HS650 Homework Header
HW ______3________________________
Winter 2018, DSPA (HS650)
Name: _Xinchun Li____
SID: _85922126____
UMich E-mail: _____xincli_______@umich.edu
I certify that the following paper represents my own independent work and conforms with the guidelines of academic honesty described in the UMich student handbook.

Problem 3.1 (Probability Distributions):

Complete the following tasks for each of the probability distributions below:

- **Generate plots of the [density, CDF, and the quantile (inverse-CDF) functions (Links to an external site.)Links to an external site.]()**
- **Report the first [4 moments (mean, variance, skewness, kurtosis) (Links to an external site.)Links to an external site.]()**
- **Complete the discrete probability distributions table below. The cell values in the table represent the values of the quantile function for the corresponding p-value (column) and distribution (row).**

 # Compute the values of the quantile function for the corresponding p-value and distribution

```
>s <- seq(0.1, 0.9, 0.1)

>sweibull <- qweibull(s, 1, 5)

>sunif <- qunif(s, 1, 10)

>sqt <- qt(s, 1)

>scauchy <- qcauchy(s)

>sbinom <- qnbinom(s, 10, 0.5)

>schisq <- qchisq(s, 10)
```

```
>stable <- cbind(sweibull, sunif, sqt, scauchy, sbinom, schisq)
>t(stable)
```

Output

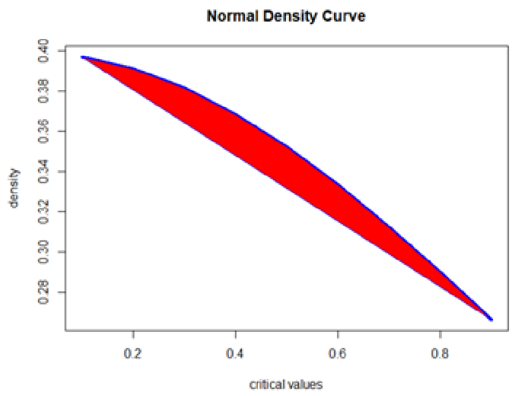
```
sweibull sunif           sqt      scauchy sbinom      schisq
 [1,]   0.5268026    1.9 -3.0776835 -3.0776835        5   4.865182
 [2,]   1.1157178    2.8 -1.3763819 -1.3763819        6   6.179079
 [3,]   1.7833747    3.7 -0.7265425 -0.7265425        7   7.267218
 [4,]   2.5541281    4.6 -0.3249197 -0.3249197        8   8.295472
 [5,]   3.4657359    5.5   0.0000000   0.0000000        9   9.341818
 [6,]   4.5814537    6.4   0.3249197   0.3249197       11 10.473236
 [7,]   6.0198640    7.3   0.7265425   0.7265425       12 11.780723
 [8,]   8.0471896    8.2   1.3763819   1.3763819       14 13.441958
 [9,] 11.5129255     9.1   3.0776835   3.0776835       16 15.987179
```

Distribution Table

```
>smatrix <- as.matrix(t(stable))
>stable1 <- knitr::kable(smatrix, digits = 4, col.names = c("0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9"))
> dnorm(s)
[1] 0.3969525 0.3910427 0.3813878 0.3682701 0.3520653 0.3332246 0.3122539 0.2896916 0.2660852
```

Density Plot

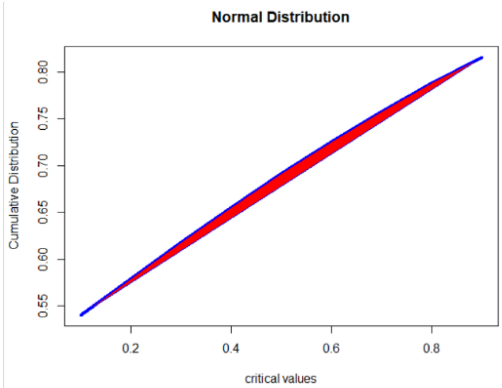
```
>dStandardNormal <- data.frame(Z=s, Density=dnorm(s, mean=0, sd=1), Distribution=pnorm(s, mean=0, sd=1))
>plot(s, dStandardNormal$Density, main="Normal Density Curve", type = "l", xlab = "critical values", ylab="density", lwd=4, col="blue")
>polygon(s, dStandardNormal$Density, col="red", border="blue")
```



Cumulative Distribution Function Plot

>plot(s, dStandardNormal$Distribution, main="Normal Distribution", type = "l", xlab = "critical values", ylab="Cumulative Distribution", lwd=4, col="blue")

>polygon(s, dStandardNormal$Distribution, col="red", border="blue")

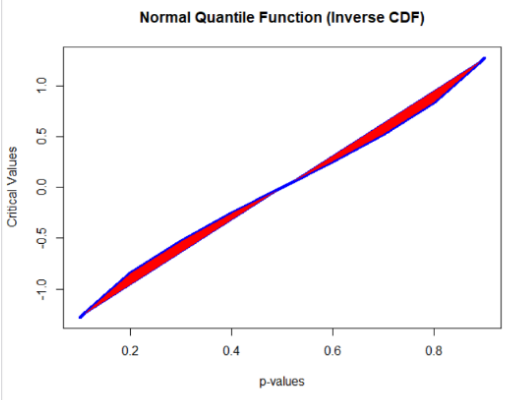


The Quantile Function Plot

>qStandardNormal <- data.frame(Q=s, Quantile=qnorm(s, mean=0, sd=1))

>plot(s, qStandardNormal$Quantile, main="Normal Quantile Function (Inverse CDF)", type = "l", xlab = "p-values", ylab="Critical Values", lwd=4, col="blue")

>polygon(s, qStandardNormal$Quantile, col="red", border="blue")



Report Four Moments (mean, variance, skewness, kurtosis) for each set of data in a table

>weibull_moments <- c(mean(sweibull), var(sweibull), skewness(sweibull), kurtosis(sweibull))

>unif_moments <- c(mean(sunif), var(sunif), skewness(sunif), kurtosis(sunif))

>qt_moments <- c(mean(sqt), var(sqt), skewness(sqt), kurtosis(sqt))

>cauchy_moments <- c(mean(scauchy), var(scauchy), skewness(scauchy), kurtosis(scauchy))

>binom_moments <- c(mean(sbinom), var(sbinom), skewness(sbinom), kurtosis(sbinom))

>chisq_moments <- c(mean(schisq), var(schisq), skewness(schisq), kurtosis(schisq))

>four_moments <- c(weibull_moments, unif_moments, qt_moments, cauchy_moments, binom_moments, chisq_moments)

>four_moments

[1] 4.400799e+00 1.293345e+01 7.019402e-01 -9.121017e-01 5.500000e+00 6.075000e+00 -2.397404e-16 -1.601481e+00

[9] 1.356457e-16 3.000000e+00 2.845866e-16 -6.888889e-01 1.356457e-16 3.000000e+00 2.845866e-16 -6.888889e-01

[17] 9.777778e+00 1.394444e+01 2.954668e-01 -1.492591e+00 9.736874e+00 1.277971e+01 3.044958e-01 -1.300956e+00

Problem 3.2 (Matrix equation solution):

Solve the following system of linear equations and validate your solution. Validate your solution.

6x + 3y - 3z + w = 2

7x + y + 2z + 2w = 5

5x + 3y - 3z + w = 3

-6x - 2y + 3z = 6

```
A_matrix_values <- c(6, 3, -3, 1, 7, 1, 2, 2, 5, 3, -3, 1, -6, -2, 3, 0)

A <- matrix(A_matrix_values, nrow = 4, ncol = 4)

b <- c(2, 5, 3, 6)

# to solve Ax = b, x = A ^ {-1} * b

x <- solve(A, b)

# Ax = b ==> x = A ^ {-1} * b

x

[1] -25    3   25   -1
```

Verification:

```
> A.inverse <- solve(A) # the inverse matrix A^{-1}

> x1 <- A.inverse %*% b

> x1

     [,1]

[1,]  -25

[2,]    3

[3,]   25

[4,]   -1
```

As we can see, the x and x1 are the same.

Problem 3.3 (Dimensionality reduction)

Use PCA and t-SNE to analyze and interpret the monthly US Federal Reserve Monetary-Base Data (1959-2009)Links to an external site.

 # PCA

> ReserveData.sub <- ReserveData[, -1]

We need to center the ReserveData.sub by subtracting the average of all column means from each element in the column. Next, we cast ReserveData.sub as a matrix and compute its variance covariance matrix, S. Finally, we can calculate the corresponding eigenvalues and eigenvectors of S.

> mu <- apply(ReserveData.sub, 2, mean)

> mean(mu)

[1] 729.8481

> ReserveData.center <- as.matrix(ReserveData.sub)-mean(mu)

> S <- cov(ReserveData.center)

> eigen(S)

eigen() decomposition

$values

[1] 6.459371e+06 8.105094e+04 1.954341e+04 5.516879e+03 1.436806e+03 7.460492e+02 1.106305e+00

$vectors

[,1] [,2] [,3] [,4] [,5] [,6] [,7]

[1,] 0.43802915 0.840194054 0.15824598 -0.22991060 -0.154725598 0.01831078 0.005333293

[2,] 0.86861886 -0.366656200 0.02497568 0.32952256 0.008871566 0.04212471 -0.001550690

[3,] 0.17100350 -0.373093272 -0.03754267 -0.84575739 -0.331941670 -0.06604044 -0.017250689

[4,] 0.11643248 0.097111962 -0.60687967 -0.17742808 0.488043646 0.06809777 -0.578277773

[5,] 0.01957359 0.071528131 -0.64779682 0.02767588 -0.186822349 0.44733754 0.582306156

[6,] 0.01238692 0.076528374 -0.42823496 0.19664207 -0.373796867 -0.79510424 -0.004673683

[7,] 0.10147146 0.005915522 0.03995173 -0.22876986 0.673039912 -0.39571449 0.571107765

The next step would be calculating the PCs using the prcomp() function in R. Note that we will use the raw (uncentered) version of the data and have to specify the center=TRUE option to ensure the column means are trivial. We can save the model information into pca1 where pca1$rotation provides the loadings for each PC.

> pca1<-prcomp(as.matrix(ReserveData.sub), center = T)

> summary(pca1)

Importance of components:

PC1 PC2 PC3 PC4 PC5 PC6 PC7

Standard deviation 2541.5292 284.69447 139.79773 74.27570 37.90522 27.31390 1.052

Proportion of Variance 0.9835 0.01234 0.00298 0.00084 0.00022 0.00011 0.000

Cumulative Proportion 0.9835 0.99585 0.99883 0.99967 0.99989 1.00000 1.000

> pca1$rotation

PC1 PC2 PC3 PC4
PC5 PC6 PC7

SAVINGSL 0.43802915 0.840194054 0.15824598 0.22991060 -
0.154725598 -0.01831078 0.005333293

M2SL 0.86861886 -0.366656200 0.02497568 -0.32952256
0.008871566 -0.04212471 -0.001550690

M1NS 0.17100350 -0.373093272 -0.03754267 0.84575739 -
0.331941670 0.06604044 -0.017250689

BOGAMBSL 0.11643248 0.097111962 -0.60687967 0.17742808
0.488043646 -0.06809777 -0.578277773

TRARR 0.01957359 0.071528131 -0.64779682 -0.02767588 -
0.186822349 -0.44733754 0.582306156

BORROW 0.01238692 0.076528374 -0.42823496 -0.19664207 -
0.373796867 0.79510424 -0.004673683

CURRCIR 0.10147146 0.005915522 0.03995173 0.22876986
0.673039912 0.39571449 0.571107765

We notice that the loadings are just the eigenvectors times -1. These loadings represent a vector in 6D space (we have 6 columns in the original data). The scale factor -1 just represents the opposite direction of the eigenvector. We can also load the factoextra package an compute the eigenvalues of each PC.

> library(factoextra)

> eigen<-get_eigenvalue(pca1)

> eigen

eigenvalue variance.percent cumulative.variance.percent

Dim.1 6.459371e+06 9.835109e+01
98.35109

Dim.2 8.105094e+04 1.234090e+00
99.58518

Dim.3 1.954341e+04 2.975700e-01
99.88275

Dim.4 5.516879e+03 8.400060e-02
99.96675

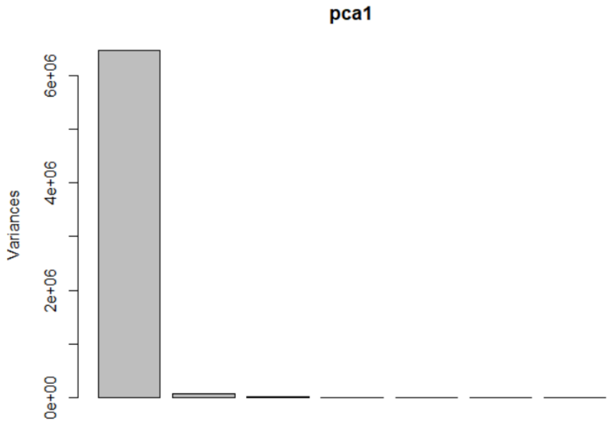
Dim.5 1.436806e+03 2.187696e-02
99.98862

Dim.6 7.460492e+02 1.135943e-02
99.99998

Dim.7 1.106305e+00 1.684473e-05
100.00000

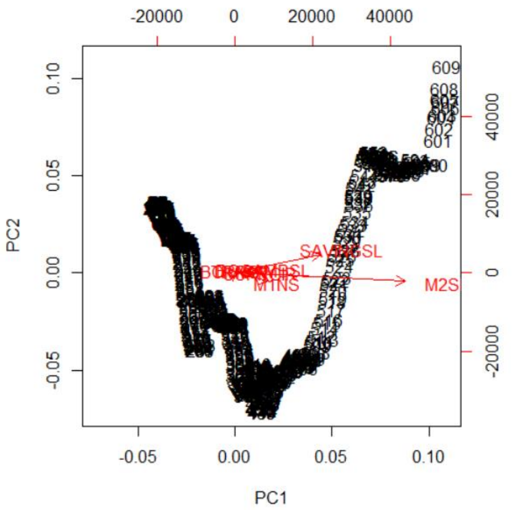
To see a detailed information about the variances that each PC explain we utilize the plot() function to visualize the PC loadings.

>plot(pca1)



>library(graphics)

>biplot(pca1, choices = 1:2, scale = 1, pc.biplot = F)



library("factoextra")

Data for the supplementary qualitative variables

```
>qualit_vars <- as.factor(ReserveData.sub$CURRCIR)
>head(qualit_vars)
>fviz_pca_biplot(pca1, axes = c(1, 2), geom = c("point", "text"),
                 col.ind = "black", col.var = "steelblue", label = "all",
                 invisible = "none", repel = T, habillage =
qualit_vars,
                 palette = NULL, addEllipses = TRUE, title = "PCA -
Biplot")

>install.packages("tsne")
>library(tsne)
>install.packages("Rtsne")
>library(Rtsne)
> dim(ReserveData)
[1] 609    8
```

 # t-SNE

Identify the label-nomenclature - digits 0, 1, 2, ..., 9 - and map to diff colors

```
> ReserveData.borrow<-ReserveData$BORROW

> ReserveData$BORROW<-as.factor(ReserveData$BORROW)

> ReserveData.borrow.colors =
rainbow(length(unique(ReserveData$BORROW)))

> names(ReserveData.borrow.colors) =
unique(ReserveData$BORROW)
```

May need to check and increase the RAM allocation

```
> memory.limit()

[1] 4012

> memory.limit(50000)

[1] 50000
```

Run the t-SNE, tracking the execution time (artificially reducing the sample-size to get reasonable calculation time)

```
> execTime_tSNE <- system.time(tsne_digits <-
Rtsne(ReserveData[1:10000 ,-1], dims = 2, perplexity=30,
verbose=TRUE, max_iter = 500))

Read the 609 x 50 data matrix successfully!

Using no_dims = 2, perplexity = 30.000000, and theta = 0.500000

Computing input similarities...

Normalizing input...

Building tree...

 - point 0 of 609

Done in 0.35 seconds (sparsity = 0.172109)!
```

Learning embedding...

Iteration 50: error is 54.090328 (50 iterations in 0.68 seconds)

Iteration 100: error is 47.739200 (50 iterations in 0.54 seconds)

Iteration 150: error is 46.353491 (50 iterations in 0.55 seconds)

Iteration 200: error is 45.691023 (50 iterations in 0.56 seconds)

Iteration 250: error is 45.212232 (50 iterations in 0.60 seconds)

Iteration 300: error is 0.268417 (50 iterations in 0.62 seconds)

Iteration 350: error is 0.188355 (50 iterations in 0.75 seconds)

Iteration 400: error is 0.166797 (50 iterations in 0.57 seconds)

Iteration 450: error is 0.157891 (50 iterations in 0.57 seconds)

Iteration 500: error is 0.151636 (50 iterations in 0.57 seconds)

Fitting performed in 6.02 seconds.

> execTime_tSNE

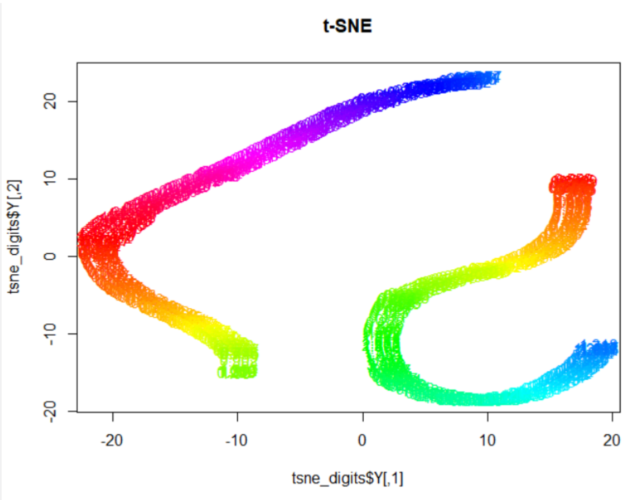
 user   system   elapsed

 8.23   0.08 11.17

> plot(tsne_digits$Y, t='n', main="t-SNE") # don't plot the points to avoid clutter

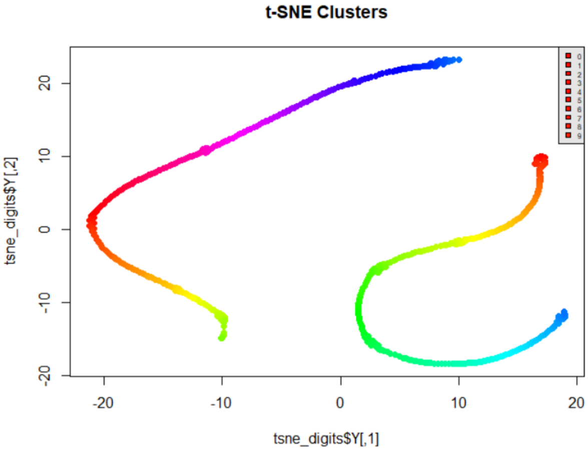
> text(tsne_digits$Y, labels=names(ReserveData.borrow.colors), col=ReserveData.borrow.colors)

> Y



>plot(tsne_digits$Y, main="t-SNE Clusters", col=ReserveData.borrow.colors, pch = 19)

>legend("topright", c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9"), fill=ReserveData.borrow.colors, bg='gray90', cex=0.5)



Problem 3.4 (Least Squares Estimation)

Use the SOCR Knee Pain datasetLinks to an external site., extract the RB = Right-Back locations (x,y), and fit in a linear model for vertical location (y) in terms of the horizontal location (x). Display the linear model on top of the scatter plot of the paired data.

 # Load the SOCR Knee Pain Data

>wiki_url = read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_KneePainData_041409")

>html_nodes(wiki_url, "#content")

>KneePain = html_table(html_nodes(wiki_url, "table")[[2]])

>KneePainData = as.data.frame(KneePain)

>KneePainData_sub = subset(KneePainData, View = RB)

>x=KneePainData_sub$x

>y=KneePainData_sub$Y>X <- cbind(1, x)

>beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y

###or

>beta_hat <- solve(crossprod(X)) %*% crossprod(X, y)

 # Now we can see the results of this by computing the estimated β^0+β^1xβ^0+β^1x for any value of xx:

>newx <- seq(min(x), max(x), len=100)

>X <- cbind(1, newx)

>fitted <- X%*%beta_hat

>plot(x, y)

>lines(newx, fitted, col=2)