Work Learn 2021 Presentation

Meiying Ding

August 24, 2021

Presentation Agenda

- Introduction
- Data Simulation and Weibull Distribution Exercises
- Cluster Effect Calculations and the Practice R Package **clustertest**
- R Package **permutest**

Introduction

- Project Objectives: To monitor the breaking strength (especially the Modulus of Rupture) of lumber over the years, especially their 5th percentile, median, and mean. To conduct permutation hypothesis tests for changes in the mean or quantiles of clustered data collected via a rotating sample design. To create an associated R Package.
- Progress Breakdown: Understanding Project Goals, Reproducing and Enhancing the Simulations, Reading Chen et al.'s Paper, Understanding and Calculating Cluster Effect, Practicing Writing an R Package, Writing the **permutest** R Package, Creating Presentation.

Weibull Distribution Exercises

(More Weibull Exercises See Overleaf or OneDrive)

- Weibull Exercises Example:

Part I:
Using the property of PDF:

$$\int_{-\infty}^{\infty} f(t)dt = 1,$$

we set

$$\int_0^{\infty} C\left(\frac{t}{\lambda}\right)^{\beta-1} e^{-\left(\frac{t}{\lambda}\right)^{\beta}} dt = 1,$$

here Weibull PDF is defined for $x \geq 0$.

Solve the integral by substituting $u = -\left(\frac{t}{\lambda}\right)^{\beta}$, we obtain $-\frac{\lambda}{\beta}C\int_0^{-\infty} e^u du = \frac{\lambda}{\beta}C = 1$, so $C = \frac{\beta}{\lambda}$. Therefore, the Weibull PDF is

$$f_X(x \mid \lambda, \beta) = \begin{cases} \frac{\beta}{\lambda}\left(\frac{x}{\lambda}\right)^{\beta-1} e^{-\left(\frac{x}{\lambda}\right)^{\beta}}, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2}$$

Part II:
Next, find expectation of a Weibull random variable.

$$\begin{aligned} E(X \mid \lambda, \beta) &= \int_0^{\infty} t f(t \mid \lambda, \beta)dt \\ &= \int_0^{\infty} t\frac{\beta}{\lambda}\left(\frac{t}{\lambda}\right)^{\beta-1} e^{-\left(\frac{t}{\lambda}\right)^{\beta}} dt \\ &= \int_0^{\infty} t\frac{\beta}{\lambda}\frac{\lambda}{\beta}e^{-y}dy \qquad \text{where } y = \left(\frac{t}{\lambda}\right)^{\beta} \\ &= \int_0^{\infty} \lambda y^{\frac{1}{\beta}} e^{-y}dy \\ &= \lambda \cdot \Gamma\left(1 + \frac{1}{\beta}\right) \qquad \text{where } \Gamma(n) = \int_0^{\infty} e^{-y}y^{n-1}dy \text{ is the Gamma function} \end{aligned}$$

Part III:
Find variance of a Weibull random variable by first finding the second mo-

Data Simulation

Specification 1: to set some input parameters as default values so that users would not need to input the arguments if they are not different from the default values (including the three moderators for random effects). Similar simulation functions for gamma and general data are omitted here.

- Normal Data Simulation:

```
# Long Form Output to Normal Data Simulation in One Function
# Model: y_{k,i,u} = mu_{k} + sigma_{k,1} * eta_{i} + sigma_{k,2} * eta_{k,i} + sigma_{k,3} * epsilon_{k,i,u}

# Parameters:
#   Compare to Permu.test-separate file's original NormalObs.gen function, removed sigma123, sz, and added no.y, no.c, no.d, size.c.
#   Three moderators default to be unchanged for occasions as in Original Example 1's values, but user can input vectors if needed.
#   Boolean values for the random effects (eta_{i}, eta_{k,i}) follow standard normal distributions if input TRUE,
#         rep(someconstant, somesize) if input FALSE. Assume epsilon_{k,i,u} follows standard normal distribution.
#   Indicator matrix and sz defaults to scenarios with balanced samples, but user can input c(c0, d0) to specify the imbalance.

# Added outputlong to the return list (same data as the returned obs but in easier to read data.frame form). See examples below to view outputlong's format.


NormalObs.gen<-function(mu, no.y, no.c, no.d, size.c, modcluster, modoccasion, modpiece,
                        indicator, ceffect, vceffect)
{
  if (missing(modcluster)) {
    sigma1 = rep(1,no.y)
  } else {
    sigma1 = modcluster
  }

  if (missing(modoccasion)) {
    sigma2 = rep(2,no.y)
  } else {
    sigma2 = modoccasion
  }

  if (missing(modpiece)) {
    sigma3 = rep(3,no.y)
  } else {
    sigma3 = modpiece
  }

  if(missing(indicator)) { # balanced sample
    indicator = NULL
    for(i in 1:no.y) {
      tmp=c(rep(0, (i-1)*no.d),rep(1, no.c), rep(0, (no.y-i)*no.d))   # generate row i in the indicator matrix each time,
      indicator =  rbind(indicator, tmp)                              # change tmp if the structure of adding and dropping specific mills need to be different
      cluster.total = no.c+(no.y-1)*no.d
```

Normal Data Simulation Examples

- Normal Data Simulation:

```
# Original Example 1 in Permu_test (balanced) default moderator values
# Balanced means equal number of mills (clusters) each year, equal number of clusters dropped and added
# each year and equal cluster size across all clusters. Sample size in each cluster keeps unchanged over years if the cluster stays
# in the sample.

data1 <- NormalObs.gen(mu = c(8, 8+0.4, 8+0.5*rnorm(no.y-2)),
                       no.y = 5, no.c = 36, no.d = 6, size.c = 5, # leave as default values for the three moderators
                       ceffect = TRUE, yceffect = TRUE) # specify that there are (normal) random effects eta_{i} and eta_{k,i}.
View(data1)
outputlong1 <- as.data.frame(t(data1[[4]]))
colnames(outputlong1) <- paste("Year",c(1:no.y), sep = "_")
View(outputlong1)
```

- Normal Data Simulation:

```
# Original Example 2 in Permu_test (imbalanced) except the three moderators' values are also changing across occasions.
# Imbalanced means unequal number of mills (clusters) each year, 4 to 7 mills randomly dropped each year, 4 to 7 mills randomly added each year,
# unequal cluster sizes across all clusters. Sample size in each cluster keeps unchanged over years if the cluster stays
# in the sample.

data2 <- NormalObs.gen(mu = c(8,7.9,7.9,7.8,7.7),
                       no.y = 5, no.c = 33, no.d = 3, size.c = 3,
                       modcluster = c(1,2,3,4,5), modoccasion = c(2,3,4,5,6), modpiece = c(3,4,5,6,7),
                       indicator = c(6,4),
                       ceffect = TRUE,yceffect = TRUE)
View(data2)
outputlong2 <- as.data.frame(t(data2[[4]]))
colnames(outputlong2) <- paste("Year",c(1:no.y), sep = "_")
View(outputlong2)
```

Data Simulation Con't

Specification 2: to output the data in an easy-to-use format.

- Normal Data Simulation:

```
  outputlong = NULL
  for(i in 1:no.y) {
    obs.year=list()
    obslong.year = list()
    count = 0
    count2 = 0
    for(j in 1:cluster.total) {
      count2 = count2+1
      if (indicator[i, j]) {
        count=count+1
        obs.year[[count]] = mu[i] + cluster.effect[j]*sigma1[i] +
          yc.effect[i, j]*sigma2[i]+ rnorm(sz[j])*sigma3[i] # within each mill j sample size unchanged if it remains in the sample

        obslong.year[[count2]] =obs.year[[count]]
      } else if (!indicator[i,j]){
        obslong.year[[count2]] = rep(NA, sz[j])
      }
    }
    obs[[i]] = obs.year
    obslong[[i]] = obslong.year

    Long =NULL
    for (y in 1:cluster.total) {
      Long = c(Long, list(obslong.year[[y]])) # list of observations from the same mill same year
    }
    outputlong= rbind(outputlong,Long)
  }

  data = list(obs=obs, indicator=indicator, sz=sz, outputlong= outputlong)
}
```

Data Simulation Con't

- Added Outputlong Element to the Normal Data Simulation:

	Year_1	Year_2	Year_3	Year_4	Year_5
1	c(10.4014739673762, 7.20445094713321, 6.89905...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
2	c(9.8496639067189, 6.76834390974684, 7.784007...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
3	c(9.31698398731589, 9.59681331878429, 5.42967...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
4	c(3.49889688427106, 4.98945842407765, 4.60253...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
5	c(13.4233752896591, 9.17947658137748, 10.5784...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
6	c(7.67211891404194, 11.8683873474529, 6.81115...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
7	c(5.56878917138511, 15.4012486891338, 13.1949...	c(10.1149071307695, 3.4213964035681, 11.97735...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
8	c(8.22614005668514, 10.2842458436714, 6.49555...	c(5.53421725749035, 2.74969916676947, 6.53350...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
9	c(3.98631435209151, 8.73527707121514, 0.04555...	c(2.62713683749137, 5.63870394966444, 10.7562...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
10	c(17.420894636336, 5.46046446737268, 12.04947...	c(4.85094646646801, 9.68048036997635, 1.17069...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
11	c(6.09259106960608, 9.11539027090333, 11.7755...	c(5.67000183069197, 4.23666089944926, 5.30176...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
12	c(9.50919263670397, 12.8093591388103, 6.14495...	c(-0.169828966234136, 3.45664264754388, 1.155...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
13	c(6.24192980848061, 9.36782102949901, 7.13536...	c(8.52167918189832, 15.6374709153415, 15.3869...	c(12.7670097101054, 6.14918914731942, 9.89240...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
14	c(3.3743474364625, 8.18823197159926, 5.389008...	c(3.0726444357512, 10.4251554716788, 12.24533...	c(10.2432842052705, 13.0846232625828, 6.35637...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
15	c(13.8983021863471, 11.3595542013658, 6.04911...	c(14.6520266367508, 12.3404331514441, 8.21467...	c(16.9398821222826, 13.1041707399261, 12.6175...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
16	c(10.287391863867, 1.24894487986098, 4.044055...	c(5.80472056054925, 2.09054558051767, -0.5917...	c(3.64472304626471, 3.12159013168344, 9.30723...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
17	c(7.72957888542321, 2.66033949029209, 1.04010...	c(4.87839522516789, 12.056144586824, 13.79200...	c(8.19816208966667, 6.86982148353423, 9.12887...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
18	c(14.0941394001855, 9.67231672957499, 8.06073...	c(11.389248189246, 12.3899959561887, 14.86490...	c(3.5324285960012, 8.235400345795556, 7.786521...	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)
19	c(7.17705103673616, 7.63767249897919, 4.39039...	c(13.9460803412383, 10.4546336995039, 8.13379...	c(2.7997360243046 9, 5.4459536403314, 5.650674...	c(7.25258794572155, 4.78336355864501, -0.2943...	c(NA, NA, NA, NA, NA)
20	c(6.46978087981249, 0.998879461633056, 4.1704...	c(7.64561594545107, 11.1342102354264, 14.1542...	c(10.9045919716468, -2.3755217923253, 3.67767...	c(8.08397490170575, 5.07928979444612, 2.50911...	c(NA, NA, NA, NA, NA)
21	c(6.52242159264258, 2.69523810654631, 6.84433...	c(2.40897148104904, 14.7642255757373, 7.59838...	c(4.57909177599341, -2.0603786704026, 6.84677...	c(9.68153903616922, 12.9127366375993, 12.3103...	c(NA, NA, NA, NA, NA)
22	c(7.47596556328142, 7.04141404296719, 15.8495...	c(4.36450723612807, 7.50751942620628, 3.09694...	c(6.5017445463039, 4.92766379159345, 4.164959...	c(6.88236156855718, 7.44409155752742, 2.63268...	c(NA, NA, NA, NA, NA)
23	c(4.71583516421808, 2.94767116750963, 3.62365...	c(6.10622512411535, 13.9707189066689, 3.44779...	c(9.12549691406074, 5.11824803318157, 9.78690...	c(7.653574356958 65, 10.2645384990636, 5.07525...	c(NA, NA, NA, NA, NA)
24	c(3.13848246382453, 8.96936654383484, 5.21887...	c(9.6066738230605, 9.88193812199597, 5.142963...	c(8.96564569865858, 9.87919159752405, 8.15581...	c(8.87358897821398, 12.8374655593831, 10.2138...	c(NA, NA, NA, NA, NA)
25	c(10.607581510 17, 4.338023333371867, 6.3400268...	c(10.6593411166902, 13.0588220478625, 8.31433...	c(8.68717203781185, 4.63636834074646, 6.28666...	c(11.7424783621821, 6.58673915051114, 16.0245...	c(9.33568530169201, 9.44704562493951, 7.10670...
26	c(5.33979246510731, 1.79571264159281, 6.30431...	c(-0.753304403237893, 10.8188113717146, 3.312...	c(5.30818363881584, 3.80576691678913, 9.55597...	c(10.3043008691673, 13.4567614169906, 11.7721...	c(5.61622523896456, 12.7214685980508, 9.22874...
27	c(13.4107514644917, 3.45353444057879, 5.68287...	c(9.88133168426024, 12.4954012658206, 10.1342...	c(10.5347964611335, 12.8090940416349, 8.60130...	c(6.22545480387766, 4.87954723273719, 2.88844...	c(6.90978255498003, 2.35611734624117, 9.41137...
28	c(4.53804552591856, 3.67863080680358, 5.70812...	c(3.46209766006143, 4.15952837159853, 8.15741...	c(5.61382919670128, 9.9960937932371, 6.293102...	c(5.26560143702859, 0.633665429679549, 5.9205...	c(7.04544152316722, 5.90079545028875, 11.8879...
29	c(7.48207320393665, 8.07832302670255, 11.2591...	c(10.8605003233933, 10.5008985217371, 10.2824...	c(4.03183478168338, 9.3288923986259, 8.440392...	c(7.80849107424137, 10.0018930891806, 8.08739...	c(12.1057874670772, 6.31727652462498, 3.88496...
30	c(4.78668972999976, 5.18974082493305, 9.16514...	c(6.40301593219989, 3.67751198636769, 3.31742...	c(7.50224535868638, 5.54054976664095, 9.70205...	c(7.66275535738762, 1.46199889225684, 4.62024...	c(8.99438875094713, 10.6227018863687, 11.2955...
31	c(13.9607033284845, 5.00597545203705, 9.68261...	c(8.60574770155785, 11.9387510898022, 8.86283...	c(9.67348961287462, 6.32804364004016, 9.68485...	c(4.23111580162398, 3.76548197795439, 9.37917...	c(13.4709587062419, 12.6727410557311, 7.57844...
32	c(15.2722660544431, 11.0771089314489, 14.0640...	c(9.49752604316864, 11.0280154954892, 10.8850...	c(9.32422398640094, 9.73067075995465, 8.73556...	c(10.3119706490328, 3.3856350615129, 6.782042...	c(8.29526213894163, 5.61693794516364, 13.6648...
33	c(7.49248966257874, 3.13865211629044, 7.29696...	c(3.94498124986327, 5.55263901000682, 3.71978...	c(10.0220027042986, 9.61241382079655, 8.11712...	c(10.5953391192136, 10.1403870333842, 4.06173...	c(10.4517423369016, 13.8869535083304, 15.6867...
34	c(4.36319994828245, 11.9400279693118, 5.00410...	c(4.9758167661381, -1.36134976785217, 3.10579...	c(5.4758523909693, 10.1500794810226, 7.893545...	c(7.54380751274431, 5.23137164884474, 3.89497...	c(5.05526822906529, 10.31423324426, 6.0968188...
35	c(12.9798616703008, 13.7020008512827, 10.3080...	c(4.81247785036489, 4.56284446510227, 3.00208...	c(4.63729725293092, 1.89763166080741, 5.42570...	c(5.29683391428604, 8.11694336984632, 8.07563...	c(7.69356767917229, 4.32743921215123, 3.86651...
36	c(14.4670833544333, 8.70113146158009, 5.46205...	c(11.677759971396, 10.1132471041861, 7.417779...	c(12.9572799314566, 14.6407589311165, 12.7954...	c(6.84596832517084, 4.56550492129083, 6.57108...	c(12.9163942903395, 7.87477249538499, 16.5842...
37	c(NA, NA, NA, NA, NA)	c(3.7510431651572, 7.46818937172085, 4.699186...	c(6.24407278828573, 8.7600765462832, 12.55482...	c(6.98331635397376, 10.7319050191377, 8.21535...	c(-1.38686012965344, 11.4643117645913, 7.9316...
38	c(NA, NA, NA, NA, NA)	c(6.23372844863178, 9.84646956722287, 12.9358...	c(11.7503692202939, 12.9217049661359, 14.0466...	c(10.5458121885631, 13.1434184648979, 8.76260...	c(12.4238532229774, 13.6942623922448, 4.95101...
39	c(NA, NA, NA, NA, NA)	c(14.3277870060789, 11.3492030118608, 15.1346...	c(4.62786561779667, 11.4231690389857, 14.3431...	c(6.77985323911719, 5.49074283513042, 8.65455...	c(1.74963186177296, 6.50476393099716, 7.27255...
40	c(NA, NA, NA, NA, NA)	c(11.8108184613747, 6.3801367960745, 5.463171...	c(9.58720463283099, 6.12000185733292, 4.61307...	c(10.4422003482001, 9.95690810953091, 12.4913...	c(7.48305207377715, 4.55382274339278, 7.94740...
41	c(NA, NA, NA, NA, NA)	c(7.90339049927146, 8.94591818355547, 9.09698...	c(4.62713072333378, 3.69293002312605, 6.189353...	c(6.77950923809045, 3.43491035082687, 6.78883...	c(11.1382721733305, 10.123357568083, 11.23961...
42	c(NA, NA, NA, NA, NA)	c(9.17648340942848, 10.8687602511306, 12.5595...	c(6.49417997384825, 11.1000611503576, 6.34352...	c(2.63751104635956, 3.47551500116275, 8.89126...	c(3.19366093479918, 9.28698421034579, 6.28756...
43	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(6.43862296830554, 8.82647039713834, 12.5225...	c(12.7050355642638, 13.9031901114639, 15.4763...	c(2.46248039401044, 9.97614511975117, 8.09273...
44	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(10.4639794691689, 6.13082604773233, 13.2999...	c(10.1954172686974, 4.98354159648623, 4.78381...	c(7.20638310899502, 8.65808069528138, 8.59878...
45	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(6.44463741982743, 7.34867739441836, 9.22088...	c(9.51800201505806, 5.70959532285932, 7.72186...	c(13.7855423920973, 11.0212983161183, 11.4696...
46	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(0.9130996427269, 7.27651971789877, 1.254328...	c(4.48362162739103, 2.65623553761359, 7.80270...	c(12.0806318664976, 5.53924744748794, 7.50801...
47	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(8.21574345643704, 1.01765500813316, 4.19723...	c(12.051267941244, 13.9551969872292, 8.584968...	c(18.5337125413944, 15.6698787651535, 11.1087...
48	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(2.50870321027854, 4.63112234201148, 1.16920...	c(9.94312089205742, 9.89035018327873, 7.35461...	c(1.6983182007499, 4.87981601001945, 4.548999...
49	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(6.24657953676917, 7.0823855041001, 10.94122...	c(9.71494052145142, 7.2244736319451, 6.886815...
50	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(NA, NA, NA, NA, NA)	c(13.4632276368272, 2.13034251487773, 0.69431...	c(4.08180186163597, 4.02398961334907, 9.17862...

Data Simulation Con't

- Versus Original Normal Data Simulation:

data1	list [4]	List of length 4
obs	list [5]	List of length 5
[[1]]	list [36]	List of length 36
[[1]]	double [5]	10.40 7.20 6.90 9.86 8.43
[[2]]	double [5]	9.85 6.77 7.78 9.81 12.52
[[3]]	double [5]	9.32 9.60 5.43 8.74 5.90
[[4]]	double [5]	3.50 4.99 4.60 8.81 8.15
[[5]]	double [5]	13.42 9.18 10.58 13.70 13.93
[[6]]	double [5]	7.67 11.87 6.81 6.45 7.74
[[7]]	double [5]	5.57 15.40 13.19 14.74 5.52
[[8]]	double [5]	8.23 10.28 6.50 12.00 5.02
[[9]]	double [5]	3.9863 8.7353 0.0456 8.5539 3.9628
[[10]]	double [5]	17.42 5.46 12.05 13.84 12.91
[[11]]	double [5]	6.09 9.12 11.78 8.28 11.58
[[12]]	double [5]	9.51 12.81 6.14 11.14 1.68
[[13]]	double [5]	6.24 9.37 7.14 1.40 3.61
[[14]]	double [5]	3.37 8.19 5.39 4.98 10.04
[[15]]	double [5]	13.90 11.36 6.05 14.66 4.87
[[16]]	double [5]	10.29 1.25 4.04 4.80 7.49
[[17]]	double [5]	7.73 2.66 1.04 1.28 11.31
[[18]]	double [5]	14.09 9.67 8.06 7.51 11.03
[[19]]	double [5]	7.177 7.638 4.390 9.421 −0.175
[[20]]	double [5]	6.470 0.999 4.170 3.228 8.903
[[21]]	double [5]	6.52 2.70 6.84 8.31 8.66
[[22]]	double [5]	7.48 7.04 15.85 1.31 6.97
[[23]]	double [5]	4.72 2.95 3.62 2.58 −1.14
[[24]]	double [5]	3.14 8.97 5.22 13.72 10.15
[[25]]	double [5]	10.61 4.34 6.34 7.01 2.81
[[26]]	double [5]	5.34 1.80 6.30 3.93 1.53
[[27]]	double [5]	13.41 3.45 5.68 8.59 7.34
[[28]]	double [5]	4.54 3.68 5.71 3.08 9.00
[[29]]	double [5]	7.482 8.078 11.259 0.583 11.043
[[30]]	double [5]	4.79 5.19 9.17 7.93 8.98
[[31]]	double [5]	13.96 5.01 9.68 5.48 4.58
[[32]]	double [5]	15.27 11.08 14.06 8.45 4.08
[[33]]	double [5]	7.49 3.14 7.30 9.36 7.47
[[34]]	double [5]	4.36 11.94 5.00 8.81 6.52
[[35]]	double [5]	12.98 13.70 10.31 9.04 7.78
[[36]]	double [5]	14.47 8.70 5.46 5.13 6.52
[[2]]	list [36]	List of length 36
[[3]]	list [36]	List of length 36
[[4]]	list [36]	List of length 36
[[5]]	list [36]	List of length 36
indicator	double [5 x 60]	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 ...
sz	double [60]	5 5 5 5 5 5 ...

Simulation Model Summarization

1.2 Gamma distribution based data

In this example, let $\boldsymbol{y}_{k,i}$ be the response vector of the ith mill sampled in occasion k is assumed to have the following structure:

$$\boldsymbol{y}_{k,i} = \lambda_k(\epsilon_i + \epsilon_{k,i} + \boldsymbol{x}_{k,i}).$$

In this expression, ϵ_i is random and it is associated with mill i: whenever a wood piece is produced by mill i, λ_k times this value is added to the final response vector $\boldsymbol{y}_{k,i}$, regardless of the occasion k or specific piece u. Also, it is mill specific: different mills have different ϵ_i values. Hence, it is longitudinal mill effect. We choose $\epsilon_i \sim Gamma(\gamma_1, 1)$, where γ_1 is a constant. In R, rgamma(cluster.total, γ_1)[i] corresponds to the simulated ϵ_i $\forall i = 1, ...$

Similarly, $\epsilon_{k,i}$ is random and it is occasion and mill specific. The term $\lambda_k\epsilon_{k,i}$ is added to the response vector $\boldsymbol{y}_{k,i}$ only if the pieces are from the kth occasion and mill i. If two wood pieces are taken from two different occasions or two different mills, they have different $\epsilon_{k,i}$ values. We assume $\epsilon_{k,i} \sim Gamma(\gamma_2, 1)$ where γ_2 is a constant. In R, each rgamma(1, γ_2) corresponds to the simulated value for cluster i on occasion k.

Finally, $\boldsymbol{x}_{k,j}$ is a vector of independent random variables with distribution

Simulation Model Summarization Con't

$Gamma(\eta_k, 1)$. η_k is a degrees of freedom vector, and $\boldsymbol{x}_{k,j}$ is specific for each specific piece u. $\boldsymbol{x}_{k,j}$ $\forall k, j$ can be simulated in R using rgamma(cluster.size, η_k).

$$\boldsymbol{x}_{k,j} = \begin{bmatrix} x_{k,j,1} \\ x_{k,j,2} \\ \vdots \end{bmatrix} \tag{1}$$

These account all random parts in the above model. We will make all random parts of $\boldsymbol{y}_{k,i}$ independent of each other and have Gamma distribution with rate parameter of 1 and different degree of freedom parameters. The non-random part of the model is λ_k.

1.3 General distribution based data

In this example, let $\boldsymbol{y}_{k,j}$ be the response vector of the ith mill sampled in occasion k is assumed to have the following structure:

$$\boldsymbol{y}_{k,i} = \varphi(x_{k,i,1}, ..., x_{k,1,r}; \epsilon_{k,i}, \epsilon_i).$$

In this expression, $\varphi(\cdot; \epsilon_{k,i}, \epsilon_i)$ is a symmetric r-dimensional vector function. For example, in the simulation, we let

$$\varphi\left(x_{k,i,1}, \ldots, x_{k,i,r}; \epsilon_{k,i}, \epsilon_i\right) = \left(\lambda_{k,i,1} x_{k,i,1}, \ldots, \lambda_{k,i,r} x_{k,i,r}\right),$$

where $\lambda_{k,i,1}, \ldots, \lambda_{k,i,r} \overset{iid}{\sim} Gamma(20, rate = 20)$. In particular, $x_{k,i,1}, \ldots, x_{k,i,r}$ is a biased sample from the finite population $\mathcal{P} = \{x_1, x_2, ...\}$ with selection probability proportional to

$$b_{k,i}(x) = \exp\{\sigma_1 \epsilon_i + \sigma_{k,2} \epsilon_{k,i} \log(x)\}$$

Simulation Model Summarization Con't

for some positive constants σ_1 and $\sigma_{k,2}$.

We choose random ϵ_i to be associated with mill i. Also, it is mill specific: different mills have different ϵ_i values. Hence, it leads to longitudinal mill effect. We make $\epsilon_i \overset{iid}{\sim} N(0,1)$.

Similarly, $\epsilon_{k,i}$ is random and it is occasion and mill specific. If two wood pieces are taken from two different occasions or two different mills, they have different $\eta_{k,i}$ values. In the simulation, we make $\epsilon_{k,i} \overset{iid}{\sim} N(0,1)$. We will do the same in the R-function.

(3D-Array Output Functions on OneDrive)

Single Occasion Cluster Effect Calculation

Let Y_{ij} represent the continuous random variable for the response value of the jth piece of lumber in mill i, where $i = 1, 2, ..., M$, $j = 1, 2, ..., n$. Let $\boldsymbol{\mu}_i \overset{iid}{\sim} N(0, \sigma_m^2)$ denote the mill random effect, and $\boldsymbol{\epsilon}_{ij} \overset{iid}{\sim} N(0, \sigma_e^2)$ denote the piece random effect. We further assume mill effect and piece effect are independent. Let

$$Y_{ij} = \mu + \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_{ij}$$

for some nonrandom constant μ.

Based on this construction, the total variance is

$$Var(Y_{ij}) = Var(\mu + \boldsymbol{\mu}_i + \boldsymbol{\epsilon}_{ij}) = \sigma_m^2 + \sigma_e^2.$$

One appropriate metric of the cluster effect's contribution to the total variance is the intra-class correlation coefficient (ICC) (Donner and Koval, 1980). The ICC is defined as $\frac{\sigma_m^2}{\sigma_m^2 + \sigma_e^2}$. If the mill effect has little impact on the outcome then this ratio should be close to 0.

Single Occasion Cluster Effect Calculation Con't

Upon derivations...

(Detailed Derivations See Overleaf or OneDrive)

$$\hat{\sigma}_e^2 = \frac{\widehat{Var}(Y_{ij} - \bar{Y}_{i\cdot})}{1 - \frac{1}{n}}.$$

Finally, we substitute $\hat{\sigma}_e^2$ into Eq.(1) to obtain

$$\hat{\sigma}_m^2 = \frac{1}{1 - \frac{1}{M}}[\widehat{Var}(\bar{Y}_i - \bar{\bar{Y}}) - (1 - \frac{1}{M})\frac{\widehat{Var}(Y_{ij} - \bar{Y}_i)}{n(1 - \frac{1}{n})}].$$

Then, the ICC can also be estimated by

$$\frac{\hat{\sigma}_m^2}{\hat{\sigma}_m^2 + \hat{\sigma}_e^2}.$$

Multiple Occasion Cluster Effect Calculation

Let Y_{kij} represent the continuous random variable for the response value of the jth piece of lumber in mill i on occasion k, where $i = 1, 2, ..., M$, $j = 1, 2, ..., n$, $k = 0, 1, ..., K$. Let $\boldsymbol{\eta}_i \overset{iid}{\sim} N(0, \sigma^2_m)$ denote the mill random effect, $\boldsymbol{\eta}_{ki} \overset{iid}{\sim} N(0, \sigma^2_{mo})$ denote the random effect specific to cluster i on occasion k, and $\boldsymbol{\epsilon}_{kij} \overset{iid}{\sim} N(0, \sigma^2_e)$ denote the piece random effect. We further assume $\boldsymbol{\eta}_i$, $\boldsymbol{\eta}_{ki}$, and $\boldsymbol{\epsilon}_{kij}$ are mutually independent.

Let

$$Y_{kij} = \boldsymbol{\mu}_k + \boldsymbol{\eta}_i + \boldsymbol{\eta}_{ki} + \boldsymbol{\epsilon}_{kij}$$

for some nonrandom constants $\boldsymbol{\mu}_k$.

Upon similar derivations as for the single occasion case, we can find estimates of $\hat{\sigma}^2_m$, $\hat{\sigma}^2_{mo}$, and $\hat{\sigma}^2_e$, and obtain the ICC. (Detailed Derivations See Overleaf or OneDrive)

Practice R Package **clustertest**

- Written for practice purpose:
  - how to create a package with R Studio (Hadley Wickham's Book - "R Packages")
  - how to sync R Studio with Git and GitHub
  - how to document each R function with Roxygen comments
  - how to add examples for each exported function
  - how to test functions in a package
- Contains one function to calculate single-occasion's ICC accompanying the derivations above, input is real world data: ex. NLGA_LTM_data_transformed.

clustertest Package in Private GitHub Repository "clusterrepo"

- Also attempted to write a function to calculate multiple-occasion's ICC, input is simulated dataset: ex. data generated by NormalGenFun().

Single and Multiple Occasion ICC functions both on OneDrive

Cluster Effect Function

cluster effect:

```r
# This is the same function as SingleOccasionClusterEffectFunction.R in a package format,
# which calculates the Variance of Cluster Effect and Cluster Effect's Contribution to the Total Variance
# for a single occasion data.

#' Calculate Variance of Cluster Effect and Cluster Effect's Contribution to the Total Variance
#' @param y column name of response in the dataframe data
#' @param cluster.by column name indicating cluster id in the dataframe data
#' @param data a dataframe containing y and cluster.by
#' @param year year of the data to be analyzed
#' @return a list where the first element is the estimated variance of the cluster effect and the second element is the ICC
#' @examples
#' cluster_effect(y = "MOR",cluster.by = "MillID", data = NLGA_LTM_data_transformed, year = 2)
#' @export
cluster_effect <- function(y, cluster.by, data=NULL, year) {
  # cluster.by is a string of the column name that contains ids representing different clusters
  # y is the column name indicating measurements in the data.frame data
  Myear <- NULL
  singleyrdata <- subset(data, Myear==year)
  singleyrdata[[cluster.by]] <- as.factor(singleyrdata[[cluster.by]])
  # Substitute $ with [[]] https://stackoverflow.com/questions/2641653/pass-a-data-frame-column-name-to-a-function
  singleyrdatao <- singleyrdata[order(singleyrdata[[cluster.by]]),] # order by increasing mill ID to match with y_bar_i_dot's ordering
  y_ij <- singleyrdatao[[y]]
  y_bar_i <- tapply(singleyrdatao[[y]], singleyrdatao[[cluster.by]], mean) # avg response for pieces in mill i
  cluster_size <- tapply(singleyrdatao[[y]], singleyrdatao[[cluster.by]], length) # each mill sample size (all equal in this case = n)
  n <- mean(cluster_size) # number in each cluster TODO: Tmp set to mean
  M <- length(y_bar_i) # number of clusters
  y_bar_i_dot <- rep(y_bar_i, times=cluster_size) # replicate average values for each mill n times each to match y_ij's length
  sigsqr_e <- stats::var(y_ij-y_bar_i_dot)/(1-1/n) # see formula derivation in overleaf
  y_bar_bar <- mean(y_ij) # overall average response of all pieces
  y_bar_bar_dot_dot <- rep(y_bar_bar, n*M) # replicate to match y_ij's length
  sigsqr_m <- (1/(1-1/M)) * (stats::var(y_bar_i_dot - y_bar_bar_dot_dot)-(1-1/M)*stats::var(y_ij- y_bar_i_dot)/(n*(1-1/n))) # see formula derivation in overleaf
  sigsqr_m/(sigsqr_e+sigsqr_m) # our "cluster effect" or ICC value

  result = list(sigsqr_m=sigsqr_m, icc= sigsqr_m/(sigsqr_e+sigsqr_m))
}
```

permutest Package Description

- The goal of **permutest** is to perform asymptotic or permutation tests, and calculate asymptotic and permutation test statistics and p-values for data in a rotating sampling plan to test whether there is a significant change between each pair of years in the mean and in the fifth and fiftieth percentiles.
- Functions in this package can be used to calculate the asymptotic and permutation test statistics and p-values in Chen et al.'s paper, including t-test statistic, Wilcoxon sum statistic (scaled to be small), modified t-statistic, studentized t-statistic for quantile change, empirical distribution based test statistics for quantile change, empirical likelihood based test statistic for quantile change under the density ratio model, and empirical likelihood ratio test statistics for mean or quantile change under the density ratio model, upon re-factoring of Yukun Liu's code.
- One can generate the three example data sets, which are described in the paper, using three data simulation functions in this package, or input their real world dataset and use the reformat function before carrying out further analyses using functions in this package.

GitHub Link

R Package **permutest**

Three .R files, permutest.R is the main one:

- permutest.R: re-factoring Yukun's code from `Permu.test-norm-pop=0.R",`Permu.test-gamma-pop=0.R", and "Permu.test-general-pop=0.R" to tease out the parts for each individual test. Previously, the code was great for an ad hoc purpose of generating all the results needed in Chen et al.'s paper. Upon re-factoring, all the main functions for permutation and different test statistics and p-values can be used separately. Users can call one/ a few, or all tests if they find particular proposed test(s) from Chen et al.'s paper is/are of interest.
  - t_mean_clus, w_mean_clus, tmod_mean_clus, lrt_mean_stat, em_quantile_stat, el_quantile_stat, lrt_quantile_stat
  - Following the principle of **low coupling** and **high cohesion**. helper functions include t_clus_data (prepares data for asymptotic tests), stat_cal_data (prepares data for permutation tests), qqfun (creates base function needed for DRM), addnames (add column names and row names to the output), mele (calculates the maximum log EL under H_0)

R Package **permutest** Con't

Remaining two .R files:

- simulatedata.R: incorporating Yukun's original simulation code (not the "long" output format or Professor Chen's suggested 3D array structure)
- reformatrealdata.R: added last, using Carolyn's "RealWorldApplAnal.Rmd" line 183-228 as a starting point and try to generalize the reordering of clusters. To convert user input data into suitable list form for further analyses. Require at least one of the cluster sampled in the first year to remain in the second year compared for this to be a rotating panel design, and for permutation to work; o.w. throw error message.

Example Code With Simulated Data

example using simulated data:

```
no.perm <- 6             ###  number of permutations, (small number for illustration purpose)
nrep <- 2              ###  number of simulation repetitions, (small number for illustration purpose)
pop.n <- 2               ###  it has four choices, -1,  0,  1,  2


#  mu <- c(8, 8-0.4*pop.n, 8+0.5*rnorm(no.y-2))
mu <- c(8, 8-0.4*pop.n, 7.697925, 8.104713, 7.534080)
start.time <- proc.time()
for(i in 1:nrep) {

  data <- NormalObs_gen(mu, sig1, sig2, sig3, indicator, sz)

  # Asymptotic tests
  result1 <- t_mean_clus(data,c(1,2), asymptotic = TRUE)
  result2 <- w_mean_clus(data,c(1,2), asymptotic = TRUE)

  result3 <- tmod_mean_clus(data,c(1,2))
  result4 <- t_quantile_clus(data,c(1,2), alpha.level = alpha.level)

  # Permutation tests, (default to asked = 1)
  result5 <- permutation_test(6,data,c(1,2), BasisFun =NULL, alpha.level = NULL, t_mean_clus, 't.pm', 'mean')
  result6 <- permutation_test(6,data,c(1,2), BasisFun =NULL, alpha.level = NULL, w_mean_clus, 'w.pm', 'mean')
  result7 <- permutation_test(6,data,c(1,2), BasisFun      , alpha.level = NULL, lrt_mean_stat, 'lrt.pm', 'mean')
  result8 <- permutation_test(6,data,c(1,2), BasisFun =NULL, alpha.level       , em_quantile_stat, 'q.em.pm', 'quantile')
  result9 <- permutation_test(6,data,c(1,2), BasisFun      , alpha.level       , el_quantile_stat, 'q.el.pm', 'quantile')
  result10 <- permutation_test(6,data,c(1,2), BasisFun      , alpha.level       , lrt_quantile_stat, 'q.lrt.pm', 'quantile
  result <- rbind(result1, result2, result3, result4, result5, result6, result7, result8, result9, result10)
  print(result)
}
end.time <- proc.time()
cpu.time <- end.time-start.time
print(cpu.time)
```

Example Code With Simulated Data Con't

continued:

```
library(permutest)
##############################################################################
##############################################################################
###################           Main program              ######################

if(!require("drmdel")){ install.packages("drmdel")}
if(!require("rootSolve")){ install.packages("rootSolve")}
library(drmdel)                          ### Dr. Song Cai's package for fitting DRM
library(rootSolve)                       ### For solving a system of equations

asked = 1                                ### should we permute clusters not shared between two years?
BasisFun = function(x){ cbind(x, x^2) }  ### needed by our density-ratio-model-based method


##############################################################################
##############################################################################
###################       Examples Using Simulated Data            ###############
no.y <- 5                 ### number of years
no.c <- 36                ### number of mills (clusters) each year
no.d <- 6                 ### number of clusters drapped and added each year

size.c <- 10               ### cluster size
sig1 <- rep(1, no.y)      ### sigma_{i1},  i=0, 1, ..., k-1
sig2 <- rep(1, no.y)      ### sigma_{i2},  i=0, 1, ..., k-1
sig3 <- rep(2, no.y)
cluster.total <- no.c+(no.y-1)*no.d ###  total number of clusters
sz <- rep(size.c, cluster.total)      ###  a vector consisting of the sizes of all clusters

indicator <- NULL
for(i in 1:no.y){
  tmp <- c(rep(0, (i-1)*no.d),rep(1, no.c), rep(0, (no.y-i)*no.d))
  indicator <- rbind(indicator, tmp)
}

year.comp <- c(1, 2)       ### which two populations are compared
###          must be in an increasig order
alpha.level <- c(0.05, 0.5)  ### at which levels quantiles are of interest
```

Example With Simulated Data Result

If we run all tests for nrep = 1 and no.perm = 6 (for illustration purpose), we will get the following results:

```
##                       stat       pvalue
## t.an             4.9209843 5.343743e-07
## w.an             0.6106404 1.385760e-07
## tmod.an          5.3485099 4.434066e-08
## q.em.an-0.05     0.5881523 2.782150e-01
## q.em.an-0.5      1.4780586 6.969605e-02
## t.pm             4.9209843 1.428571e-01
## w.pm             0.6106404 1.428571e-01
## lrt.pm           3.4549060 1.428571e-01
## q.em.pm-0.05     0.4330833 4.285714e-01
## q.em.pm-0.5      0.8090712 4.285714e-01
## q.el.pm-0.05     0.8539825 1.428571e-01
## q.el.pm-0.5      0.7823012 1.428571e-01
## q.lrt.pm-0.05 2.1253431 2.857143e-01
## q.lrt.pm-0.5  3.4976586 2.857143e-01
```

```
##    user  system elapsed
##  16.983   0.327  17.336
```

Example Code With User Input Dataset

example using user input/real-world data:

```
#############################################################################
#############################################################################
##################       Examples Using Real/User-Input Data       ##############
Year <- c(rep("2019",8),rep("2020",7), rep("2021", 1))
ID <- c(rep(9,3),rep(10,2),rep(11,2),rep(14,1),rep(11,2),rep(14,1),rep(4,2),rep(2,2),rep(3,1))
Value <- runif(16)
inputdata <- data.frame(Year, ID, Value)

realdata <- reformat(inputdata, c("2019","2020"))

tmod_mean_clus(realdata, year.comp = c(1,2))

permutedrealdata <- permute_data_rs(realdata,c(1,2), asked = 1)

permutation_test(6, realdata, year.comp = c(1,2), alpha.level = c(0.05,0.5),
                 FUN = em_quantile_stat, nm = 'q.em.pm', param = 'quantile', asked = 0)

BasisFun = function(x){ cbind(x, x^2) }
permutation_test(6, realdata, year.comp = c(1,2), BasisFun, c(0.05,0.5),
                 FUN = lrt_quantile_stat, nm = 'q.lrt.pm', param = 'quantile')
```

Example With User Input Result

```
##              stat       pvalue
## tmod.an 3.295163 0.0004918228
```

```
##                     stat    pvalue
## q.em.pm-0.05 0.4223464 0.1428571
## q.em.pm-0.5  0.3925106 0.1428571
```

```
##                     stat    pvalue
## q.lrt.pm-0.05 1.051976 0.1428571
## q.lrt.pm-0.5  2.041219 0.1428571
```

Example Code For Within the Forest Product Modelling Group

example using the NLGA_LTM_data_transformed.csv data:

```
## Example for Within the Forest Product Modeling Group
## using the NLGA_LTM_data_transformed.csv dataset
# input <- read_csv("LTM data and Simulation code/NLGA_LTM_data_transformed.csv")
# Year <- input$Myear
# ID <- input$MillID
# Value <- input$MOR
# inputdata <- data.frame(Year, ID, Value)
# realdata <- reformat(inputdata, c("2","3"))
# tmod_mean_clus(realdata, year.comp = c(1,2))
#
# permutedrealdata <- permute_data_rs(realdata,c(1,2), asked = 1)
#
# permutation_test(6, realdata, year.comp = c(1,2), alpha.level = c(0.05,0.5),
#                  FUN = em_quantile_stat, nm = 'q.em.pm', param = 'quantile', asked = 0)
#
# BasisFun = function(x){ cbind(x, x^2) }
# permutation_test(6, realdata, year.comp = c(1,2), BasisFun, c(0.05,0.5),
#                  FUN = lrt_quantile_stat, nm = 'q.lrt.pm', param = 'quantile')
```

Testing the R Package **permutest**

- Test all functions using **testthat** package.
- Test with set.seed(123)
- Test with coverage –> 100%!
- Carolyn also reminded me to test the edge case when two sampled years have all or no overlapping clusters when reformatting the input data. Catching the later case exception by throwing an error message.

Checking CRAN Specifications

- devtools
- rhub::check_for_cran()
- check syntax useful article for future submission

Future Work

- Write a conversion function between Professor Chen's 3D array structure and Yukun's nested list structure for simulation.
- Check Cluster Effect Calculation for Multiple Occasions Data.
- From the ICC, we would like to reverse-engineer to find desired argument values of the simulation models so that the generated dataset would have the given ICC.
- Change Yukun's original data generation function to my data generation functions so that users can view the rotating panel structure with ouput columns representing years and rows representing mills, as well as for more default arguments.
- Publish **permutest**.

THANK YOU!

Thank you for all your supports! Any questions?

Chopin Étude op.10 no.8

etude