DATA 609 Assignment 4: Simulation Modeling

Dan Smilowitz

February 23, 2017

Section 5.1, Problem 3

Creating the Monte Carlo method with $n = 5000$ points, the following simulation is created using the uniform distribution for generating x and y values:

```
n <- 5000  # number of simulations
Q <- 0     # counter for number in circle

for (i in 1:n) {
  # generate random numbers
  x <- runif(1, 0, 1)
  y <- runif(1, 0, 1)
  # if inside cirlce; increment
  if (x^2 + y^2 <= 1) {
    Q <- Q + 1
  }
}

# calculate ratio of areas
ratio <- Q / n
# estimate pi
pi_est <- ratio * 4
```

The estimated value produced by the simulation is $\pi \approx 3.1672$.

Section 5.2, Problem 1

A function is written to generate the numbers using the middle-square method:

```
mid_sq <- function(seed, n) {
  # start sequence with seed
  gen <- seed
  # get length of seed
  len <- nchar(as.character(seed))
  # determine which digits to grab from squared numbers
  dig_start <- (2 * len) / 4 + 1
  dig_end   <- 3 * (2 * len) / 4
  for (i in 1:n) {
    # square last number generated
    new_gen <- gen[length(gen)]^2
    # convert new number to string to subset
    new_gen <- as.character(new_gen)
    # ensure number is of proper length
    lead_zero <- len * 2 - nchar(new_gen)
    new_gen <- paste0(paste(rep('0', lead_zero), collapse = ''), new_gen)
    # get middle digits
    new_gen <- substr(new_gen, dig_start, dig_end)
    # add new number to array of generated numbers
    gen <- c(gen, as.numeric(new_gen))
  }
  gen
}
```

	a	b	c
0	1009	653217	3043
1	180	692449	2598
2	324	485617	7496
3	1049	823870	1900
4	1004	761776	6100
5	80	302674	2100
6	64	611550	4100
7	40	993402	8100
8	16	847533	6100
9	2	312186	2100
10	0	460098	4100
11		690169	8100
12		333248	6100
13		54229	2100
14		940784	4100
15		74534	8100
16		555317	
17		376970	
18		106380	
19		316704	
20		301423	

Random sequence a rapidly degenerates to zero. Sequence b exhibits degeneration, but it is far less rapid than part a. Sequence c exhibits cycling beginning with the fourth number generated.

Section 5.3, Problem 4

The total of the two dice is simulated by running a Monte Carlo simulation for each weighted die:

```r
# load probabilities
die_1 <- c(0.1, 0.1, 0.2, 0.3, 0.2, 0.1)
die_2 <- c(0.3, 0.1, 0.2, 0.1, 0.05, 0.25)
# convert to cumulative probabilities
die_1 <- cumsum(die_1)
die_2 <- cumsum(die_2)

# set up container for sums
tot <- numeric()

for (i in 1:300) {
  # generate two random numbers
  d1 <- runif(1)
  d2 <- runif(1)
  # get rolls associated with values
  d1 <- min(which(d1 < die_1))
  d2 <- min(which(d2 < die_2))
  # add sum to set of sums
  tot <- c(tot, d1 + d2)
}
```

The count of observation by sum is below:

Sum	Count	Proportion
2	4	0.01333
3	10	0.03333
4	26	0.08667
5	45	0.15
6	36	0.12
7	58	0.1933
8	34	0.1133
9	34	0.1133
10	34	0.1133
11	12	0.04
12	7	0.02333

Section 5.3, Project 4

To generate the probabilities for the Monte Carlo simulation, the provided odds must be converted to probabilies:

$$n : 1 \text{ odds} \implies \frac{1}{n+1} \text{ probability}$$

The sum of the individual probabilities do not add up to exactly 1, so each probability is divided by the sum of the probabilities to give the adjusted probability.

```
odds <- c(7, 5, 9, 12, 4, 35, 15, 4)
prob <- 1 / (odds + 1)
prob.adj <- prob / (sum(prob))
```

	Probability	Adjusted.Probability
Euler's Folly	0.125	0.1304
Leapin' Leibniz	0.1667	0.1738
Newton Lobell	0.1	0.1043
Count Cauchy	0.07692	0.08022
Pumped up Poisson	0.2	0.2086
Loping L'Hopital	0.02778	0.02897
Steaming' Stokes	0.0625	0.06518
Dancin' Dantzig	0.2	0.2086

These adjusted probabilites are used to create the simulation:

```
# set up counter for wins
wins <- rep(0, length(prob.adj))
# get cumulative distribution
cum_prob <- cumsum(prob.adj)
# run simulation
for (i in 1:1000) {
  # generate random number
  w <- runif(1)
  # determine winning horse
  w <- min(which(w < cum_prob))
  # increment horse's winnings
  wins[w] <- wins[w] + 1
}
```

	Wins
Euler's Folly	137
Leapin' Leibniz	175
Newton Lobell	108
Count Cauchy	89
Pumped up Poisson	183
Loping L'Hopital	19
Steaming' Stokes	50
Dancin' Dantzig	239

The horse with the most wins was Dancin' Dantzig – this is unsurprising, as it was tied for the best odds (4:1). Loping L'Hopital had the fewest wins – this too is unsurprising, since it had the worst odds (35:1).

Section 5.4, Problem 3

Assuming that the lag can only take the discrete values given (i.e. a minimum of 2 days, maximum of 7, and no half days), the simulation can be set up as follows:

```r
lag_count <- c(10, 25, 30, 20, 13, 2)
# calculate cumulative probabilities
lag_prob <- cumsum(lag_count / 100)
# set up counter for lags
lag_days <- rep(0, length(lag_count))
# run simulation
for (i in 1:1000) {
  # generate random number
  l <- runif(1)
  # determine lag duration based on number
  l <- min(which(l < lag_prob))
  # increment lag count
  lag_days[l] <- lag_days[l] + 1
}
```

Lag time	Occurences
2	89
3	248
4	299
5	207
6	142
7	15

Based on the observed occurences, the simulation results align with what would be expected.

Section 5.5, Problem 2

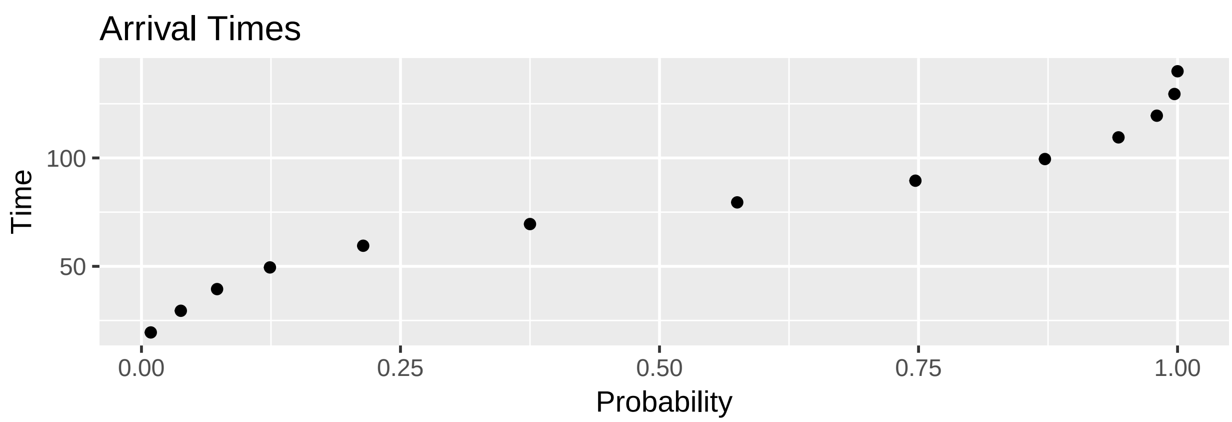
Table 5.18 contains two subtables – one for arrivals, and one for service times. Using the midpoints of each interval:

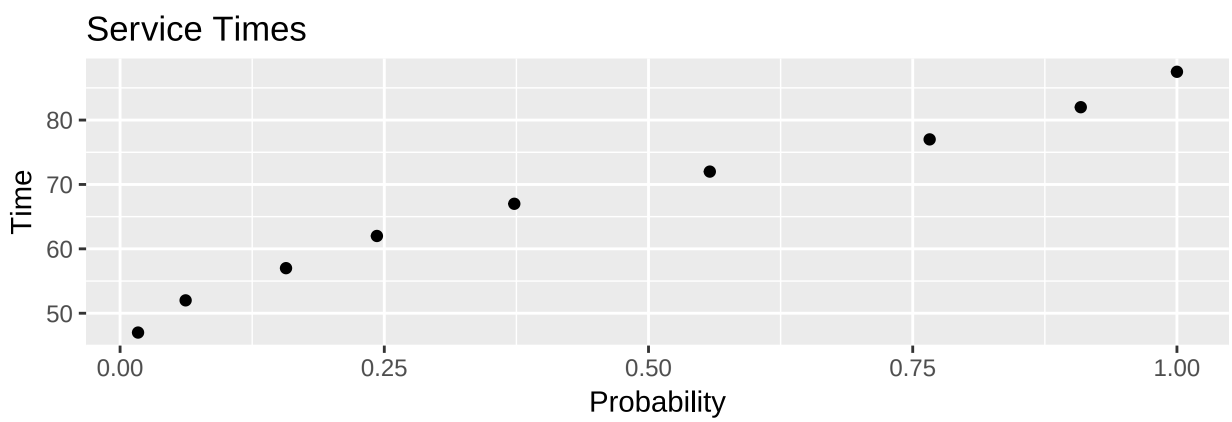
```
arrival <- data.frame(
  times = c(19.5, 29.5, 39.5, 49.5, 59.5, 69.5, 79.5, 89.5, 99.5, 109.5, 119.5, 129.5, 140),
  probs = c(0.009, 0.029, 0.035, 0.051, 0.09, 0.161, 0.2, 0.172, 0.125, 0.071, 0.037, 0.017, 0.003)
)

service <- data.frame(
  times = c(47, 52, 57, 62, 67, 72, 77, 82, 87.5),
  probs = c(0.017, 0.045, 0.095, 0.086, 0.13, 0.185, 0.208, 0.143, 0.091)
)

# convert to cumulative probabilities
arrival$probs <- cumsum(arrival$probs)
service$probs <- cumsum(service$probs)
```

To establish a smooth polynomial function, plots of times vs. probabilities are created for each set of observations:





Arrival Times

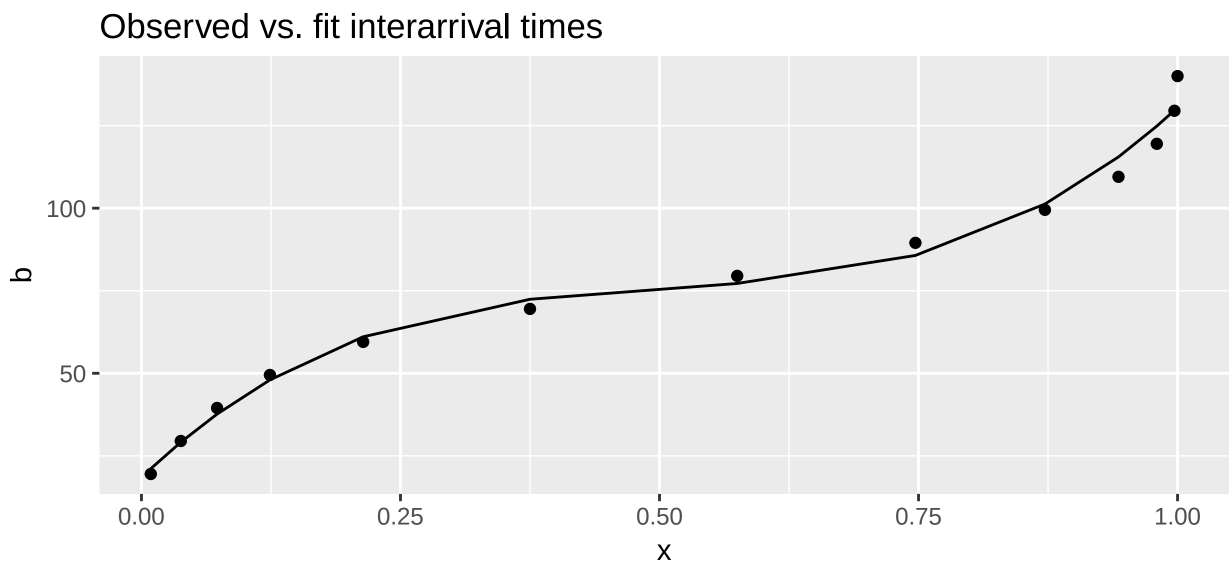
The curve for arrival times shown above appears to resemble that of a cubic equation for x vs. y^3. As such, a third-degree polynomial is fit to the data:

```
mod_arr <- lm(times ~ poly(probs, 3, raw = TRUE), data = arrival)
arrival$preds <- predict(mod_arr, arrival)
```

The equation for the best-fit third-order fit is

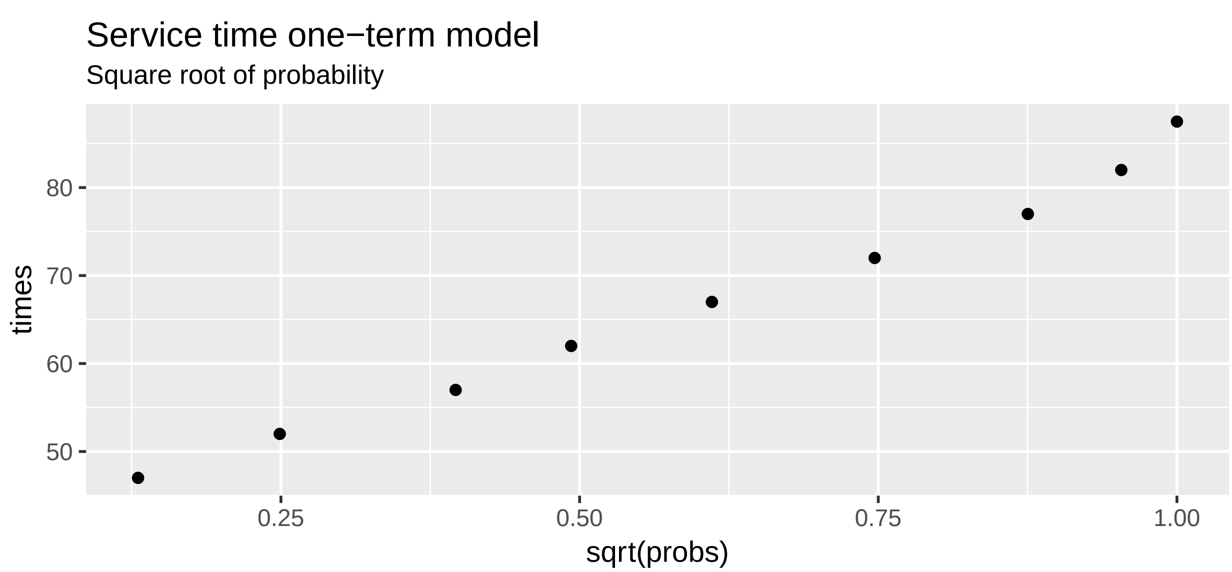
$$b = 18.4377 + 302.2950x - 561.4642x^2 + 371.2266x^3$$

This appears reasonable when plotted against the original data:



Service Times

To try fitting a one-term model, a number of transforms are examined. A transformation using the square root of probability appears to provide a near-linear relationship:



A linear equation is fit for this one-term model:

```
mod_arr_sq <- lm(times ~ I(sqrt(probs)), data = service)
```

The equation for this line is given by

$$u = 40.4257 + 43.9362\sqrt{x}$$

Plotting this equation against the original data, it appears reasonable: