Chapter 8 Simulation to Test Procedures

How do we know if a procedure "works" properly?

- Does a test give the right p-values?
- Do confidence intervals have the right coverage rates?

A typical stat article claims some new procedure is better than an old one. Arguments:

- likelihood based – often using MLE's are asymptotically normal. BUT how big does n need to be to use this argument?
- simulation based – create fake data based on an assumed parameter value.
  - Show that the procedure estimates the parameter and gives good coverage. (Repeat process many times. Examine the proportion of parameters covered by their estimating intervals.)
  - May use different n to see how it works for small sample sizes as well as large ones.

Simulation to Predict

Is our model capable of producing data like ours?

A powerful tool, needs some skill and practice for application
Not a "standard" procedure (unlike the fake data simulation).

8.1 SLR

Assume we have model $y_i = \alpha + \beta x_i + \epsilon_i$
Create fake data, check coverage rates of CI's for β.

1. Arbitrary "true" values:
   $\alpha = 1.4, \quad \beta = 2.3, \quad \sigma = 0.9, \quad \mathbf{x} = (1\ 2\ 3\ 4\ 5)^T$

```
alpha <- 1.4
beta <- 2.3
sigma <- 0.9
x <- 1:5
n <- length(x)
```

2. Generate data.

```
y <- alpha + beta * x + rnorm(n, 0, sigma)
```

3. Forget the parameters, and estimate them.

	Estimate	Std. Error	t value
(Intercept)	-0.63	0.29	-2.17
x	2.76	0.09	31.63

Table: n = 5 rank = 2 resid sd = 0.276 R-Squared = 0.997

Build a CI and Check It

```
b.hat <- coef(lm1)[2]
b.se <- se.coef(lm1)[2]
cover.68 <- abs(beta - b.hat) < b.se  # this will be TRUE or FALSE
cover.95 <- abs(beta - b.hat) < 2 * b.se  # this will be TRUE or FALSE
cat(paste("68% coverage: ", cover.68, "\n"))

## 68% coverage:  FALSE

cat(paste("95% coverage: ", cover.95, "\n"))

## 95% coverage:  FALSE
```

It worked once. Need long run coverage.

Long Run Coverage (Normal)

```
n.fake <- 1000
cover.68 <- cover.95 <- rep(NA, n.fake)
for (s in 1:n.fake) {
    y <- alpha + beta * x + rnorm(n, 0, sigma)
    lm.1 <- lm(y ~ x)
    b.hat <- coef(lm.1)[2]
    b.se <- se.coef(lm.1)[2]
    cover.68[s] <- abs(beta - b.hat) < b.se
    cover.95[s] <- abs(beta - b.hat) < 2 * b.se
}
cat(paste("68% coverage: ", mean(cover.68), "\n"))

## 68% coverage:  0.639

cat(paste("95% coverage: ", mean(cover.95), "\n"))

## 95% coverage:  0.88
```


Long Run Coverage (t_3)

```
for (s in 1:n.fake) {
    y <- alpha + beta * x + rnorm(n, 0, sigma)
    lm.1 <- lm(y ~ x)
    b.hat <- coef(lm.1)[2]
    b.se <- se.coef(lm.1)[2]
    cover.68[s] <- abs(beta - b.hat) < qt(0.84, n - 2) *
        b.se
    cover.95[s] <- abs(beta - b.hat) < qt(0.975, n - 2) *
        b.se
}
cat(paste("68% coverage: ", mean(cover.68), "\n"))

## 68% coverage:  0.652

cat(paste("95% coverage: ", mean(cover.95), "\n"))

## 95% coverage:  0.948
```


Fake Data Residuals Sim

```
midterm <- grades[, "Midterm"]
final <- grades[, "Final"]
display.xtable(lm.1 <- lm(final ~ midterm))
```

	Estimate	Std. Error	t value
(Intercept)	64.50	16.98	3.80
midterm	0.70	0.21	3.28

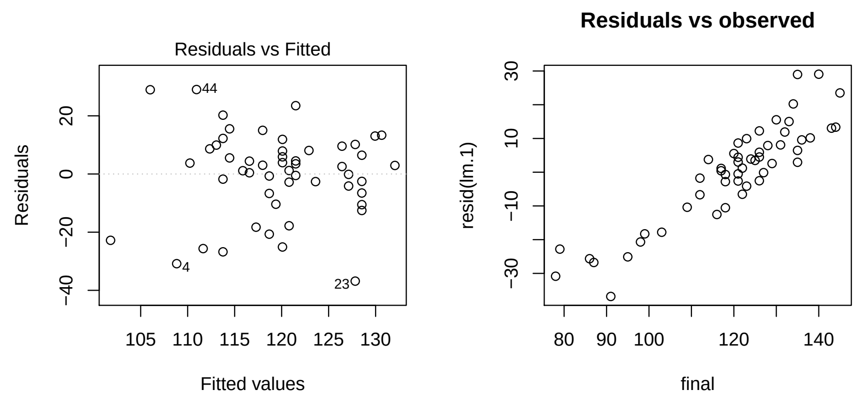
Table: n = 52 rank = 2 resid sd = 14.752 R-Squared = 0.177

```
X <- cbind(1, midterm)
n <- length(final)
```

Nothing simulated yet.

Residual Relationships

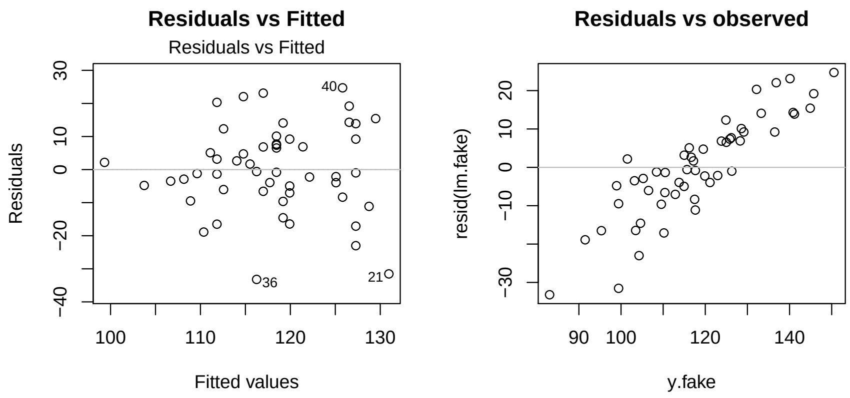
```
par(mfrow = c(1, 2))
plot(lm.1, which = 1, add.smooth = F)
plot(final, resid(lm.1), main = "Residuals vs observed")
```



Residuals are orthogonal to Fits, not to Observed.
Make a demo to convince us that's right.

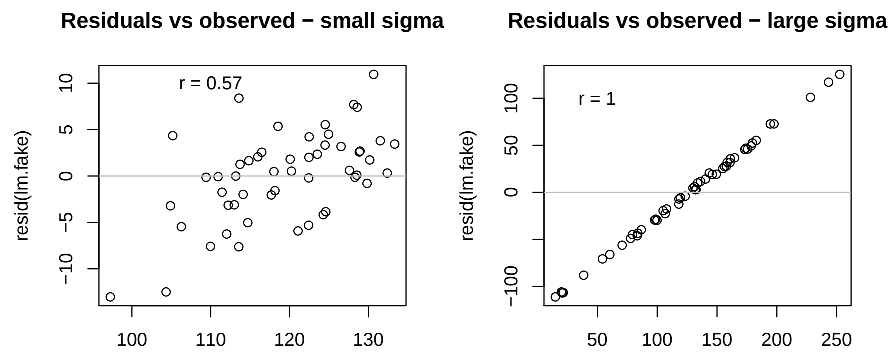
Residuals Relationships 2

```
a <- 65;  b <- 0.7;  sigma <- 15; par(mfrow=c(1,2))
y.fake <- a + b*midterm + rnorm (n, 0, sigma)
lm.fake <- lm (y.fake ~ midterm)
plot(lm.fake,which=1, main="Residuals vs Fitted", add.smooth=F,sub="")
abline(h=0, col="grey")
plot(y.fake, resid(lm.fake),  main="Residuals vs observed")
abline(h=0, col="grey") ## Pattern seen is not a failure of the model.
```



Residuals Relationships 3

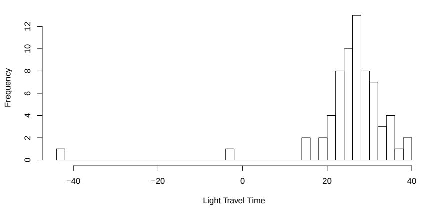
```
par(mfrow=c(1,2))
y.fake <- a + b*midterm + rnorm (n, 0, 5)
lm.fake <- lm (y.fake ~ midterm)
plot(y.fake, resid(lm.fake),  main="Residuals vs observed - small sigma")
abline(h=0, col="grey");text(110,10, paste("r =",round(cor(y.fake, resid(lm.fak
y.fake <- a + b*midterm + rnorm (n, 0, 50)
lm.fake <- lm (y.fake ~ midterm)
plot(y.fake, resid(lm.fake),  main="Residuals vs observed - large sigma")
abline(h=0, col="grey"); text(50,100, paste("r =",round(cor(y.fake, resid(lm.fa
```



§8.3 Simulated Data Compared to Actual Data

Speed of light data (Newcomb 1882) has some outliers.

```
hist((light$time - 24.8) * 1000, breaks = 35, xlab = "Light Travel Time
    main = "")
```



Fit a mean assuming (poor idea) normality.

```
display.xtable(light.fit <- lm(I((time - 24.8) * 1000) ~
    1, light))
```

	Estimate	Std. Error	t value
(Intercept)	26.21	1.32	19.82

Simulate from Model

Generate 1000 random coefficient estimates using sampling dist'n of $(\widehat{\beta}^T \quad \widehat{\sigma})$

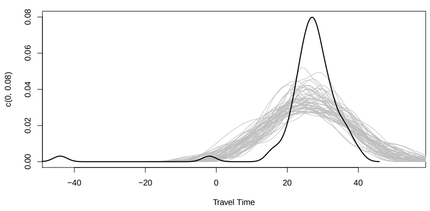
```
sim.light <- sim(light.fit, 1000)
n <- nrow(light)
```

For each simulated coefficient, generate random $\mathbf{y}_{rep}$ and plot 50 of them, comparing to the actual data.

```
y.rep <- sapply(1:1000, function(ndx) rnorm(n, sim.light@coef[ndx],
    sim.light@sigma[ndx]))
```

Plot Sims versus Actual

```
plot(c(-45, 55), c(0, 0.08), type = "n", xlab = "Travel Time",
    main = "")
plot.myDensity <- function(y, ...) {
    dense <- density(y)
    lines(dense$x, dense$y, ...)
}
for (s in 1:50) plot.myDensity(y.rep[, s], col = "grey")
plot.myDensity((light$time - 24.8) * 1000, lwd = 2)
```

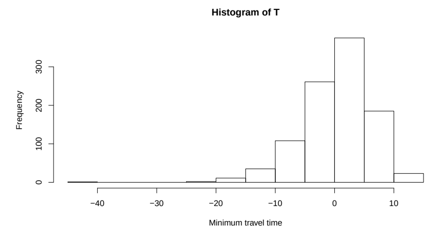


Grey lines don't ever have the two bumps, otherwise they have too much spread.

Numerical Summaries

With simple data, plot shows the observed data are quite different from simulated data. More of a challenge with complex data. Idea: find a numerical summary $T(\mathbf{y})$ which highlights some aspect of the data. Compute it on each y_{rep}. In this case, the min is a good candidate.

```
Test <- function(y) min(y)
test.rep <- apply(y.rep, 2, Test)
hist(c(min((light$time - 24.8) * 1000), test.rep), main = "Histogram of
    xlab = "Minimum travel time")
```



Conclusion of Newcomb's Data

Density plots and numerical summaries agree:
Newcomb's data don't come from a normal distribution. Could be a contaminated mixture, or a long-tailed, or ...

Count Data Example

Model number of roaches caught as $y_i \sim \text{Poisson}(u_i \exp(X_i\boldsymbol{\beta}))$ where $\mathbf{X}$ has predictors: pre-treatment roach level an indicator for treatment and an indicator for "senior" (building restricted to elderly) and an intercept. Offset u_i is "number of trap days" and $\log(u_i)$ acts like another predictor with coefficient set to be one.

```
roach.glm1 <- glm(y ~ roach1 + treatment + senior, roaches,
    family = poisson, offset = log(exposure2))
display.xtable(roach.glm1)
```

	Estimate	Std. Error	z value
(Intercept)	3.09	0.02	145.49
roach1	0.01	0.00	78.69
treatment	-0.52	0.02	-20.89
senior	-0.38	0.03	-11.37

Table: n = 262 rank = 4 Resid Deviance = 11429.467

Count Data Example: Random Poisson

Generate one set of random Poissons from this model.

```
X <- model.matrix(roach.glm1)
y.hat1 <- roaches$exposure2 * exp(X %*% coef(roach.glm1))
y.rep1 <- rpois(nrow(roaches), y.hat1)
xtable(matrix(table(roaches$y)[1:10], 1, 10, dimnames = lis
    0:9)))
```

	0	1	2	3	4	5	6	7	8	9
1	94	20	11	10	7	7	3	6	3	2

```
xtable(matrix(table(y.rep1)[1:10], 1, 10, dimnames = list(N
    names(table(y.rep1))[1:10])))
```

	0	4	5	6	7	8	9	10	11	12
1	1	1	2	6	4	10	14	18	10	13

Data has way more zeroes than random Poisson.

Count Data Example Simulation

Generate 1000 random Poissons from this model.

```
roach.sim1 <- sim(roach.glm1, 1000)
y.rep <- sapply(1:1000, function(ndx) rpois(nrow(roaches),
    roaches$exposure2 * exp(X %*% roach.sim1@coef[ndx, ])))
Test <- function(y) mean(y == 0)
Test(roaches$y)

## [1] 0.359

summary(test.rep <- apply(y.rep, 2, Test))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.00000 0.00064 0.00000 0.00763
```

How unusual is the real data? Compute a p-value.

```
mean(test.rep >= Test(roaches$y))

## [1] 0
```

Too few zeroes. One fix: add in overdispersion.

Overdispersed Poisson

```
roach.glm2 <- update(roach.glm1, family = quasipoisson)
display.xtable(roach.glm2)
```

	Estimate	Std. Error	t value
(Intercept)	3.09	0.17	17.98
roach1	0.01	0.00	9.73
treatment	-0.52	0.20	-2.58
senior	-0.38	0.27	-1.41

Table: n = 262 rank = 4 Resid Deviance = 11429.467

Overdispersed Poisson Simulated Data

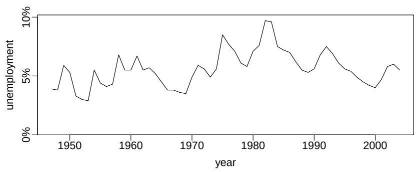
```
roach.sim2 <- sim(roach.glm2, 1000)
y.hat <- sapply(1:1000, function(ndx) roaches$exposure2 *
    exp(X %*% roach.sim2@coef[ndx, ]))
y.rep <- sapply(1:1000, function(ndx) rnegbin(nrow(roaches),
    y.hat[, ndx], y.hat[, ndx]/(65.4 - 1)))
summary(test.rep <- apply(y.rep, 2, Test))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.168   0.290   0.317   0.319   0.351   0.469
```

Is 35% zeroes unusual?

Autocorrelated Data

To use OLS regression, errors must be independent. With time series data, it's common to find that each points is correlated with its neighboring points. Positive AR: highs follow highs, lows follow lows.

```
year <- unemploy$year
y <- unemploy$unemployed.pct
plot(year, y, type = "l", ylab = "unemployment", xlab = "year",
    yaxs = "i", ylim = c(0, max(y) * 1.05), yaxt = "n",
    mgp = c(2, 0.5, 0), cex.axis = 1.2, cex.lab = 1.2)
axis(2, c(0, 5, 10), paste(c(0, 5, 10), "%", sep = ""),
    mgp = c(2, 0.5, 0), cex.axis = 1.2)
```



Autocorrelated Data Fit

Create "lagged *y*" to predict current year from previous year.

```
y.lag <- c(NA, y)[1:58]
lm.lag <- lm(y ~ y.lag)
display.xtable(lm.lag)
```

	Estimate	Std. Error	t value
(Intercept)	1.43	0.50	2.84
y.lag	0.75	0.09	8.61

Table: n = 57 rank = 2 resid sd = 0.986 R-Squared = 0.574

Is this a good fit? Simulate to see.

Simulate Autocorrelated Data

```
b.hat <- coef(lm.lag)  # vector of 2 regression coefs
s.hat <- sigma.hat(lm.lag)  # residual sd
n.sims <- 1000
y.rep <- array(NA, c(n.sims, n <- length(y)))
for (s in 1:n.sims) {
    y.rep[s, 1] <- y[1]
    for (t in 2:n) {
        prediction <- c(1, y.rep[s, t - 1]) %*% b.hat
        y.rep[s, t] <- rnorm(1, prediction, s.hat)
    }
}
```


Simulate Autocorrelated Data 2

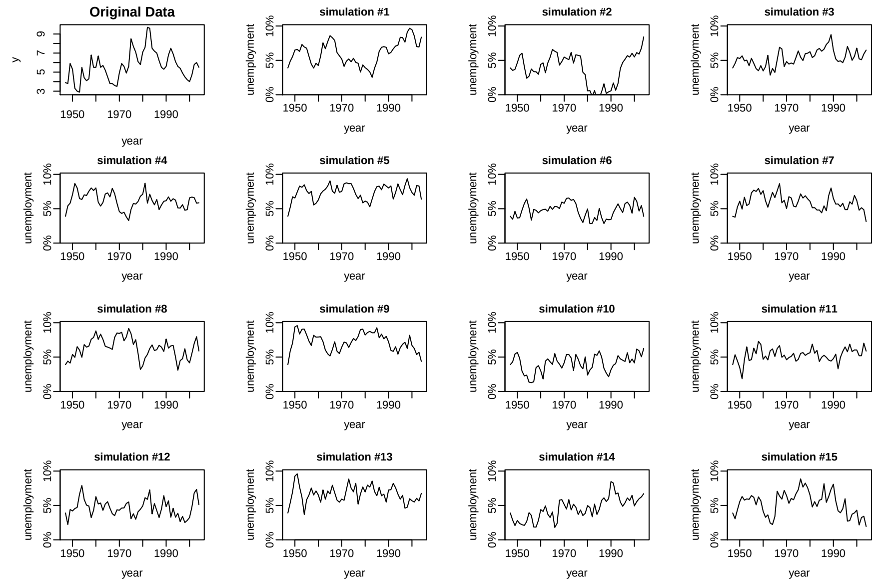
```
## Including uncertainty in the estimated parameters
lm.lag.sim <- sim(lm.lag, n.sims)  # simulations of beta and sigma
for (s in 1:n.sims) {
    y.rep[s, 1] <- y[1]
    for (t in 2:n) {
        prediction <- c(1, y.rep[s, t - 1]) %*% lm.lag.sim@coef[s,
            ]
        y.rep[s, t] <- rnorm(1, prediction, lm.lag.sim@sigma[s])
    }
}
```

Simulate Autocorrelated Data 3

Plot of simulated unemployment rate series

```
par(mfrow = c(4, 4), mar = c(4, 4, 2, 2))
plot(year, y, type = "l", main = "Original Data")
for (s in 1:15) {
    plot(year, y.rep[s, ], type = "l", ylab = "unemployment",
        xlab = "year", yaxs = "i", ylim = c(0, max(y) *
            1.05), yaxt = "n", mgp = c(2, 0.5, 0), main = paste("simula
            s, sep = ""), cex.main = 0.95)
    axis(2, c(0, 5, 10), paste(c(0, 5, 10), "%", sep = ""),
        mgp = c(2, 0.5, 0))
}
```


Plots of Simulated Autocorrelated Data



Simulate Autocorrelated Data 4

```
Test <- function(y) {
    ## count # of switches in sign of difference
    n <- length(y)
    y.lag <- c(NA, y)[1:58]
    y.lag2 <- c(NA, NA, y)[1:58]
    sum(sign(y - y.lag) != sign(y.lag - y.lag2), na.rm = TRUE)
}
## Test (y) ## 23 switches in original data
n.sims <- 1000
for (s in 1:n.sims) {
    test.rep[s] <- Test(y.rep[s, ])
}
mean(test.rep > Test(y))

## [1] 0.972

quantile(test.rep, c(0.025, 0.05, 0.5, 0.95, 0.975))

##  2.5%    5%   50%   95% 97.5%
##    23    24    31    36    38
```