Using R for Simulation

the basics

Steve Mooney

P9489
Spring 2016

Outline

Crediting those who did the real work...

Arnold *et al. BMC Medical Research Methodology* 2011, **11**:94
http://www.biomedcentral.com/1471-2288/11/94

COMMENTARY Open Access

Simulation methods to estimate design power: an overview for applied research

Benjamin F Arnold[1*†], Daniel R Hogan[2†], John M Colford Jr[1] and Alan E Hubbard[3]

`http://www.biomedcentral.com/content/pdf/1471-2288-11-94.pdf`

Power

As a reminder, power is the probability of avoiding a Type II error given that the alternative hypothesis is true.

In pseudo-math: $P(\textit{reject } H_0 | H_1 \textit{ is true})$

Traditional Power Calculations

Under a traditional aapproach, power is computed analytically. For example, for a two-sample t-test of two normally distributed variables, power $(1 - \beta)$ can be computed as follows:

$$1 - \beta = \phi(Z_{\alpha/2} + \frac{|\delta|\sqrt{n}}{\sigma})$$

The Problem...

Power calculations get increasingly complex as analytic techniques get more complex. How do you calculate power for:

- Observational studies with strong or time-varying confounding
- Analyses using principal components to predict an outcome
- Studies where complex survey samples were used
- A scenario where you expect heteroskedasticity
- etc.

The Generalized Simulation Approach

Power is defined in relation to a particular statistical test, so...

... if you know how to compute that test...
... and you know how to simulate your data generating process...
... then you can compute power by simulation rather than work through (potentially untenable) math.

Admittedly, at the cost of CPU time. But CPU time is CHEAP!

The simulation approach

Because the two-sample t-test is simple, we'll first explore simulation for power for that scenario to compare the simulation approach to the analytic approach.

The value of the simulation approach is that it scales better with analytic complexity, so starting simple may seem like a waste of time. But simple also gives you a baseline to tweak the code from.

The simulation code

```
fnPower <- function(intercept, beta, sd, nTotalSubjects, nTreated,
nIterations, alpha=0.05){
  pVec <- betaVec <- seVec <- rep(NA,nIterations)
  tx <- c(rep(1,nTreated),rep(0,nTotalSubjects - nTreated))
  for(i in 1:nIterations){
    resid <- rep(rnorm(nTotalSubjects,0,sd))
    y <- intercept + beta*tx + resid
    o <- ols(y~tx,x=TRUE,y=TRUE)
    v <- robcov(fit=o, cluster=1:length(tx))
    pVec[i] <- 2*pnorm(-abs(v$coeff[2]/sqrt(diag(v$var))[2]))
    betaVec[i] <- v$coef[2]
    seVec[i] <- sqrt(diag(v$var))[2]
  }
  power <- length(pVec[pVec<alpha])/length(pVec)
  return(list(power=power,p=pVec,beta=betaVec,se=seVec))
}
```

Note that the intercept parameter does not affect power, but helps to see how the simualtion process works...

The simulation logic

Where the simulation magic actually happens

```
for(i in 1:nIterations){
  resid <- rep(rnorm(nTotalSubjects,0,sd))
  y <- intercept + beta*tx + resid
  o <- ols(y~tx,x=TRUE,y=TRUE)
  v <- robcov(fit=o, cluster=1:length(tx))
  pVec[i] <- 2*pnorm(-abs(v$coeff[2]/sqrt(diag(v$var))[2]))
  betaVec[i] <- v$coef[2]
  seVec[i] <- sqrt(diag(v$var))[2]
}
```

- Get some residuals
- Compute an outcome
- Fit a regression model, keeping the covariance structure
- Compute a p-value with robust standard errors
- And store effect size and standard error

Checking the results I

Should we trust the simulation? Some empirical tests:

Because the true beta (i.e. effect size) is zero, the power calculated in this simulation should be appproximately the alpha value.

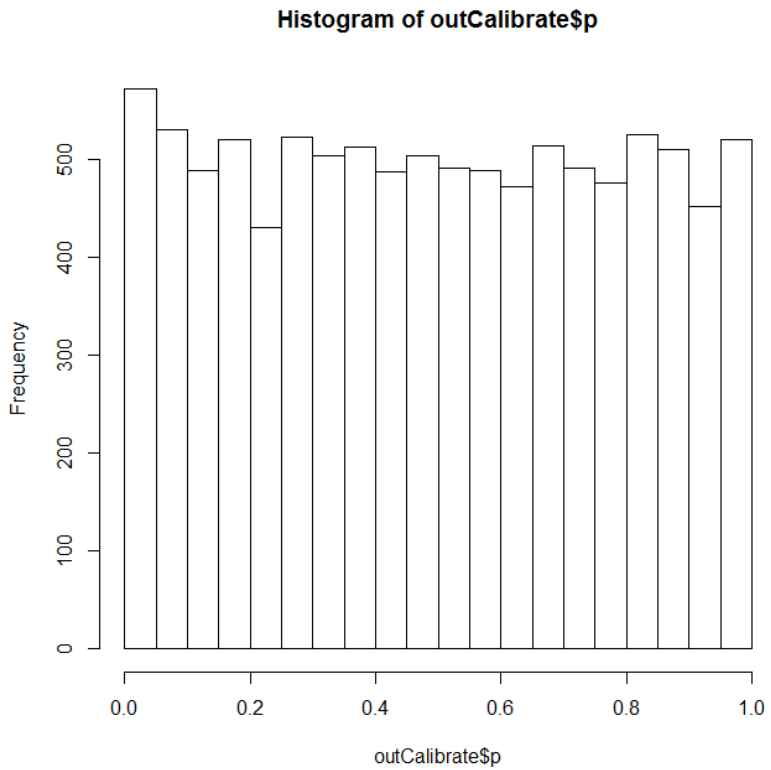
```
outCalibrate <- fnPower(intercept=-0.25,beta=0,sd=1,
  nTotalSubjects=100,nTreated=50, nIterations=10000, alpha=0.1)
outCalibrate$power
```

Note: this test (null effect size implies power equal to the type I error rate) should apply for any study/analytic design

Checking the results II

Similarly, a histogram of p-values should be a uniform distribution between 0 and 1

```
hist(outCalibrate$p)
```



Checking the results III

Because we're just doing a 2-sample t-test, we can also check our results against an analytic solution. Of course, this isn't a good test when the whole point is that the analytic solution is tricky to implement in code.

```
#Assume an effect size of 0.15 and compute power analytically
library(pwr)
pwr.t.test(500, 0.15)
#Then test our power calculation
outCalibrate <- fnPower(intercept=-0.25,beta=0.15,sd=1,
  nTotalSubjects=1000,nTreated=500, nIterations=10000, alpha=0.05)
outCalibrate$power
```

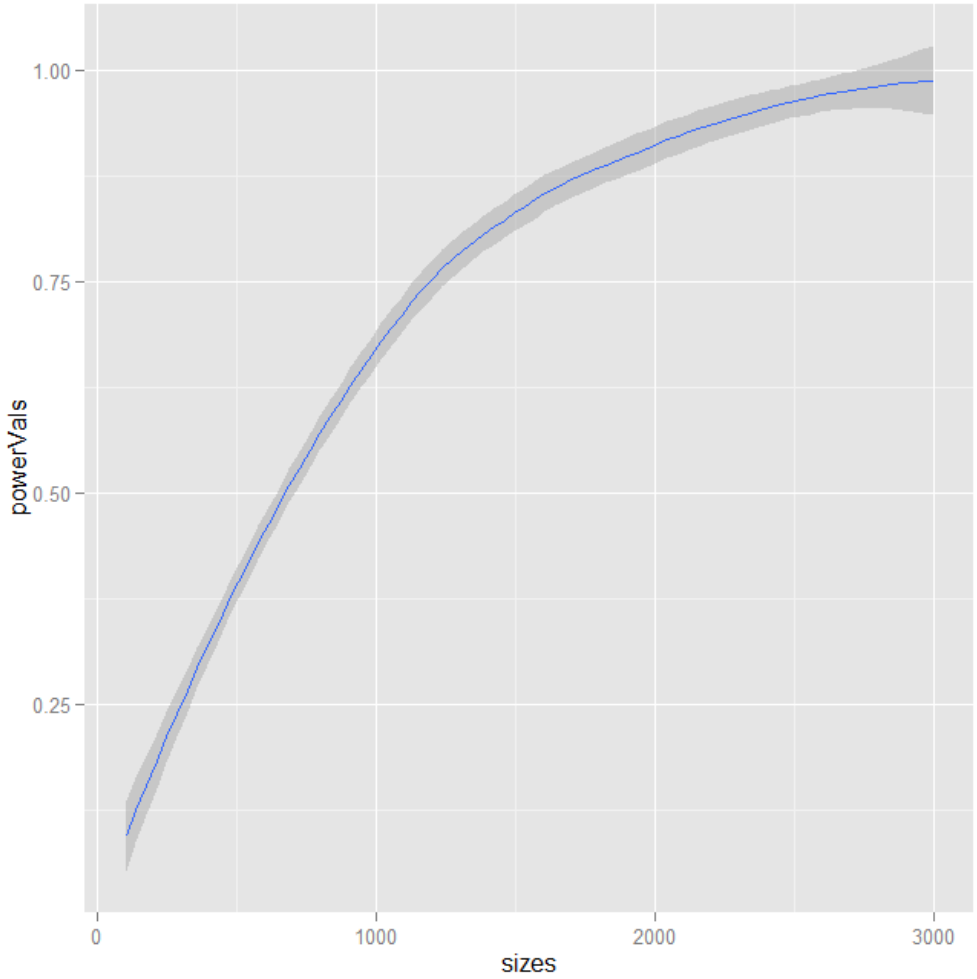
A Power curve

Sometimes (e.g. when designing a study) it's of interest to plot a power curve – the curve of the relationship of sample size to power
This is as simple as calling the power function in a loop:

```
# Power curve for sample size
sizes <- seq(100, 3000, by=50)
powerVals <- vector(length=length(sizes))
for (i in 1:length(sizes)) {
  powerVals[i] <- fnPower(intercept=1,beta=0.15,sd=1,
  nTotalSubjects=sizes[i],nTreated=sizes[i]/2,
nIterations=100)$power
}
powerCalc <- data.frame(sizes, powerVals)
ggplot(powerCalc) + aes(x=sizes, y=powerVals) + geom_smooth()
```

Power Curve

My power curve looks like



Try it out!

Use the simulation skills you've developed in this workshop to simulate for another scenario. Consider:

- Heteroskedasticity
- Interactions
- Survival Data
- Repeated Measures
- Anything else

See `http://www.biomedcentral.com/1471-2288/11/94/` for a more extensive simulation handling factorial randomization within clusters (which I pared down to make this example)