**Distributions to use:**

Normal - look at general performance

Poisson - standard error depends on true value of mean - so look at performance of BCa in relation to other methods

Gamma - skewed, non-symmetric distribution - see how all methods perform

**Methods:**

BCa - should deal with poisson better, better at low n and B

Percentile - should not deal well with poisson or gamma or low n or B

t - similar to percentile I think, but should be a bit better

smooth - same as percentile probably? Will widen CI I think, better at low n too

VERSION 1

**Functions I need:**

1. **Samples from data and creates bootstrap estimates**
   * needs data, B, n, method, h for smooth
   * n changes, different size vector of estimates. How do I store it?
   * list?
2. **Finds alpha for right method**
   * needs alpha, estimates, data
   * alpha different for bca and t from perc
   * need their own fns
3. **Gets CI using the right alpha - quantile on estimates**
   * needs alpha, estimates
   * t CI different - will need own fn
4. **Simulation function - have B change and n stay constant, vice versa**
   * B and n are vectors
   * Need inputs for B and n when the other is changing
   * Generate data from right dist
   * Don't want method as a vector input, would rather pick and choose one at a time
   * data as an input
   * will need alpha to pass to alpha fns
5. **Coverage** 
   * will need simulation object and true mean value
   * checks each CI and see if true mean in that interval

**b.ests.np(data, B, B.v, n, n.v, method, h)**

* for loop for each level in B.v
  + replicate sample B.v[i] from data
  + add noise if smooth
  + apply mean on columns of replication
  + BUT if t, apply t fn instead
  + store mean vector as element in list for B
* for loop for each level in n.v
  + replicate sample n.v[j] from data
  + add noise if smooth
  + apply mean on columns of replication
  + BUT if t, apply t fn instead
  + store mean vector as element in list for n
* keep for loops separate so that B.v an n.v can be different length
* combine lists in another list
* return list

**noise fn(x, h)**

* use in apply across rows and columns in b.ests.np
* adds rnorm(0,h) to each entry

**t fn(x, data)**

* use in apply across cols in b.ests.np
* generates t statistics for each sample
* (mean(sample)-mean(data))/sd(sample)

**zhat fn(est, boot.est)**

* will need to perform get.zhat0 on each list independently
* use 2 vectors to store zhats for B and n
* combine them in list

**ahat - unchanged**

**alpha.perc(alpha)**

* list of 2 vectors a.B<-a.n<-list(c(alpha, 1-alpha))

**alpha.bca(alpha, est, data, boot.est)**

* get zhats from get.zhat0
* get ahat from get.ahat
* for loop1 : get right zhat for level of B and calculate correct percentiles
* store in list
* for loop2: get right zhat for level of n and calculate correct percentiles
* store in list
* combine lists

**CI.bca(alpha, boot.est)**

* for loop to get right alpha for B level - use lapply with quantile
* for loop to get right alpha for n level - use apply with quantile
* combine these CIs together in a list

**UPDATE -**  going to look for another way to do this, I'm not happy with the large amounts of for loops being used. Must be a better way than using so many lists.

**UPDATE -** I could just keep B and n as scalars in b.ests.np. Then just introduce B.v and n.v into the simulation function, so it just calls the bootstrap function #sims times for B = 10, stores this, and then moves on to B = 20 etc. Do same for n. This will get rid of the for loops and lists. Yay!

VERSION 2

**UPDATE -** getting coverage of 1 a lot of the time, even for large numbers of simulations. I'm happy with how I'm creating my coverage is being calculated, must be how I'm creating my intervals.

**UPDATE -** Realised I can generate a random sample from rnorm etc. of size n instead of sampling from a single dataset, so need data generating fn and need to get rid of data arg in do.sim.np. - FIXED coverage=1 problem! :-)

**UPDATE -** ggplot fn will not work. )-:

**b.ests.np(B = 99, data, h = sd(data)/3, method)**

* replicate B samples of size = length(data) - ~~use replicate~~
* creating a matrix is actually faster
* Calculate mean on each sample
* if using a smooth, add noise
* if method is t, calculate t statistics instead of mean
* return vector of bootstrap estimates

**get.zhat0**

* change fun to mean

**get.ahat**

* unchanged

**alpha.perc(alpha)**

* return a vector of c(alpha, 1-alpha)

**alpha.bca(alpha, est, data, boot.est)**

* get zhat from get.zhat0
* get ahat from get.ahat
* calculate alpha1 and alpha2 using equations from lecture slides
* return vector of alpha1 and alpha2

**CI(alpha, data, boot.est)**

* quantile on boot.est with the right alpha for BCa, smooth or percentile

**CI.T(alpha, est, data, boot.est)**

* quantile on boot.est using percentiles from percentile method but with (1-alpha, alpha) instead
* then calculate mean(data)-sd(data)\*quantiles

**bootstrap.ci(B, data, alpha = 0.025, method, dist = "normal", h = sd(data)/3)**

* will need to error check data and h now since they'll have values by now
* get bootstrap estimates - call b.ests.np
* get est - mean(data)
* get right alpha for the method - use switch and do.call
* get right CI for method - switch and do.call
* return CI

**input.checks(sims, B.v, B, n.v, n, alpha, method, dist, mean, sd, lambda, shape, rate)**

* check all these inputs in do.sim.np since they'll be assigned values by user
* sims - scalar, integer, positive non-zero, numeric
* B.v - numeric, all entries are positive non-zero
* n.v - numeric, all entries are positive non-zero
* n - scalar, integer, positive non-zero, numeric
* B - scalar, integer, positive non-zero, numeric
* alpha - scalar, numeric, between 0 and 1
* method - character, is it one of bca, percentile, t and smooth
* dist - character, is it one of normal, poisson and gamma
* mean - numeric, scalar
* sd - numeric, scalar, positive non-zero
* lambda - numeric, positive non-zero
* shape- numeric, positive non-zero
* rate - numeric, positive non-zero

**data.gen(n, dist, mean, sd, lambda, shape, rate)**

* error check n
* get correct random deviate generating function for the distribution given - use switch and do.call
* use correct arguments for random generating function - switch
* get n deviates

**do.sim.np(sims, B.v, B, n.v, n, alpha, method, dist, mean, sd, lambda, shape, rate, check, h)**

* error check "check" - is it T/F?
* for loop for B in B.v
* nested for loop for i in 1:sims
  + generate data for the required distribution - data.gen
  + get CI for this data and for this B in B.v - bootstrap.ci
  + How to store? Matrix?
* for loop for n in n.v
* nested for loop for i in 1:sims
  + generate data for the required distribution - data.gen
  + get CI for this data and for this n in n.v - bootstrap.ci
  + store in matrix
* combine these two matrices in a list
* UPDATE: will need to return levels of B and n, the distribution of the data and the method, makes it easier to construct a data frame so I can use ggplot2

**coverage(true\_mean, simulation)**

* will need to extract some info from the simulation
* How many levels of B and n were there in the simulation?
* How many simulations were there?
* Nested for loop, for this level of B and for this simulation
  + is true\_mean in the CI?
  + store in vector
  + vector of length sims
  + sum the TRUEs in this vector and divide by its length = COVERAGE
* Outside the sim loop but in B loop, place this proportion in vector
* Do same for n
* combine in these two coverage vectors in list along with distribution and method

SIMULATION FUNCTIONS DONE!

**Plot functions**

**format.data(cov\_objects)**

* need list of outputs from coverage
* for each element in list, take the levels of B and n, the coverages for each of them respectively, the distribution and method used and creates a data frame
* B and n must be of same length

**plot.fn(df, x, y, method.plot, dist.plot )**

* needs data frame from format.data
* do I want a plot comparing methods for one distribution? or a plot comparing how each distribution fares for each method?
* if method.plot =T or dist.plot=T, then get right ggplot
* **UPDATE -** doesn't seem to want to work and I just can't see why, I can take the code out of the function and enter in the arguments I want and run as plain code and it works. But if I enter the same arguments into the function I get an error. )-:<