Analyzing Small Sample Experimental Data

Session 1 - Part 2: Tools and applications

Dominik Duell (University of Essex)

July 15, 2017

Part II: Tools and applications

Monte Carlo simulations

What?

- Model the world – assume exogenous part of that model
- Generate data according to that model– drawing from a (pseudo-)random sample
- Calculate endogenous part of model and generate estimate of interest
- Repeat S times
- Summarize or plot the empirical distribution of the S values

Why?

- Helpful when no data available
- Approximates what frequentist statistics is all about: sampling
- Study finite sample properties of estimators/statistics
- Compare the power of tests
- Allows to built counterfactuals think about it as robustness check or experimental lab

(Pseudo) - Random number generator

- deterministic approximation of a random number, uses `runiform()`
- `set seed 01010` for replication but not too often!
- all the distributions you want: `runiform()`, `rnormal(m, s)`, `rt(n)`, `rchi2(m)`, `rbeta(a,b)`, `rbinomial(n,p)`, `rgamma(a,b)`, `rhypergeometric(N,K,n)`, `rpoisson(m)`
- Could generate many distributions as transformation of the `runiform()` but less efficient

`simulate`

- Runs a Stata `command` or a user written `program` *s* times
- Results saved in data set
- Clear memory to evaluate the generated data set of simulations

postfile

- Posts data in a saved data set
- Can be run from within another data set, memory not cleared to post
- Can be embedded in a loop to run s times
- Load data set of posts to manipulate/analyse

A few more Stata necessities

- Macros
  - Globals
  - Locals
- `programs`
- `loopss`
  - `foreach`
  - `forval`
  - `while`

Macros in Stata

- `global` *macroName* = *string* accessable across `programs` and do-files
- `local` *macroName* = *string* accessable within `programs` and do-files
- `tempvar` *string* assigns name to a temporary variable within `programs` and do-files
- `tempname` *string* assigns name to a temporary scalar or matrix within `programs` and do-files
- `tempfile` *string* assigns name to a temporary file within `programs` and do-files

`programs`

- How to input?
  - uses data in memory
  - `args`
- How to access output?
  - `rclass|eclass|sclass` returns results in `r()|e()|s()`
  - when declared, it modifies results already in `r()|e()|s()`

programs

```
program programName, rclass|eclass|sclass
args argument1, ..., argumentN
... stuff happens ... that generates/plots/etc. something
return|ereturn|sreturn scalar|matrix returnName
end
```

- Before writing `programs`, test contents outside

program example

```
program spitOutBootstrappedCIs, rclass
        args B function statistic
        qui bootstrap `statistic', reps(`B') seed(01010): `function'
        mat result = r(table)
        return scalar theta = result[1,1]
        return scalar lb = result[5,1]
        return scalar ub = result[6,1]
end
```

program extensions

- define syntax, e.g.

  `syntax varlist [if] [in] [, DOF(integer 50) Beta(real 1.0)]`

- define properties, e.g.

  `program logit, ... properties(or svyb svyj svyr mi)`

Loops

- `foreach`, `forvalue` loops – repeat for a fixed number of iterations
- `while` loop – repeat until a certain condition is satisfied

Loops

- foreach – repeat for a fixed number of iterations

```
foreach item in local itemList {
        something happens with `item'
}
```

Loops

- `foreach` loop over list of strings with count

```
local i = 1
foreach item in local itemList {
        something happens with ‘item’
        something happens with ‘i’
        local i = ‘i’ + 1
}
```

Loops

- `forvalue` loop – repeat for a fixed number of iterations

```
forvalue i = minimum(step)maximum|minimum/maximum {
        something happens with ‘i’
}
```

Loops

- `while` loop – repeat until a certain condition is satisfied

```
while statementAbouti {
        something happens with ‘i’
}
```

Loops

`while` example:

```
local i = 1
while `i' < 40 {
        g u`i' = runiform()
    local i = `i' + 1
}
```

Stata's Monte Carlo simulations command

Basic syntax:

`simulate [`*exp_list*`], reps(#) [`*options*`] :` *command*

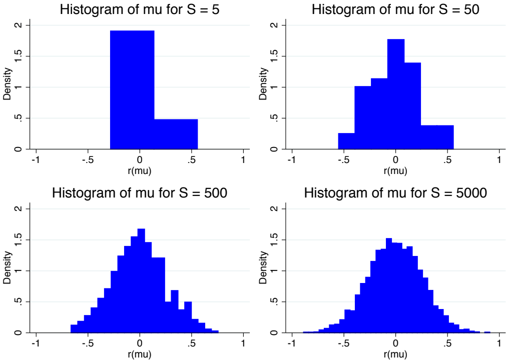
simulate example

```
program define normalDistribution, rclass
        syntax [, obs(integer 1) mean(real 0) sd(real 1)]
        drop _all
        set obs ‘obs’
        tempvar mu
        g ‘mu’ = rnormal(‘mean’,‘sd’)
        sum ‘mu’;
        return scalar mu = r(mean)
end
```

simulate example

```
foreach s in 5 50 500 5000 {
        simulate mu=r(mu), reps(‘s’) seed(010101) saving(sim, replace):
        normalDistribution, obs(15) mean(0) sd(1)
        use sim, clear
        hist mu, ‘graphr’ col(blue) name(hist‘s’, replace)
        ti("Histogram of mu for S = ‘s’", col(black))
}

gr combine hist5 hist50 hist500 hist5000, ‘graphr’ ‘grcom’ rows(2)
```

`simulate` example



Stata command to post results to saved data set

`postfile` *namePostRoutine listOfVariables* `using` *nameOfFile*`[, every(#) replace]`

to declare variable names, data set name

`post` *postname* (*value of variable1*) ... (*value of variableN*)

to add a new observation

`postclose` *postname*

to declare end of posting

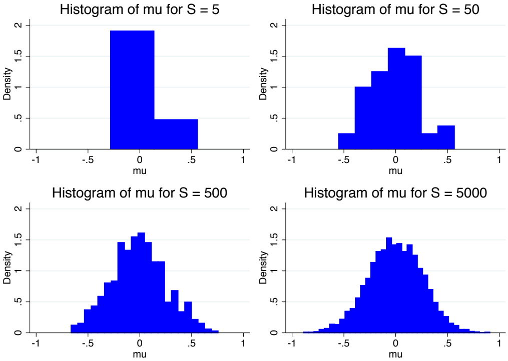
`postfile` example

```
set seed 010101
local obs = 15
local mean = 0
local sd = 1
local nSimsList = "5 50 500 5000"
```

postfile example

```
foreach s in `nSimsList' {
        tempname normalDistribution
        postfile `normalDistribution' mu using sim, replace
        forvalue i = 1/`s' {
                drop _all
                set obs `obs'
                tempvar mu
                g `mu' = rnormal(`mean',`sd')
                sum `mu'
                post `normalDistribution' (r(mean))
        };
        postclose `normalDistribution'

        use sim, clear
        hist mu, `graphr' col(blue) name(hist`s', replace) \\\
        ti("Histogram of mu for S = `s'", col(black))
}

gr combine hist5 hist50 hist500 hist5000, `graphr' `grcom' rows(2)
```

`postfile` example

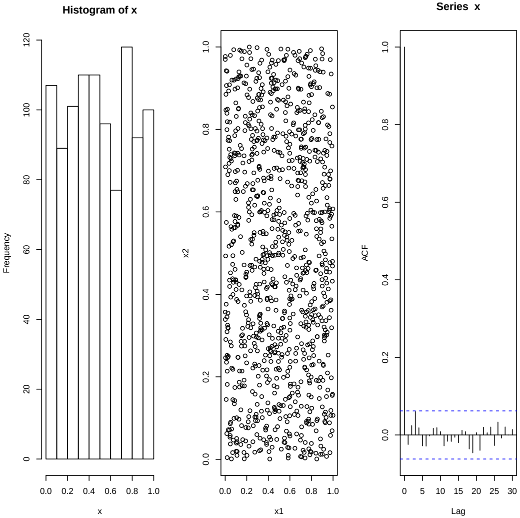


(Pseudo) - Random number generator

- All tools you want: `runif`, `rpois`, `rnorm`, `rbinom`, `rgamma`, `rbeta`, ...
- Could generate many distributions as transformation of the `runif` but less efficient
- Always check out what is generated:

```
x = runif(1000)
x2 = x[-1]
par(mfrow=c(1,3))
hist(x)
plot(x1,x2)
acf(x)
```

(Pseudo) - Random number generator



Loops

- `for` loop – repeat for a fixed number of iterations
- `while` loop – repeat until a certain condition is satisfied

Loops

- `for` loop – over a list of items

```
for (item in c(item1, item2, ..., itemN)) {
        something happens with item
}
```

Loops

- `for` loop – repeat for a fixed number of iterations

```
for (i in 1:10) {
      something happens with i
}
```

Loops

- `while` loop – repeat until a certain condition is satisfied

```
i = 1
while (i < 10) {
        something happens
        i <- i + 1
}
```

Functions

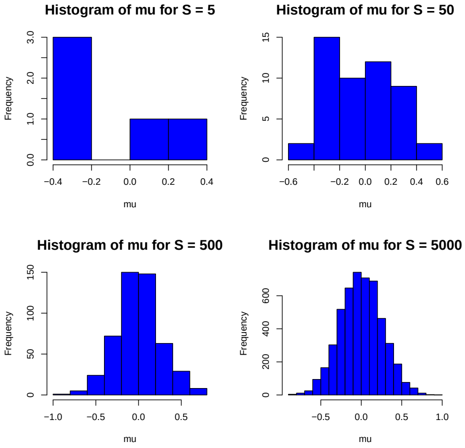
```
functionName <- function(argument1,...) {
        # something happens here
}
```

- Before writing `functions`, test contents outside

Simulations using loops

```
set.seed(010101)
par(mfrow=c(2,2))
for (s in c(5, 50, 500, 5000)) {
        nSims = s
        mu = rep(NA,nSims) # sets the vector to be filled
        nSample=15
        for(i in 1:nSims){
                x = rnorm(n=nSample,mean=0,sd=1)
                mu[i] = mean(x)
        }
        hist(mu,main=paste("Histogram of mu for S =",nSims),col="blue",
        cex.main=1.5)
}
```

Simulations using loops

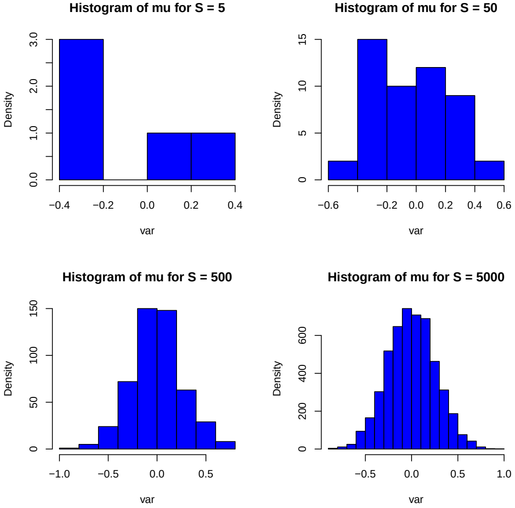


- But, loops can be slow!

Simulations using functions and replicate

```
set.seed(010101)
normalDistr.sim <- function(x){
        var <- rnorm(x)
        return(mean(var))
}

numObs <- 15
par(mfrow = c(2,2))
for(s in c(5,50,500,5000)) {
        sim <- replicate(s, normalDistr.sim(numObs))
        hist(sim, main = paste("Histogram of mu for S =", s), ylab="Density",
        xlab="var", col="blue")
}
```

Simulations using `functions` and `replicate`

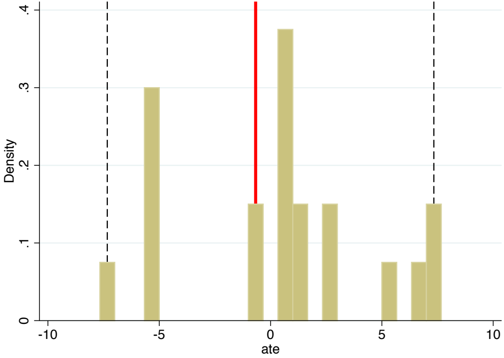


Randomization inference

Randomization inference in Stata

permute

```
permute cat ate=(r(mu_2)-r(mu_1)), reps(20) sav(permuteTTest, replace) nodots nowarn nodrop
left: ttest var, by(cat)
```

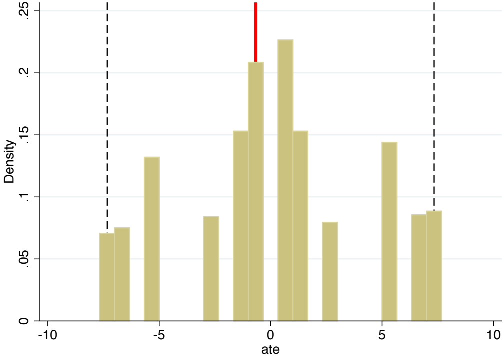


permute

```
Monte Carlo permutation results                  Number of obs    =          6

      command:  ttest var, by(cat)
          ate:  r(mu_2)-r(mu_1)
  permute var:  cat

------------------------------------------------------------------------------
T            |     T(obs)       c       n   p=c/n   SE(p) [95% Conf. Interval]
-------------+----------------------------------------------------------------
         ate |  -.6666667       7      20  0.3500  0.1067  .1539092   .5921885
------------------------------------------------------------------------------
Note:  confidence interval is with respect to p=c/n.
Note:  c = #{T <= T(obs)}
```

permute

```
permute cat ate=(r(mu_2)-r(mu_1)), reps(1000) sav(permuteTTest, replace) nodots nowarn nodrop
left: ttest var, by(cat)
```



permute

```
Monte Carlo permutation results                    Number of obs   =          6

      command:  ttest var, by(cat)
          ate:  r(mu_2)-r(mu_1)
  permute var:  cat

------------------------------------------------------------------------------
T            |     T(obs)       c       n   p=c/n   SE(p) [95% Conf. Interval]
-------------+----------------------------------------------------------------
         ate |  -.6666667     482    1000  0.4820  0.0158  .4506223   .5134839
------------------------------------------------------------------------------
Note:  confidence interval is with respect to p=c/n.
Note:  c = #{T <= T(obs)}
```

- Note, approximation of p-value

tsrtest

- Obtain exact p-value

```
program drop _all
program diffInMeans, rclass
        sum var if(cat==0)
    local control=r(mean)
        sum var if(cat==1)
    local treatment=r(mean)
    return scalar ate = `treatment'-`control'
end

tsrtest cat r(ate), reps(1000) nullvalue(-.67) exact: diffInMeans;
```

tsrtest

```
Two-sample randomization test for theta=r(ate) of diffInMeans by cat

Combinations:   20 = (6 choose 3)
Observed theta: -.6667

Minimum time needed for exact test (h:m:s):  0:00:00
Mode: exact

progress: |....................|

 p=0.65000 [one-tailed test of Ho:  theta(cat==0)<=theta(cat==1)]
 p=0.50000 [one-tailed test of Ho:  theta(cat==0)>=theta(cat==1)]
 p=1.00000 [two-tailed test of Ho:  theta(cat==0)==theta(cat==1)]
```

More randomization inference tools

- `permtest1` – randomization inference for Wilcoxon sign-ranked test (`signrank`)
- `permtest2` – randomization inference for Wilcoxon/Mann-Whitney rank-sum test (`ranksum`)

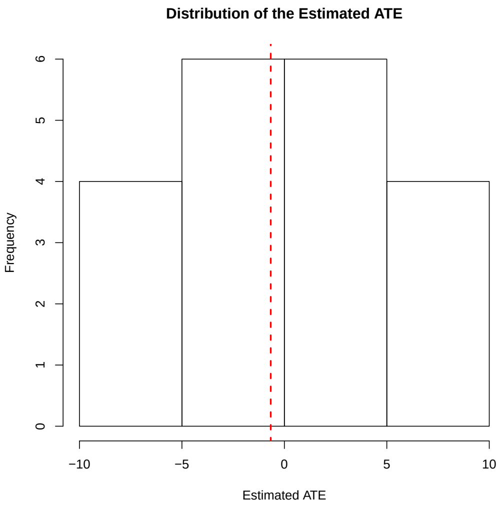
  both programs optionally return exact p-values
- `ritest` – allows for more complex resampling (e.g., stratifying, clustering)

Randomization inference in R

ri-package

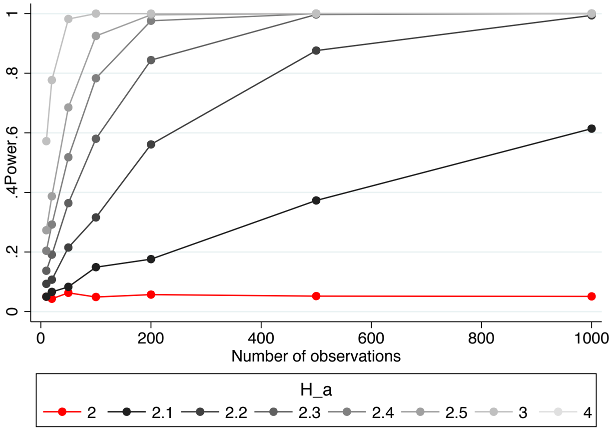
```
library(ri) # Check out package by Aronow/Samii: https://cran.r-project.org/web/packages/ri/ri.pdf
data <- read.dta("data/fakeData.dta")
data <- matrix(data[1:6,1:2])

y <- data[[1]]
t <- data[[2]]
cluster <- seq(1,6) # we do not cluster in this example
block <- c(rep(1,6)) # we do not block in this example
permutations <- genperms(t,block,cluster)
probability <- genprobexact(t,block,cluster)
ate <- estate(y,t,prob=probability)
permutations
probability
ate
potentialOutcomes <- genouts(y,t,ate=0)
distributionUnderH0 <- gendist(potentialOutcomes,permutations,prob=probability)
dispdist(distributionUnderH0, ate, quantiles=c(.025,0.975))
```

ri-package


Distribution of the Estimated ATE
Frequency
0
1
2
3
4
5
6
−10
−5
0
5
10
Estimated ATE

Statistical power evaluation in Stata

```
* Check out Cameron/Trivedi: Microeconomics using Stata, pp.135ff, 140ff, 408ff
program drop _all;
program powerCalculation, rclass;
        syntax [, numSim(integer 100) obs(integer 1) h0(real 0) ha(real 0) alpha(real .05)];
        tempname sim;
        postfile `sim' pvalues using powerResults, replace;
        forvalues i = 1/`numSim' {;
                drop _all;
                qui {;
                        set obs `obs';
                        g double var = rnormal();
                        g y = `ha'*var + rchi2(1);
                        reg y var;
                        test var= `h0';
                        scalar p = r(p);
                        post `sim' (p);
                };
        };
        postclose `sim';
        use powerResults, clear;
        qui count if(pvalues < `alpha');
        return scalar power = r(N)/`numSim';
end;
```


Power
0
.2
.4
.6
.8
1
0
200
400
600
800
1000
Number of observations
H_a
2
2.1
2.2
2.3
2.4
2.5
3
4

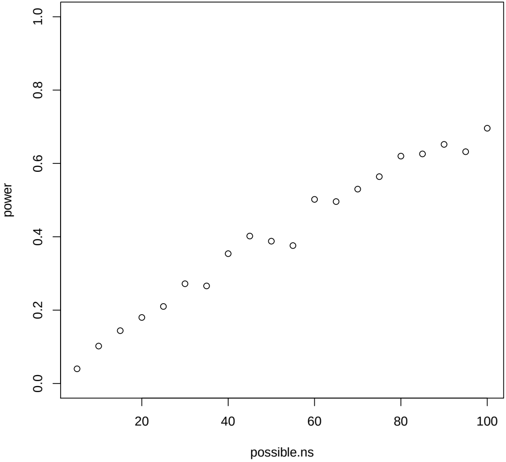
More tools to evaluate statistical power

- `power` command
- Example: `power twomeans 2 2.5 sd(1) sd(10)`
- Compute power, required sample size, largest expected effect

```
# Check out EGAP: http://egap.org/content/power-analysis-simulations-r
possible.ns <- seq(from=5, to=100, by=5)
power <- rep(NA, length(possible.ns))
alpha <- 0.05
sims <- 500
H_a <- .5

for (j in 1:length(possible.ns)){
  N <- possible.ns[j]
  significant.experiments <- rep(NA, sims)

  for (i in 1:sims){
    p.value <- t.test(rnorm(N/2),rnorm(N/2,H_a))$p.value
    significant.experiments[i] <- (p.value <= alpha)
  }

  power[j] <- mean(significant.experiments)
}
plot(possible.ns, power, ylim=c(0,1))
```

More tools to evaluate statistical power

- Many
- `pwr`-package
- `PoweR`-package – package links to many tests
- `power.t.test`


power
possible.ns
0.0
0.2
0.4
0.6
0.8
1.0
20
40
60
80
100

Performance of standard estimators in small samples

How good is your standard test?

- The textbook claim of $n \approx 30$ such that, under random sampling, samples statistics approach a normal distribution build on studies of a few distributions
- Many distribution converge slower (some faster)
- Even if sample statistic approaches normal, test statistic may not (i.e., t-statistic)
- What is a good test/estimator?

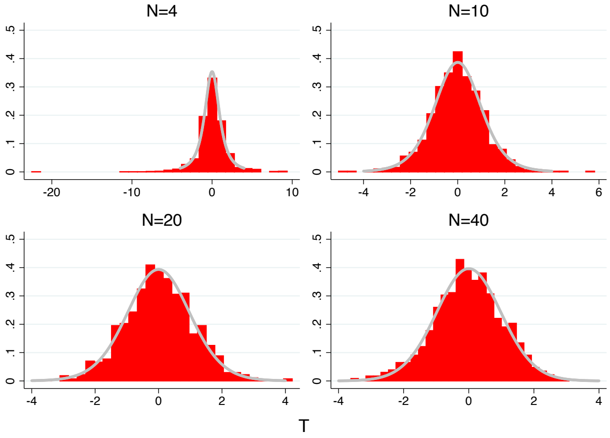
Assessing performance (in small samples):

- Robust: statistics are robust if small changes in distribution of the underlying sample have only small effects on their value
- Small Type I error – small rate rejection of true H_0
- High statistical power – high rate rejection of false H_0

Robustness

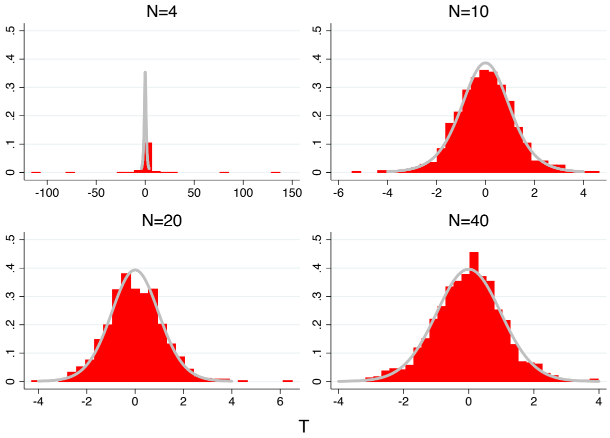
Exercise 1

- In Stata or R, write a function to
  - generate two normally distributed random variables with $n_1 = n_2 = 4$ observations, $\mu_1 = \mu_2 = 0$ (i.e., H_0 : no difference), and $\sigma_1 = \sigma_2 = 1$
  - conduct a t-test of equality of means
  - repeat for $N = \{5, 10, 20\}$ and extract t-statistic
  - How does the distribution of the test statistic vary with sample size?

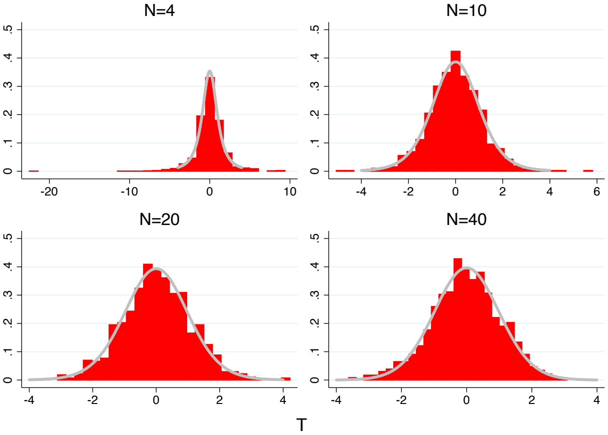

N=4
N=10
N=20
N=40
0
.1
.2
.3
.4
.5
-20
-10
0
10
-4
-2
0
2
4
6
T

Exercise 2

- Adjust the function and vary the distribution of the two random variables
- How does the distribution of the test statistic vary with sample size and variations in the data generating process?



- skewed population distribution, outliers

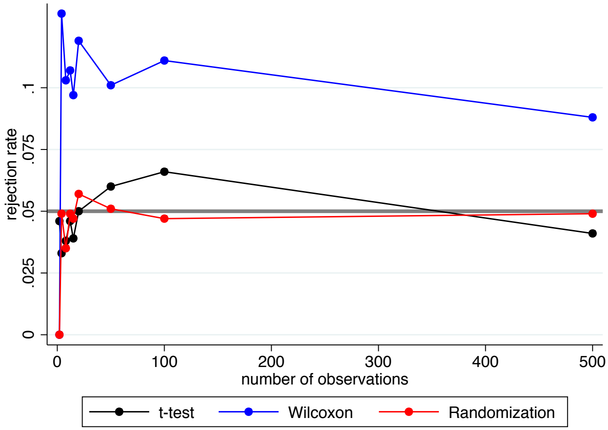


- normal population distribution, no outliers

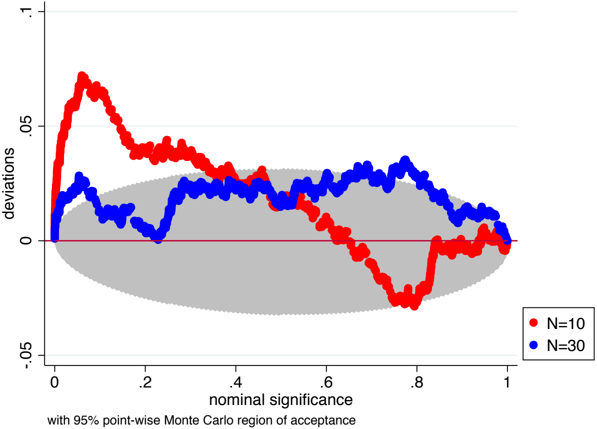
Small type 1 error

Exercise 3

- In Stata or R, write a function to
  - generate two normally distributed random variables with $n_1 = n_2 = 2$ observations, $\mu_1 = \mu_2$ (i.e., H_0 : no difference), and $\sigma_1 = \sigma_2 = 1$ but add outliers
  - simulate a t-test of equality of means, a Wilcoxon test, and a difference in means test based on randomization inference S times
    (Wilcoxon tests for shift in distribution but lets keep it here for the sake of illustration)
  - repeat for $N = \{4, 8, 12, 15, 20, 50, 100, 500\}$ and extract proportion H_0 rejected
  - How does occurrence of a type 1 error change with sample size across tests?


rejection rate
0
.025
.05
.075
.1
0
100
200
300
400
500
number of observations
t-test
Wilcoxon
Randomization

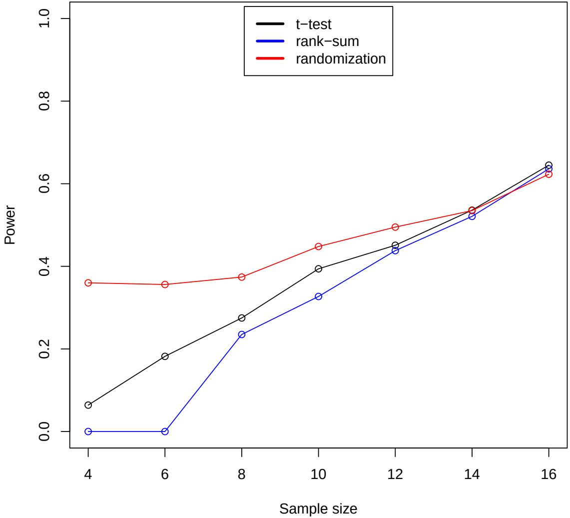
```
program drop _all;
program define simTTest, rclass;
        drop _all;
    set obs 30;
    g x = rchi2(2);
    ttest x=2 in 1/10;
        return scalar p10 = r(p);
    ttest x=2;
    return scalar p30 = r(p);
end;

simulate p10=r(p10) p30=r(p30), reps(1000) seed(010101): simTTest;

lab var p10 "N=10";
lab var p30 "N=30";

simpplot p10 p30, `graphr' main1opt(mcolor(red)) main2opt(mcolor(blue))
```


-.05
0
.05
.1
deviations
0
.2
.4
.6
.8
1
nominal significance
with 95% point-wise Monte Carlo region of acceptance
N=10
N=30

High statistical power

Exercise 4

- Create a population with two groups and outliers in one of them in the main variable of interest. Conduct t-test, Wilcoxon rank-sum, and differences in mean based on randomization inference. Extract statistical power
- Allow function output to vary with sample size


t−test
rank−sum
randomization
Power
0.0
0.2
0.4
0.6
0.8
1.0
4
6
8
10
12
14
16
Sample size

Random numbers, simulations

- STATA blog posts on random numbers
- Baum: Simulation for estimation and testing
- Carsey: Simulations
- Robert/Casella: Introducing Monte Carlo Methods with R

Randomization inference

- Kaiser/Lacy (2009): A general-purpose method for two-group randomization tests
- Aronow/Samii (2012): `ri`-package for R
- Bowers/Fredrickson/Hansen (2016): `RItools`-package for R

Standard estimators and sample size

- Imbens/Rubin (2015): Causal inference in Statistics, Social, and Biomedical Science
- Wilcox (2012): Introduction to Robust Estimation and Hypothesis Testing

Power analysis

- EGAP: 10 Things You Need to Know About Statistical Power
- EGAP: Power Analysis Simulations in R