Math 297 HW1 EM Algorithm and Bootstrap

He Jiang

25 April 2020

1 Part One

1.1 Single Component Gaussian

$d1.csv$ consists of 500 observations with numeric values. First we want to test the fit of a single Gaussian on $dat1$, another name for $d1.csv$. We do so by first estimating the MLE for μ and σ, recorded as μ_{MLE} and σ_{MLE}. Then use the Kolmogorov Smirnov Test to decide the Goodness of Fit of $dat1$ and $Gaussian(\mu_{MLE}, \sigma_{MLE})$.
MLE and KS Test

```
        One-sample Kolmogorov-Smirnov test

data:  dat1_vec
D = 0.13069, p-value = 7.642e-08
alternative hypothesis: two-sided
```

The KS Test here computes the maximum difference in absolute value of the empirical distribution function of our $dat1$ vector, and the theoretical $N(\mu_{MLE}, \sigma_{MLE})$ cdf. H_0 here is that the above ecdf and theoretical cdf belong to the same distribution, and H_1 is that they belong to different distributions.
As the p value is extremely small, we reject H_0. This means fitting a one component Gaussian by MLE to the data is a very bad choice, and the 1st dataset is very unlikely coming from a single Gaussian Distribution.

1.2 EM Algorithm Estimates

In the EM Algorithm, x is the input data vector, K is number of components, *initmean* is the intial mean vector with length K, *initsigsq* is the intial variance vector with length K, and *initpi* is the intial proportion of the components, *tol* is the tolerance bound, and *yesprint* determines whether we want to print each steps estimators
First is the EM with Different Variance:

```
> EM_Al_Diff_Var = function(x, K, init_mu, init_sigsq, init_pi, tol, yes_print){
+
+   # initialization
```

```
+   N = length(x)
+   mu = init_mu
+   sig_sq = init_sigsq
+   pi = init_pi
+   s = 1 # s counts the iteration of the EM Algorithm
+   sat = FALSE # satisfy the tol is set to false before EM starts
+   w_mat = matrix(NA, nrow = N, ncol = K) # stores the weights in E Step
+   dens_mat = matrix(NA, nrow = N, ncol = K) # stores the density of each data belong
+
+   # begin EM
+   while (sat == FALSE){
+
+     # E Step: update density matrix in order to update weight matrix
+     for (k in 1:K){
+       dens_mat[ , k] = dnorm(x , mu[k] , sqrt(sig_sq[k]) , log=FALSE) # intialize Ga
+     }
+     for (i in 1:N){
+       for (k in 1:K){
+         w_mat[i,k] = pi[k] * dens_mat[i,k]
+       }
+       denom = sum(w_mat[i,])
+       w_mat[i,] = w_mat[i,] / denom
+     }
+
+     # M Step:
+     N_k = numeric(K) # stores sum of w_mat
+     mu_new = numeric(K)
+     sig_sq_new = numeric(K)
+     for (k in 1:K){
+       N_k[k] = sum(w_mat[,k])
+     }
+     pi_new = N_k / N # update pi
+     for (k in 1:K){
+       mu_new[k] = as.numeric(x %*% w_mat[,k])
+     }
+     mu_new = mu_new / N_k # update mu
+     for (k in 1:K){
+       diff_sq = (x - mu_new[k])^2
+       sig_sq_new[k] = as.numeric(diff_sq %*% w_mat[,k])
+     }
+     sig_sq_new = sig_sq_new / N_k # update sig_sq
+
+     # check whether the tolerence is satisfied or not
+     sat = all(abs(pi_new - pi) <= tol) && all(abs(mu_new - mu) <= tol) && all(abs(si
+
```

```
+     # update the parameters
+     pi = pi_new
+     mu = mu_new
+     sig_sq = sig_sq_new
+
+     if (yes_print == TRUE){
+       print(paste0("iteration: ", s))
+       print(paste0("weights: ", pi, " means: ", mu, " variances: ", sig_sq))
+       print("###############################################")
+     }
+
+     s = s + 1
+
+     if (sat == TRUE){
+       return(list(pi, mu, sig_sq))
+     }
+   }
+ }
>
```

Next is the EM with Same Variance:
Notice here π and μ are computed in the same fashion, but the σ^2 is computed as follows:
Using the newly updated π and μ, we compute the variance as the weighted sum of squares, with $K = 2$:

$$\frac{\sum_{i=1}^{N}\sum_{k=1}^{K} w_{i,k}(x_i - \mu_k)^2}{N} \tag{1}$$

```
> EM_Al_Same_Var = function(x, K, init_mu, init_sigsq, init_pi, tol, yes_print){
+
+   # initialization
+   N = length(x)
+   mu = init_mu
+   sig_sq = init_sigsq
+   pi = init_pi
+   s = 1 # s counts the iteration of the EM Algorithm
+   sat = FALSE # satisfy the tol is set to false before EM starts
+   w_mat = matrix(NA, nrow = N, ncol = K) # stores the weights in E Step
+   dens_mat = matrix(NA, nrow = N, ncol = K) # stores the density of each data belong
+
+   # begin EM
+   while (sat == FALSE){
+
+     # E Step: update density matrix in order to update weight matrix
+     for (k in 1:K){
+       dens_mat[ , k] = dnorm(x , mu[k] , sqrt(sig_sq[k]) , log=FALSE) # intialize Ga
+     }
+     for (i in 1:N){
```

```
+       for (k in 1:K){
+         w_mat[i,k] = pi[k] * dens_mat[i,k]
+       }
+       denom = sum(w_mat[i,])
+       w_mat[i,] = w_mat[i,] / denom
+     }
+
+     # M Step:
+     N_k = numeric(K) # stores sum of w_mat
+     mu_new = numeric(K)
+     sig_sq_new = numeric(K)
+     for (k in 1:K){
+       N_k[k] = sum(w_mat[,k])
+     }
+     pi_new = N_k / N # update pi
+     for (k in 1:K){
+       mu_new[k] = as.numeric(x %*% w_mat[,k])
+     }
+     mu_new = mu_new / N_k # update mu
+
+     for (k in 1:K){
+       diff_sq = (x - mu_new[k])^2
+       sig_sq_new[k] = as.numeric(diff_sq %*% w_mat[,k])
+     }
+     sig_sq_new_num = sum(sig_sq_new) / N # same variance assumption
+     sig_sq_new = rep(sig_sq_new_num, K)
+
+     # check whether the tolerence is satisfied or not
+     sat = all(abs(pi_new - pi) <= tol) && all(abs(mu_new - mu) <= tol) && all(abs(si
+
+     # update the parameters
+     pi = pi_new
+     mu = mu_new
+     sig_sq = sig_sq_new
+
+     if (yes_print == TRUE){
+       print(paste0("iteration: ", s))
+       print(paste0("weights: ", pi, " means: ", mu, " variances: ", sig_sq))
+       print("###############################################")
+     }
+
+     s = s + 1
+
+     if (sat == TRUE){
+       return(list(pi, mu, sig_sq))
```

```
+     }
+   }
+ }
>
```

Apply the General Variance EM to the dat1 dataset, with required intialization

```
[1] "iteration: 1"
[1] "weights: 0.493782945770368 means: 1.51200057517549 variances: 2.58109269302747"
[2] "weights: 0.506217054229632 means: 4.51961442797522 variances: 5.6705835859641"
[1] "################################################"
[1] "iteration: 2"
[1] "weights: 0.49938545136379 means: 1.41346085856142 variances: 1.90055023068401"
[2] "weights: 0.50061454863621 means: 4.65157118886381 variances: 5.66441489785352"
[1] "################################################"
[1] "iteration: 3"
[1] "weights: 0.49972842784668 means: 1.29348715718948 variances: 1.50640392382594"
[2] "weights: 0.50027157215332 means: 4.77363462052954 variances: 5.24811501020242"
[1] "################################################"
[1] "iteration: 4"
[1] "weights: 0.498641678385251 means: 1.19194788484245 variances: 1.2534758242137"
[2] "weights: 0.501358321614749 means: 4.86708009212912 variances: 4.79591472290431"
[1] "################################################"
[1] "iteration: 5"
[1] "weights: 0.497368584441498 means: 1.11896944341799 variances: 1.09687584771592"
[2] "weights: 0.502631415558502 means: 4.92998582100264 variances: 4.43609126661097"
[1] "################################################"
[1] "iteration: 6"
[1] "weights: 0.496194910397758 means: 1.0719618544495 variances: 1.00501335957717"
[2] "weights: 0.503805089602242 means: 4.96740512418691 variances: 4.19616430952775"
[1] "################################################"
[1] "iteration: 7"
[1] "weights: 0.495142630092706 means: 1.04384863956105 variances: 0.953222154102932"
[2] "weights: 0.504857369907294 means: 4.98685804993074 variances: 4.0559701199017"
[1] "################################################"
[1] "iteration: 8"
[1] "weights: 0.49418709321021 means: 1.02766716026021 variances: 0.924563650395655"
[2] "weights: 0.50581290678979 means: 4.99521882373291 variances: 3.98247892017756"
[1] "################################################"
[1] "iteration: 9"
[1] "weights: 0.493309804056546 means: 1.01838426243413 variances: 0.908645596860902"
[2] "weights: 0.506690195943454 means: 4.99738712001234 variances: 3.94813290314469"
[1] "################################################"
[1] "iteration: 10"
[1] "weights: 0.492501819138217 means: 1.01290777002697 variances: 0.899561000702816"
[2] "weights: 0.507498180861783 means: 4.99636683689525 variances: 3.93499240131289"
```

```
[1] "################################################"
[1] "iteration: 11"
[1] "weights: 0.491758217481792 means: 1.00949624813877 variances: 0.894107237725883"
[2] "weights: 0.508241782518208 means: 4.99383956959461 variances: 3.93272889336403"
[1] "################################################"
[1] "iteration: 12"
[1] "weights: 0.491074891411452 means: 1.00721043584267 variances: 0.890593271278186"
[2] "weights: 0.508925108588548 means: 4.99069549080454 variances: 3.9357638379745"
[1] "################################################"
[1] "iteration: 13"
[1] "weights: 0.490447686855454 means: 1.00555305253746 variances: 0.888137085191302"
[2] "weights: 0.509552313144546 means: 4.98738748861622 variances: 3.94118881760188"
[1] "################################################"
[1] "iteration: 14"
[1] "weights: 0.489872404114083 means: 1.00426186035412 variances: 0.886279692997737"
[2] "weights: 0.510127595885917 means: 4.98413700535591 variances: 3.94752301028688"
[1] "################################################"
[1] "iteration: 15"
[1] "weights: 0.489344945252645 means: 1.0031976350915 variances: 0.884780841643996"
[2] "weights: 0.510655054747355 means: 4.98104598110168 variances: 3.95402985329778"
[1] "################################################"
[1] "iteration: 16"
[1] "weights: 0.488861420308857 means: 1.00228516444344 variances: 0.883512932671651"
[2] "weights: 0.511138579691143 means: 4.97815573321273 variances: 3.96035540733631"
[1] "################################################"
[1] "iteration: 17"
[1] "weights: 0.488418198020262 means: 1.00148256476889 variances: 0.882406441245575"
[2] "weights: 0.511581801979738 means: 4.97547739289538 variances: 3.96634049391042"
[1] "################################################"
[1] "iteration: 18"
[1] "weights: 0.488011920272382 means: 1.00076540819846 variances: 0.881421932931805"
[2] "weights: 0.511988079727618 means: 4.97300748243485 variances: 3.97192401215754"
[1] "################################################"
[1] "iteration: 19"
[1] "weights: 0.487639497233102 means: 1.00011852055957 variances: 0.88053572213787"
[2] "weights: 0.512360502766898 means: 4.97073582703447 variances: 3.97709344237634"
[1] "################################################"
[1] "iteration: 20"
[1] "weights: 0.487298093452684 means: 0.999531735532814 variances: 0.879732507970283"
[2] "weights: 0.512701906547316 means: 4.96864953767109 variances: 3.98185960899512"
[1] "################################################"
[1] "iteration: 21"
[1] "weights: 0.486985110404737 means: 0.998997686514358 variances: 0.879001577223894"
[2] "weights: 0.513014889595263 means: 4.96673498765104 variances: 3.98624387593394"
[1] "################################################"
```

```
[1] "iteration: 22"
[1] "weights: 0.486698168195206 means: 0.998510648492055 variances: 0.87833483270971"
[2] "weights: 0.513301831804794 means: 4.96497876794 variances: 3.9902717020939"
[1] "################################################"
[1] "iteration: 23"
[1] "weights: 0.486435087724266 means: 0.998065922021978 variances: 0.877725757528545"
[2] "weights: 0.513564912275734 means: 4.96336812491985 variances: 3.99396944571797"
[1] "################################################"
[1] "iteration: 24"
[1] "weights: 0.486193873860151 means: 0.997659498095432 variances: 0.87716886084479"
[2] "weights: 0.513806126139849 means: 4.96189113777458 variances: 3.99736282479109"
[1] "################################################"
[1] "iteration: 25"
[1] "weights: 0.485972699828329 means: 0.997287869437811 variances: 0.87665937265491"
[2] "weights: 0.514027300171671 means: 4.96053676668863 variances: 4.00047621828578"
[1] "################################################"
[1] "iteration: 26"
[1] "weights: 0.485769892849817 means: 0.99694791890475 variances: 0.876193068368978"
[2] "weights: 0.514230107150183 means: 4.95929483862755 variances: 4.00333239118945"
[1] "################################################"
[1] "iteration: 27"
[1] "weights: 0.485583920984947 means: 0.996636849120037 variances: 0.87576616194693"
[2] "weights: 0.514416079015053 means: 4.95815600455128 variances: 4.00595243010362"
[1] "################################################"
[1] "iteration: 28"
[1] "weights: 0.485413381106466 means: 0.996352134727815 variances: 0.875375235977503"
[2] "weights: 0.514586618893534 means: 4.95711168509201 variances: 4.00835578061055"
[1] "################################################"
[1] "iteration: 29"
[1] "weights: 0.485256987915169 means: 0.996091487510887 variances: 0.875017192287508"
[2] "weights: 0.514743012084831 means: 4.95615401314489 variances: 4.01056033108407"
[1] "################################################"
[1] "iteration: 30"
[1] "weights: 0.48511356391076 means: 0.995852829213914 variances: 0.874689214479795"
[2] "weights: 0.51488643608924 means: 4.95527577744777 variances: 4.01258251501371"
[1] "################################################"
[1] "iteration: 31"
[1] "weights: 0.484982030234765 means: 0.995634269288556 variances: 0.874388737825193"
[2] "weights: 0.515017969765235 means: 4.95447036900919 variances: 4.01443741793221"
[1] "################################################"
[1] "iteration: 32"
[1] "weights: 0.484861398308175 means: 0.995434086017578 variances: 0.874113424019541"
[2] "weights: 0.515138601691825 means: 4.95373173112838 variances: 4.01613888220194"
[1] "################################################"
[1] "iteration: 33"
```

```
[1] "weights: 0.484750762192918 means: 0.995250710127431 variances: 0.873861139405483"
[2] "weights: 0.515249237807082 means: 4.95305431319865 variances: 4.01769960656412"
[1] "################################################"
[1] "iteration: 34"
[1] "weights: 0.48464929161253 means: 0.995082710346781 variances: 0.873629935833099"
[2] "weights: 0.51535070838747 means: 4.95243302821976 variances: 4.01913123920058"
[1] "################################################"
[1] "iteration: 35"
[1] "weights: 0.484556225573285 means: 0.99492878055826 variances: 0.873418033641796"
[2] "weights: 0.515443774426715 means: 4.95186321382442 variances: 4.02044446397902"
[1] "################################################"
[1] "iteration: 36"
[1] "weights: 0.484470866532495 means: 0.994787728297522 variances: 0.873223806415976"
[2] "weights: 0.515529133467505 means: 4.95134059657698 variances: 4.02164908000108"
[1] "################################################"
[1] "iteration: 37"
[1] "weights: 0.484392575065606 means: 0.994658464416477 variances: 0.873045767264147"
[2] "weights: 0.515607424934394 means: 4.95086125929103 variances: 4.0227540747796"
[1] "################################################"
[1] "iteration: 38"
[1] "weights: 0.484320764988202 means: 0.994539993766734 variances: 0.872882556429724"
[2] "weights: 0.515679235011798 means: 4.95042161111864 variances: 4.02376769145642"
[1] "################################################"
[1] "iteration: 39"
[1] "weights: 0.484254898893084 means: 0.994431406785349 variances: 0.872732930079185"
[2] "weights: 0.515745101106916 means: 4.95001836017792 variances: 4.02469749049525"
[1] "################################################"

[1] "The result of the General EM Algorithm, in the order of proportion, mean, variance,

[[1]]
[1] 0.4842549 0.5157451

[[2]]
[1] 0.9944314 4.9500184

[[3]]
[1] 0.8727329 4.0246975
```

Apply the Same Variance EM to the dat1 dataset, with required intialization

```
[1] "iteration: 1"
[1] "weights: 0.493782945770368 means: 1.51200057517549 variances: 4.14504567191913"
[2] "weights: 0.506217054229632 means: 4.51961442797522 variances: 4.14504567191913"
[1] "################################################"
[1] "iteration: 2"
```

```
[1] "weights: 0.517044122159714 means: 1.51797123389531 variances: 3.94392244290039"
[2] "weights: 0.482955877840286 means: 4.65808161599985 variances: 3.94392244290039"
[1] "################################################"
[1] "iteration: 3"
[1] "weights: 0.541325011443517 means: 1.51113696129686 variances: 3.66731261134693"
[2] "weights: 0.458674988556483 means: 4.83237547813293 variances: 3.66731261134693"
[1] "################################################"
[1] "iteration: 4"
[1] "weights: 0.566285751948276 means: 1.49481520392686 variances: 3.31085821636618"
[2] "weights: 0.433714248051724 means: 5.04482722910137 variances: 3.31085821636618"
[1] "################################################"
[1] "iteration: 5"
[1] "weights: 0.59054372786602 means: 1.47403156986224 variances: 2.89410430547432"
[2] "weights: 0.40945627213398 means: 5.28512090061048 variances: 2.89410430547432"
[1] "################################################"
[1] "iteration: 6"
[1] "weights: 0.611435667842153 means: 1.4555882467911 variances: 2.48323478436073"
[2] "weights: 0.388564332157847 means: 5.51905373380728 variances: 2.48323478436073"
[1] "################################################"
[1] "iteration: 7"
[1] "weights: 0.626317490496789 means: 1.44521165585085 variances: 2.17262272464186"
[2] "weights: 0.37368250950321 means: 5.69827221857835 variances: 2.17262272464186"
[1] "################################################"
[1] "iteration: 8"
[1] "weights: 0.635006936210988 means: 1.44331970853363 variances: 2.00123271571036"
[2] "weights: 0.364993063789012 means: 5.80281705061511 variances: 2.00123271571036"
[1] "################################################"
[1] "iteration: 9"
[1] "weights: 0.639541694761488 means: 1.44593462413079 variances: 1.92871689998644"
[2] "weights: 0.360458305238512 means: 5.85302235878034 variances: 1.92871689998644"
[1] "################################################"
[1] "iteration: 10"
[1] "weights: 0.641894142615829 means: 1.44950245118995 variances: 1.90302224228518"
[2] "weights: 0.358105857384171 means: 5.87557791498691 variances: 1.90302224228518"
[1] "################################################"
[1] "iteration: 11"
[1] "weights: 0.643174557840606 means: 1.45253604874985 variances: 1.89516582610816"
[2] "weights: 0.356825442159394 means: 5.88599219940035 variances: 1.89516582610816"
[1] "################################################"
[1] "iteration: 12"
[1] "weights: 0.643910848019119 means: 1.45475087677824 variances: 1.89332656537845"
[2] "weights: 0.356089151980881 means: 5.89115427304507 variances: 1.89332656537845"
[1] "################################################"
[1] "iteration: 13"
[1] "weights: 0.644352701256648 means: 1.4562661490496 variances: 1.89328909268347"
```

```
[2] "weights: 0.355647298743352 means: 5.89392069198292 variances: 1.89328909268347"
[1] "################################################"
[1] "iteration: 14"
[1] "weights: 0.644625430586107 means: 1.45727137659097 variances: 1.89366791155652"
[2] "weights: 0.355374569413893 means: 5.89550291918647 variances: 1.89366791155652"
[1] "################################################"
[1] "iteration: 15"
[1] "weights: 0.644796671710943 means: 1.45792804559262 variances: 1.89405267826753"
[2] "weights: 0.355203328289057 means: 5.89645051665571 variances: 1.89405267826753"
[1] "################################################"

[1] "The result of the Same Variance EM Algorithm, in the order of proportion, mean, var

[[1]]
[1] 0.6447967 0.3552033

[[2]]
[1] 1.457928 5.896451

[[3]]
[1] 1.894053 1.894053
```

1.3 Histogram and Fitted Curve

Plot the resulting fit (the figure is on the next page)

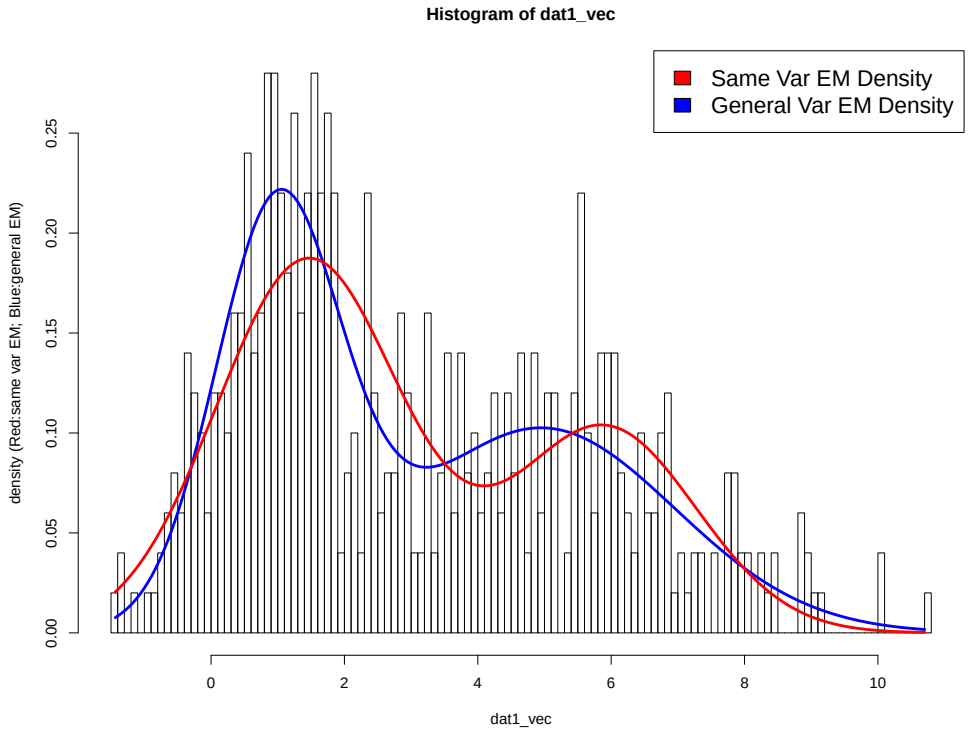
2 Part Two

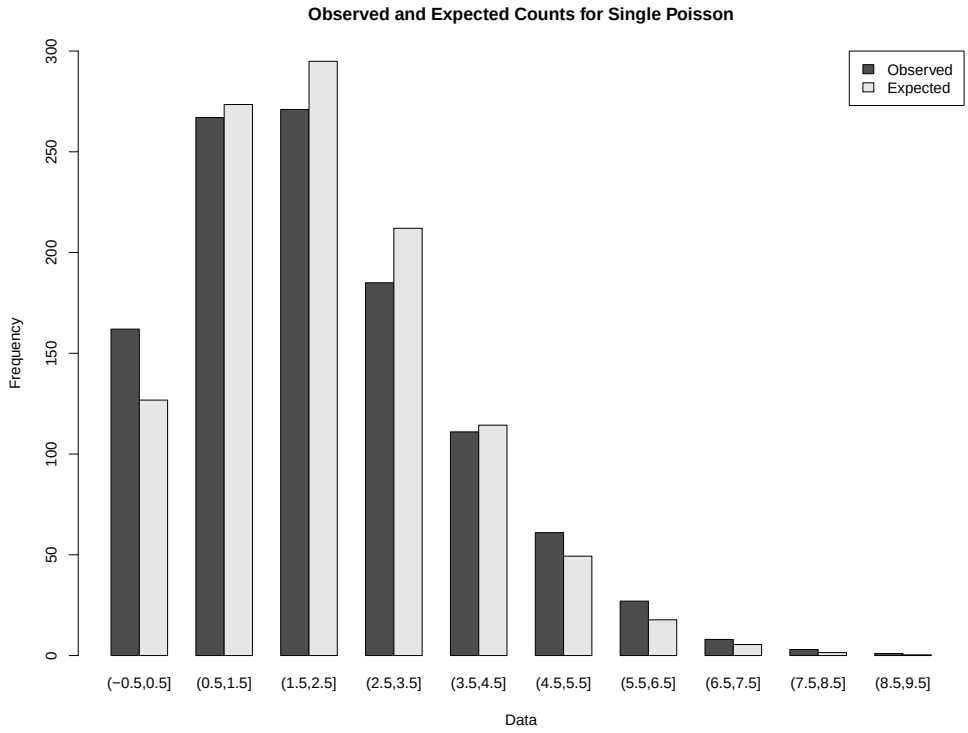
2.1 Single Component Poisson

First we want to test the fit of a single component Poisson on $dat2$. We do so by first estimating the MLE for λ, and then use the Chi-Squared Goodness of Fit to examine the fit between the given data vector and $Poisson(\lambda_{MLE})$. The test uses $0, 1, 2, ..., 9$ as the 10 bins. d2.csv comes from London Times, and consists of two columns, with the left column the count of death of women over 80 years old, the the right column the frequency(i.e. amount of days that this happened).
Since some of the expected counts at the right tail are less then 5, we use the Chi-square GOF test with resampling:

```
        Chi-squared test for given probabilities with simulated p-value (based
        on 2000 replicates)

data:  o_ct
X-squared = 26.972, df = NA, p-value = 0.007996
```

As can be seen by both the picture and p-value, the fit of a single component Poisson is not good.


Histogram of dat1_vec
Same Var EM Density
General Var EM Density
density (Red:same var EM; Blue:general EM)
0.00
0.05
0.10
0.15
0.20
0.25
0
2
4
6
8
10
dat1_vec


Observed and Expected Counts for Single Poisson
Observed
Expected
Frequency
0
50
100
150
200
250
300
(−0.5,0.5]
(0.5,1.5]
(1.5,2.5]
(2.5,3.5]
(3.5,4.5]
(4.5,5.5]
(5.5,6.5]
(6.5,7.5]
(7.5,8.5]
(8.5,9.5]
Data

2.2 Poisson EM Algorithm

There is an extremely helpful website on the parameters of Poisson Mixture EM:
https://www.cs.helsinki.fi/u/bmmalone/probabilistic-models-spring-2014/mixture-of-poissons.pdf
The E Step here is the same as Gaussian Mixture EM except that when computing the density we use $Pois(\mu_k)$ instead of $N(\mu_k, \sigma_k)$. In the M Step, The λ are represented as μ and updated as:

$$\mu_k = \frac{\sum_{i=1}^{N} w_{i,k} x_i}{\sum_{i=1}^{N} w_{i,k}} \tag{2}$$

x is data vector, K is number of components, *initmu* is the initial guess of the λ values, *initpi* the intial guess for proportion, and *tol* is the value such that the change is not considered significant when less than it, and finally *yesprint* determines whether we print the result of each iteration

```
> EM_Al_Pois = function(x, K, init_mu, init_pi, tol, yes_print){
+
+   # initialization
+   N = length(x)
+   mu = init_mu # mu is the vector of lambdas
+   pi = init_pi
+   s = 1 # s counts the iteration of the EM Algorithm
+   sat = FALSE # satisfy the tol is set to false before EM starts
+   w_mat = matrix(NA, nrow = N, ncol = K) # stores the weights in E Step
+   dens_mat = matrix(NA, nrow = N, ncol = K) # stores the density of each data belong
+
+   # begin EM
+   while (sat == FALSE){
+
+     # E Step: update density matrix in order to update weight matrix
+     for (k in 1:K){
+       dens_mat[ , k] = dpois(x, mu[k], log = FALSE)   # intialize Poisson density
+     }
+     for (i in 1:N){
+       for (k in 1:K){
+         w_mat[i,k] = pi[k] * dens_mat[i,k]
+       }
+       denom = sum(w_mat[i,])
+       w_mat[i,] = w_mat[i,] / denom
+     }
+
+     # M Step:
+     N_k = numeric(K) # stores sum of w_mat
+     mu_new = numeric(K)
+     for (k in 1:K){
+       N_k[k] = sum(w_mat[,k])
+     }
```

```
+     pi_new = N_k / N # update pi
+     for (k in 1:K){
+       mu_new[k] = as.numeric(x %*% w_mat[,k])
+     }
+     mu_new = mu_new / N_k # update Poisson lambda
+
+     # check whether the tolerence is satisfied or not
+     sat = all(abs(pi_new - pi) <= tol) && all(abs(mu_new - mu) <= tol)
+
+
+     # update the parameters
+     pi = pi_new
+     mu = mu_new
+
+     l_l = 0
+     for (id in 1:N){
+       dens_id = 0
+       for (k in 1:K){
+         dens_id = dens_id + dens_mat[id, k] * pi[k]
+       }
+       l_l = l_l + log(dens_id)
+     }
+
+     if (yes_print == TRUE){
+       print(paste0("iteration: ", s))
+       print( paste0("weights: ", pi, " lambdas: ", mu) )
+       print( paste0("log likelihood: ", l_l) )
+       print("################################################")
+     }
+
+     s = s + 1
+
+     if (sat == TRUE){
+       return(list(pi, mu))
+     }
+   }
+ }
>

[1] "iteration: 1"
[1] "weights: 0.285690438368875 lambdas: 1.06138980766247"
[2] "weights: 0.714309561631125 lambdas: 2.5951009012186"
[1] "log likelihood: -1991.76793162164"
[1] "################################################"
[1] "iteration: 2"
```

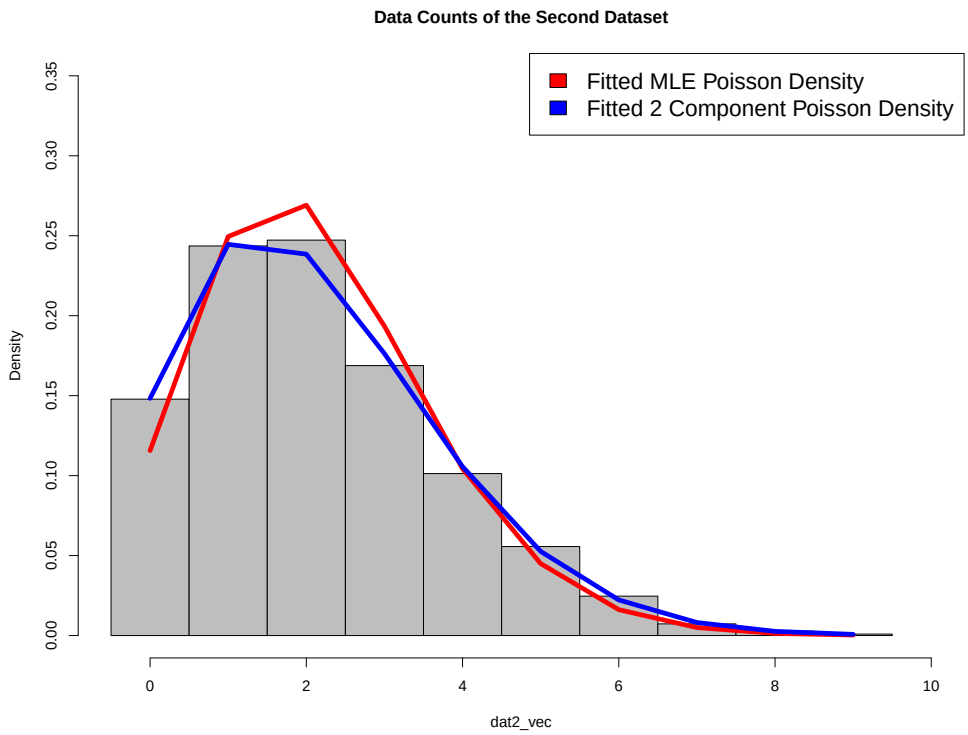
```
[1] "weights: 0.28500064657844 lambdas: 1.07497712002747"
[2] "weights: 0.71499935342156 lambdas: 2.5882053227724"
[1] "log likelihood: -1990.153379554"
[1] "################################################"
[1] "iteration: 3"
[1] "weights: 0.284636815376832 lambdas: 1.08475381775126"
[2] "weights: 0.715363184623168 lambdas: 2.58354563687858"
[1] "log likelihood: -1990.09466715838"
[1] "################################################"
[1] "iteration: 4"
[1] "weights: 0.284489104119896 lambdas: 1.09190312291263"
[2] "weights: 0.715510895880104 lambdas: 2.58039364037245"
[1] "log likelihood: -1990.06529344879"
[1] "################################################"
[1] "iteration: 5"
[1] "weights: 0.284487982259559 lambdas: 1.09721522559405"
[2] "weights: 0.715512017740441 lambdas: 2.57827921152378"
[1] "log likelihood: -1990.05006002607"
[1] "################################################"
[1] "iteration: 6"
[1] "weights: 0.28458783854997 lambdas: 1.10122965755476"
[2] "weights: 0.71541216145003 lambdas: 2.57688901296023"
[1] "log likelihood: -1990.04182869831"
[1] "################################################"
[1] "iteration: 7"
[1] "weights: 0.284757958678348 lambdas: 1.10432073891716"
[2] "weights: 0.715242041321652 lambdas: 2.57600935177769"
[1] "log likelihood: -1990.03713621266"
[1] "################################################"
[1] "iteration: 8"
[1] "weights: 0.284977258824135 lambdas: 1.10675118614324"
[2] "weights: 0.715022741175865 lambdas: 2.57549205262122"
[1] "log likelihood: -1990.03426203418"
[1] "################################################"
[1] "iteration: 9"
[1] "weights: 0.285231061410218 lambdas: 1.10870693519082"
[2] "weights: 0.714768938589782 lambdas: 2.5752331295812"
[1] "log likelihood: -1990.0323361088"
[1] "################################################"
[1] "iteration: 10"
[1] "weights: 0.285509043622458 lambdas: 1.11032038077418"
[2] "weights: 0.714490956377542 lambdas: 2.57515897173616"
[1] "log likelihood: -1990.03091233202"
[1] "################################################"
[1] "iteration: 11"
```

```
[1] "weights: 0.285803891436593 lambdas: 1.1116862090097"
[2] "weights: 0.714196108563407 lambdas: 2.5752171426382"
[1] "log likelihood: -1990.02975933774"
[1] "################################################"
[1] "iteration: 12"
[1] "weights: 0.286110395007866 lambdas: 1.11287235420028"
[2] "weights: 0.713889604992134 lambdas: 2.57537012005661"
[1] "log likelihood: -1990.02875589141"
[1] "################################################"
[1] "iteration: 13"
[1] "weights: 0.286424829286311 lambdas: 1.11392767083542"
[2] "weights: 0.713575170713689 lambdas: 2.5755909665058"
[1] "log likelihood: -1990.02783781946"
[1] "################################################"
[1] "iteration: 14"
[1] "weights: 0.286744523880888 lambdas: 1.11488735261393"
[2] "weights: 0.713255476119112 lambdas: 2.57586029827612"
[1] "log likelihood: -1990.02697077853"
[1] "################################################"

[1] "The result of the EM Poisson Algorithm, in the order of proportion, lambda, is:"

[[1]]
[1] 0.2867445 0.7132555

[[2]]
[1] 1.114887 2.575860
```

2.3 Tabulate Data

To create a table of the values of expected counts, we use the single component Poisson and Mixture Poisson densities, in estimating the expected count of $0, 1, ..., 9$.

```
                [,1]      [,2]      [,3]      [,4]      [,5]       [,6]
values      0.0000000 1.0000000 2.0000000 3.0000000 4.0000000 5.00000000
o_pro       0.1478102 0.2436131 0.2472628 0.1687956 0.1012774 0.05565693
exp_ct_1_po 0.1156792 0.2495125 0.2690910 0.1934705 0.1043258 0.04500478
exp_ct_2_po 0.1483089 0.2446354 0.2384874 0.1763085 0.1056036 0.05263520
                 [,7]        [,8]        [,9]        [,10]
values      6.00000000 7.000000000 8.000000000 9.0000000000
o_pro       0.02463504 0.007299270 0.002737226 0.0009124088
exp_ct_1_po 0.01617873 0.004985207 0.001344096 0.0003221251
exp_ct_2_po 0.02226814 0.008141884 0.002614249 0.0007473119
```



2.4 2 Component Goodness of Fit

Plotting the data in histogram, with overlaying density curves for single Poisson and a mixture of two Poissons. The plot could be found above.
The expected and observed counts in Poisson Mixture Model could be found on the next page.

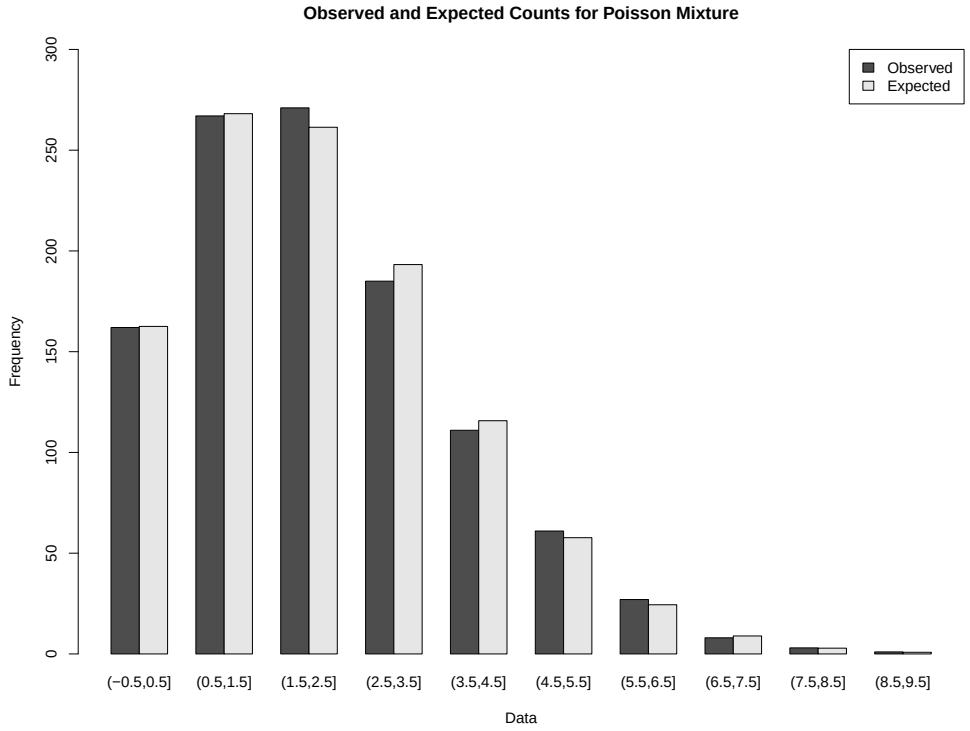
```
        Chi-squared test for given probabilities with simulated p-value (based
        on 2000 replicates)

data:  o_ct
X-squared = 1.5128, df = NA, p-value = 0.998
```

We use here again the Chi Squared GOF test, with $0, 1, 2, ..., 9$ as the 10 bins. From the result, the mixture of Poisson fits the data very well. There is also a picture of the observed and expected counts on the next page to show this point.

3 Bootstrap

3.1 Difference in σ Confidence Interval

The idea for the Bootstrap when estimating the CI for the difference in variance is as follows:



Once we get a nonparametric Bootstrap sample from the original sample, we use the EM Algorithm(General Variance) of that Bootstrapped sample and acquire values σ_1 and σ_0. We do the bootstrap B times, every time computing $(\sigma_1 - \sigma_0)$. Since there will be B values, we select the 0.025 and 0.975 percentile of those B values as the Confidence Interval estimate.

```
[1] "The CI for difference in sigma in dat1 is: 0.695146431357102 1.37626487098609"
```

As 0 is not inside the Confidence Interval, by the duality of $1 - \alpha$ level Confidence Interval and α level hypothesis tests with $\alpha = 0.05$, we conclude the Null with $\sigma_1 = \sigma_0$ should be rejected at the 0.05 level. Thus we conclude σ_1 and σ_0 differ.

3.2 Median and IQR Confidence Interval

Using the Bootstrap, we can get B Bootstrap Samples. In every Bootstrap sample, we compute the median and IQR, and we will get B median and IQR in total. We than select the 0.025 and 0.975 of those B values as the 0.95 Confidence Interval Estimate.

```
[1] "The CI for median in dat1 is: 1.91 2.8195"
```

```
[1] "The CI for interquartile range in dat1 is: 3.64999375 4.49150625"
```

Another way for finding a CI for the median is using the Binomial Distribution and Order Statistic. The idea is that $P(X_i > F_{1/2})$, where $F_{1/2}$ is the median, can be expressed as a sum of Binomial probabilities with the the p parameter as $P(X_i \leq F_{1/2})$. The probability parts of

the summation could be computed to 2^{-n}, and the remaining part forms a sum of Binomial Coefficients. This could lead us to finding $P(X_u > F_{1/2})$ and independently $P(X_l < F_{1/2})$, combining this we can set $P(X_l < F_{1/2} < X_u) = 0.95$ and find the corresponding order statistics through the Binomial Distribution.
A very detailed explaination could be found at this website:
https://stats.stackexchange.com/questions/122001/confidence-intervals-for-median

```
> sort(dat1_vec)[qbinom(c(.025,.975), length(dat1_vec), 0.5)]

[1] 1.894 2.819
```

As a comparison, the result of this method is very similar to the one obtained by Bootstrap, with significant computation advantages, and much easier to implement. However, the idea is much less straightforward then Boostrap and the derivation is much less friendlier than the simple idea of replicating the samples. Seems that's the tradeoff.

Thank you very much for answering my questions and for looking over my report.