Practical R #3: Gaining insight from simulations

7 November, 2017

1 Motivation

It is very useful in science to construct mathematical models of the systems that we are investigating. As the complexity of these systems increases, it can often be difficult or impractical to derive or locate in the literature exact mathematical solutions of our model systems. While one option is to incorporate simplifying assumptions (such as that of normality) that make analytical solutions tractable, an alternative approach is to more directly simulate real world phenomena, and use numerical techniques to obtain solutions and insights.

For example, we saw how a t-test (which uses an analytical solution based on the Student's t-distribution) can also be cast as a data shuffling problem, and solved by repeated numerical (random or exhaustive) sampling of such shufflings via the randomization test. Similarly, we've seen how simulations could be used to explore power calculations for two groups with different means and different standard deviations, and to illustrate how the Bonferroni and FDR multiple hypothesis corrections were controlling the error for our entire experiment.

In this lecture, we'll develop the tools that you can use to build and explore your own models.

2 The Optimal Stopping Problem

Algorithms to Live By, by Brian Christian and Tom Griffiths, explores how people solve all sorts of problems that are defined by a limited amount of time, space, information or some combination of the above. The first chapter describes the so-called "secretary problem", also called the "optimal stopping problem". Although its origins are obscured by the mists of history, it was first described in print by Martin Gardner in his famous

Mathematical Games column in a 1960 issue of *Scientific American*. We'll frame it as a "dating problem".

Along the way, we will also explore some useful constructs in R, i.e., for loops and user-defined functions.

2.1 Define the problem

Let's first define the problem: We want to maximize our chance of finding our soul mate in a limited pool of candidates. Each potential mate in the pool has a compatibility score, and our goal is to find (and then propose to) the person in the pool with the highest score for us. We can only ascertain a person's compatibility score by dating them, and once we move on to the next candidate, we can never go back to someone we've dated in the past. So we need to decide if we want to propose to someone without knowing for sure if there is a better match still to come.

We'll attack this problem using a so-called "look then leap" strategy, whereby we'll first date a predetermined number of people just to gather data about the range of compatibility scores. Once our data gathering phase is over, we'll then continue to date the remaining people from our pool until we find someone whose compatibility score higher than any we've seen in the data gathering phase, and commit to that person. In this model, if we propose to somebody, they always accept, and if we reject them, we can never get them back.

The question now becomes: how many potential mates should be included in the data gathering phase, and how many should be left for the commitment phase?

In this model, there are two ways we can fail to meet our soul mate using this strategy: a) we meet our soul mate during the data gathering phase, and end up moving on to date others when we shouldn't have, or b) we can commit too early in the commitment phase, and never meet our soul mate. Note that in this model, success is binary and only occurs if we commit to our optimal match; committing to the second-best match is considered a failure, regardless of their numerical compatibility score. The optimal strategy requires finding the right balance between the two phases, or deriving the optimal proportion of the population that we should sample before committing.

2.2 The simulation

It turns out that this problem does have a well-known analytical solution; the math (which in its exact form is a Riemann approximation to an integral) says that the optimal proportion of people we gather data on before switching to the commitment phase is $1/e$ (cf

Thomas Ferguson, Who Solved the Secretary Problem? Statistical Science, 4(3):282-289 (1989)). Let's see if we can use simulations to verify this result.

We first develop the code for one scenario. Say we have 100 people in our potential candidate dating space, each with a score representing how good a match they are. The person with the best score is our real soul mate.

```
N <- 100
score <- rnorm(N)
optimal.score <- max(score)
```

We date the first n of the group and note the maximum score in that group...

```
n <- N * 1 / exp(1)
cutoff.score <- max(score[1:n])
```

Now select as your life partner the next date with a better score.

```
spouse.index <- n + which(score[n+1:N] > cutoff.score)[1]
spouse.score <- score[spouse.index]
```

Note that in the above code snippet, the `which()` function returns the indices of the truncated array `score[n+1:N]`. So, if your soul mate is found at position `n + 1` in the whole array, that will correspond to position 1 in the truncated array; adding `n` to the result of the `which()` function accounts for this shift.

We have successfully found our soul mate if the score of the match we found is the same as the optimal score.

```
spouse.score == optimal.score
```

If you run this a few times, you'll find that when our soul mate is encountered in the data gathering phase, we don't commit to anyone at all and the `spouse.score` is `NA`, which becomes an `NA` in our logical expression. We really want this to be recorded as `FALSE`, so let's set the score to the score of the final candidate if we haven't found anybody else by then.

```
if (is.na(spouse.index)) {
  spouse.index <- N
}
```

We probably want to run this multiple times, so that we can see how many times, on average, we find our soul mate. To do this, we'll write a for loop to run the code 1000 times.

```r
niter <- 1000
spouse.scores <- numeric(length = niter)

for (case.idx in 1:niter) {
  # scores of potential mates
  score <- rnorm(N)

  # we date the first n and note the maximum score in that group
  cutoff.score <- max(score[1:n])
  optimal.score <- max(score)

  # now select as your life partner the next date with a better score
    spouse.index <- n + which(score[n+1:N] > cutoff.score)[1]

    # pick the last one if nobody better came along before then
    if (is.na(spouse.index)) {
      spouse.index <- N
    }

    spouse.scores[case.idx] <- (score[spouse.index] == optimal.score)
}
```

and calculate the mean of all the scores:

```r
mean(spouse.scores)

## [1] 0.377
```

Note that in the code above, we have declared `spouse.scores` to be a numeric vector, so `TRUE`/`FALSE` values are converted to `0`/`1` values; the motivation for doing this will become apparent soon.

We can see that the mean of the scores turns out to be approximately 0.37, or we find our soul mate 37% of the time, if we sample $1/e$ of the population before committing to somebody…which is coincidentally also the proportion of the sample we date before committing.

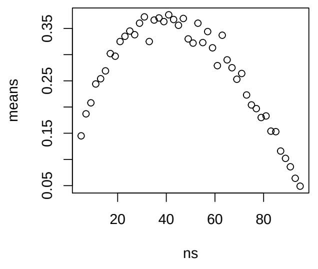
We probably want to explore this simulation a bit. What happens if we change the fraction of the population that we sample before switching to the commitment phase? Does the size of the candidate pool make a difference? We can answer questions such as these by casting our code as a function, and then varying the parameters it is called with.

```r
simulate.dating <- function(N = 1000, n = N / exp(1), niter = 1) {
  spouse.scores <- numeric(length = niter)

  for (case.idx in 1:niter) {
    # scores of potential mates
    score <- rnorm(N)

    # we date the first n and note the maximum score in that group
    cutoff.score <- max(score[1:n])
    optimal.score <- max(score)

    # now select as your life partner the next date with a better score
    spouse.index <- n + which(score[n+1:N] > cutoff.score)[1]

    # pick the last one if nobody better came along before then
    if (is.na(spouse.index)) {
      spouse.index <- N
    }

    spouse.scores[case.idx] <- (score[spouse.index] == optimal.score)
  }

  mean(spouse.scores)
}
```

Note here that we have assigned default values to all of the function's arguments. This way, when using the function, we only have to specify the arguments that differ.

Now we can call this function inside of another `for` loop.

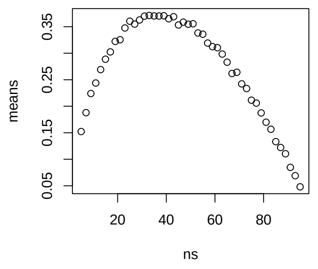
```r
ns <- seq(5, 95, 2)
means <- numeric(length = length(ns))
for (idx in 1:length(ns)) {
  means[idx] <- simulate.dating(N = 100, n = ns[idx], niter = 1000)
}
plot(means ~ ns)
```



```
ns[which.max(means)]

## [1] 41
```

There is definitely a pattern, but the graph is a bit bumpy, so let's recompute this distribution with more samples to get a smoother curve.

```
ns <- seq(5, 95, 2)
means <- numeric(length = length(ns))
for (idx in 1:length(ns)) {
  means[idx] <- simulate.dating(N = 100, n = ns[idx], niter = 10000)
}
plot(means ~ ns)
```



```
ns[which.max(means)]

## [1] 33
```

The `n` that corresponds to the peak of the graph is around 30-40. . . , confirming the analytical solution of $1/e$!

The optimal strategy gives us a 37% chance of finding our soul mate. However, this also implies a 63% failure rate (given our current definition of success)!

2.3 Exploring an alternative model

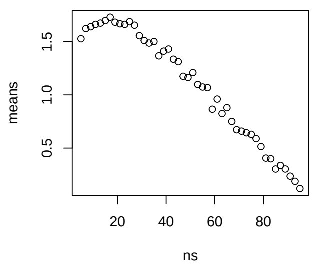
Let's consider an alternative model with a less stringent definition of "success". Instead of defining success as finding our soul mate, and pairing up with anybody else as failure, we might consider the score of the person that we ultimately pair off with as a numerical measure of the success of our endeavor. In other words, commiting to somebody with a relatively high score is still considered a pretty good outcome, even if it wasn't the optimal one.

While the underlying math for this optimization is perhaps very different, modifying our simulation to reflect this different objective is almost trivial.

```
# I am content with any decent match
spouse.scores[case.idx] <- score[spouse.index]
```

Here, `spouse.score` is a numeric value, not just a zero or one.

As the simulation below demonstrates, the optimal fraction of the population that we should sample before we switch to a commitment phase is now much lower!



```
ns[which.max(means)]

## [1] 17
```

So depending on your preferred criterion for "success", you'll adopt a different strategy.

There are other riffs and improvements on our model that we could imagine. For example, a previous assumption was that once you reject somebody, their feelings are irreversibly hurt and you can never go back to them. However, with a few more lines of code, you could simulate the situation where you would be able to select any of the last three people (for example) that you saw.

3 Problem Set #2

1. One of the advantages of using a simulation to model your system is that you have access to all information about the system. In the above analyses, we only considered the mean scores for each n. For each of the two strategies considered above, prepare a histogram of the scores at the optimal n. Does this new information influence which strategy you might adopt or recommend?
2. The simulation we described in class assumes that the candidates come from a normal distribution, with a standard deviation of 1. For each of the strategies, do the results change if the standard deviation is different? What about if the distribution is different, e.g., a uniform or poisson distribution? Can you reason out why?
3. One of the assumptions in the classic statement of this problem is that the candidate will always accept when you propose to them. How would you change the model to include the possibility of rejection? Do you think that a constant probability of rejection is a good model? It may be that if you find somebody very compatible, chances are good that they like you a lot too, and vice versa. How might you model the case where the chance of rejection is proportional to how much you like them?
4. Extra credit: Extend the model in any way you think might be fun or interesting! For example, a group of 100 people are all trying to get paired up. What would be the objective function? Total happiness? Or would you try to maximize individual's happiness?

Submit your response as an Rmd file by 11:59pm Monday, 27 November 2017. You must work in groups of four or five, with a single submission per group.

To help you along, here's the complete code from the above discussion, with an added argument that allows you to specify whether the function returns the mean of all the spouse scores, or the full distribution.

```r
simulate.dating <-
  function(N = 1000, n = N / exp(1), niter = 1, mean = TRUE) {

  spouse.scores <- numeric(length = niter)

  for (case.idx in 1:niter) {
    # scores of potential mates
    score <- rnorm(N)
    # score of soul mate
    optimal.score <- max(score)
    # we date the first n and note the maximum score in that group
    cutoff.score <- max(score[1:n])
    # now select as your life partner the next date with a better score
    spouse.index <- n + which(score[n+1:N] > cutoff.score)[1]
    if (is.na(spouse.index)) {
      spouse.index <- N
    }
    # Soulmate or bust
    spouse.scores[case.idx] <- (score[spouse.index] == optimal.score)
    # I am content with any decent match
    # spouse.scores[case.idx] <- score[spouse.index]
  }

  if (mean) {
    mean(spouse.scores)
  } else {
    spouse.scores
  }
}

sampled <- seq(5, 95, 5)
means <- numeric(length = length(sampled))
for (idx in 1:length(sampled)) {
  means[idx] <- simulate.dating(N = 100, n = sampled[idx], niter = 1000)
}
plot(means ~ sampled)
sampled[which.max(means)]
```