Date Science and Applications with R

Simulation methods

Peng Zhang

School of Mathematical Sciences, Zhejiang University

2020/07/12

Agenda

- Writing simulations using random number generation
- From the jackknife to the bootstrap
- Processes that have memory
- Finally, `for` loops!

`[dpqr]unif`, etc.

Recall how we drew from various distributions: `runif()`: draws from the uniform distribution (others follow)

Home-baked discrete distributions: Use `sample()`

```
population.values <- 1:3
probs <- c(5,2,3)
my.draw <- sample (population.values, 100000, probs, replace=TRUE)
table(my.draw)
```

```
## my.draw
##     1     2     3
## 49889 20162 29949
```

Permutations with `sample()`

`sample()` is powerful -- it works on any object that has a defined `length()`. Permutations:

```
sample(1:6)
```

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

```
replicate(3,sample(c("Curly","Larry","Moe","Shemp")))
```

```
##      [,1]    [,2]    [,3]
## [1,] "Shemp" "Moe"   "Shemp"
## [2,] "Moe"   "Larry" "Curly"
## [3,] "Larry" "Shemp" "Moe"
## [4,] "Curly" "Curly" "Larry"
```

```
sample(list("A",3,sum))
```

```
## [[1]]
## [1] "A"
##
## [[2]]
## function (..., na.rm = FALSE)  .Primitive("sum")
```

Resampling with `sample()`

Resampling from any existing distribution gets us the **bootstrap** family of estimators:

```
bootstrap.resample <- function (object){
  sample (object, length(object), replace=TRUE)}
replicate(5, bootstrap.resample (6:10))
```

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

Recall: the *jackknife* worked by removing one point from the sample and recalculating the statistic of interest. Here we resample the same length with replacement.

Bootstrap test

The 2-sample `t`-test checks for differences in means according to a known null distribution. Let's resample and generate the sampling distribution under the bootstrap assumption:

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
diff.in.means <- function(df) {
  mean(df[df$Sex=="M","Hwt"]) - mean(df[df$Sex=="F","Hwt"])
}
resample.diffs <- replicate(1000,
  diff.in.means(cats[bootstrap.resample(1:nrow(cats)),]))
```

```
tibble(d = resample.diffs) %>% ggplot(mapping = aes(x = d))+
  geom_histogram(bins = 25)+
  geom_vline(aes(xintercept = diff.in.means(cats)), col='red')
```

Dependence

Most interesting examples in probability have a little dependence added in: "If it rained yesterday, what is the probability it rains today?"

Use this to generate weather patterns and probabilities for some time in the future. Almost always occurs with time series; can occur with other dependence (spatial -- if it's sunny in Dallas today, will it also be sunny in Fort Worth?)

Markov Dependence

Suppose we have a sequence of observations that are dependent. In a time series, what happens next depends on what happened before:

$$p(X_1, X_2, \ldots, X_n) = p(X_1)p(X_2|X_1)\ldots p(X_n|X_{n-1}, \ldots, X_1)$$

(Note: you could, of course, condition on the future to predict the past, if you had a time machine.)

Markov dependence: each outcome only depends on the one that came before.

$$p(X_1, X_2, \ldots, X_n) = p(X_1)\prod_{i=2}^{n} p(X_i|X_{i-1})$$

Generating a Markov Chain

1. Set up the conditional distribution.
2. Draw the initial state of the chain.
3. For every additional draw, use the previous draw to inform the new one.

Simple weather model:

- if it was sunny yesterday, today's chance of sun is 80%.
- if it wasn't sunny yesterday, today's chance of sun is 20%.

Simulate for one year (at the equator?)

```
sunny.year <- rep(NA, 365)
sunny.year[1] <- 1
for (day in 2:365) {
  sunny.year[day] <- rbinom(1,1,0.2 + 0.6*sunny.year[day-1])}
```

And `for` loops are back in play!

```
tibble(x = 1:365, y = sunny.year) %>% ggplot(aes(x = x, y = y))+
  geom_line()+
  labs(title="Sunny Days in An Equatorial City", x="Day", y="Sunshi
```

Different From Independence

```
boring.year <- tibble(day = 1:365, rain = rbinom(365, 1, 0.5))
boring.year %>% ggplot(aes(x = day, y = rain))+
  geom_line()+
  labs(title="Sunny Days in A Coin Flip City", x="Day", ylab="Sunsh
```

The above chain

Transitions are represented as a matrix: Q_{ij} is $P(X_t = j | X_{t-1} = i)$.

```
##      [,1] [,2]
## [1,]  0.8  0.2
## [2,]  0.2  0.8
```

Flippy chain

```r
weird.year <- rep(NA, 365)
weird.year[1] <- 1
transition.matrix <- matrix (c(0.2, 0.8, 0.8, 0.2), nrow=2)
for (day in 2:365) weird.year[day] <-
  sample(1:2, 1, prob=transition.matrix[weird.year[day-1],])
```

```
tibble(day = 1:365, rain = weird.year) %>%
ggplot(aes(x = day, y = rain))+
  geom_line()+
  labs(title="Sunny Days in Al Gore's Nightmare", x="Day", y="Sunsh
```

General Markov Chain

```
rmarkovchain <- function (nn, transition.matrix,
    start=sample(1:nrow(transition.matrix), 1)) {
  output <- rep (NA, nn)
  output[1] <- start
  for (day in 2:nn) output[day] <-
    sample(ncol(transition.matrix), 1,
           prob=transition.matrix[output[day-1],])
}
```

Simple Unbounded Markov Chain

"Unbiased Random Walk": Independent events atop a dependent structure.

```
randomwalk <- function (nn, upprob=0.5, start=50) {
  output <- rep (NA, nn)
  output[1] <- start
  for (iteration in 2:nn)
    output[iteration] <-
      output[iteration-1] + 2*rbinom(1, 1, upprob) - 1
  output
}
```

```r
tibble(x = 1:10000, y = randomwalk(10000, start=200)) %>%
  ggplot(aes(x, y))+ geom_line()+
  labs(title="Simple Random Walk")
```

Simple Unbounded Markov Chain

```
tibble(x = 1:10000, y = randomwalk(10000, start=200)) %>%
  ggplot(aes(x, y))+ geom_line()+
  labs(title="Simple Random Walk")
```

Simple Unbounded Markov Chain

```
tibble(x = 1:10000, y = randomwalk(10000, start=200)) %>%
  ggplot(aes(x, y))+ geom_line()+
  labs(title="Simple Random Walk")
```

Simple Unbounded Markov Chain

```
tibble(x = 1:10000, y = randomwalk(10000, start=200)) %>%
  ggplot(aes(x, y))+ geom_line()+
  labs(title="Simple Random Walk")
```

What's available?

Integration by Simulation

Law of large numbers: if $X_1, X_2, \ldots X_n$ all IID with p.d.f. p,

$$\frac{1}{n}\sum_{i=1}^{n} f(X_i) \to \mathbb{E}_p[f(X)] = \int f(x)p(x)dx$$

The **Monte Carlo principle**: to find $\int g(x)dx$, draw from p and take the sample mean of $f(x) = g(x)/p(x)$.

And it works, too

Central limit theorem:

$$\frac{1}{n}\sum_{i=1}^{n} \frac{g(x_i)}{p(x_i)} \rightsquigarrow \mathcal{N}\left(\int g(x)dx, \frac{\sigma^2_{g/p}}{n}\right)$$

The Monte Carlo approximation to the integral is unbiased, with root mean square error $\propto n^{-1/2}$ -- just keep taking Monte Carlo draws, and the error gets as small as you like, even if g or x are very complicated.

Taking unknown samples

Generating from p is easy if it's a standard distribution or we have a nice, invertible CDF (quantile method). What can we do if all we've got is the probability density function p?

Suppose the pdf f is zero outside an interval $[c, d]$, and $\leq M$ on the interval.

Rejection sampling

We know how to draw from uniform distributions in any dimension. Do it in two:

```
x1 <- runif(300, 0, 1); y1 <- runif(300, 0, 2.6);
selected <- y1 < dbeta(x1, 3, 6)
```

Rejection sampling

```
mean(selected)
```

```
## [1] 0.4166667
```

```
accepted.points <- x1[selected]
mean(accepted.points < 0.5)
```

```
## [1] 0.784
```

```
pbeta(0.5, 3, 6)
```

```
## [1] 0.8554688
```

For this to work efficiently: we have to cover the target distribution with one that sits close to it.

```
x2 <- runif(100000, 0, 1); y2 <- runif(100000, 0, 10);
selected <- y2 < dbeta(x2, 3, 6)
mean(selected)
```

```
## [1] 0.09915
```

Rejection sampling

Metropolis Algorithm

(Really: Metropolis, Rosenbluth x2, Teller x2, 1953)

Rejection sampling works if we know the complete distribution $p(x)$, with $\int p(x) = 1$. For many, many problems, we know the function up to the normalizing constant.

Example: Equations of the canonical ensemble of energy formations in the state of uncontrolled hydrogen fusion,

$$p(E) = \frac{1}{|Z|} exp(f(E)/kT)$$

$|Z|$ is legendarily hard to calculate precisely.

Basic Metropolis

On integer steps:

```
metropolis.one <- function (xx, fun=dpois, ...) {
  prop <- 2*rbinom(1,1,0.5) - 1 + xx
  acc.ratio <- fun(prop, ...)/fun(xx, ...)
  output <- if (acc.ratio > runif(1)) prop else xx
  output
}
replicate (10, metropolis.one(10, lambda=5))
```

```
##  [1] 11 11  9  9  9  9  9 10 11  9
```

```
start <- 50
draws <- rep(NA, 10000)
for (iteration in 1:10000) {
  start <- metropolis.one (start, lambda=10)
  draws[iteration] <- start
}
```

Basic Metropolis

```
tibble(n = 1:10000, draws = draws) %>%
  ggplot(aes(n, draws))+ geom_point()
```

Basic Metropolis

Have to discard the initial state for "burn-in"

```
tibble(n = 101:10000, draws = draws[-(1:100)]) %>%
  ggplot(aes(n, draws))+ geom_point()
```

Basic Metropolis

Have to discard the initial state for "burn-in"

```
tibble(draws = draws[-(1:100)], poir = rpois(9900, 10)) %>%
  ggplot()+
  geom_histogram(aes(x = draws), bins = 20, alpha = 0.6)+
  geom_histogram(aes(x = poir), bins = 20, col='red', alpha = 0.6)
```

Basic Metropolis

On integer steps:

```
metropolis.cts <- function (xx, fun=dgamma, ...) {
  prop <- rnorm(1, xx, 1)
  acc.ratio <- fun(prop, ...)/fun(xx, ...)
  output <- if (acc.ratio > runif(1)) prop else xx
  output
}
replicate (20, metropolis.cts(20, shape=100, rate=10))
```

```
##  [1] 20.00000 20.00000 20.00000 18.63705 20.00000 20.00000 20.00000 20.00
##  [9] 20.29867 18.79457 19.20427 18.99566 18.59367 19.81806 20.00000 20.00
## [17] 19.04624 18.16218 20.00000 20.00000
```

```
start <- 50
draws <- rep(NA, 100000)
for (iteration in 1:100000) {
  start <- metropolis.one (start, fun=dgamma, shape=100, rate=10)
  draws[iteration] <- start
}
```

Basic Metropolis

```
tibble(n = 1:100000, draws = draws) %>%
  ggplot(aes(n, draws))+ geom_point()
```

Basic Metropolis

Have to discard the initial state for "burn-in"

```
tibble(n = 201:100000, draws = draws[-(1:200)]) %>%
  ggplot(aes(n, draws))+ geom_point()
```

Basic Metropolis

Have to discard the initial state for "burn-in"

```
tibble(draws = draws[-(1:200)], gamr = rgamma(99800, 100, 10)) %>%
  ggplot()+
  geom_histogram(aes(x = draws), bins = 20, alpha = 0.6)+
  geom_histogram(aes(x = gamr), bins = 20, col='red', alpha = 0.6)
```

Example: Prediction Intervals

Stock returns of the Corrections Corp. of America:

```
# Corrections Corp. of America
CXW <- pdfetch_YAHOO("CXW",fields="adjclose",
  from=as.Date("2000-01-01",
  to=as.Date("2020-06-20")))
CXW.returns <- na.omit(as.vector(diff(log(CXW$CXW))))
```

Prediction Intervals

Regress tomorrow's returns on today's:

```
CXW.ar <- lm(CXW.returns[-1] ~ CXW.returns[-length(CXW.returns)])
coefficients(CXW.ar)
```

```
##                        (Intercept) CXW.returns[-length(CXW.returns)]
##                       0.0002357051                      -0.0642756879
```

Example: Prediction Intervals

```
CXWrf <- tibble(day = 1:(length(CXW.returns)-1), obs = CXW.returns[
  fitted = fitted(CXW.ar))
CXWrf %>%  ggplot() + geom_point(aes(x = day, y = obs))+
  geom_point(aes(x = day, y = fitted), col="blue")
```

Example: Prediction Intervals

Could work out prediction intervals from theory, if they were Gaussian ... or just simulate

```
prediction.interval <- function(model, confidence=0.95, n=1e3) {
  residuals <- residuals(model)
  sim <- sample(residuals, size=n, replace=TRUE)
  lower.limit <- (1-confidence)/2
  upper.limit <- 1-lower.limit
  return(quantile(sim, c(lower.limit, upper.limit)))
}
(CXW.interval95 <- prediction.interval(CXW.ar))
```

```
##        2.5%       97.5%
## -0.05092949  0.06062655
```

Example: Prediction Intervals

```
CXWrf %>% mutate(lowbound = fitted + CXW.interval95[1],
  upbound = fitted + CXW.interval95[2])  %>% ggplot() +
  geom_point(aes(x = day, y = obs))+
  geom_point(aes(x = day, y = fitted), col="blue")+
  geom_line(aes(x = day, y = lowbound), linetype= 2, col="blue")+
  geom_line(aes(x = day, y = upbound), linetype= 2, col="blue")
```

Simulations as Models

- Sometimes the only convenient way to specify the statistical model is as a simulation
  - Lots of details, no simplifying math
- Running the simulation is then how we see what the model does at given parameters
- Fit by matching simulation output to data

Example: Antibiotics Again

- Doctors have either adopted or they haven't
- Every day, two random doctors meet
- If one has adopted but the other hasn't, the hold-out adopts with probability p
- Look at number of adoptions over time

This was originally a model of disease spread, but now the "disease" is adopting a new drug

Example: Antibiotics Again

Code written by section 1:

```
sim_doctors_1 <- function(num.doctors, num.days, initial_doctors, p
  # Remember to set up all_doctors
  all_doctors <- 1:num.doctors
  # Remember to set up has_adopted as binary vector
  has_adopted <- matrix(0,nrow=num.doctors,ncol=num.days)
  # Set some doctors to have initially adopted
    # initial_doctors are indices of doctors who are using as of da
  has_adopted[initial_doctors,1:num.days] <- 1
  for (today in 1:num.days) {
    # pull two random doctors
    todays_doctors <- sample(all_doctors,size=2,replace=FALSE)
    # check that one has adopted and the other hasn't
    if(sum(has_adopted[todays_doctors,today])==1) {
      # make the non-adopter adopt with probability p
      which_of_todays <- which(has_adopted[todays_doctors,today]==0
      receiver <- todays_doctors[which_of_todays]
      has_adopted[receiver,today:num.days] <- rbinom(n=1,size=1,prob
    }
  }
  return(has_adopted)
}
```

Example: Antibiotics Again

Code written by section 2:

```
sim_doctors_2 <- function(num.doctors,num.meetings,starting_adopter
  # vector to keep track of which doctors have adopted
  has_adopted <- rep(FALSE, num.doctors)
  # Set some to have adopted initially
  has_adopted[1:round(starting_adopters*num.doctors)] <- TRUE
  # matrix to keep track of adoptions over time
  adoptions_vs_time <- matrix(FALSE,nrow=num.doctors,ncol=num.meeti
  for (meeting in 1:num.meetings) {
    # select 2 doctors at random
    meeting_pair <- sample(1:num.doctors,size=2)
    # check whether exactly one selected doctor has adopted
    if(has_adopted[meeting_pair[1]] != has_adopted[meeting_pair[2]]
      # With probability prob., update the vector of adopters
      if(rbinom(n=1,size=1,prob=prob)) { has_adopted[meeting_pair]
    }
    # update adoptions_vs_time matrix
    adoptions_vs_time[,meeting] <- has_adopted
  }
  return(adoptions_vs_time)
}
```

Example: Antibiotics Again

```
sim2 <- sim_doctors_2(num.doctors=10,num.meetings=10,starting_adopt
```

- Start small for debugging
- Making transmission always successful lets us see about updating

```r
tibble(m = 1:ncol(sim2), n = colSums(sim2)) %>%
  ggplot(aes(m, n))+
  geom_point() + labs(x = "Meeting", y ="Number of adoptees")
```

Example: Antibiotics Again

Maybe not that small? With only 10 time-steps it's pretty likely we never pick the one initial adoptee

```r
sim2.1 <- sim_doctors_2(num.doctors=10,num.meetings=100,starting_ad
tibble(m = 1:ncol(sim2.1), n = colSums(sim2.1)) %>%
  ggplot(aes(m, n)) +
  geom_point() + labs(x = "Meeting", y ="Number of adoptees")
```

Example: Antibiotics Again

```
sim.big <- sim_doctors_2(num.doctors=1000,num.meetings=10000,starti
tibble(m = 1:ncol(sim.big), n = colSums(sim.big)) %>%
  ggplot(aes(m, n)) +
  geom_point() + labs(x="Meeting",y="Number of adoptees")
```

Elapsed time from starting to code to final figure: 20 minutes

Example: Antibiotics Again

- These **logistic** or **sigmoid** curves are very characteristic of actual product-adoption curves, and of a lot of epidemiology
- This is the "susceptible-infectious" (SI) model Lots of variants
  - susceptible-infectious-susceptible (SIS), susceptible-infectious-recovered (SIR)
  - multiple stages of infectiousness
  - competing infections
  - can only transmit to network neighbors, not random pairing
  - etc., etc.

Summary

- When we don't have exact probability formulas or they don't apply, we can simulate to get arbitrarily-good approximations
- If we can describe the process of our model, we can set up a simulation of it