SSSA
Small Sample Statistical Analysis

Day 1
Get to work

Dominik Duell (University of Essex)

September 26, 2016

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

Monte Carlo simulations

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

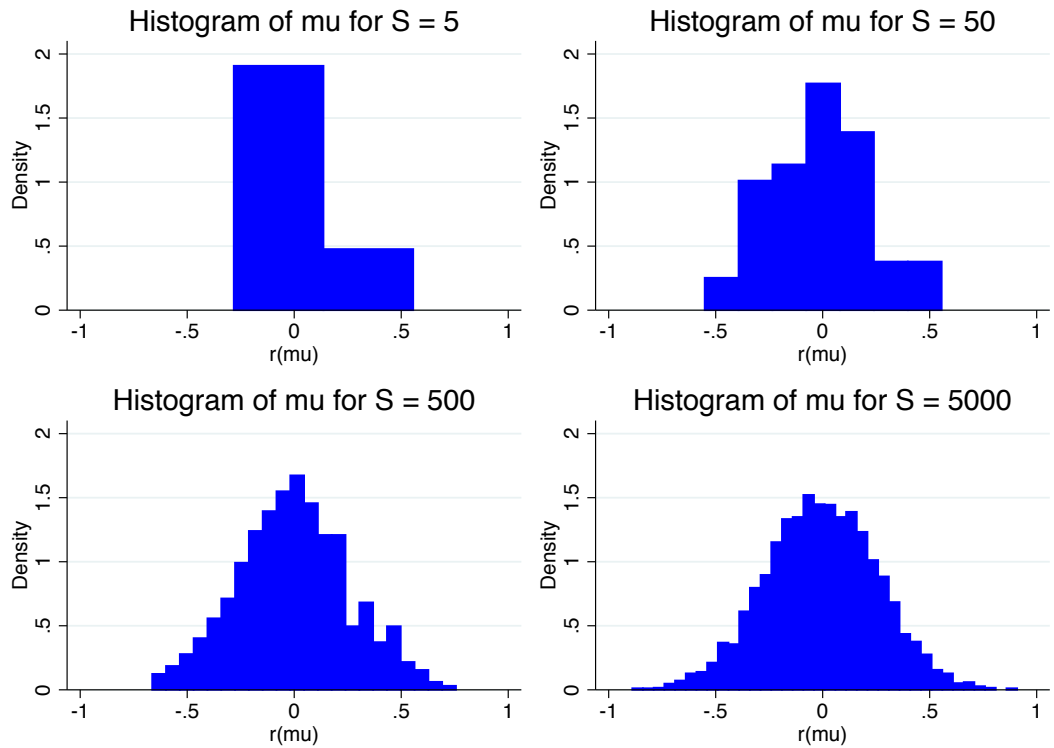
```
program define normalDistribution, rclass
        syntax [, obs(integer 1) mean(real 0) sd(real 1)]
        drop _all
        set obs `obs'
        tempvar mu
        g `mu' = rnormal(`mean',`sd')
        sum `mu';
        return scalar mu = r(mean)
end
```

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"
```

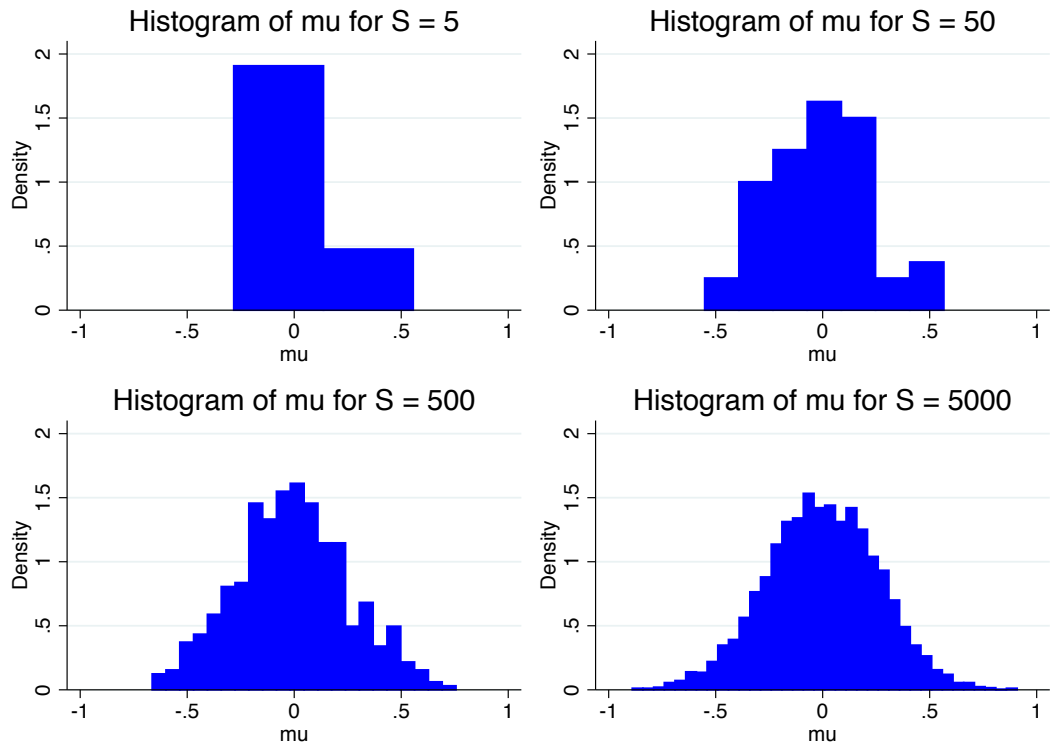
postfile example

```
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

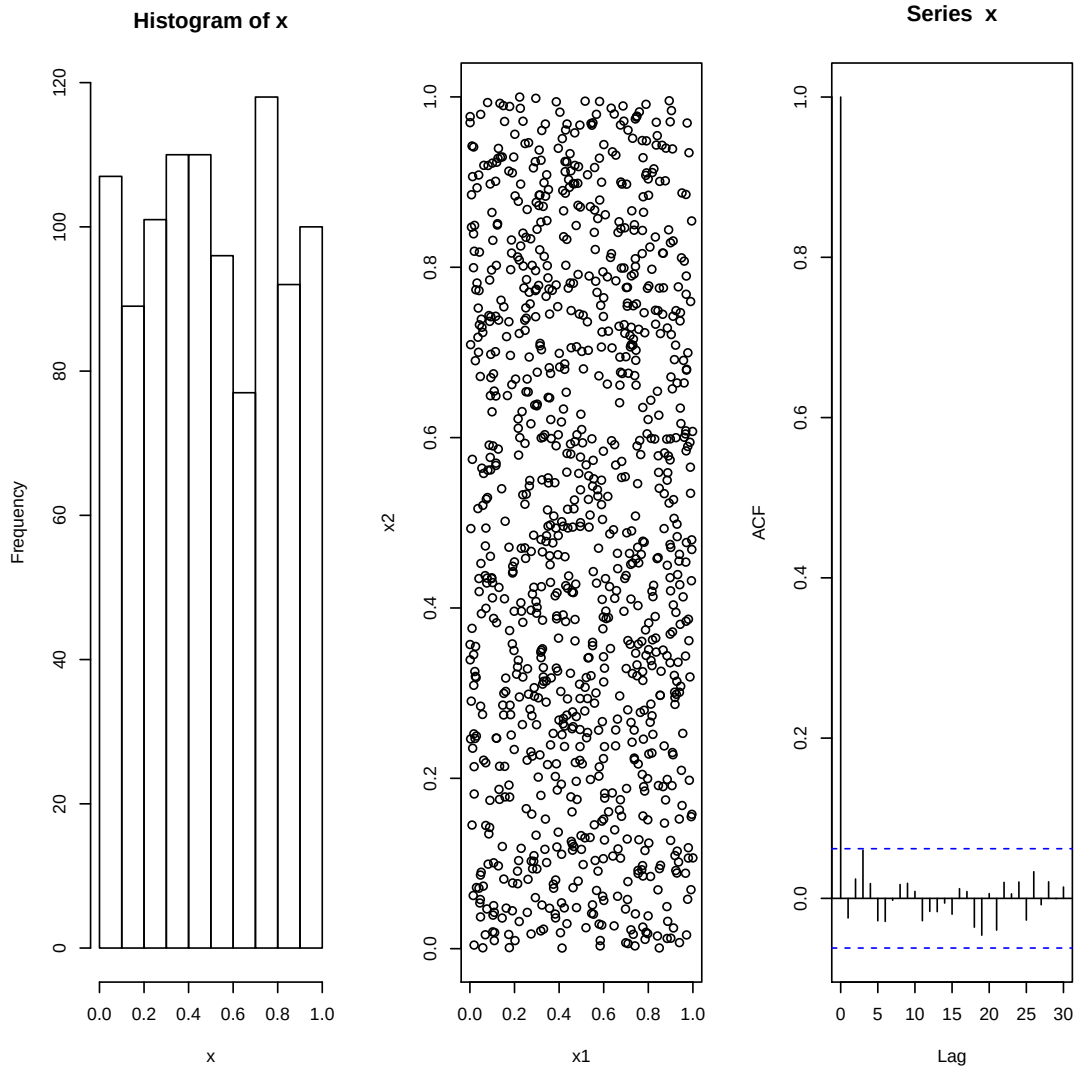


(Pseudo) - Random number generator

- 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

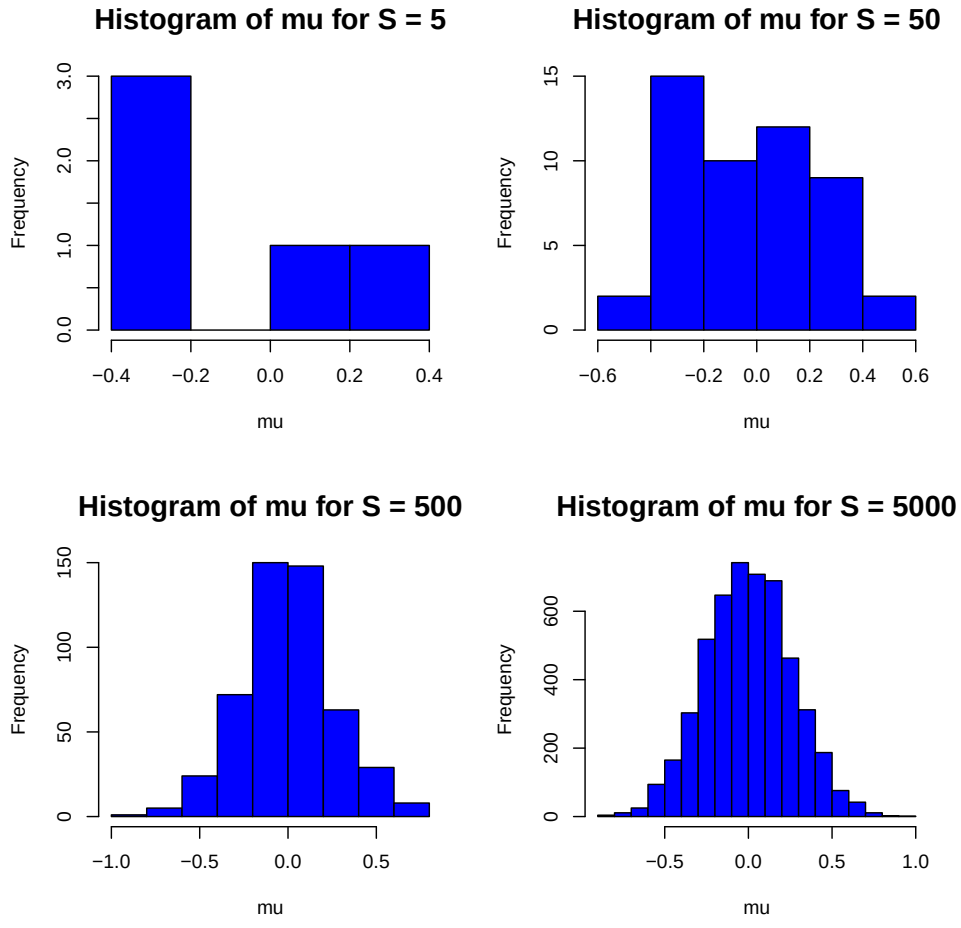
```
functionName <- function(argument1,...) {
        # something happens here
}
```

- 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



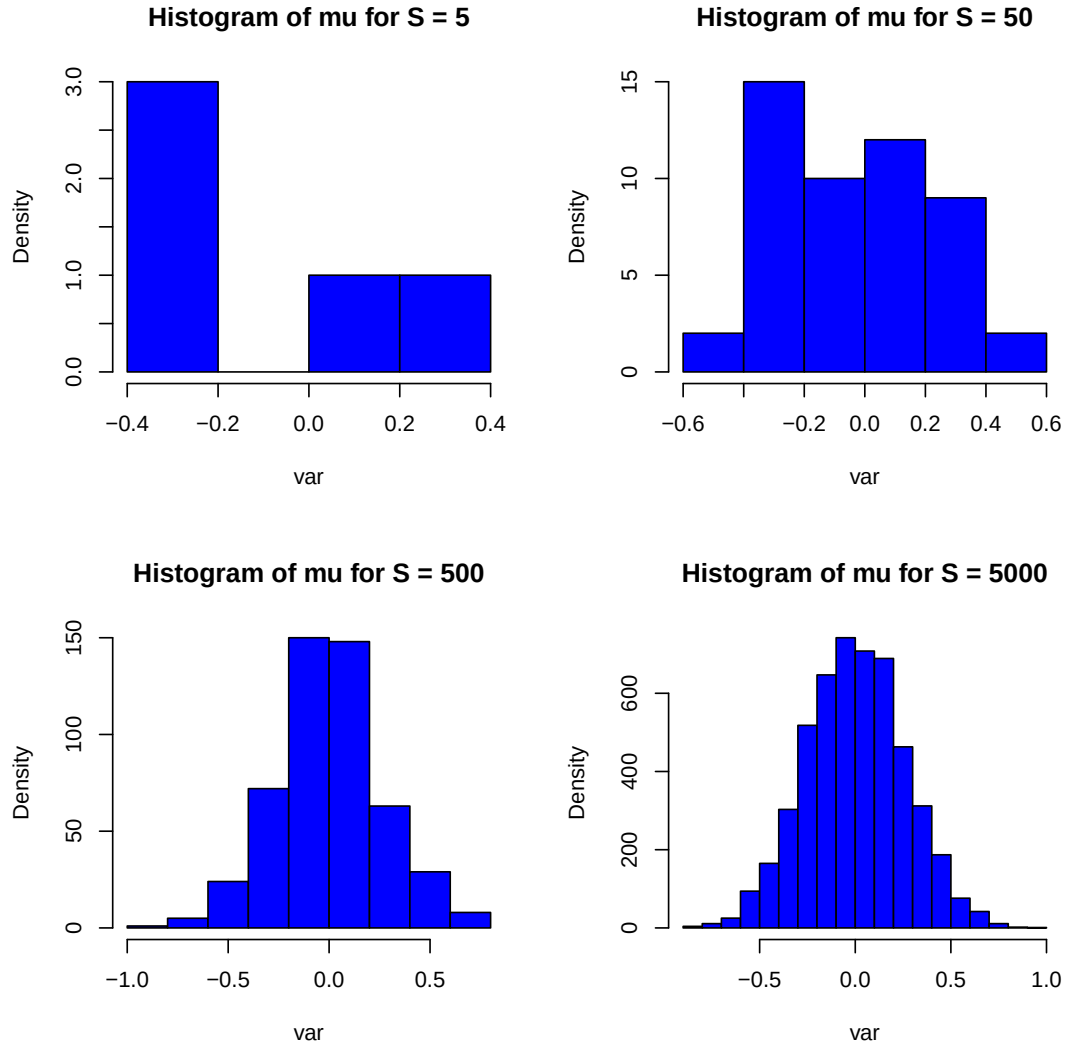
- But, loops can be slow!

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`



Task 1

- In Stata or R, write a function to
  - generate two normally distributed random variables with $n_1 = n_2 = 8$ observations, $\mu_1 = \mu_2 = 0$, and $\sigma_1 = \sigma_2 = 1$
  - conduct a t-test of equality of means
  - repeat $S = \{5, 50, 500, 5000\}$ times and extract t-statistic

Task 2

- Adjust the function and vary mean and standard deviation of the two random variables

Task 3

- Adjust the function and introduce
  - $\mu_1 \neq \mu_2$
  - $\sigma_1 \neq \sigma_2$
  - $n_1 \neg n_2$

Task 4

- Adjust the function and vary the distributions of the random variables to a non-normal distribution (e.g., uniform, χ^2, poisson)?

Now, how is the distribution of the t-statistic varying with sample size and various differences in the data generating process?

Task 5

- In Stata or R, write a function
  - to generate a normally distributed variable x
  - to define $y = 2 * x + \epsilon$
  - to run a least square regression of x on y
  - with normally distributed error term
  - repeat $S = \{5, 50, 500, 5000\}$ times and extract t-statistic
  - Allow function to set number of observations (start with $n = 100$)

Task 6

- Adjust the function and introduce an error term that is distributed χ^2

Task 7

- Set $n = 15$

How good are the estimates across error structures and sample sizes?

Task 1

- Take the function from Task 1 (t-test) and extract probability of type 1 error and statistical power
- Allow function output to vary with sample size

Task 2

- Take the function from Task 1 (regression) and extract probability of type 1 error and statistical power
- Allow function output to vary with 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