson's r (correlation)
```

```
     Conventional effect size from Cohen (1982)

           test = r
           size = large
    effect.size = 0.5
```

```
cohen.ES(test = "r", size = "medium")
```

```
     Conventional effect size from Cohen (1982)

           test = r
           size = medium
    effect.size = 0.3
```

Normative standards in pwr ...

```
cohen.ES(test = "r", size = "small")
```

```
     Conventional effect size from Cohen (1982)

           test = r
           size = small
    effect.size = 0.1
```

```
cohen.ES(test = "anov", size = "small") # Cohen's f_squared
```

```
     Conventional effect size from Cohen (1982)

           test = anov
           size = small
    effect.size = 0.1
```

```
cohen.ES(test = "f2", size = "small")   # Cohen's f_squared
```

```
     Conventional effect size from Cohen (1982)

           test = f2
           size = small
    effect.size = 0.02
```

Approximate Power Guesses for Simple Stats

```
## Find power for a given sample size, effect size, and alpha level
pwr.r.test(n = 30, r = .1, sig.level = .05)
```

```
     approximate correlation power calculation (arctangh transformation)

              n = 30
              r = 0.1
      sig.level = 0.05
          power = 0.08208191
    alternative = two.sided
```

```
## A priori power analysis: Find sample size required for a given
## level of power, alpha, and effect size
pwr.r.test(power = .80, r = .1, sig.level = .05)
```

```
     approximate correlation power calculation (arctangh transformation)

              n = 781.7516
              r = 0.1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided
```

Approximate Power Guesses for Simple Stats ...

```
## Do the same with Cohen's D for a t test
pwr.t.test(n = 30, d = .2, sig.level = .05)
```

```
     Two-sample t test power calculation

              n = 30
              d = 0.2
      sig.level = 0.05
          power = 0.1186794
    alternative = two.sided

NOTE: n is number in *each* group
```

```
pwr.t.test(power = .80, d = .2, sig.level = .05)
```

Approximate Power Guesses for Simple Stats ...

```
     Two-sample t test power calculation

              n = 393.4057
              d = 0.2
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

```
## repeat without the requirement for equal group sizes
pwr.t2n.test(n1 = 20, n2 = 12, d = .2, sig.level = .05)
```

```
     t test power calculation

             n1 = 20
             n2 = 12
              d = 0.2
      sig.level = 0.05
          power = 0.08280013
    alternative = two.sided
```

```
pwr.t2n.test(power = .8, n1 = 40, d = .5, sig.level = .05)
```

Approximate Power Guesses for Simple Stats ...

```
     t test power calculation

             n1 = 40
             n2 = 153.0969
              d = 0.5
      sig.level = 0.05
          power = 0.8
    alternative = two.sided
```

Outline

Monte Carlo Power Analysis

- A Monte Carlo study where:
  - The outcome of interest is statistical power
  - The main manipulated factor is N
- Useful because analytical methods only cover simple cases
  - Power $=$ the proportion of samples in a condition for which H_0 was rejected
- Can manipulate other factors
  - Effect size, alpha, variability, missing data, etc.

Explore the Two-Group Simulation

- In the Monte Carlo lecture, we developed a series of functions that can estimate the H_0 rejection rates for a problem with normally distributed data in which two groups are observed.
- We developed an idiom to describe the group
  - Sample size: N
  - Mean: M
  - Standard Deviation: SD

tPowerSim

Here is how we might fit the various functions together more tightly

```
##' Monte Carlo simulation for 2 group t test
##' @param conds = a conditions data frame
##' @param var.equal: should the t.test use the equal variance
##' assumption or the Welch corrected calculation (if FALSE).
##' Note default TRUE is different from R base.
##' @return a matrix summarizing rejection rates
tPowerSim <- function(conds, var.equal = TRUE){
     ## Creates data by parsing N, M and SD strings
     getTdata <- function(rep, N, M, SD) {
         Nvec <- as.numeric(unlist(strsplit(N, ":")))
         Mvec <- as.numeric(unlist(strsplit(M, ":")))
         SDvec <- as.numeric(unlist(strsplit(SD, ":")))

         dat <- data.frame(first = c(rep(0, times = Nvec[1]),
                                     rep(1, times = Nvec[2])))
         dat$IQ <- rnorm(sum(Nvec), m = Mvec[(dat$first + 1)],
                         sd = SDvec[(dat$first + 1)])
         dat$IQ <- round(dat$IQ)
         attr(dat, "rep") <- rep
         attr(dat, "parms") <- c(N = N, M = M, SD = SD)
         dat
     }
     ## conducts T test, keeps only p value
```

tPowerSim ...

```
conductTtest <- function (dframe, y = "IQ", x = "first",
     var.equal){
    t.test(formula(paste(y, "~", x)),
           data = dframe, var.equal = var.equal)$p.value
}
## orchestrates the data pull, analysis, and summary
runOneSim <- function(rep, N, M, SD, var.equal){
    dframe <- getTdata(rep, N = N, M = M, SD = SD)
    reslt <- conductTtest(dframe, var.equal = var.equal)
    parms <- attr(dframe, "parms")
    dframe2 <- data.frame(rep = attr(dframe, "rep"),
                          pvalue = reslt,
                          reject.05 = if (reslt <= 0.05) 1 else 0,
                          reject.1 = if (reslt <= 0.10) 1 else 0,
                          N = parms["N"], M = parms["M"], SD =
                              parms["SD"])
    dframe2
}

# Reads the condition matrix, runs one row from it
runOneCondition <- function(i, conds, var.equal){
    x <- conds[i, ]
    result.list <- lapply(1:x$nReps, runOneSim,
                          N = x$N, M = x$M, SD = x$SD, var.equal
                              = var.equal)
    do.call("rbind", result.list)
```

tPowerSim ...

```
      }
       # Run all of the rows in the condition matrix
      listofresults <- lapply(1:NROW(conds), runOneCondition, conds,
           var.equal = var.equal)
      stackedResults <- do.call(rbind, listofresults)
      output <- aggregate(stackedResults[, c("reject.05", "reject.1")],
                          by = list(N = stackedResults$N, SD =
                               stackedResults$SD, M = stackedResults$M),
                          mean)
      names(output) <- c("N", "SD", "M", "reject.05.mean",
           "reject.1.mean")
      output
  }
```

Input is a conds matrix

```
cond.N <- c("30:30", "40:20")
cond.SD <- c("10:20", "15:15", "20:10")
cond.M <- c("100:100") # for now, mean-difference
   = 0
conds <- expand.grid(nReps = 1000, SD = cond.SD,
   N = cond.N, M = cond.M,
                      stringsAsFactors = FALSE)
head(conds)
```

```
  nReps    SD     N       M
1  1000 10:20 30:30 100:100
2  1000 15:15 30:30 100:100
3  1000 20:10 30:30 100:100
4  1000 10:20 40:20 100:100
5  1000 15:15 40:20 100:100
6  1000 20:10 40:20 100:100
```

Example Run

```
tPowerSim(conds, var.equal = TRUE)
```

```
      N    SD        M reject.05.mean reject.1.mean
1 30:30 10:20 100:100           0.060         0.111
2 40:20 10:20 100:100           0.125         0.185
3 30:30 15:15 100:100           0.040         0.090
4 40:20 15:15 100:100           0.041         0.105
5 30:30 20:10 100:100           0.052         0.103
6 40:20 20:10 100:100           0.025         0.049
```

```
tPowerSim(conds, var.equal = FALSE)
```

```
      N    SD        M reject.05.mean reject.1.mean
1 30:30 10:20 100:100           0.055         0.105
2 40:20 10:20 100:100           0.051         0.100
3 30:30 15:15 100:100           0.034         0.094
4 40:20 15:15 100:100           0.052         0.113
5 30:30 20:10 100:100           0.059         0.111
6 40:20 20:10 100:100           0.038         0.096
```

```
tPowerSim(conds, var.equal = TRUE)
```

Example Run ...

```
      N    SD       M reject.05.mean reject.1.mean
1 30:30 10:20 100:100          0.060         0.111
2 40:20 10:20 100:100          0.125         0.185
3 30:30 15:15 100:100          0.040         0.090
4 40:20 15:15 100:100          0.041         0.105
5 30:30 20:10 100:100          0.052         0.103
6 40:20 20:10 100:100          0.025         0.049
```

```
tPowerSim(conds, var.equal = FALSE)
```

```
      N    SD       M reject.05.mean reject.1.mean
1 30:30 10:20 100:100          0.055         0.105
2 40:20 10:20 100:100          0.051         0.100
3 30:30 15:15 100:100          0.034         0.094
4 40:20 15:15 100:100          0.052         0.113
5 30:30 20:10 100:100          0.059         0.111
6 40:20 20:10 100:100          0.038         0.096
```

Experiment with That

```
## Play with tPowerSim.
## I just "noodled" around a while
## You can be systematic :)
##
## Power analysis is the study of data group sizes

cond.N <- c("30:30", "40:20", "100:100")
cond.SD <- c("10:20", "15:15", "20:10")
cond.M <- c("100:105")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
             M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

```
        N    SD       M reject.05.mean reject.1.mean
1   30:30 10:20 100:105          0.234         0.335
2   40:20 10:20 100:105          0.285         0.375
3 100:100 10:20 100:105          0.618         0.716
4   30:30 15:15 100:105          0.226         0.348
5   40:20 15:15 100:105          0.238         0.350
6 100:100 15:15 100:105          0.644         0.774
7   30:30 20:10 100:105          0.213         0.326
8   40:20 20:10 100:105          0.113         0.203
9 100:100 20:10 100:105          0.621         0.725
```

Experiment with That ...

```
tPowerSim(conds, var.equal = FALSE)
```

```
        N    SD       M reject.05.mean reject.1.mean
1   30:30 10:20 100:105          0.218         0.319
2   40:20 10:20 100:105          0.178         0.268
3 100:100 10:20 100:105          0.622         0.724
4   30:30 15:15 100:105          0.220         0.343
5   40:20 15:15 100:105          0.210         0.317
6 100:100 15:15 100:105          0.657         0.751
7   30:30 20:10 100:105          0.220         0.338
8   40:20 20:10 100:105          0.247         0.363
9 100:100 20:10 100:105          0.612         0.726
```

```
cond.M <- c("100:106")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
                     M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

Experiment with That ...

```
          N     SD        M reject.05.mean reject.1.mean
1    30:30 10:20 100:106          0.336         0.447
2    40:20 10:20 100:106          0.347         0.446
3  100:100 10:20 100:106          0.753         0.840
4    30:30 15:15 100:106          0.305         0.433
5    40:20 15:15 100:106          0.298         0.412
6  100:100 15:15 100:106          0.784         0.869
7    30:30 20:10 100:106          0.297         0.429
8    40:20 20:10 100:106          0.183         0.299
9  100:100 20:10 100:106          0.746         0.833
```

```
tPowerSim(conds, var.equal = FALSE)
```

```
          N     SD        M reject.05.mean reject.1.mean
1    30:30 10:20 100:106          0.302         0.425
2    40:20 10:20 100:106          0.232         0.359
3  100:100 10:20 100:106          0.746         0.842
4    30:30 15:15 100:106          0.341         0.459
5    40:20 15:15 100:106          0.272         0.396
6  100:100 15:15 100:106          0.803         0.875
7    30:30 20:10 100:106          0.278         0.396
8    40:20 20:10 100:106          0.344         0.459
9  100:100 20:10 100:106          0.758         0.846
```

Experiment with That ...

```
cond.N <- c("100:100", "150:150", "150:100", "100:150")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
                     M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

```
         N    SD       M reject.05.mean reject.1.mean
1  100:100 10:20 100:106          0.734         0.827
2  150:150 10:20 100:106          0.899         0.944
3  150:100 10:20 100:106          0.851         0.906
4  100:150 10:20 100:106          0.821         0.905
5  100:100 15:15 100:106          0.809         0.891
6  150:150 15:15 100:106          0.922         0.955
7  150:100 15:15 100:106          0.875         0.928
8  100:150 15:15 100:106          0.874         0.922
9  100:100 20:10 100:106          0.739         0.837
10 150:150 20:10 100:106          0.908         0.956
11 150:100 20:10 100:106          0.810         0.884
12 100:150 20:10 100:106          0.847         0.905
```

```
tPowerSim(conds, var.equal = FALSE)
```

Experiment with That ...

```
         N     SD         M reject.05.mean reject.1.mean
1  100:100 10:20 100:106          0.763         0.865
2  150:150 10:20 100:106          0.902         0.936
3  150:100 10:20 100:106          0.766         0.851
4  100:150 10:20 100:106          0.883         0.932
5  100:100 15:15 100:106          0.804         0.871
6  150:150 15:15 100:106          0.935         0.968
7  150:100 15:15 100:106          0.865         0.922
8  100:150 15:15 100:106          0.864         0.907
9  100:100 20:10 100:106          0.766         0.852
10 150:150 20:10 100:106          0.885         0.939
11 150:100 20:10 100:106          0.902         0.950
12 100:150 20:10 100:106          0.795         0.874
```

```
cond.M <- c("100:110")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
                     M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

Experiment with That ...

```
         N    SD       M reject.05.mean reject.1.mean
1  100:100 10:20 100:110          0.998         1.000
2  150:150 10:20 100:110          1.000         1.000
3  150:100 10:20 100:110          0.995         0.998
4  100:150 10:20 100:110          1.000         1.000
5  100:100 15:15 100:110          0.997         0.999
6  150:150 15:15 100:110          1.000         1.000
7  150:100 15:15 100:110          0.999         0.999
8  100:150 15:15 100:110          0.999         0.999
9  100:100 20:10 100:110          0.995         0.998
10 150:150 20:10 100:110          1.000         1.000
11 150:100 20:10 100:110          0.998         1.000
12 100:150 20:10 100:110          0.996         0.999
```

```
 ## Play with tPowerSim.
 ## I just "noodled" around a while
 ## You can be systematic :)
 ##
 ## Power analysis is the study of data group sizes

 cond.N <- c("30:30", "40:20", "100:100")
 cond.SD <- c("10:20", "15:15", "20:10")
 cond.M <- c("100:105")
 conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
```

Experiment with That ...

```
                M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

```
        N    SD       M reject.05.mean reject.1.mean
1   30:30 10:20 100:105          0.234         0.335
2   40:20 10:20 100:105          0.285         0.375
3 100:100 10:20 100:105          0.618         0.716
4   30:30 15:15 100:105          0.226         0.348
5   40:20 15:15 100:105          0.238         0.350
6 100:100 15:15 100:105          0.644         0.774
7   30:30 20:10 100:105          0.213         0.326
8   40:20 20:10 100:105          0.113         0.203
9 100:100 20:10 100:105          0.621         0.725
```

```
tPowerSim(conds, var.equal = FALSE)
```

Experiment with That ...

```
        N    SD       M reject.05.mean reject.1.mean
1   30:30 10:20 100:105          0.218         0.319
2   40:20 10:20 100:105          0.178         0.268
3 100:100 10:20 100:105          0.622         0.724
4   30:30 15:15 100:105          0.220         0.343
5   40:20 15:15 100:105          0.210         0.317
6 100:100 15:15 100:105          0.657         0.751
7   30:30 20:10 100:105          0.220         0.338
8   40:20 20:10 100:105          0.247         0.363
9 100:100 20:10 100:105          0.612         0.726
```

```r
cond.M <- c("100:106")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
                     M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

Experiment with That ...

```
        N    SD       M reject.05.mean reject.1.mean
1   30:30 10:20 100:106          0.336         0.447
2   40:20 10:20 100:106          0.347         0.446
3 100:100 10:20 100:106          0.753         0.840
4   30:30 15:15 100:106          0.305         0.433
5   40:20 15:15 100:106          0.298         0.412
6 100:100 15:15 100:106          0.784         0.869
7   30:30 20:10 100:106          0.297         0.429
8   40:20 20:10 100:106          0.183         0.299
9 100:100 20:10 100:106          0.746         0.833
```

```
tPowerSim(conds, var.equal = FALSE)
```

```
        N    SD       M reject.05.mean reject.1.mean
1   30:30 10:20 100:106          0.302         0.425
2   40:20 10:20 100:106          0.232         0.359
3 100:100 10:20 100:106          0.746         0.842
4   30:30 15:15 100:106          0.341         0.459
5   40:20 15:15 100:106          0.272         0.396
6 100:100 15:15 100:106          0.803         0.875
7   30:30 20:10 100:106          0.278         0.396
8   40:20 20:10 100:106          0.344         0.459
9 100:100 20:10 100:106          0.758         0.846
```

Experiment with That ...

```
cond.N <- c("100:100", "150:150", "150:100", "100:150")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
                        M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

```
         N    SD       M reject.05.mean reject.1.mean
1  100:100 10:20 100:106          0.734         0.827
2  150:150 10:20 100:106          0.899         0.944
3  150:100 10:20 100:106          0.851         0.906
4  100:150 10:20 100:106          0.821         0.905
5  100:100 15:15 100:106          0.809         0.891
6  150:150 15:15 100:106          0.922         0.955
7  150:100 15:15 100:106          0.875         0.928
8  100:150 15:15 100:106          0.874         0.922
9  100:100 20:10 100:106          0.739         0.837
10 150:150 20:10 100:106          0.908         0.956
11 150:100 20:10 100:106          0.810         0.884
12 100:150 20:10 100:106          0.847         0.905
```

```
tPowerSim(conds, var.equal = FALSE)
```

Experiment with That ...

```
          N     SD        M reject.05.mean reject.1.mean
1   100:100 10:20 100:106          0.763         0.865
2   150:150 10:20 100:106          0.902         0.936
3   150:100 10:20 100:106          0.766         0.851
4   100:150 10:20 100:106          0.883         0.932
5   100:100 15:15 100:106          0.804         0.871
6   150:150 15:15 100:106          0.935         0.968
7   150:100 15:15 100:106          0.865         0.922
8   100:150 15:15 100:106          0.864         0.907
9   100:100 20:10 100:106          0.766         0.852
10  150:150 20:10 100:106          0.885         0.939
11  150:100 20:10 100:106          0.902         0.950
12  100:150 20:10 100:106          0.795         0.874
```

```r
cond.M <- c("100:110")
conds <- expand.grid(nReps = 1000, SD = cond.SD, N = cond.N,
                     M = cond.M, stringsAsFactors = FALSE)
tPowerSim(conds, var.equal = TRUE)
```

Experiment with That ...

```
          N     SD         M reject.05.mean reject.1.mean
1   100:100 10:20 100:110          0.998         1.000
2   150:150 10:20 100:110          1.000         1.000
3   150:100 10:20 100:110          0.995         0.998
4   100:150 10:20 100:110          1.000         1.000
5   100:100 15:15 100:110          0.997         0.999
6   150:150 15:15 100:110          1.000         1.000
7   150:100 15:15 100:110          0.999         0.999
8   100:150 15:15 100:110          0.999         0.999
9   100:100 20:10 100:110          0.995         0.998
10  150:150 20:10 100:110          1.000         1.000
11  150:100 20:10 100:110          0.998         1.000
12  100:150 20:10 100:110          0.996         0.999
```

IQ study

- We are planning a study about parental IQ
- This is a study focused on a correlation, R, between the IQs of the parents (which are positively correlated).
- Cohen's guidelines say a "medium" correlation would be 0.30, so we will use that to manufacture data.
- We are going to manufacture data using a multivariate normal distribution

Digression: Drawing MVN samples

- A vector of means μ="mu" and a covariance matrix Σ="Sigma"

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = MVN\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & \sigma_{12} & & \sigma_{1p} \\ \sigma_{12} & \sigma_2^2 & & \sigma_{2p} \\ & & \ddots & \\ \sigma_{1p} & \sigma_{2p} & & \sigma_p^2 \end{bmatrix} \right)$$

- The one-variable formula for the probability density of the Normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \text{ or } \frac{1}{(2\pi)^{1/2}\sigma} e^{-\frac{1}{2}(x-\mu)\sigma^{-1}(x-\mu)}$$

- The multivariate one looks almost the same

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} e^{\frac{-1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

where p is the number of elements in μ.

Digression: Drawing MVN samples ...

- We will create Sigma by specifying standard deviations and a correlation matrix, then we use this handy formula

$$SD.diagonal \times Corr.matrix \times SD.diagonal$$

$$Sigma = \begin{bmatrix} \sigma_{x1} & 0 & 0 & 0 & 0 \\ 0 & \sigma_{x2} & 0 & 0 & 0 \\ 0 & 0 & \sigma_{x3} & 0 & 0 \\ 0 & 0 & 0 & \sigma_{x4} & 0 \\ 0 & 0 & 0 & 0 & \sigma_{x5} \end{bmatrix} \times \begin{bmatrix} 1 & \rho_{12} & \rho_{13} & \cdots & \rho_{1p} \\ \rho_{21} & 1 & \rho_{23} & & \rho_{2p} \\ \rho_{31} & \ddots & 1 & & \rho_{3p} \\ \vdots & 11 & \rho_{11} & \ddots & \\ \rho_{p1} & \rho_{11} & \rho_{11} & & 1 \end{bmatrix} \times \begin{bmatrix} \sigma_{x1} & 0 & 0 & 0 & 0 \\ 0 & \sigma_{x2} & 0 & 0 & 0 \\ 0 & 0 & \sigma_{x3} & 0 & 0 \\ 0 & 0 & 0 & \sigma_{x4} & 0 \\ 0 & 0 & 0 & 0 & \sigma_{x5} \end{bmatrix} \tag{1}$$

- The R implementation. We'll use `mvrnorm` from the `rockchalk` package, a slightly adjusted version of the one in MASS.

Digression: Drawing MVN samples ...

- rockchalk has convenience functions I created because I got tired of writing out matrix function calls

```
library(rockchalk)
myR <- lazyCor(X = 0.3, d = 5)
myR
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  1.0  0.3  0.3  0.3  0.3
[2,]  0.3  1.0  0.3  0.3  0.3
[3,]  0.3  0.3  1.0  0.3  0.3
[4,]  0.3  0.3  0.3  1.0  0.3
[5,]  0.3  0.3  0.3  0.3  1.0
```

```
mySD <- c(0.5, 0.5, 0.5, 1.5, 1.5)
myCov <- lazyCov(Rho = myR, Sd = mySD)
myCov
```

Digression: Drawing MVN samples ...

```
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 0.250 0.075 0.075 0.225 0.225
[2,] 0.075 0.250 0.075 0.225 0.225
[3,] 0.075 0.075 0.250 0.225 0.225
[4,] 0.225 0.225 0.225 2.250 0.675
[5,] 0.225 0.225 0.225 0.675 2.250
```

```
myMu <- c(1.1, 2.0, 1.1, -0.2, 0)
## Draw one to see what that does
set.seed(123123)
mvrnorm(1, mu = myMu, myCov)
```

```
[1]  1.2809807  2.1131828  0.9241272 -0.2839534 -0.4944292
```

Now create 1000 rows of that (5 columns)

```
N <- 1000
X <- mvrnorm(N, mu = myMu, myCov)
```

Digression: Drawing MVN samples ...

```
## Check column means
colMeans(X)
```

```
[1]  1.1008238  1.9931514  1.1021634 -0.2079248 -0.1264216
```

```
## Check Pearson Correlations
cor(X)
```

```
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 1.0000000 0.3581434 0.3040862 0.3085298 0.3271159
[2,] 0.3581434 1.0000000 0.2762719 0.3117241 0.3361439
[3,] 0.3040862 0.2762719 1.0000000 0.3015170 0.2782909
[4,] 0.3085298 0.3117241 0.3015170 1.0000000 0.3168756
[5,] 0.3271159 0.3361439 0.2782909 0.3168756 1.0000000
```

Fiddle to find create IQ data generator

```
R <- 0.30 #"medium" in Cohen's opinion
## Start with a correlation matrix of predictors
## If you do this the standard R way
IQ.cor <- matrix(c(1, R, R, 1), nrow = 2, ncol = 2,
                 dimnames = list(c("dadIQ", "momIQ"),
                                 c("dadIQ", "momIQ")))
IQ.cor
```

```
      dadIQ momIQ
dadIQ   1.0   0.3
momIQ   0.3   1.0
```

```
## My equivalent method in rockchalk
IQ.cor <- lazyCor(X = R, d = 2)
IQ.cor
```

```
     [,1] [,2]
[1,]  1.0  0.3
[2,]  0.3  1.0
```

Fiddle to find create IQ data generator ...

```
## these are the assumed means of momIQ and dadIQ
IQ.M <- c(dadIQ = 100, momIQ = 100)
## mvrnorm will take these as column names in the
## output. That's why those are named

## these are the standard devations of momIQ and dadIQ
IQ.SD <- c(15, 15)
## The diagonal matrix we need will be
diag(IQ.SD)
```

```
     [,1] [,2]
[1,]   15    0
[2,]    0   15
```

```
## Create the covariance matrix
IQ.cov <- diag(IQ.SD) %*% IQ.cor %*% diag(IQ.SD)
IQ.cov
```

```
      [,1]  [,2]
[1,] 225.0  67.5
[2,]  67.5 225.0
```

Fiddle to find create IQ data generator ...

```
## Use R's builtin cov2cor to double-check thc
## correlations
cov2cor(IQ.cov)
```

```
     [,1] [,2]
[1,]  1.0  0.3
[2,]  0.3  1.0
```

```
N <- 100
set.seed(123)
dat <- mvrnorm(n = N, mu = IQ.M, Sigma = IQ.cov)
dat <- as.data.frame(round(dat))
head(dat)
```

```
  dadIQ momIQ
1    95    91
2   118   119
3    86   117
4   117    94
5    96    88
6   112   118
```

Fiddle to find create IQ data generator ...

```
## Do observed means and correlations reflect
## the population parameters? Rounding is not too harmful
colMeans(dat)
```

```
 dadIQ  momIQ
100.23  99.86
```

```
cor(dat)
```

```
          dadIQ     momIQ
dadIQ 1.0000000 0.2170863
momIQ 0.2170863 1.0000000
```

Fiddle to find create IQ data generator ...

```
## Now we will once again draw random birth-orders
dat$first <- rbinom(n = N, size = 1, prob = .4)
## Now that we have our multiple predictors, we
## specify a model to generate outcomes (child's IQ)
## with random sampling error.

stde <- 9
b <- c(-3, 5, .5, .5)
## parameters designed so child IQ is average of parents
dat$IQnoe <- b[1] + b[2]*dat$first + b[3]*dat$dadIQ + b[4]*dat$momIQ
dat$IQ <- dat$IQnoe + rnorm(N, m = 0, sd = stde)
dat$IQ <- round(dat$IQ)
head(dat)
```

```
  dadIQ momIQ first IQnoe  IQ
1    95    91     1  95.0  92
2   118   119     0 115.5 110
3    86   117     1 103.5 100
4   117    94     0 102.5 103
5    96    88     0  89.0 103
6   112   118     0 112.0 111
```

Fiddle to find create IQ data generator ...

```
## Do sample statistics match data generator parameters?
mod0 <- lm(IQ ~ first + momIQ + dadIQ, data = dat)
summary(mod0)
```

```
Call:
lm(formula = IQ ~ first + momIQ + dadIQ, data = dat)

Residuals:
     Min        1Q    Median        3Q       Max
-20.0270   -5.5010   0.1397   6.3073   19.2770

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.73128    8.59794   0.318   0.7514
first        4.26077    1.86205   2.288   0.0243 *
momIQ        0.46511    0.07340   6.336 7.60e-09 ***
dadIQ        0.49168    0.06181   7.955 3.53e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.074 on 96 degrees of freedom
Multiple R-squared:  0.5867,  Adjusted R-squared:  0.5738
F-statistic: 45.42 on 3 and 96 DF,  p-value: < 2.2e-16
```

Anticipate Data Output Requirements

Monte Carlo methods can be used to check the power simultaneously for all effects (i.e., each slope estimates).

```
## Inspect data output, figure out what we want.
##
summary(mod0)$coef
```

```
              Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 2.7312803 8.59793508 0.317667 7.514275e-01
first       4.2607692 1.86205478 2.288208 2.431902e-02
momIQ       0.4651099 0.07340265 6.336418 7.602465e-09
dadIQ       0.4916806 0.06180512 7.955337 3.531280e-12
```

```
## get p values
summary(mod0)$coef[2:4, 4]
```

```
       first        momIQ        dadIQ
2.431902e-02 7.602465e-09 3.531280e-12
```

Anticipate Data Output Requirements ...

```
## check whether they meet significance criterion
alpha <- .05
alpha
```

```
[1] 0.05
```

```
summary(mod0)$coef[2:4, 4] < alpha
```

```
first momIQ dadIQ
 TRUE  TRUE  TRUE
```

Data Generator Program

```
makeData <- function(rep, N, R, stde = 9, b = c(-3, 5, .5, .5)) {
     require(rockchalk) # will load the package if it's not already
         loaded
     IQ.cor <- lazyCor(R, 2)
     IQ.M <- c(dadIQ = 100, momIQ = 100)
     IQ.SD <- c(15, 15)
     IQ.cov <- diag(c(15, 15)) %*% IQ.cor %*% diag(c(15, 15))
     dat <- mvrnorm(n = N, mu = IQ.M, Sigma = IQ.cov)
     dat <- as.data.frame(round(dat))
     dat$first <- rbinom(n = N, size = 1, prob = .4)
     dat$IQnoe <- b[1] + b[2]*dat$first + b[3]*dat$dadIQ +
         b[4]*dat$momIQ
     dat$IQ <- dat$IQnoe + rnorm(N, m = 0, sd = stde)
     dat$IQ <- round(dat$IQ)
     dat$rep <- rep
     dat
 }
## Test it once
set.seed(123)
dat <- makeData(1, N = 20, R = .3)
head(dat)
```

Data Generator Program ...

```
  dadIQ momIQ first IQnoe  IQ rep
1    95    91     0  90.0  92   1
2   118   119     1 120.5 120   1
3    86   117     0  98.5  98   1
4   117    94     1 107.5 120   1
5    96    88     0  89.0  87   1
6   112   118     0 112.0 126   1
```

```
## To analyze that data and return rejection decisions
getDecision <- function(data, alpha = .05) {
     ## run a regression on sample data
     mod <- lm(IQ ~ first + momIQ + dadIQ, data = data)
     ## return decisions about whether null was rejected for each slope
     summary(mod)$coef[2:4, 4] < alpha
 }
## Test it once (on the data we just generated)
getDecision(data = dat)
```

```
first momIQ dadIQ
FALSE  TRUE  TRUE
```

Data Generator Program ...

```
## Test it on 5 replications to see the format of the output
dataList <- lapply(1:5, makeData, N = 100, R = 0.30)
lapply(dataList, head)
```

```
[[1]]
  dadIQ momIQ first IQnoe  IQ rep
1   115    74     1  96.5  99   1
2   118   106     0 109.0 120   1
3    83   101     1  94.0  82   1
4   107    86     0  93.5  99   1
5   103   101     1 104.0  99   1
6    97   103     1 102.0 108   1

[[2]]
  dadIQ momIQ first IQnoe  IQ rep
1   119   124     0 118.5 114   2
2    90   113     0  98.5  96   2
3    95    87     1  93.0  97   2
4   129   129     0 126.0 121   2
5   133   107     1 122.0 123   2
6    94   101     0  94.5  77   2

[[3]]
  dadIQ momIQ first IQnoe  IQ rep
```

Data Generator Program ...

```
1     84    76      0  77.0  76    3
2    120    91      0 102.5 101    3
3     87   120      0 100.5 113    3
4    102    97      1 101.5 110    3
5    108   103      0 102.5  88    3
6    102   118      1 112.0 114    3

[[4]]
  dadIQ momIQ first IQnoe  IQ rep
1   105    79     0  89.0  85   4
2   119   115     1 119.0 107   4
3    65   122     0  90.5 102   4
4   103    95     0  96.0  83   4
5   118    90     1 106.0  98   4
6   107   105     1 108.0 114   4

[[5]]
  dadIQ momIQ first IQnoe  IQ rep
1   108    85     0  93.5  96   5
2    90    87     1  90.5  92   5
3    93    63     1  80.0  77   5
4   118   108     0 110.0 100   5
5    79    77     0  75.0  77   5
6   110   122     1 118.0  99   5
```

Data Generator Program ...

```
do.call(rbind, lapply(dataList, getDecision))
```

```
     first momIQ dadIQ
[1,]  TRUE  TRUE  TRUE
[2,]  TRUE  TRUE  TRUE
[3,]  TRUE  TRUE  TRUE
[4,]  TRUE  TRUE  TRUE
[5,]  TRUE  TRUE  TRUE
```

```
## Define a function that gets rejections for nReps replications per
## condition
runCond <- function(nReps, N, R) {
     ## generate nReps data sets
     dataList <- lapply(1:nReps, makeData, N = N, R = R)
     ## run regression and get rejection decisions for each data set
     out <- data.frame(do.call(rbind, lapply(dataList, getDecision)))
     ## record conditions
     out$N <- N
     out$R <- R
     ## return results
     out
 }
## Test it on 10 replications
runCond(nReps = 10, N = 100, R = .3)
```

Data Generator Program ...

```
   first momIQ dadIQ   N   R
1  FALSE  TRUE  TRUE 100 0.3
2   TRUE  TRUE  TRUE 100 0.3
3   TRUE  TRUE  TRUE 100 0.3
4   TRUE  TRUE  TRUE 100 0.3
5  FALSE  TRUE  TRUE 100 0.3
6  FALSE  TRUE  TRUE 100 0.3
7   TRUE  TRUE  TRUE 100 0.3
8   TRUE  TRUE  TRUE 100 0.3
9   TRUE  TRUE  TRUE 100 0.3
10 FALSE  TRUE  TRUE 100 0.3
```

Build a condition object

```
## Using a Monte Carlo design, we can calculate
   power across sample
## sizes as the proportion of cases where the
   null was rejected in
## each condition.
cond.N <- seq(from = 20, to = 150, by = 10)
cond.N
```

```
[1]  20  30  40  50  60  70  80  90 100 110 120 130 140 150
```

```
cond.R <- c(0.30) # for now, don't vary the
   correlation between predictors
conds <- expand.grid(N = cond.N, R = cond.R)
conds$nReps <- 1000
conds
```

Build a condition object ...

```
      N   R nReps
1    20 0.3  1000
2    30 0.3  1000
3    40 0.3  1000
4    50 0.3  1000
5    60 0.3  1000
6    70 0.3  1000
7    80 0.3  1000
8    90 0.3  1000
9   100 0.3  1000
10  110 0.3  1000
11  120 0.3  1000
12  130 0.3  1000
13  140 0.3  1000
14  150 0.3  1000
```

Run a simulation

```
## Run 1000 t tests in each condition to see how
    often the null is
## rejected
set.seed(123)
out <- apply(conds, MARGIN = 1,
             FUN = function(x) do.call(runCond,
                 as.list(x)))
out <- do.call(rbind, out)
head(out)
```

```
  first momIQ dadIQ  N   R
1 FALSE  TRUE  TRUE 20 0.3
2 FALSE  TRUE FALSE 20 0.3
3 FALSE  TRUE  TRUE 20 0.3
4 FALSE FALSE  TRUE 20 0.3
5 FALSE  TRUE  TRUE 20 0.3
6 FALSE  TRUE  TRUE 20 0.3
```

Run a simulation ...

```
## TRUE == 1 and FALSE == 0, so the mean of each
   outcome is the
## proportion of samples for which the null was
   rejected (in each
## condition).
rates <- aggregate(cbind(first, momIQ, dadIQ) ~
   N, data = out, mean)
rates
```

```
     N first momIQ dadIQ
1   20 0.149 0.805 0.825
2   30 0.281 0.963 0.959
3   40 0.375 0.994 0.989
4   50 0.448 0.997 0.996
5   60 0.529 1.000 1.000
6   70 0.581 1.000 1.000
7   80 0.656 1.000 1.000
8   90 0.723 1.000 1.000
9  100 0.762 1.000 1.000
10 110 0.813 1.000 1.000
```

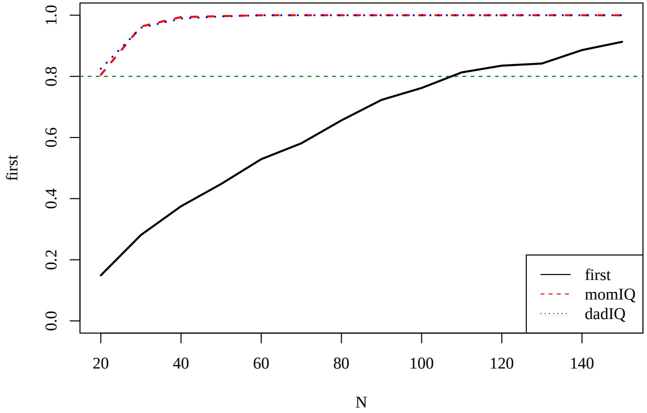
Run a simulation ...

```
11 120 0.835 1.000 1.000
12 130 0.842 1.000 1.000
13 140 0.886 1.000 1.000
14 150 0.913 1.000 1.000
```

Did we find out anything?

```
## plot power by sample size
plot(first ~ N, data = rates, type = "l", lwd =
    2, ylim = 0:1)
abline(h = .8, col = "darkgreen", lty = "dashed")
lines(momIQ ~ N, data = rates, col = "red", lwd =
    2, lty = 2)
lines(dadIQ ~ N, data = rates, col = "blue", lwd
    = 2, lty = 3)
legend("bottomright", legend = c("first",
    "momIQ", "dadIQ"),
        col = c("black" , "red", "blue"), lty =
            c(1,2,3))
```

Did we find out anything? ...



1. What sample size is required to detect the effect of mom's IQ at least 80% of the time?
2. Dad's IQ?

Did we find out anything? ...

3. First-born status?

Do we need to add more sample size conditions?

More complete power analyses might take into more population parameters, such as the correlation among other predictors, or the effects (slopes) of each predictor, or the variance of the random errors, or whether interactions exist, etc.

The function above already allows you to manipulate the correlation between predictors, but you can add arguments to manipulate other characteristics of importance.

Outline

Cookbook. And Beyond!

- Power analysis is easy for simple problems, we have a good deal of experience with 1-predictor models with simple designs
- More complicated models don't fit into the easy-to-use cookbooks
- Monte Carlo Simulation can be a way to understand the ability of a proposed study to detect statistically significant findings.

Can't live with it. Can't live without it

- Power analysis involves guessing about parameters and distributions of predictors
- Much of the work seems onerous or silly to researchers, who say they "just don't know," yet
- Nevertheless, project planners (and funders) need to be assured that the study will, if correctly executed, recover statistically significant evidence.
- If a study ever concludes with a comment like

  > *We did not find statistically significant differences between groups, but we still believe there are effects worth finding. The likely explanation for our difficulty is the small number of participants in each group.*

  we should blame the people who carried out the study for poor planning and inadequate power analysis.

References

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Session

```
sessionInfo()
```

```
R version 3.4.4 (2018-03-15)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04 LTS

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
     LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8
     LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                   LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8
    LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  base

other attached packages:
[1] rockchalk_1.8.111 pwr_1.2-2
```

Session ...

```
loaded via a namespace (and not attached):
 [1] Rcpp_0.12.15        lattice_0.20-35     MASS_7.3-49
      grid_3.4.4          MatrixModels_0.4-1
 [6] nlme_3.1-137        SparseM_1.77        minqa_1.2.4
      nloptr_1.0.4        car_2.1-6
[11] Matrix_1.2-14       splines_3.4.4       lme4_1.1-17
     tools_3.4.4         pbkrtest_0.4-7
[16] parallel_3.4.4      compiler_3.4.4      mgcv_1.8-23
     nnet_7.3-12         quantreg_5.35
[21] methods_3.4.4
```