IS 604 – Homework #1

J. Hamski

September 3, 2016

```
library(ggplot2)
```

1.1) Name several entities, attributes, activities, events, and state variables for the following systems:

a. Cafeteria
entities: diners, cash register attendants, cooks
attributes: tables vacancies, line wait time
activities: sitting down, eating, waiting to check out
events: arrive at seat, depart seat, checking out
state variables: time, diner number

b. Grocery store
entities: shelves, items on shelves (UPC codes)
attributes: quantity of product available
activities: restocking, picking out items
events: taking an item off a shelf
state variables: time, shift

c. laudromat
entities: customers, washers, dryers
attributes: wash/dry time
activities: washing, drying
events: wash/dry started, wash/dry ended
state variables: time, wash/dry cycles

d. Fast-food restaurant *entities*: cars in a drive through window
attributes: order preparation time
activities: ordering, preparing order
events: order entered, order prepared
state variables: car number, time

e. Hospital emergency room *entities*: patients
attributes: time of treatment, severity (priority for treatment)
activities: treatment, waiting
events: entering into queue, triage, wait time, treatment start, treatment end
state variables: time, waiting order, priority

f. Taxicab company with 10 taxis *entities*: cars, riders, drivers
attributes: hours of operation, breaks
activities: picking up fares, driving them to destination
events: pickup, drop-off
state variables: taxi occupancy, time, trip length

g. Automobile assembly line *entities*: cars, workers, robots
attributes: percent assembled, maintenance requirements, shift changes
activities: assembling a single car
events: assembly of a specific section of car, fixing a specific part
state variables: time, begin assebly, end essembly, ready for assembly action

2.1)

The Excel examples would not allow me to do any editing on MacOS, so the remaining problems were attempted in R.

```
set.seed(243)
# time unit is minutes

between.arrivals <- c(0,60,120,180)
between.arrivals.prob <- c(0.23,0.37,0.28,0.12)


shop.model <- function(max.customers){

  customer <- 1
  arrival.time <- 0
  service.time <- 50
  service.time.begins <- 25
  service.time.ends <- NULL
  time.in.system <- 25
  time.in.queue <- NULL
  int.arrival.time <- NULL

  for (i in 1:(max.customers-1)) {


    customer <- c(customer, customer[i] + 1)
    #print(customer)

    sim.service.time <- round(rnorm(n = 1, mean = (50), sd = (8)),0)
    service.time <- c(service.time, sim.service.time)

    int.arrival.time.customer <- sample(x = between.arrivals, size = 1, prob = between.arrivals.prob)
    int.arrival.time <- c(int.arrival.time, int.arrival.time.customer)

    arrival.time <- c(arrival.time, arrival.time[i-1] + int.arrival.time.customer)

    service.time.begins <- c(service.time.begins, max(arrival.time[i], service.time.ends[i - 1]))

    service.time.ends <- c(service.time.ends, service.time.begins[i] + service.time[i])

    time.in.system <- c(time.in.system, service.time.ends[i] - arrival.time[i])

    time.in.queue <- c(time.in.queue, arrival.time[i] - service.time.begins[i])

    #print(service.time[i])
  }
  output <- list(customer, time.in.queue, service.time, time.in.system)
  return(output)
}

a <- shop.model(10)
```

Average time in queue:

```r
mean(a[[1]])
```

```
## [1] 5.5
```

Average processing time:

```r
mean(a[[2]])
```

```
## [1] 43.22222
```

Maximum time in system:

```r
max(a[[3]])
```

```
## [1] 62
```

This model shows reasonable results for the average and max processing times, but certainly has an error (that I failed to debug) since time in system shows occasonal negative numbers.

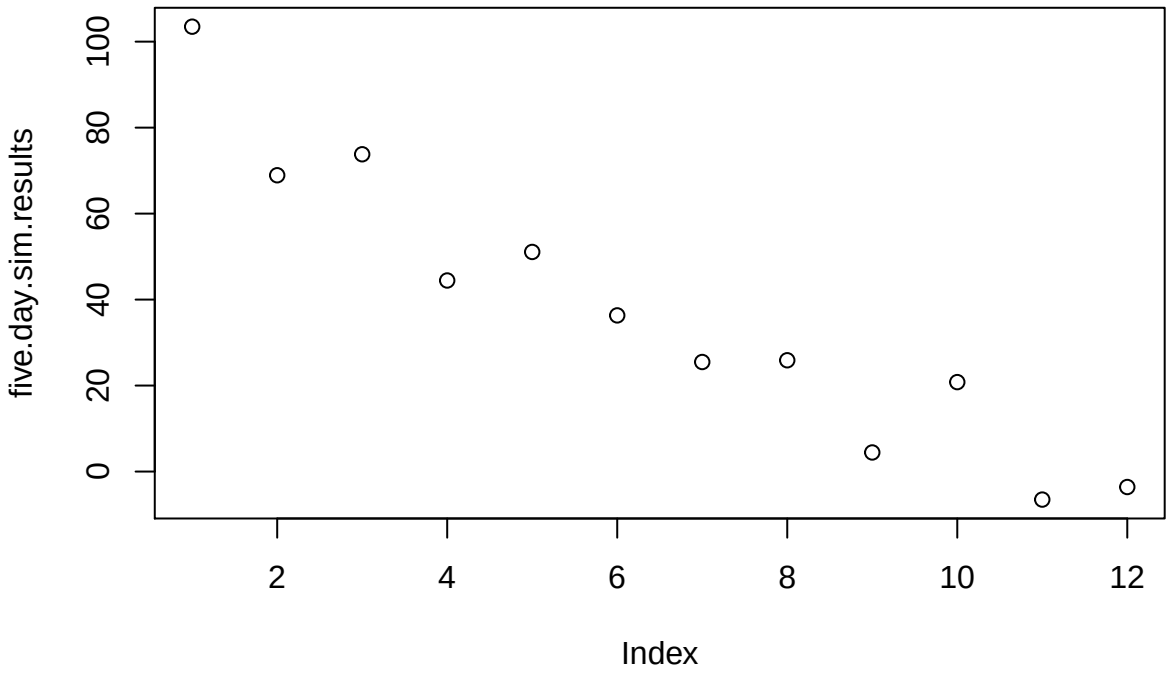
2.2)

```r
#bagel units are dozens

customers.day <- c(8,10,12,14)
customer.prob <- c(0.35,0.30,0.25,0.10)

dozens.ordered <- c(1,2,3,4)
ordered.prob <- c(0.4,0.3,0.2,0.1)

bagel.demand.sim <- function(quantity, sim.days){

  day <- 1
  demand <- NULL
  revenue <- NULL
  lost.profit <- NULL
  salvage <- NULL
  daily.cost <- NULL
  daily.profit <- NULL


  for (i in 1:(sim.days)) {
    day <- c(day, day[i] + 1)
    #print(day)
    #print(i)

    #start demand subroutine
    customers <- sample(x = customers.day, prob = customer.prob, size = 1)
    bagels.sold.day <- NULL
    for (j in 1:(customers)) {
      bagels.sold.day <- c(bagels.sold.day, sample(x = dozens.ordered, prob = ordered.prob, size = 1))
    }
```

```
    #print(bagels.sold.day)
    bagels.sold <- sum(bagels.sold.day)
    #end demand subroutine

    #print(bagels.sold)

    demand <- c(demand, bagels.sold)

    revenue <- c(revenue, bagels.sold * 8.4)

    daily.cost.day <- quantity * 5.8
    daily.cost <- c(daily.cost, daily.cost.day)

    lost.profit.day <- ifelse(quantity < bagels.sold, (bagels.sold - quantity) * 2.6, 0)
    lost.profit <- c(lost.profit, lost.profit.day)
    #print(lost.profit.day)

    salvage.day <- ifelse(quantity > bagels.sold, (quantity - bagels.sold)*4.2, 0)
    salvage <- c(salvage, salvage.day)
    #print(salvage.day)

    daily.profit <- c(daily.profit, bagels.sold * 8.4 - lost.profit.day + salvage.day - daily.cost.day)

  }
  return(mean(daily.profit))
}
```

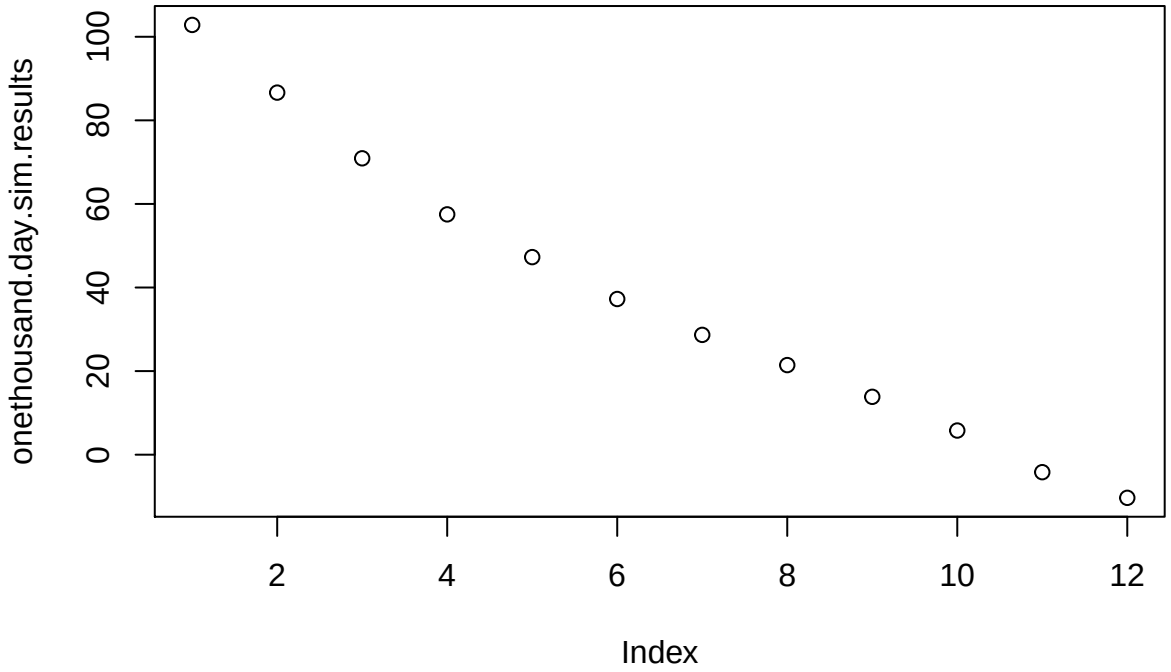
```
max <- 5
min <- 60

quantities <- seq(max, min, 5)
```

```
five.day.sim.results <- sapply(quantities, FUN=bagel.demand.sim, sim.days = 5)
plot(five.day.sim.results)
```



Limiting to five days simulation only makes sense with Excel. Here, 1,000 days are simulated for each 5 dozen bin.

```
onethousand.day.sim.results <- sapply(quantities, FUN=bagel.demand.sim, sim.days = 1000)
plot(onethousand.day.sim.results)
```



It looks like only 5 dozen is the right quantity to bake each day. This didn't sound right so I took a look at the demand generation subroutine.

```r
demand.sim <- function() {
  #start demand subroutine
    customers <- sample(x = customers.day, prob = customer.prob, size = 1)
    bagels.sold.day <- NULL
    for (j in 1:(customers)) {
      bagels.sold.day <- c(bagels.sold.day, sample(x = dozens.ordered, prob = ordered.prob, size = 1))
    }
    #print(bagels.sold.day)
    bagels.sold <- sum(bagels.sold.day)
    #end demand subroutine
    return(bagels.sold)
}

demand.distribution <- replicate(1000, demand.sim())
```

```r
summary(demand.distribution)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   17.00   20.00   20.53   24.00   38.00
```

This suggests the actual best quantity is 20 dozen per day. I cannot figure out which part of my simulation is wrong, but this means more work is needed.

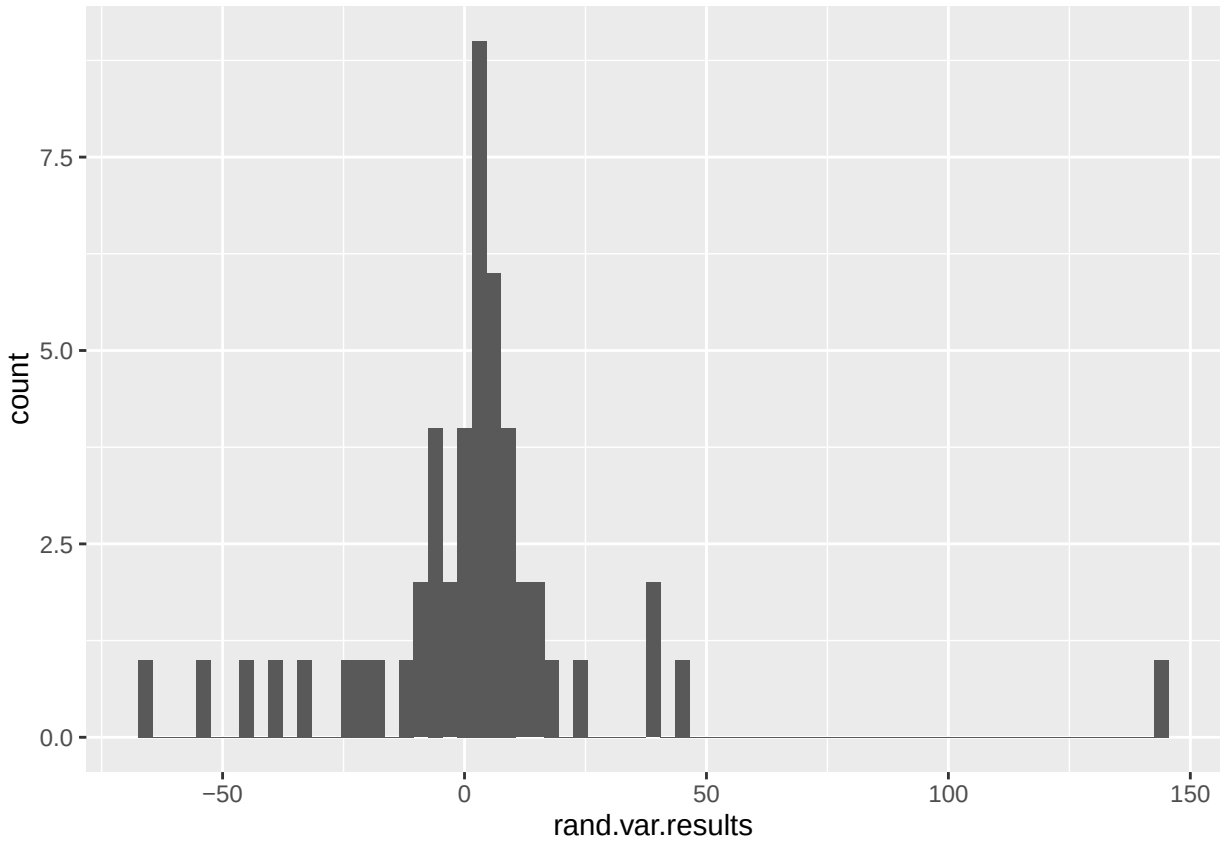
2.4)

```r
interarrival <- c(15,20,25,30,35)
inter.prob <- c(0.14,0.22,0.43,0.17,0.04)

service.time <- c(5,15,25,35,45)
service.prob <- c(0.12,0.35,0.43,0.06,0.04)

one.taxi <- function(){


}
```

2.5)

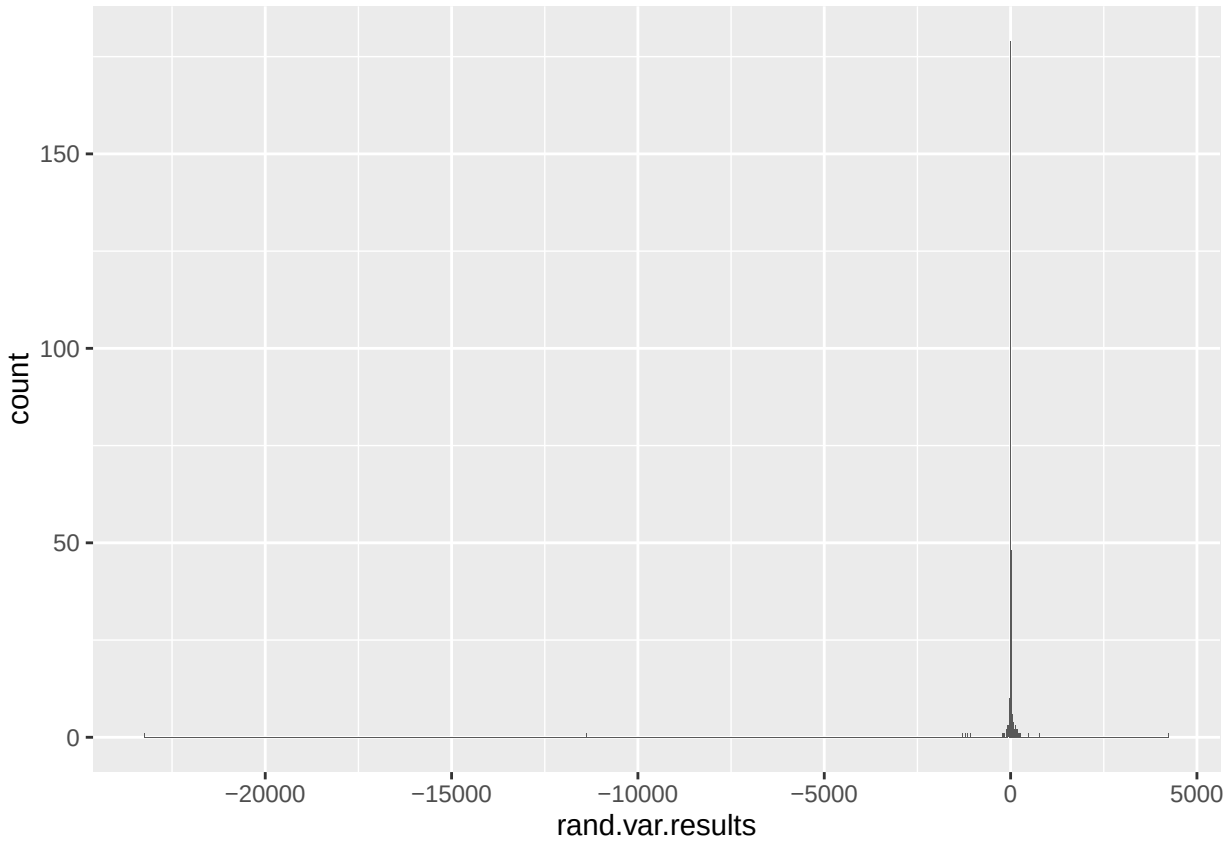
```r
rand.var.sim <- function(){
  X <- rnorm(1, mean = 100, sd = 100)
  Y <- rnorm(1, mean = 300, sd = 225)
  Z <- rnorm(1, mean = 40, sd = 64)

  W <- (X + Y)/Z
  return(W)
}
```

```r
rand.var.results <- replicate(50, rand.var.sim())
qplot(rand.var.results, geom = "histogram", binwidth = 3)
```



Note that 50 is not really enough replications to show the full variance of this distribution. Simulating the random variable 1000 times shows this random variable has a tendency to produce some extreme values.

```
rand.var.results <- replicate(1000, rand.var.sim())
qplot(rand.var.results, geom = "histogram", binwidth = 3)
```




```
var(rand.var.results)
```

```
## [1] 695395.9
```

2.6)

```
B.values <- c(0,1,2,3,4)
B.probs <- rep(1/5, length(B.values))

C.values <- c(10,20,30,40)
C.probs <- c(0.1,0.25,0.5,0.15)

D.sim <- function() {
  A <- rnorm(1, mean = 100, sd = 400)
  B <- sample(x = B.values, prob = B.probs, size = 1)
  C <- sample(x = C.values, prob = C.probs, size = 1)

  D <- (A - 25*B) / (2*C)
  return(D)
}
```

```
D.sim.results <- replicate(100, D.sim())

mean(D.sim.results)
```

```
## [1] -0.3419739
```

2.7

```
# output goal: average number of lost sales per week

inventory.sim <- function(sim.weeks){
  day <- 1
  inventory <- 18
  demand <- NULL
  lost.sales <- NULL
  order.days <- NULL
  delivery.day <- NULL
  inventory.record <- NULL

  #simulation time unit is 1 day. Loop starts on day 2
  for (i in 1:(sim.weeks*7)) {

    demand.day <- round(rnorm(1, mean = 5, sd = 1.5), digits = 0)
    demand <- c(demand, demand.day)

    lost.sales.day <- ifelse(demand.day > inventory, demand.day - inventory, 0)
    lost.sales <- c(lost.sales, lost.sales.day)

    #recalculate inventory
    inventory <- ifelse(demand.day < inventory, inventory - demand.day, 0)
    inventory.record <- c(inventory.record, inventory)


    #begin ordering subroutine
    if (inventory < 10 && is.null(delivery.day) == TRUE) {
      delivery.time.period <- round(runif(1, min = 0, max = 5), digits = 0)
      delivery.day <- day + delivery.time.period
      order.size <- 20 - inventory
    }

    if (is.null(delivery.day) == FALSE && delivery.day == day) {
      inventory <- inventory + order.size
      delivery.day <- NULL
    }
    #end ordering subroutine
    day <- day + 1
  }
  return(list(lost.sales, demand, inventory.record))
}

lost.sales.model <- inventory.sim(5)
round(mean(lost.sales.model[[1]]), 0)
```

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

This is giving the average as per-day, the time unit used in the simulation. Multiplying by 7 give the per-week value.

```r
round(mean(lost.sales.model[[1]]), 0) * 7
```

```
## [1] 7
```

```r
summary(lost.sales.model[[1]]) * 7
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  4.7999  0.0000 35.0000
```

Because we're using R, we can run the simulation for more iterations. Here we find the mean for 1000 weeks.

```r
lost.sales.model.1K.weeks <- inventory.sim(1000)
summary(lost.sales.model.1K.weeks[[1]]) * 7
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   0.000  11.956  28.000  70.000
```

2.8)

```r
#time unit is minutes

elevator.sim <- function(sim.min) {

  simulated.arrivals.total <- 1000
  A <- runif(simulated.arrivals.total, min = 3, max = 7)
  B <- rep(6, simulated.arrivals.total)
  C <- sample(x = c(2, 3), prob = c(0.33, 0.67), replace = TRUE, size = simulated.arrivals.total)

  time <- 0


  #calculate loading
  while (time < 60) {

    #A.ready <- ifelse(A[time])



    time <- time + 1
  }


}

elevator <- elevator.sim(60)
```