Linear Non-Linear Course Project

Madhavi Polisetti

June 28, 2018

MScA, Linear and Nonlinear Models (31010)

Course Assignment. Part 1.

1. Problem Description

The business analytics group of a company is asked to investigate causes of malfunctions in technological process of one of the manufacturing plants that result in significant increase of cost for the end product of the business.

One of suspected reasons for malfunctions is deviation of temperature during the technological process from optimal levels. The sample in the provided file contains times of malfunctions in seconds since the start of measurement and minute records of temperature.

2 . Data

The csv contains time stamps of events expressed in seconds. Read and prepare the data.

```
dataPath <- "/Users/mspolisetti/Desktop/R Studio/Linear-Class/Course Project/"
Course.Project.Data<-read.csv(file=paste(dataPath,"MScA_LinearNonLinear_MalfunctionData.csv",sep="/"))
```

3. Create Counting Process, Explore Cumulative Intensity

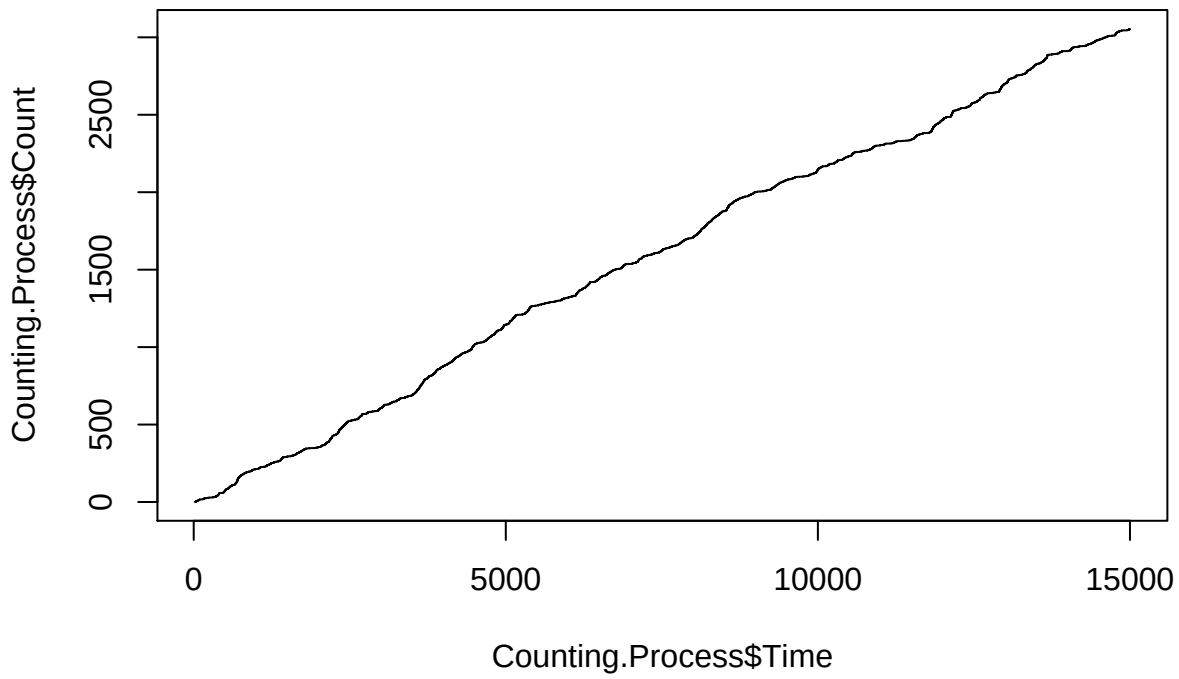
Counting Process is a step function that jumps by 1 at every moment of new event. #### Step 3.0: Data preparation

```
Counting.Process<-as.data.frame(cbind(Time=Course.Project.Data$Time,Count=1:length(Course.Project.Data$
Counting.Process[1:20,]
```

```
##          Time Count
## 1   18.08567     1
## 2   28.74417     2
## 3   34.23941     3
## 4   36.87944     4
## 5   37.84399     5
## 6   41.37885     6
## 7   45.19283     7
## 8   60.94242     8
## 9   66.33539     9
## 10  69.95667    10
## 11  76.17420    11
## 12  80.48524    12
## 13  81.29133    13
## 14  86.18149    14
## 15  91.28642    15
## 16  91.75162    16
## 17  98.29452    17
## 18 142.58741    18
```

```
## 19 149.82484    19
## 20 151.58587    20
```

```
plot(Counting.Process$Time,Counting.Process$Count,type="s")
```



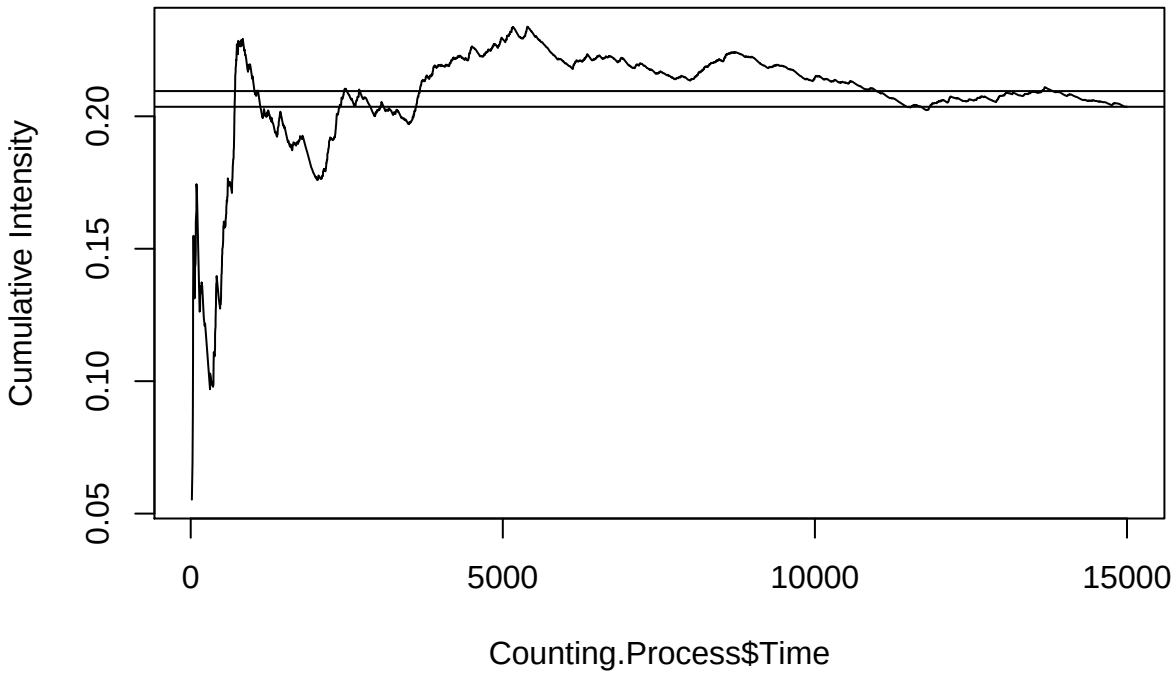
The counting process trajectory looks pretty smooth and grows steadily. ###### 3.0.1: What does it tell you about the character of malfunctions and the reasons causing them? We can tell that the intensity of malfunctions - While highly volatile in the begining,over time, converges to mean value of 0.2. Malfunctions seeem to be happening constantly (at a contant rate) and do not stop over time. Reason for malfunction seems like a manufacturing defect and not wear and tear. If wear and tear were to be the reason for malfunction, we would have seen an exponential curve as frequency of failures would have increased with time.

3.1 Explore cumulative intensity of the process.

Cumulative intensity is calculated as Lambda(t)=frac{N_t}/{t}, where N_t is the number of events during the time interval [0,t].

For our data t is the sequence of time stamps and Nt is the count up until t.

```
plot(Counting.Process$Time,Counting.Process$Count/Counting.Process$Time,type="l",ylab="Cumulative Intens
abline(h=Counting.Process$Count[length(Counting.Process$Count)]/
         Counting.Process$Time[length(Counting.Process$Time)])
abline(h=mean(Counting.Process$Count/Counting.Process$Time))
```



The two horizontal lines on the graph are at the mean cumulative intensity and last cumulative intensity levels. The cumulative intensity seems to converge to a stable level.

```
c(Last.Intensity=Counting.Process$Count[length(Counting.Process$Count)]/
         Counting.Process$Time[length(Counting.Process$Time)],
  Mean.Intensity=mean(Counting.Process$Count/Counting.Process$Time))
```

```
## Last.Intensity Mean.Intensity
##      0.2036008      0.2095305
```

4. Check for over-dispersion.

In order to do that calculate one-minute event counts and temperatures.

Step 4.0: Data Preparation

For example, look at the first 20 rows of the data.

```
Course.Project.Data[1:29,]
```

```
##          Time Temperature
## 1    18.08567    91.59307
## 2    28.74417    91.59307
## 3    34.23941    91.59307
## 4    36.87944    91.59307
## 5    37.84399    91.59307
## 6    41.37885    91.59307
## 7    45.19283    91.59307
## 8    60.94242    97.30860
## 9    66.33539    97.30860
## 10   69.95667    97.30860
## 11   76.17420    97.30860
## 12   80.48524    97.30860
## 13   81.29133    97.30860
```

```
## 14  86.18149    97.30860
## 15  91.28642    97.30860
## 16  91.75162    97.30860
## 17  98.29452    97.30860
## 18 142.58741    95.98865
## 19 149.82484    95.98865
## 20 151.58587    95.98865
## 21 156.37781    95.98865
## 22 161.97298    95.98865
## 23 172.42610    95.98865
## 24 174.79452    95.98865
## 25 184.07435   100.38440
## 26 209.82744   100.38440
## 27 223.35757   100.38440
## 28 230.02632   100.38440
## 29 252.39556    99.98330
```

The Time column is in seconds.

Note that the first 7 rows (events) occurred during the first minute. The temperature measurement for the first minute was 91.59307°F. The following 10 rows happen during the second minute and the second minute temperature is 97.3086°F. The third minute had 7 events at temperature 95.98865°F. The fourth minute had 4 events at 100.3844°F. And the following fifth minute had only 1 event at 99.9833°F.

After constructing a data frame of one-minute counts and the corresponding temperatures we should see. #### 4.0.1: Recreate One Minute Counts Temps table (show first 10 rows)

```
#install.packages("tidyr")
library("tidyr")
library("plyr")
```

```
##
## Attaching package: 'plyr'
```

```
## The following object is masked from 'package:faraway':
##
##     ozone
```

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following object is masked from 'package:car':
##
##     recode
```

```
## The following object is masked from 'package:MASS':
##
##     select
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

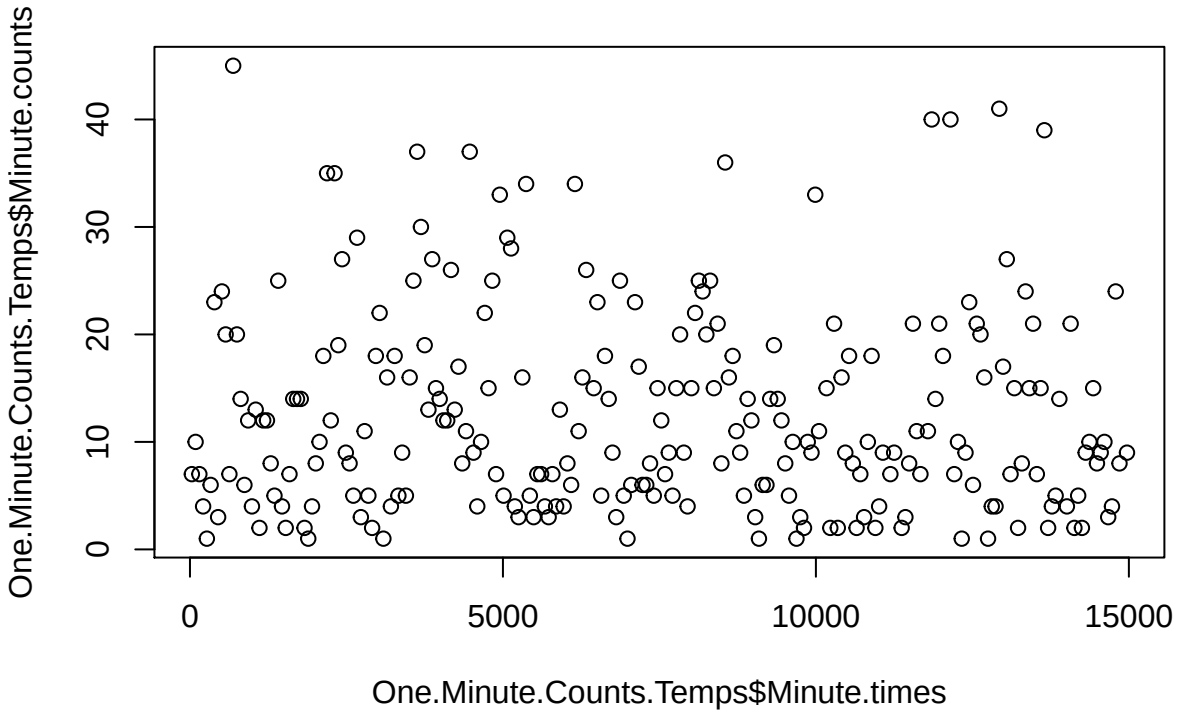
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
g <- group_by(Course.Project.Data, Temperature, Time = as.integer(ceiling(Time/60)* 60 - 30)  )

One.Minute.Counts.Temps <- aggregate( g, by = list(Temperature = g$Temperature,Time = g$Time),
          FUN = function(x){NROW(x)})[,1:3]
names(One.Minute.Counts.Temps) <- c("Minute.Temps", "Minute.times", "Minute.counts" )
head(One.Minute.Counts.Temps)
```

```
##   Minute.Temps Minute.times Minute.counts
## 1     91.59307           30             7
## 2     97.30860           90            10
## 3     95.98865          150             7
## 4    100.38440          210             4
## 5     99.98330          270             1
## 6    102.54126          330             6
```

Plot the one-minute counts.

```
plot(One.Minute.Counts.Temps$Minute.times , One.Minute.Counts.Temps$Minute.counts)
```



4.1 Methods for Testing Over-Dispersion ####4.1.1 A quick and rough method.

Test the method on simulated Poisson data.

```
Test.Deviance.Overdispersion.Poisson<-function(Sample.Size,Parameter.Lambda){
  my.Sample<-rpois(Sample.Size,Parameter.Lambda)
  Model<-glm(my.Sample~1,family=poisson)
  Dev<-Model$deviance
  Deg.Fred<-Model$df.residual
  (((Dev/Deg.Fred-1)/sqrt(2/Deg.Fred)>-1.96)&((Dev/Deg.Fred-1)/sqrt(2/Deg.Fred)<=1.96))*1
}
Test.Deviance.Overdispersion.Poisson(100,1)
```

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

4.1.1.1: Show the number of times the function returned one out of 300 times for the Poisson simulated data

Now repeat the call of the function 300 times to see how many times it returns one and how many times zero.

```
sum(replicate(300,Test.Deviance.Overdispersion.Poisson(100,1)))
```

```
## [1] 264
```

4.1.1.2: Explain the conclusion you would draw based on the number you observe above in 4.1.1.

For Poisson, Deviance should be chi-square distributed with mean equal to the degrees of freedom. So if Dev/Deg.Fred is within 1.96 standard deviations, the data is not over-dispersed.

Results show that 259 out of 300 times the simulated Poisson data passed the test and the sample is not Over dispersed.

4.1.1.3: Estimate and interpret the parameter (does not have to match precisely but should be in ball-park)

The estimate of the parameter lambda given by glm() is e^Coefficient:

```
exp(glm(rpois(1000,2)~1,family=poisson)$coeff)
```

```
## (Intercept)
##       1.971
```

Perform the same test on negative binomial data #### 4.1.1.4: Show the number of times the function returned one out of 300 times for the Negative Binomial simulated data

```
Test.Deviance.Overdispersion.NBinom<-function(Sample.Size,Parameter.prob){
  my.Sample<-rnbinom(Sample.Size,2,Parameter.prob)
  Model<-glm(my.Sample~1,family=poisson)
  Dev<-Model$deviance
  Deg.Fred<-Model$df.residual
  (((Dev/Deg.Fred-1)/sqrt(2/Deg.Fred)>-1.96)&((Dev/Deg.Fred-1)/sqrt(2/Deg.Fred)<=1.96))*1
}
sum(replicate(300,Test.Deviance.Overdispersion.NBinom(100,.2)))
```

```
## [1] 0
```

We see that the over-dispersed negative binomial distribution sample never passes the test. #### 4.1.1.5: Fit a generalized linear model (Poisson regression) to the one-minute event counts that were created in 4.0.1 above. Match all the parameters and output in the summary of the GLM model – coefficient, degrees of freedom, null/residual deviance and AIC
Now apply the test to the one-minute event counts.

```
GLM.model<-glm(One.Minute.Counts.Temps$Minute.counts~1,family=poisson)
GLM.model
```

```
##
## Call:  glm(formula = One.Minute.Counts.Temps$Minute.counts ~ 1, family = poisson)
```

```
##
## Coefficients:
## (Intercept)
##       2.535
##
## Degrees of Freedom: 241 Total (i.e. Null);  241 Residual
## Null Deviance:       1600
## Residual Deviance: 1600  AIC: 2591
```

4.1.1.6: Respond to the question: do you see signs of over-dispersion? If you answer yes, explain how you arrived at that conclusion. If you answer no, explain as well. You must show your method and not just say "yes" or "no"

Do you see signs of over-dispersion?

See below. yes, This is a sign of over-dispersion as the data doesn't follow the quick and rough method. Deviance is greater than 249 (no. of observations - parameter)

```
d<-GLM.model$deviance
 df<-GLM.model$df.residual
  (((d/df-1)/sqrt(2/df)>-1.96)&((d/df-1)/sqrt(2/df)<=1.96))*1
```

```
## [1] 0
```

4.1.2 Regression test by Cameron-Trivedi

4.1.2.1: Apply the dispersiontest() function from the AER package to the GLM model created in 4.1.1.5 above (Poisson regression model). Match all the parameters and output in the summary – z score, p-value, dispersion parameter

```
#install.packages("AER")
suppressWarnings(library(AER))
#?dispersiontest

Disp.Test <- dispersiontest(GLM.model, alternative = c("two.sided"))
Disp.Test
```

```
##
##  Dispersion test
##
## data:  GLM.model
## z = 8.3391, p-value < 2.2e-16
## alternative hypothesis: true dispersion is not equal to 1
## sample estimates:
## dispersion
##    6.97414
```

Does the test show overdispersion?

4.1.2.2: Respond to the question: does the test show over-dispersion? If you answer yes, explain how you arrived at that conclusion. If you answer no, explain as well. You must show your method and not just say "yes" or "no"

Test is not significant. Null hypothesis (standard Poisson GLM models the (conditional) mean E[y] = mu which is assumed to be equal to the variance VAR[y] = mu.) is rejected. Test shows that there is over-dispersion as it supports the alternative hypothesis with a low p-value.

4.1.3 Test against Negative Binomial Distribution

4.1.3.1: Apply the glm.nb() function from the MASS package to fit a negative binomial model the one-minute counts. Comment on the summary of this model (i.e. how did it fit the data?)

The R parameter (theta) is equal to the inverse of the dispersion parameter (alpha) estimated in these other software packages. Thus, the theta value of 2.105 seen here is equivalent to overdispersion of 0.476 because $1/2.105 = 0.476$.

Negative binomial models assume the conditional means are not equal to the conditional variances. This inequality is captured by estimating a dispersion parameter that is held constant in a Poisson model.

```
glm.nb.fit <- glm.nb(One.Minute.Counts.Temps$Minute.counts~1)
summary(glm.nb.fit)
```

```
##
## Call:
## glm.nb(formula = One.Minute.Counts.Temps$Minute.counts ~ 1, init.theta = 2.10539409,
##     link = log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1438  -1.0475  -0.2991   0.5109   2.2079
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.53527    0.04786   52.98   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(2.1054) family taken to be 1)
##
##     Null deviance: 256.3  on 241  degrees of freedom
## Residual deviance: 256.3  on 241  degrees of freedom
## AIC: 1685
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  2.105
##           Std. Err.:  0.219
##
##  2 x log-likelihood:  -1681.007
```

4.1.3.2: Apply the odTest() function from the pscl package to test if the one-minute counts data can be described by Poisson distribution (no over-dispersion) or not (over-dispersion). Match the parameters and output in the summer of the odTest() – test statistic critical value, chi-square test statistic, and p-value.

Does this test show overdispersion?

YES, this test (see below) also shows over dispersion. Here we clearly see that there is evidence of overdispersion (c is estimated to be 2.7055) which speaks quite strongly against the assumption of equidispersion (i.e. c=0).

```
#install.packages("pscl")
suppressWarnings(library(MASS))
suppressWarnings(library(pscl))
odTest(glm.nb.fit)$estimate
```

```
## Likelihood ratio test of H0: Poisson, as restricted NB model:
## n.b., the distribution of the test-statistic under H0 is non-standard
## e.g., see help(odTest) for details/references
##
## Critical value of test statistic at the alpha= 0.05 level: 2.7055
## Chi-Square Test Statistic =  907.6241 p-value = < 2.2e-16
```

```
## NULL
```

5. Find the distribution of Poisson intensity.

5.1. Kolmlgorov-Smirnov test.

Kolmogorov-Smirnov test is used to test hypotheses of equivalence between two empirical distributions or equivalence between one empirical distribution and one theoretical distribution. #####5.1.1: Plot the empirical CDF plot for both Sample 1 and Sample 2 using the ecdfplot() function from the latticeExtra package (you do not need to match the graph exactly since the samples are randomly generated) ######5.1.2: Check equivalence of empirical distributions for the 2 samples by performing the KS Test (you do not need to match the D test statistic or p-value

```
#install.packages("lattice")
#install.packages("latticeExtra")
suppressWarnings(library(lattice))
```

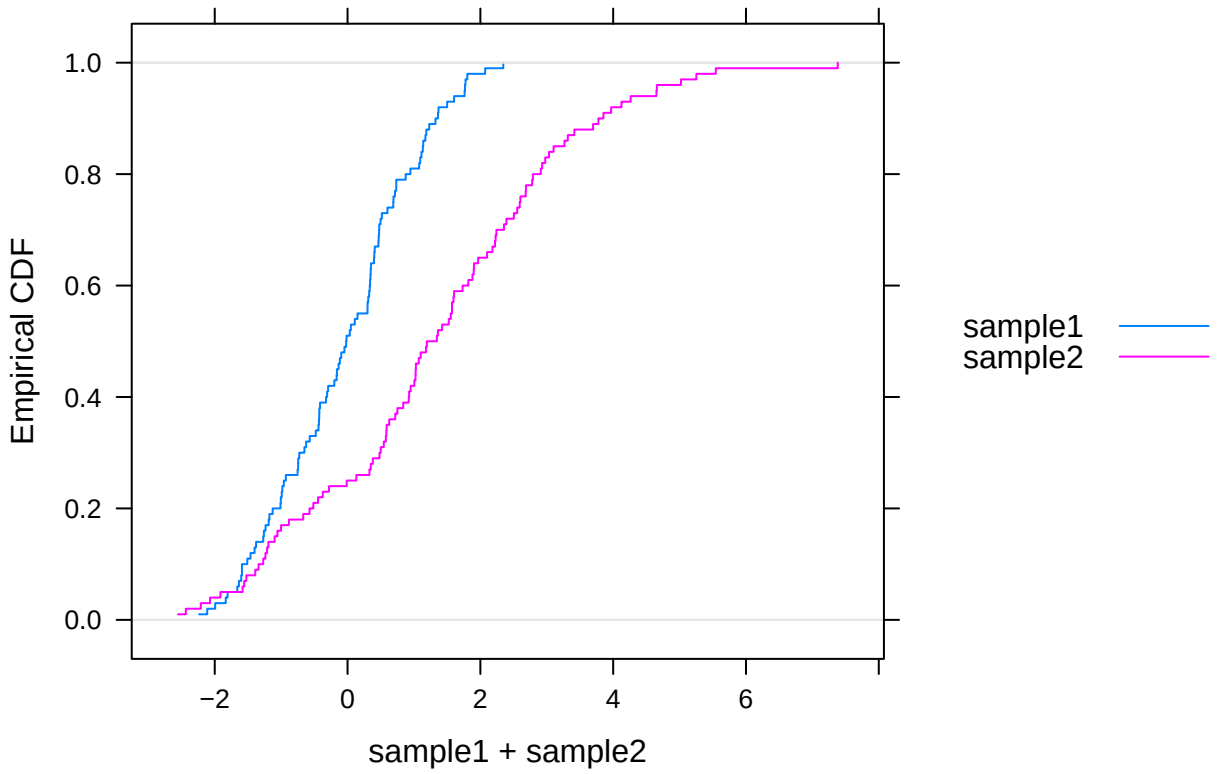
```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:faraway':
##
##     melanoma
```

```
suppressWarnings(library(latticeExtra))
```

```
## Loading required package: RColorBrewer
```

```
sample1=rnorm(100)
sample2=rnorm(100,1,2)
Cum.Distr.Functions <- data.frame(sample1,sample2)
ecdfplot(~ sample1 + sample2, data=Cum.Distr.Functions, auto.key=list(space='right'))
```




```
#Check equivalence of empirical distributions for the two samples.
ks.test(sample1,sample2)
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  sample1 and sample2
## D = 0.42, p-value = 4.366e-08
## alternative hypothesis: two-sided
```

5.1.3: Respond to the question: what does the output tell you about equivalence of the two distributions? Be specific with your response and don't state simply that the two samples are equivalent or not equivalent

What does this output tell you about equivalence of the two distributions?

The null hypothesis is that both datasets were sampled from populations with identical distributions. Small p-value rejects this hypothesis here. Hence, the test says that samples are from different populations with different means/variances

5.1.4: Check equivalence of empirical distributions for the 2 samples and the CDF of a standard normal distribution by performing the KS Test (you do not need to match the D test statistic or p-value and in fact, your conclusion may be different than that which is stated in the assignment). This will consist of 2 separate KS Tests (Sample 1 vs. CDF of Standard Normal and Sample 2 vs. CDF of Standard Normal)

Check equivalence of empirical distribution of sample1 and theoretical distribution Norm(0,1).

```
ks.test(sample1,"pnorm",mean=0,sd=1)
```

```
##
```

```
##  One-sample Kolmogorov-Smirnov test
##
## data:  sample1
## D = 0.0837, p-value = 0.4853
## alternative hypothesis: two-sided
```

What does this output tell you?

Large p-value from the test confirms that the sample is from a standard Normal distribution = Norm(0,1).

```
ks.test(sample2,"pnorm",mean=0,sd=1)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  sample2
## D = 0.43165, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

What does this output tell you?

Small p-value from the test says that the sample is **NOT** from a standard Normal distrubution

5.1.5: Respond to the question: what does the output tell you about equivalence of the two distributions? Be specific with your response and don't state simply that the two distributions are equivalent or not equivalent. Answer the question for each test above in 5.1.4 – 2 points per test (4 POINTS)

While first sample (sample1) is from Standard Normal distribution~Norm(0,1), sample2 is not from Standard Normal distribution ~ Norm(0,1). Hence, the two samples are not equivalent distributions.

5.2. Check the distribution for the entire period.Create the empirical CDF based on time intervals between malfunctions

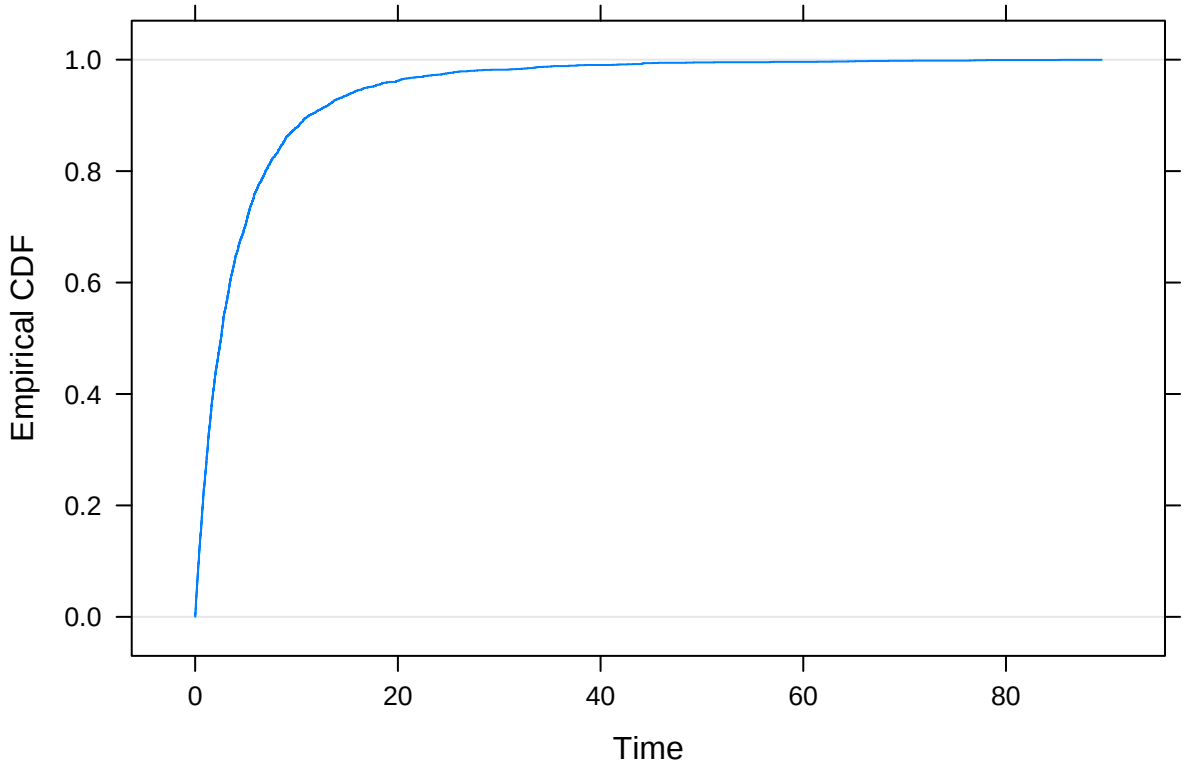
Apply Kolmogorov-Smirnov test to Counting.Process$Time and theoretical exponential distribution with parameter equal to average intensity. Hint: the empirical distribution should be estimated for time intervals between malfunctions.

```
KS.Test.Event.Intervals <- ks.test(Counting.Process$Time,"pexp",rate=0.2095305)

c(KS.Test.Event.Intervals$statistic,p.value=KS.Test.Event.Intervals$p.value)
```

```
##        D  p.value
## 0.998579 0.000000
```

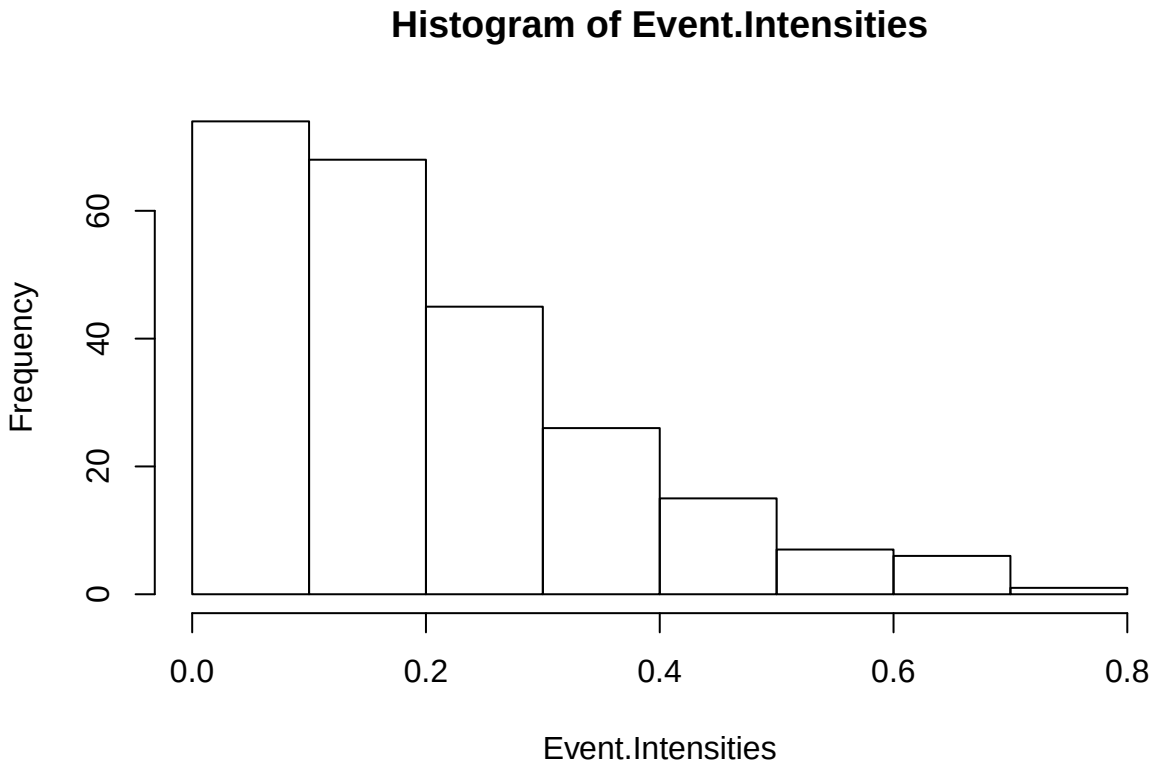
Plot empirical cumulative distribution function for time intervals between malfunctions.

```
Time.Interval.Bet.Malfunctions<-   Counting.Process[-1,] - Counting.Process[-nrow(Counting.Process),]
ecdfplot(~  Time, data=Time.Interval.Bet.Malfunctions, auto.key=list(space='right'))
```



#5.3. Check distribution of one-minute periods Use at least 5 different candidates for distribution of Poisson intensity of malfunctions. Find one-minute intensities Event.Intensities. Hint. One-minute intensity by definition is the number of events per unit of time (second).

```
Event.Intensities <- na.omit(as.vector(One.Minute.Counts.Temps$Minute.counts/(60)))

hist(Event.Intensities  )
```



####What distribution does this histogram remind you of? Even though this reminds me of exponential distribution, this could be gamma/Normal/Binomial/Weibull distributions.

Suggest 5 candidates for the distribution.

Exponential distribution/gamma/Normal/Binomial/Weibull

Fit each of you 5 candidate distributions to Event.Intensities using fitdistr() from MASS. Recommendation: start with fitting normal and exponential distributions first.

5.3.4: Fit the one-minute intensities to a normal distribution using the fitdistr() function from the package MASS. Match the estimated parameters (mean and sd)

5.3.6: Fit the one-minute intensities to an exponential distribution using the fitdistr() function from the package MASS. Match the estimated parameter (rate)

```
#install.packages("MASS")
#Fitting Norma distribution first
suppressWarnings(library(MASS))
Fitting.Normal<-fitdistr(Event.Intensities,"normal")
Fitting.Exponential<-fitdistr(Event.Intensities,"exponential")

#Normal - Compare estimated parameter
Fitting.Normal$estimate
```

```
##      mean        sd
## 0.2103306 0.1563583
```

```
mean(Event.Intensities)
```

```
## [1] 0.2103306
```

```r
sd(Event.Intensities)
```

```
## [1] 0.1566824
```

```r
#Exponential - Match the estimated parameter (rate)
Fitting.Exponential$estimate
```

```
##    rate
## 4.75442
```

5.3.5: Perform the KS Test on the one-minute intensities (empirical distribution) and a theoretical normal distribution. Match the D test statistic and p-value and comment on the results

5.3.7: Perform the KS Test on the one-minute intensities (empirical distribution) and a theoretical exponential distribution. Match the D test statistic and p-value and comment on the results

```r
#Test the fitted distributions with Kolmogorov-Smirnov test.
KS.Normal <- ks.test(Event.Intensities,"pnorm", mean=mean(Event.Intensities), sd=sd(Event.Intensities))
```

```
## Warning in ks.test(Event.Intensities, "pnorm", mean =
## mean(Event.Intensities), : ties should not be present for the Kolmogorov-
## Smirnov test
```

```r
KS.Exp <- ks.test(Event.Intensities,"pexp",rate= Fitting.Exponential$estimate)
```

```
## Warning in ks.test(Event.Intensities, "pexp", rate =
## Fitting.Exponential$estimate): ties should not be present for the
## Kolmogorov-Smirnov test
```

```r
#Print the kstest results
c(KS.Normal$statistic,P.Value=KS.Normal$p.value)
```

```
##            D      P.Value
## 0.1333709787 0.0003648017
```

```r
c(KS.Exp$statistic,P.Value=KS.Exp$p.value)
```

```
##            D      P.Value
## 0.1352774668 0.0002847072
```

What do you conclude from these tests?

Low p-values suggest that we don't have enough evidence to say that the data is from Normal & Exponential distributions.

5.3.8: Estimate the parameters of a Gamma distribution for the one-minute intensities using the method of moments. Match the estimated parameters (rate and shape)

Try to fit gamma distribution directly using fitdistr()

```r
#install.packages("fitdistrplus")
suppressWarnings(library(fitdistrplus))
```

```
 n <- length(Event.Intensities)
 n
```

```
## [1] 242
```

```
#Using method of moments
gamma.mean <- mean(Event.Intensities)
gamma.variance <- (sd(Event.Intensities)^2) * ((n-1)/n)

#scale <- v/m
#shape <- m*m/v


#shape
(Moments.Shape <- gamma.mean ^2 / gamma.variance)
```

```
## [1] 1.809518
```

```
#rate
(Moments.Rate <- gamma.mean/gamma.variance)
```

```
## [1] 8.603211
```

```
#Check gamma distribution with these parameters as a theoretical distribution using Kolmogorov-Smirnov
(KS.Test.Moments <-  ks.test(Event.Intensities,"pgamma",rate = Moments.Rate,shape=Moments.Shape))
```

```
## Warning in ks.test(Event.Intensities, "pgamma", rate = Moments.Rate, shape
## = Moments.Shape): ties should not be present for the Kolmogorov-Smirnov
## test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  Event.Intensities
## D = 0.061123, p-value = 0.3265
## alternative hypothesis: two-sided
```

Find at least 2 more candidates and test them by Kolmogorov-Smirnov.

```
(Fitting.LogNormal<-fitdistr(Event.Intensities,  "log-normal"))
```

```
##      meanlog        sdlog
##   -1.87594922    0.86727364
##  ( 0.05575046) ( 0.03942153)
```

```
(KS.LogNormal <- ks.test(Event.Intensities,"plnorm",Fitting.LogNormal$estimate))
```

```
## Warning in ks.test(Event.Intensities, "plnorm",
## Fitting.LogNormal$estimate): ties should not be present for the Kolmogorov-
## Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  Event.Intensities
## D = 0.89018, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```r
#(Fitting.Beta<-fitdistr(Event.Intensities,  "beta", start=list(shape1=0.1,shape2=0.1)))
#(KS.Beta <- ks.test(Event.Intensities,"pbeta",shape1=Fitting.Beta$estimate[0],shape2=Fitting.Beta$esti

(Fitting.weibull <- fitdistr(Event.Intensities,densfun=dweibull,start=list(scale=1,shape=2)))
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

## Warning in densfun(x, parm[1], parm[2], ...): NaNs produced

##      scale        shape
##   0.23082533   1.38023856
##  (0.01134642) (0.06900230)
```

```r
(ks.wei <- ks.test(Event.Intensities,"pweibull", shape=2,scale=1))
```

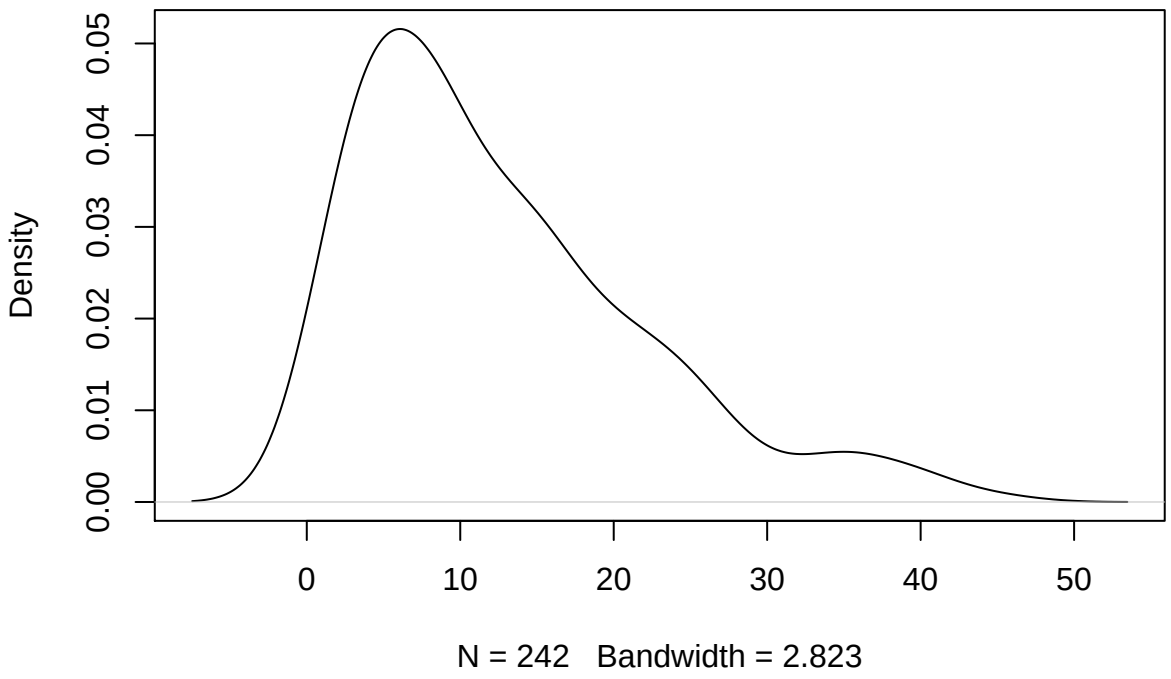
```
## Warning in ks.test(Event.Intensities, "pweibull", shape = 2, scale = 1):
## ties should not be present for the Kolmogorov-Smirnov test

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  Event.Intensities
## D = 0.74558, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```r
#(KS.Beta <- ks.test(Event.Intensities,"pbeta",shape1=Fitting.Beta$estimate[0],shape2=Fitting.Beta$esti
```

What distribution for the one-minute intensity of malfunctions do you choose?

I chose Gamma distrubition because ks.test returns significant results.

```r
plot(density(One.Minute.Counts.Temps$Minute.counts))
```



##What distribution of one-minute malfunctions counts follow from your choice? Based on the plot, counts seem to be following normal distribution.

```
suppressWarnings(library(fitdistrplus))


#Using method of moments
gamma.mean.malfcounts <- mean(Event.Intensities * 60)
gamma.variance.malfcounts <- sd(Event.Intensities * 60)^2

#scale <- v/m
#shape <- m*m/v


#shape
(Moments.Shape.malfcounts <- gamma.mean.malfcounts ^2 / gamma.variance.malfcounts)
```

```
## [1] 1.802041
```

```
#rate
(Moments.Rate.malfcounts <- gamma.mean.malfcounts/gamma.variance.malfcounts)
```

```
## [1] 0.1427943
```

```
#Check gamma distribution with these parameters as a theoretical distribution using Kolmogorov-Smirnov
(KS.Test.Moments.malfcounts <-  ks.test(Event.Intensities * 60,"pgamma",rate = Moments.Rate.malfcounts,s
```

```
## Warning in ks.test(Event.Intensities * 60, "pgamma", rate =
## Moments.Rate.malfcounts, : ties should not be present for the Kolmogorov-
## Smirnov test
```

```
##
##  One-sample Kolmogorov-Smirnov test
```

```
##
## data:  Event.Intensities * 60
## D = 0.060326, p-value = 0.3419
## alternative hypothesis: two-sided
```

Write One.Minute.Counts.Temps to file OneMinuteCountsTemps.csv to continue working on Part 2.

```
write.csv(One.Minute.Counts.Temps,file="OneMinuteCountsTemps.csv",row.names=FALSE)
```