

# Lecture 10

Brad McNeney

Random number generation

Permutation tests

# Load packages

```
library(dplyr)
library(broom)
library(Stat2Data)
data(AutoPollution)
data(CloudSeeding)
data(Hawks)
```

## Random number generation

# Random number generation

- ▶ Reference: Chapter 2 of Robert and Casella (2004),  
Introducing Monte Carlo Methods with R
  - ▶ Online copy available from the library: [<http://search.lib.sfu.ca/?q=introducing%20monte%20carlo%20methods%20with%20r>]

# Why generate random numbers?

- ▶ Simulating from a data-generating process allows us to evaluate the performance of methods for making inference from such data.
  - ▶ E.G., to evaluate the performance of least squares regression for inference of regression coefficients, we can repeatedly generate datasets, perform inference, and summarize the results (example later).
- ▶ Many modern statistical methods are simulation-based; e.g.,
  - ▶ permutation tests
  - ▶ bootstrap tests and confidence intervals
  - ▶ cross validation for selecting “tuning” parameters

# Random number generation

- ▶ We can't generate truly random numbers.
- ▶ Instead we generate deterministic sequences or *streams* of pseudo-random numbers.
  - ▶ Goal: Every  $n$ -tuple from the stream should be statistically indistinguishable from a random sequence of size  $n$  drawn from a uniform distribution on  $(0,1)$ , denoted  $U(0,1)$ .
- ▶ Basic approach is to
  - ▶ Generate  $U(0,1)$  deviates
  - ▶ Transform to a sample from the distribution of interest.
- ▶ Generating  $U(0,1)$  deviates
  - ▶ Several random number generators (RNGs) have been developed
  - ▶ No consensus on which is “best”
  - ▶ We'll use the default in R

# Random number seed

- ▶ The current position in the random number stream is called the *seed*.
- ▶ Information on the seed is contained in the variable `.Random.seed` in your workspace.
  - ▶ If we don't set the seed, R will do it for us.
  - ▶ Each time we generate random numbers, the seed is incremented.
  - ▶ When we save the workspace, the seed is saved too.
  - ▶ **Warning:** If you start R from a previously-saved workspace, you will get the random seed. Starting multiple simulations from the same seed gives the same simulations
- ▶ You can set the seed with `set.seed()`.



# Setting the seed

```
set.seed(123)  
runif(10)
```

```
## [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565 0.5281055  
## [8] 0.8924190 0.5514350 0.4566147
```

```
set.seed(123)  
runif(10)
```

```
## [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565 0.5281055  
## [8] 0.8924190 0.5514350 0.4566147
```

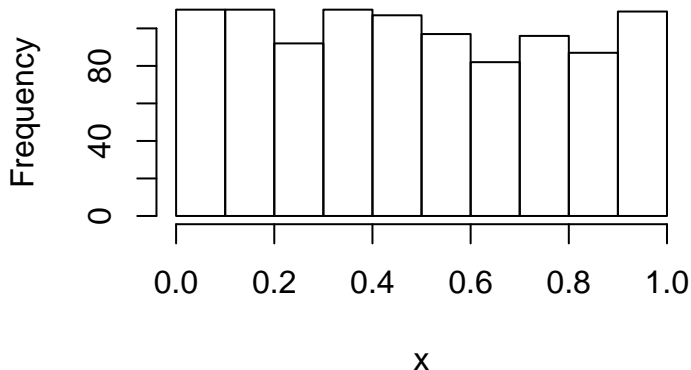
```
set.seed(42)  
runif(10)
```

```
## [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455 0.5190959 0.7365883  
## [8] 0.1346666 0.6569923 0.7050648
```

# Generating uniforms

```
Nsim <- 1000  
x <- runif(Nsim)  
hist(x)
```

**Histogram of x**



## Example: Assessing uniformity with a test

```
ks.test(x, "punif")
```

```
##  
##  One-sample Kolmogorov-Smirnov test  
##  
## data:  x  
## D = 0.036422, p-value = 0.1408  
## alternative hypothesis: two-sided
```

# Generating from other distributions

- ▶ For common distributions, we don't have to work out the transformations to convert from  $U(0, 1)$  deviates to the distribution of interest.
- ▶ Built-in functions in R have names like `rnorm()`, `rt()`, `rchisq()`

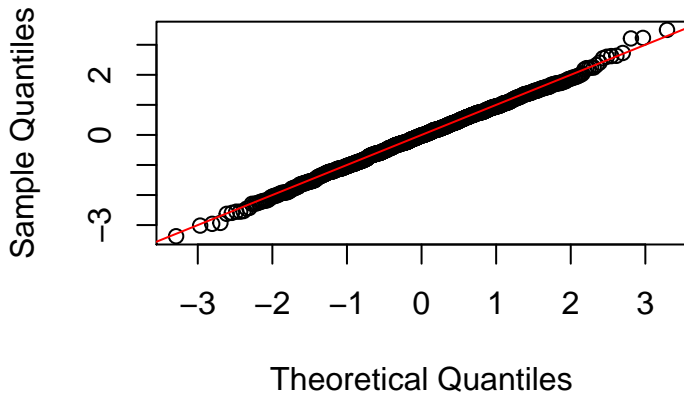
## Example: Generating from a Normal distribution

```
Nsim <- 1000; mu <- 0; sig <- 1  
x <- rnorm(Nsim, mean=mu, sd=sig)
```

## Example: Assessing normality graphically

```
qqnorm(x); abline(a=0,b=1,col="red")
```

**Normal Q-Q Plot**



## Example: Assessing normality with a test

```
ks.test(x, "pnorm")
```

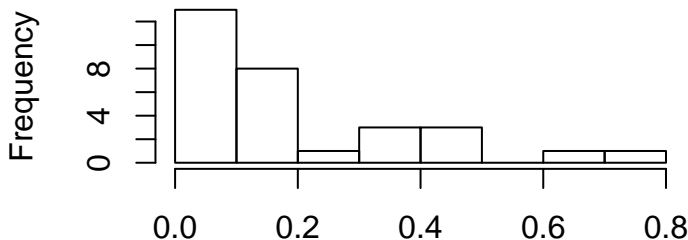
```
##  
##  One-sample Kolmogorov-Smirnov test  
##  
## data:  x  
## D = 0.019345, p-value = 0.8483  
## alternative hypothesis: two-sided
```

## Application: Demonstrate the central limit theorem (CLT)

- ▶ The CLT says that the sampling distribution of an average is approximately normal for large sample sizes, regardless of the parent population from which the sample is drawn.
- ▶ Example data-generating process: Samples of size  $n = 30$  from an exponential distribution with rate 5.

```
x <- rexp(n=30,rate=5)  
hist(x)
```

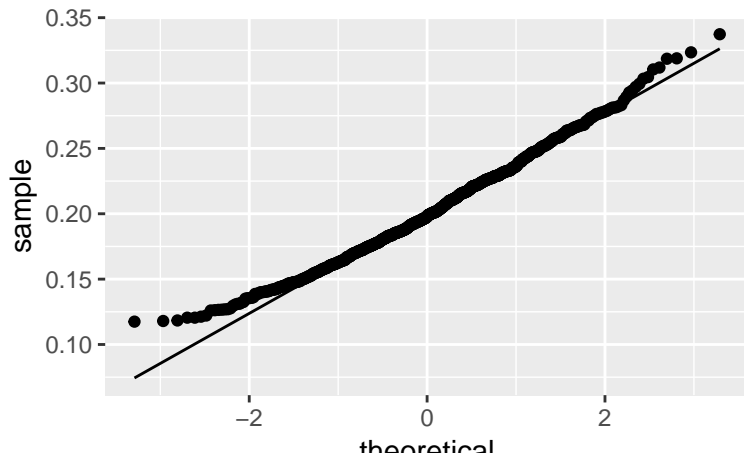
**Histogram of x**





# CLT: Sampling distribution of averages

```
library(ggplot2)
set.seed(8675309)
xbar <- vector(mode="numeric",length=Nsim)
for(i in 1:Nsim) {
  xbar[i] <- mean(rexp(n=30,rate=5))
}
ggplot(data.frame(xbar=xbar),aes(sample=xbar)) + geom_qq() + geom_qq_line()
```



# replicate()

- ▶ The replicate() function can be used for simulation.
- ▶ The following is equivalent to the for loop in the previous slide.

```
set.seed(8675309)
expMean <- function(n=30,rate=5) { mean(rexp(n=n,rate=rate)) }
xbar2 <- replicate(Nsim,expMean())
all.equal(xbar,xbar2)
```

```
## [1] TRUE
```

## Application: Simulation study

- ▶ We can conduct a simulation study to evaluate the type 1 error of the t test of the null hypothesis that a regression coefficient is zero.
  - ▶ Simulate under the null hypothesis, but with non-normal errors
- ▶ Fix
  - ▶ sample size  $n=30$
  - ▶ values of a single covariate  $x=(1:n)/n$
- ▶ Replicate the following simulation  $N_{sim}$  times
  - ▶ Simulate responses  $Y$  as  $Z - 1$  where  $Z$  has as exponential distribution with rate 1 (mean 1).
  - ▶ Fit the regression model by least squares and return the p-value from the test of association with  $x$ .
- ▶ Calculate the proportion of p-values less than 0.05.
  - ▶ Should be 5%

## Simulation code: simulating one dataset

```
n <- 30; x <- (1:n)/n
y <- rexp(n)-1
fit <- lm(y~x)
tidy(fit)
```

```
##           term      estimate std.error  statistic   p.value
## 1 (Intercept)  0.05388653  0.3368775   0.1599589  0.8740619
## 2              x -0.11807275  0.5692762  -0.2074085  0.8371919
```

```
tidy(fit) %>% filter(term=="x") %>% select(p.value)
```

```
##      p.value
## 1 0.8371919
```

```
# or
tidy(fit)[2,5]
```

```
## [1] 0.8371919
```

# Simulation study

```
Nsim <- 1000
simfunc <- function() {
  y <- rexp(n)-1
  tidy(lm(y~x))[2,5]
}
simout <- replicate(Nsim,simfunc())
alphaNominal <- 0.05
alphahat <- mean(simout<alphaNominal)
alphahat
```

```
## [1] 0.043
```

```
SE <- sqrt(alphahat*(1-alphahat)/Nsim)
c(alphahat - 1.96*SE, alphahat + 1.96*SE)
```

```
## [1] 0.03042679 0.05557321
```

## Note on simulation error

- ▶ Use the normal approximation to the mean of  $N_{sim}$  binary variables.
  - ▶ Let  $\alpha$  be the true type 1 error of the procedure, and  $\hat{\alpha}$  be our simulation-based estimate
  - ▶  $\hat{\alpha}$  is a mean and can be shown to be approximately Normal with mean  $\alpha$  and SD  $\sqrt{\alpha(1 - \alpha)/N_{sim}}$ .
  - ▶ SE is obtained by plugging in  $\hat{\alpha}$
- ▶ Can define simulation error as the margin of error in a 95% confidence interval for the parameter being estimated.
  - ▶ In our study, the CI covers 0.05, so we say that the empirical type 1 error is within simulation error of the nominal level.

# Sampling with and without replacement

- ▶ Draw probability weighted samples of size  $n$  from a set with `sample()`
  - ▶ Sampling can be with replacement or without.

```
myset <- 1:10; nset <- length(myset)
probwts <- rep(1/nset,nset)
n<-8
sample(myset,size = n, replace = TRUE, prob=probwts)
```

```
## [1] 6 3 4 2 8 5 2 2
```

```
sample(myset,size = n, replace = FALSE, prob=probwts)
```

```
## [1] 9 6 1 4 8 7 2 10
```

# Notes on sampling with and without replacement

- ▶ Weights need not sum to one
  - ▶ They will be normalized
- ▶ If sampling with replacement, the size of the sample can't exceed the size of the set.
- ▶ Sampling  $n$  without replacement from a set of size  $n$  is a *permutation* of the set.
  - ▶ Using all the defaults of `sample()` gives a permutation.

```
sample(1:10)
```

```
## [1] 9 6 10 5 1 3 2 4 7 8
```

```
sample(c("cat", "dog", "fish", "zebra"))
```

```
## [1] "zebra" "dog" "fish" "cat"
```



## Permutation tests

## Example

- ▶ Makes most sense in the context of a designed experiment.
- ▶ Example experiment:
  - ▶ 36 cars were randomly assigned to receive one of two types of car muffler.
  - ▶ The noise in decibels of each car was measured.
  - ▶ Is there a difference in sound level between the two mufflers?
- ▶ A standard ANOVA for these data is as follows.

```
mfit <- lm(Noise ~ Type, data=AutoPollution)
tidy(anova(mfit)) # F stat is 1,5 element
```

##	term	df	sumsq	meansq	statistic	p.value
## 1	Type	1	1056.25	1056.2500	1.246181	0.2721107
## 2	Residuals	34	28818.06	847.5899	NA	NA

```
Fstat <- tidy(anova(mfit))[1,5]
```

# Permutation distribution of F

- ▶ Under the null hypothesis the noise of cars doesn't depend on the muffler type, we just have 36 cars assigned randomly to two groups.
  - ▶ Cars are said to be *exchangeable* (with respect to noise)
  - ▶ The distribution of the F statistic under all possible re-randomizations is the randomization distribution.
- ▶ Randomization can be achieved by permuting the Noise variable.
  - ▶ The randomization distribution can also be called a permutation distribution and the test that compares the observed F to this distribution a permutation test.

# Permutation test on example

```
fstatPerm <- function() {  
  permdat <- data.frame(Type = AutoPollution$Type,  
                        Noise = sample(AutoPollution$Noise))  
  mm <- lm(Noise ~ Type, data=permdat)  
  tidy(anova(mm))[1,5]  
}  
set.seed(1)  
pdist <- replicate(1000,fstatPerm())  
mean(pdist > Fstat) # permutation p-value
```

```
## [1] 0.276
```

# Exchangeability is key assumption

- ▶ Responses must be exchangeable under the null hypothesis of no treatment effect
- ▶ Cloud seeding data
  - ▶ Clouds randomly seeded (S) or not (U)
  - ▶ Clouds from several areas of Tasmania; focus on Tasmania East (TE).

```
CloudSeeding %>% group_by(Seeded) %>%  
  summarize(n=n(),mean=mean(TE),sd=sd(TE))
```

```
## # A tibble: 2 x 4  
##   Seeded     n  mean    sd  
##   <fct> <int> <dbl> <dbl>  
## 1 S      14  1.09 0.671  
## 2 U      14  1.60 1.63
```

- ▶ Unseeded clouds are more variable in rainfall, so they are not exchangeable.