# Introduction to Monte Carlo Methods

## Lecture 3

Kevin McAlister

University of Michigan

January 27, 2017

# Back to Numerical Calculus

# Numerical Integration

- Recall that last week we discussed methods for solving numerical derivatives.
- What about the other side?
- Most problems in statistics require utilizing integration.

# Numerical Integration

- Assume that the height of all males in the US is $\sim N(70, 10)$. What is the probability that I see a male that is shorter than 60 inches?
- Recall the normal distribution:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right] \mathbb{1}(-\infty < x < \infty)$$

- We can answer our question using integration:

$$P(x < 60) = \int_{-\infty}^{60} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]$$

- Can we integrate that analytically?

# Numerical Integration

- One approach to solving the above problem computationally is using interpolation methods.
- We can utilize numerical integration methods like quadrature approximations, Simpson's methods, etc.
- These are limited in usage, though, as they require knowing the functional form of the *distribution* of interest.

# Numerical Integration

- I run a casino in Branson, MO that has a $n \times k$ grid of slot machines. Old people love coming to my casino and spending their pennies in my penny slots. The number of people that show up each hour, $S \sim Poiss(\lambda)$. These individuals only look at the edge of the grid and if they see a seat, they sit down and play. Each hour, anyone currently playing slots decides to either stay where they're at, move one seat away in any direction, or leave the slots, with equal probability. If the edge is totally full at time $t$, then no new people can play. How long will it take for the edge of the slot floor to fill up? What is the probability that my casino can accept new busses of old people for more than 40 hours?
- Time to failure follows a distribution. What distribution is this?
- No idea.

# Monte Carlo Methods

- One approach to abstract problems is Monte Carlo simulation.
- Monte Carlo simulation has the following structure:
  1. Think about your system of interest. What distributions dictate your system?
  2. Randomly draw a sample from each distribution.
  3. Compute the quantity of interest.
  4. Repeat this a large number of times.
- This procedure is very general and can be used to solve just about anything in applied statistical modelling!

# Monte Carlo Methods

- Statistics is the study of uncertainty.
- Any quantity that we compute in a statistical environment is associated with some amount of uncertainty.
- Assume we have $N$ i.i.d draws from a normal distribution:

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

$$S_x = \sqrt{\frac{\sum(x_i - \bar{x})}{N-1}}$$

$$var(\bar{x}) = \frac{S_x}{\sqrt{N}}$$

$$\mu \sim N(\bar{X}, var(\bar{X}))$$

# Monte Carlo Methods

- We know these results from asymptotic theory.
- What about other statistics?
  - Median
  - Quantiles
  - Trimean
  - Harmonic mean
- Many statistics don't have easily attainable asymptotic distributions.
- Monte Carlo simulations make quantifying uncertainty much easier in these cases.

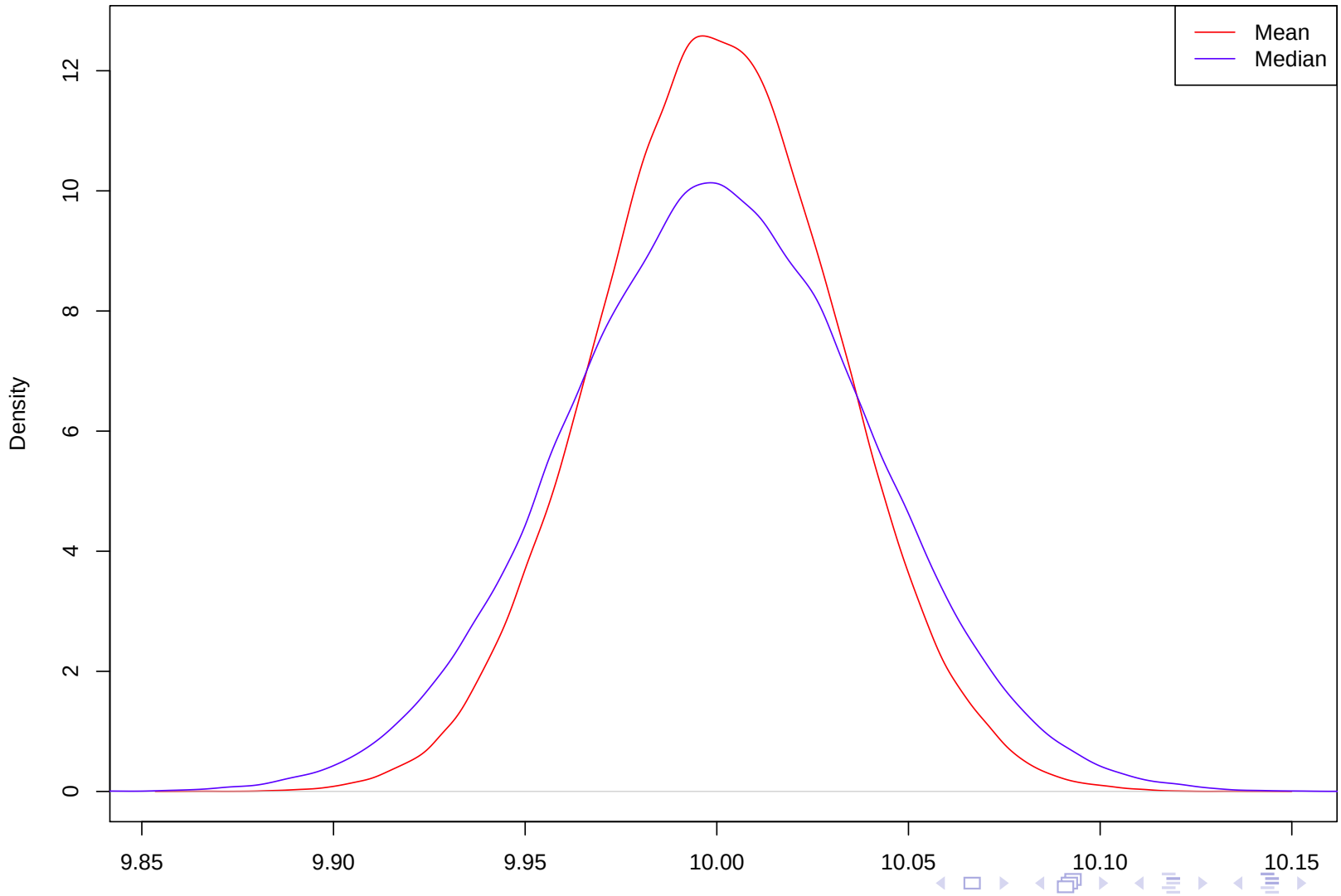
# Seeing is Believing

- Start with $X \sim N(10, 1)$
- Say we draw 1000 samples. What is the standard error of our estimate of the mean?
- Now write a function in `R` that draws 1000 samples and calculates the mean 100000 times. What is the standard deviation of your draws?
- A good function for replications in `R` is `replicate(n, FUN)`. Faster than a for loop.
- Now write a function that calculates the median and replicate it 100000 times. What is the distribution of your resulting variable?
- Plot the density of your estimated means with the density of your estimated medians? Do the distributions look the same?

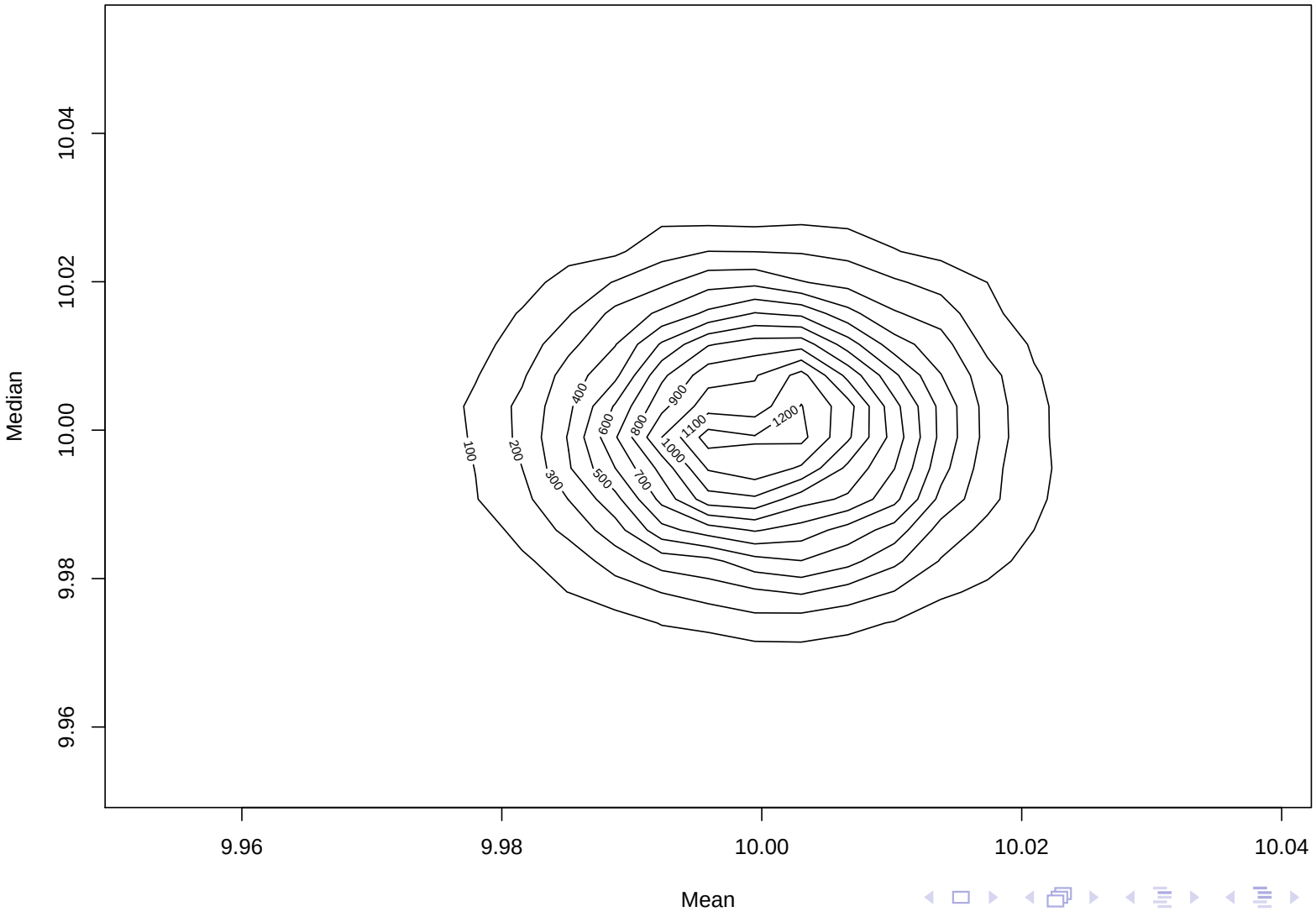
# Seeing is Believing

```
rm(list=ls())
calc.mean <- function(n){
  return(mean(rnorm(n,10,1)))
}
calc.median <- function(n){
  return(median(rnorm(n,10,1)))
}
mds <- replicate(1e5,calc.median())
mns <- replicate(1e5,calc.mean())
plot(density(mns),col = "red", main = "")
lines(density(mds), col = "blue")
legend("topright",c("Mean","Median"),
    lty = c(1,1), col = c("red","blue"))
contour(kde2d(mns,mds),xlab = "Mean", ylab = "Median")
```

# Seeing is Believing



# Seeing is Believing



# A Little More Difficult

- Jason Street gets bored during the day and wheels around town. He starts at his house and chooses to visit a truant Tim Riggins, bother the hard working Tyra Collette, yell at little Matty Saracen while he throws footballs, or stare at Lyla Garrity through her window as she's trying to move on with her life with equal probabilities. After doing one of these things, he then chooses to visit another person or go home, each with equal probabilities. He continues doing this until he decides to go home. Using Monte Carlo to estimate, what is the distribution of the number of young women that Jason chooses to visit? What is the probability that he visits a lady more than 5 times?

# A Little More Difficult

```
rm(list=ls())
jason.sim <- function(){
  peeps <- c("TR","TC","MS","LG")
  visits <- c()
  visits[1] <- sample(peeps,1)
  zz <- 0
  cc <- 1
  while(zz == 0){
    cc <- cc + 1
    next.visit <- sample(c(peeps[peeps != visits[cc - 1]],"H")
                         ,1)
    visits <- c(visits,next.visit)
    if(next.visit == "H"){
      zz <- 1
    }
  }
  visits <- visits[visits == "LG"|visits == "TC"]
  return(length(visits))
}
reps <- replicate(1e5,jason.sim())
```

# How it works

- Monte Carlo is simple!
- The only limit is how fast your computer can generate random samples from distributions.
- This statement seems simple, but there's a lot going on.
  1. Random Samples
  2. From Distribution

# Random Values

- I bet you've never thought about how a computer generates "random" values before.
- A Monte Carlo simulation is only as good as our ability to randomly draw values.
- If not random, we introduce bias.
- Most languages have a routine for generating a random number between 0 and 1.
- There is no way for a computer to generate truly random values.
- Use a seed value that passes into an algorithm and produce a series of unique values between 0 and machine max. Map them to unit line.
- Generally $2^{63} - 1$ on 64-bit systems.
- 9223372036854775807
- That's huge!

# Random Values

- Even if the domain of possible values is large, there is still an algorithm that dictates the order in which values arise.
- Algorithms are inherently predictable.
- Divide unit line into $2^{63} - 1$ even pieces. Given drawn value, return corresponding value on unit line.
- Say the algorithm returns $2, 4, 6, ...,$, what is the issue?
- There is a whole subset of CS that is dedicated to RNGs.
- Fortunately, `R`'s RNG is very good.
- Mersenne Twister is a gold standard. Has a very long sequence length.

# Drawing from Distributions

- We can draw random values from unit line. Easy peasy.
- How do we get from uniform values to a random draw from the normal distribution?
- Many methods. Many really difficult methods.

# Discrete Distributions

- Simplest problem.
- $K$ atoms in PMF.
- Divide unit line into $K$ parts proportional to probability of getting each value.
- Draw random value.
- Map random value to PMF value using $K$ segments.

# Inverse CDF Trick

- What is the probability that I draw 3.2 from a normal distribution?
- Zero!
- We can divide the unit line into infinite parts.
- Drawing from continuous PDFs is really hard.
- One trick utilizes CDFs.

# Inverse CDF Trick

- Recall that a CDF is a one-to-one function with domain $X$.
- $0 \leq f(x) \leq 1$.
- If a function is one-to-one, we can define an inverse function s.t. $f(x) = y$ implies $f^{-1}(y) = x$.
- CDFs are generally one-to-one functions.
- Areas of the density function with high values result in high derivative areas of the CDF.
- i.e. If there is a lot of mass around a point in the PDF, the corresponding inverse CDF has many values which map to areas local to the high density point.

# Inverse CDF Trick

- We want to draw random samples from $X \sim Exp(1)$.
  - $f(x) = \exp(-x)$
  - $F(x) = 1 - \exp(-x)$.
  - $F^{-1}(x) = -\log(1 - F(X))$
- Using the inverse CDF, draw a uniform random value, $U$, then sample from $Exp(1)$:

$$R = -\log(1 - U)$$

- Write a function in R that samples $N$ values from this distribution.

# What if that doesn't work?

- Sometimes, the inverse CDF isn't easy to find.
- Move to more complicated methods of sampling.
  - Accept-Reject
  - Importance
  - MCMC
- Sampling from a distribution is difficult!
- Fortunately, `R` has a full suite of sampling algorithms: `r.(n,params)`

# Point Estimates and Confidence Intervals

- Given a set of Monte Carlo replicates, we can an estimate of the distribution of a statistic.
- Summarize estimates with a point estimator and some quantification of the associated error.
- Summarize with point estimates based on some loss function:
  - Mean minimizes MSE
  - Median minimizes Absolute Error
  - Mode minimizes Posterior Loss

# Point Estimates and Confidence Intervals

- We can also summarize uncertainty using MC output.
- Given large $N$, find 2.5% and 97.5%.
- Equivalent to 95% confidence interval.
- What is the probability that our confidence interval contains the true value of our quantity of interest?
- Links to next week's discussion on resampling methods.

# Monte Carlo Error

- Let's go back to estimating the mean value from a given distribution.
- We know that the standard error associated with a mean estimate decreases at $\sqrt{N}$ rate.
- With Monte Carlo estimates, there's a little more error involved.
- Say that we take sample sizes of 10000. We expect that the standard error of the mean is $\frac{1}{100}$.
- Using your previously made function, take 10000 samples from a normal distribution and calculate the mean 100 times. What is the standard deviation? Now take it 1000 times. And then 10000. Does it get close to the expected SD?
- This implies that we should take as many samples as possible when using Monte Carlo.