Lecture 19: Simulation

Statistical Computing, 36-350

Wednesday November 30, 2015

Outline

- Quick review of random number generation
- Why simulate?
- Basics of Monte Carlo sampling, applied to darts scoring

Random number generation in R

Throughout, we've simulated random numbers in R according to various distributions

- `rnorm()`: generate normal random variables
- `pnorm()`: normal distribution function, $\Phi(x) = P(Z \leq x)$
- `dnorm()`: normal density function, $\phi(x) = \Phi'(x)$
- `qnorm()`: normal quantile function, $q(y) = F^{-1}(y)$, i.e., $F(q(y)) = y$

Replace "norm" with the name of another distribution, all the same functions apply. E.g., "t", "exp", "gamma", "chisq", etc.

Simple examples

t random variables, with 5 df:

```
n = 1000
t.draws = rt(n,df=5) # t5 random variables
mean(t.draws)  # Check: mean approx 0
```

```
## [1] 0.0524523
```

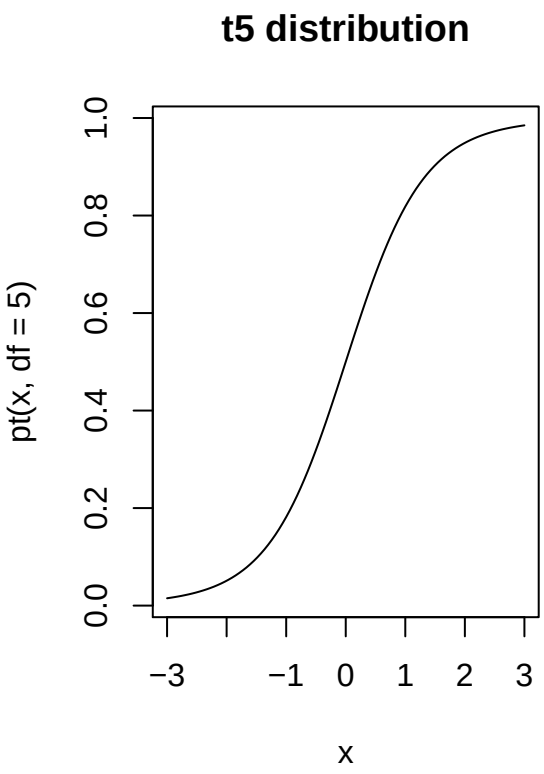
```
var(t.draws)   # Check: variance approx 5/3
```

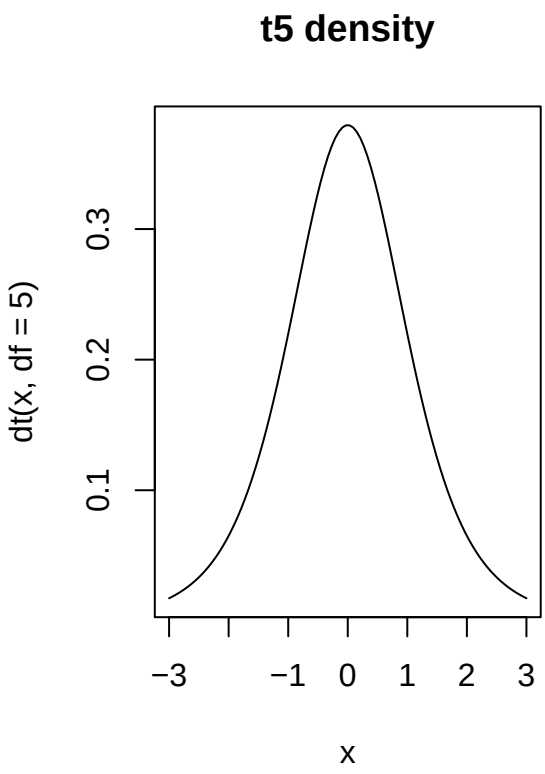
```
## [1] 1.835184
```

---

t distribution, density functions:

```
x = seq(-3,3,length=100)
par(mfrow=c(1,2))
plot(x, pt(x,df=5), main="t5 distribution", type="l")
plot(x, dt(x,df=5), main="t5 density", type="l")
```



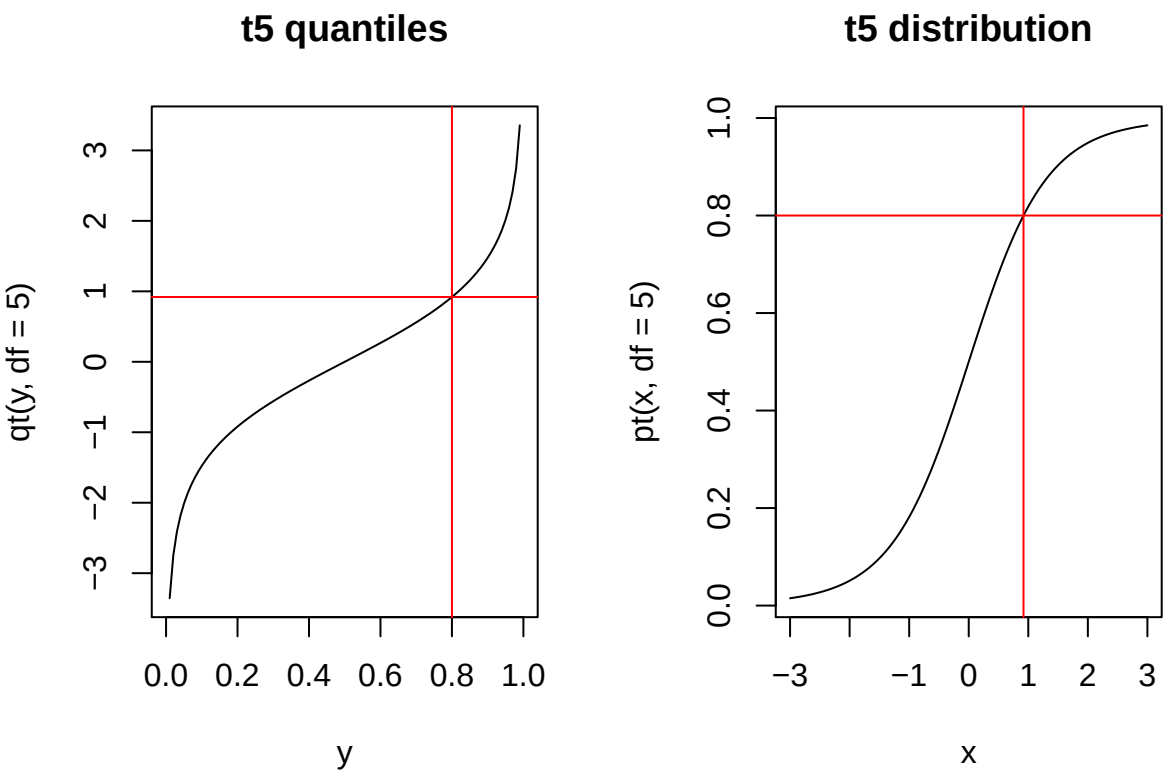


---

t5 quantile function:

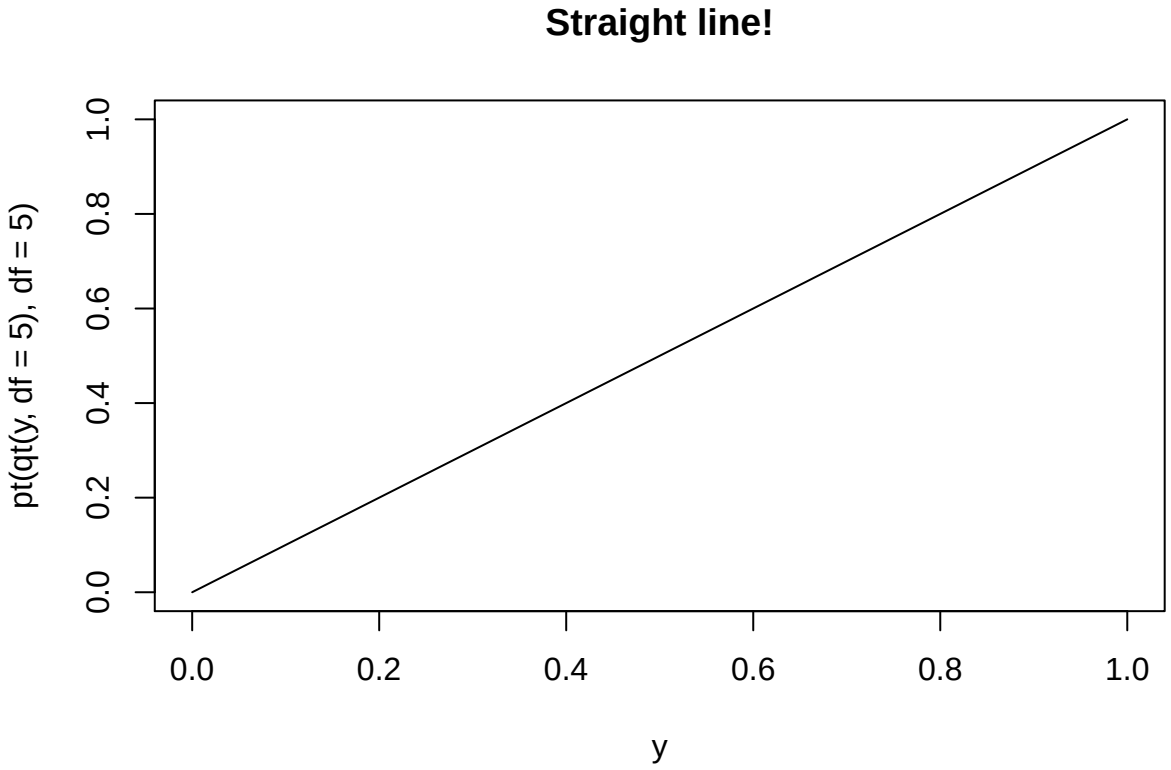
```
par(mfrow=c(1,2))
y = seq(0,1,length=100)
plot(y, qt(y,df=5), main="t5 quantiles", type="l")
xval = qt(0.8,df=5)
abline(v=0.8, col="red")
abline(h=xval, col="red")

plot(x, pt(x,df=5), main="t5 distribution", type="l")
abline(v=xval, col="red")
abline(h=0.8, col="red")
```



Check that `qt()` and `pt()` are inverses:

```
par(mfrow=c(1,1))
plot(y, pt(qt(y,df=5),df=5), type="l", main="Straight line!")
```



Why simulate?

R gives us unique access to great simulation tools (unique compared to other languages). Why simulate? Welcome to the 21st century! Two reasons:

- Often times simulations can be easier than hand calculations
- Often times simulations can be made more realistic than hand calculations

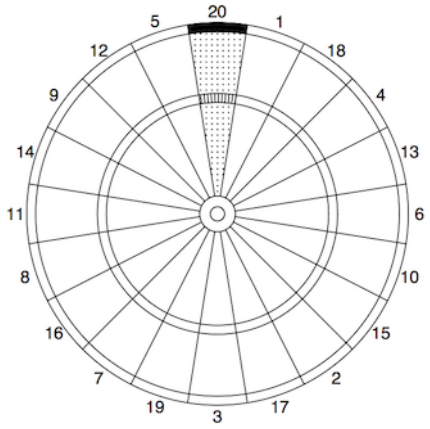
Two motivating examples

This week we'll go through two motivating examples, from two different sides of the statistics universe: fun and games, and medical statistics! Former today (and in lab), latter next time

- Simulation for optimal darts strategies
- Simulation and power calculations for drug effects

Darts strategies

- Darts is a game where you throw metal darts at a dartboard, and receive different scores for landing the dart in different regions



- Land in a pie slice, obtain a score of the corresponding number
- Land in the "double ring", get double the score
- Land in the "triple ring", get triple the score
- Land in the "double bullseye" (DB), get 50 points
- Land in the "single bullseye" (DB), get 25 points

Some questions we might have:

- Suppose you threw uniformly at random over the board, or aimed at the center and had normal distributed throws, with a certain variance. What would give a better average score?
- If you had a certain variance, where would be the best spot to aim?
- If you had recorded certain scores, aiming at the center of the board, how could you estimate your variance?

(For more, you can check out this paper: http://www.stat.cmu.edu/~ryantibs/papers/darts.pdf, and also check out the R package `darts` on CRAN)

Board measurements

We'll have to start off by learning the board measurements. These are easy to look up (all measured in mm). We'll also have to store the order of the numbers, starting at the top (12 o'clock)

```
board = list(
  R1 = 6.35,   # center to DB wire
  R2 = 15.9,   # center to SB wire
  R3 = 99,     # center to inner triple ring
  R4 = 107,    # center to outer triple ring
  R5 = 162,    # center to inner double ring
  R = 170,     # center to outer double ring
  nums = c(20,1,18,4,13,6,10,15,2,17,3,19,
    7,16,8,11,14,9,12,5)) # numbers in order
```

Score function

Now we'll have to define a scoring function:

```
# Inputs:
# - x: vector, horizontal positions of dart throws in mm,
#   with the center of the board being 0
# - y: vector, vertical positions of dart throws in mm,
#   with the center of the board being 0
# - board: list, containing dart board measurements
# Outputs: vector of scores, same length as x (and as y)
scorePositions = function(x, y, board) {
  R1 = board$R1
  R2 = board$R2
  R3 = board$R3
  R4 = board$R4
  R5 = board$R5
  R = board$R
  nums = board$nums

  n = length(x)
  rad = sqrt(x^2 + y^2)
  raw.angles = atan2(x,y)
  slice = 2*pi/20
  titled.angles = (raw.angles + slice/2) %% (2*pi)
  scores = nums[floor(titled.angles/slice) + 1]

  # Bullseyes
  scores[rad <= R1] = 50
  scores[R1 < rad & rad <= R2] = 25

  # Triples
  scores[R3 < rad & rad <= R4] = 3*scores[R3 < rad & rad <= R4]

  # Doubles
  scores[R5 < rad & rad <= R] = 2*scores[R5 < rad & rad <= R]

  # Zeros (off the board)
  scores[R < rad] = 0

  return(scores)
}
```

Plotting the board

Lastly, we'll need to know how to plot the board:

```
# Inputs:
# - board: list, list, containing dart board measurements
# Outputs: none, side effect is a plot of the dart board,
#  where further plotting can take place on top as usual
#  with points(), lines(), etc. In the plotting region,
```

```
#  (0,0) marks the center of the dart board
drawBoard = function(board) {
  R1 = board$R1
  R2 = board$R2
  R3 = board$R3
  R4 = board$R4
  R5 = board$R5
  R = board$R
  nums = board$nums

  mar.orig = par()$mar
  par(mar = c(0, 0, 0, 0))
  plot(c(), c(), axes = FALSE, xlim = c(-R - 15, R + 15),
       ylim = c(-R - 15, R + 15))
  t = seq(0, 2 * pi, length = 5000)
  x = cos(t)
  y = sin(t)
  points(R * x, R * y, type = "l")
  points(R5 * x, R5 * y, type = "l")
  points(R4 * x, R4 * y, type = "l")
  points(R3 * x, R3 * y, type = "l")
  points(R2 * x, R2 * y, type = "l")
  points(R1 * x, R1 * y, type = "l")
  t0 = pi/2 + 2 * pi/40
  points(c(R2 * cos(t0), R * cos(t0)), c(R2 * sin(t0),
         R * sin(t0)), type = "l")
  for (i in 1:19) {
    t1 = t0 - i * 2 * pi/20
    points(c(R2 * cos(t1), R * cos(t1)), c(R2 * sin(t1),
    R * sin(t1)), type = "l")
  }

  r = R + 10
  for (i in 1:20) {
    t1 = pi/2 - (i - 1) * 2 * pi/20
    text(r * cos(t1), r * sin(t1), nums[i])
  }

  par(mar=mar.orig)
  invisible()
}
```

Practice problems

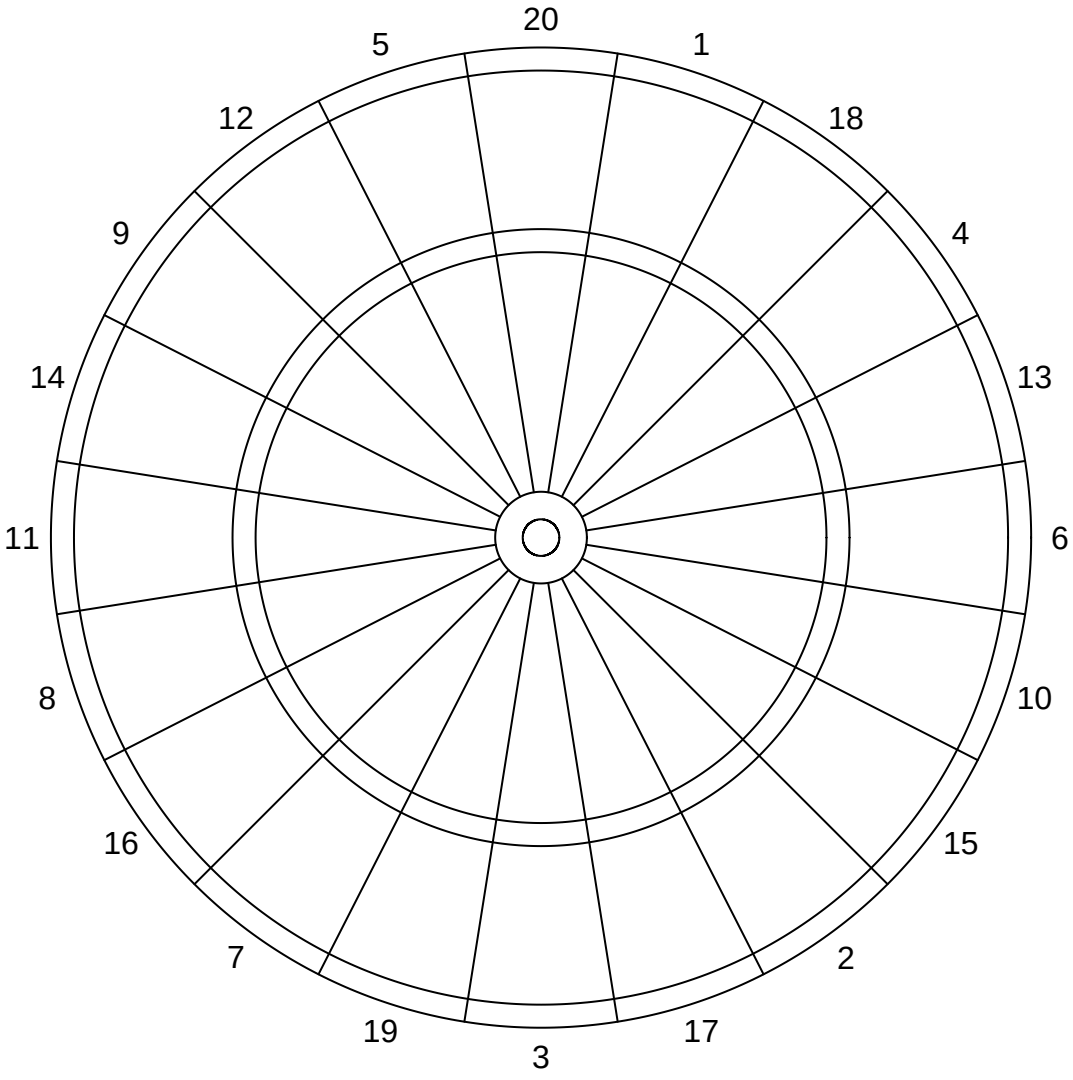
Enter your unique ID here:

Let's try some of these functions out. Work through the following problems (go ahead and fill in the code below)

```
# 1. Let x,y denote the horizontal and vertical positions of the throws,
# measured from the center of the board. Consider the model:
#    x and y are both N(0,50^2)
```

```
# (so that the standard deviation here is sd = 50mm). Generate 100 throws
# from this model. Plot the resulting positions on top of the dart board,
# with pch=20 (to make small solid dots). Then, compute the scores of your
# 100 throws, and draw in text, the scores corresponding to each throw.
# (Hint: for this last part, use text().)

# Draw the dartboard
drawBoard(board)
```



```
# 2. Now simulate 10,000 darts throws according to two possible models:
# (a) x and y are both N(0,50^2)
# (b) x and y are both Uniform(-R,R)
# Note that model (b) describes throws that uniformly distributed over
# the smallest rectangle that contains the dart board. Compute the scores
# from each set of throw. Then, answer the following:
# - what are the average scores under each model, and which is higher?
```

```
# - what proportion of scores from model (b) missed the board?
# - what you expect for the latter, theoretically, under model (b)?
#   (hint: basic geometry needed, area of circle, area of square ...)
# - what proportion of score from model (a) missed the board?
# - bonus: what do you expect for the latter, theoretically, under
#   model (a)? (hint: consider using pchisq() ...)
# - which throws missed the board more often, those from model (a) or
#   (b)? and are you surprised?
```

```
# 3. For 25 values of sd, in between 5 and 150 mm, draw 10,000 throws from
# model (a) as described above. For each value of sd, compute the average
# score, and the proportion of zero scores encountered. Then make a plot
# showing the average score as a function of sd, drawn as a black line. Label
# the axes appropriately, and draw the average score under the uniform model
# (b) as a horizontal line, in red. Then, answer the following:
# - at around what value of sd does it become better to throw uniformly at the
#   board?
# - at this value of sd, what is the average proportion of zero scores?
# - at the highest value of sd, what is the average propotion of zero scores?
# - compare the latter to the average proportion of zero scores from the
#   uniform model. which is higher? why?
```

```
# 4. Consider fixed the value sd = 35mm, and consider a normal model for
# throws where x is N(mean.x, 35^2) and y is N(mean.y, 35^2), with
# - (a) (mean.x,mean.y) = (0,0)
# - (a) (mean.x,mean.y) = (0,103)
# - (b) (mean.x,mean.y) = (-32,-98)
# First, either by plotting or by looking at scores, determine where the throws
# are being aimed in each of the three models (a),(b),(c). That is, the choice
# of (mean.x,mean.y)=(0,0) in model (a) means that we are aiming at the center
# of the board. Where are we aiming in models (a) and (b)? Then, for each of the
# models (a),(b),(c) specified above, draw 10,000 throws, and compute the average
# score and proportion of zeros. How do they compare? Can you explain the results
# from an intuitive perspective? (Hint: look at a picture of the board.)
```

```
# Bonus. The normal model that we used in the problem 3 was a bit too pessimistic,
# when compared to the uniform model. This is because the normal model can
# produce throws that are arbitrarily far away from the center, but the uniform
# model is restricted to have throws that lie inside [-R,R] in each direction.
# A way to fix this: restrict the normal model so that each sampled coordinate
# also lies in [-R,R]. Technically, this means sampling from a random variable
# from a normal distribution, **conditional** on it lying with [-R,R].

# A simple way to achieve this is called **rejection sampling**. That is, just
# sample from a normal distribution as usual, but if the draw lies outside of
```

```
# [-R,R], then toss it out and resample, until you find a draw that lies in this
# interval. Write a function rnorm.reject() that takes in the arguments:
# - n, mean, sd: just as rnorm() does
# - min.val, max.val: upper and lower limits for rejection sampling, i.e., the
#   goal is to return normal draws that lie in [min.val,max.val].
# First, try out a few examples to verify that your function actually works as
# claimed. Then, modify your code from  problem 3 to use rnorm.reject() for
# rejection sampling over [-R,R], for all sd values. Produce the same plot and
# answer the same questions as before.
```