Assignment 3: Multidimensional Scaling

Ankit Tewari and Hong-Tan Lam

1

Let us first import the data file of cereals in our R environment. The following achieves this objective and prints the first five observations.

```
cereals <- read.table("http://www.public.iastate.edu/~maitra/stat501/datasets/cereals.dat", header = F)
names(cereals) <- c("Brand", "Manufacturer", "Calories", "Protein", "Fat", "Sodium", "Fiber", "Carbohyd
head(cereals, 5)
```

```
##           Brand Manufacturer Calories Protein Fat Sodium Fiber
## 1    ACCheerios            G      110       2   2    180   1.5
## 2      Cheerios            G      110       6   2    290   2.0
## 3    CocoaPuffs            G      110       1   1    180   0.0
## 4  CountChocula            G      110       1   1    180   0.0
## 5 GoldenGrahams            G      110       1   1    280   0.0
##   Carbohydrates Sugar Potassium Class
## 1          10.5    10        70     1
## 2          17.0     1       105     1
## 3          12.0    13        55     1
## 4          12.0    13        65     1
## 5          15.0     9        45     1
```

2

Now, we will first standardized the variable under consideration according to our problem statement. The results of this operation are printed using the following code.

```
cereals_standardized <- scale(cereals[,3:10], center = TRUE, scale = TRUE)
head(cereals_standardized)
```

```
##       Calories    Protein       Fat       Sodium      Fiber Carbohydrates
## [1,] 0.1103426 -0.3806804 1.2767742 -0.005871679 -0.1189104    -0.8822330
## [2,] 0.1103426  2.8931707 1.2767742  1.382780515  0.1589780     0.6446037
## [3,] 0.1103426 -1.1991431 0.0290176 -0.005871679 -0.9525758    -0.5298861
## [4,] 0.1103426 -1.1991431 0.0290176 -0.005871679 -0.9525758    -0.5298861
## [5,] 0.1103426 -1.1991431 0.0290176  1.256539406 -0.9525758     0.1748078
## [6,] 0.1103426  0.4377824 0.0290176  0.877816080 -0.1189104    -0.6473351
##           Sugar   Potassium
## [1,]  0.5280395 -0.21810133
## [2,] -1.4559536  0.31132205
## [3,]  1.1893705 -0.44499706
## [4,]  1.1893705 -0.29373324
## [5,]  0.3075958 -0.59626088
## [6,]  0.5280395  0.08442632
```

Now, we will compute the distance matrix of our standardized variables and paste the specimen of first 5 observations from our distance matrix named `distance` in this report.

```
distance <- dist(cereals_standardized, method = "euclidean", upper = TRUE, diag = TRUE)
distance <- as.matrix(distance)
head(distance,5)
```

```
##          1        2         3         4        5        6        7
## 1 0.000000 4.389923 1.8800970 1.8678873 2.413378 1.776058 3.455752
## 2 4.389923 0.000000 5.5151628 5.4964619 4.869285 3.684407 3.962058
## 3 1.880097 5.515163 0.0000000 0.1512638 1.700201 2.210626 3.327191
## 4 1.867887 5.496462 0.1512638 0.0000000 1.720269 2.179354 3.340917
## 5 2.413378 4.869285 1.7002007 1.7202688 0.000000 2.169286 2.115457
##           8        9       10       11       12       13       14
## 1 1.6192301 2.037995 1.692973 1.681436 3.325714 3.798287 2.771250
## 2 4.8239397 3.849793 3.765158 3.868147 4.123082 4.992484 3.117500
## 3 0.8476301 2.365674 2.754506 3.113207 3.186540 4.206123 3.497958
## 4 0.8610212 2.336478 2.704207 3.053845 3.204440 4.112606 3.465098
## 5 1.7945462 1.912533 3.071040 3.577630 2.334208 4.523253 3.086864
##          15       16       17       18       19       20       21       22
## 1 2.0443430 2.888244 1.893207 6.577956 2.818565 4.240844 3.062161 2.591681
## 2 5.5993313 3.108638 4.095663 6.390159 5.751471 4.425505 6.170433 4.189560
## 3 0.7514574 3.568236 1.773684 7.554684 1.806802 3.896519 1.881682 4.312430
## 4 0.8513833 3.536029 1.773684 7.475524 1.844401 3.911172 1.929708 4.259042
## 5 1.9680463 3.113561 1.467644 7.561212 2.917822 2.686999 2.901732 4.644644
##         23       24       25       26       27       28       29       30
## 1 3.923052 1.712732 2.868150 3.788553 3.962087 2.186708 3.616425 1.979298
## 2 4.516607 5.214771 5.603114 5.338347 4.872641 4.069030 4.909135 4.436180
## 3 3.501791 1.288037 1.597044 3.858931 4.153263 2.181978 4.243838 1.513060
## 4 3.521339 1.340270 1.646424 3.835141 4.081017 2.181978 4.189576 1.543008
## 5 2.626153 2.553318 1.796336 4.553077 4.218383 1.901755 4.547773 1.498137
##         31       32       33       34       35       36       37       38
## 1 3.386087 3.876270 4.228492 3.730262 4.337000 2.259963 4.639299 1.580069
## 2 3.520707 4.015880 3.808077 4.712787 4.569048 5.858332 2.816341 5.236607
## 3 4.156754 4.036184 4.044825 4.350095 3.824121 1.910388 4.975300 1.493297
## 4 4.118043 4.019142 4.053302 4.254365 3.839050 1.934193 4.977599 1.531124
## 5 3.731677 3.660273 2.864802 4.544612 2.592047 3.427599 4.564946 1.834499
##         39       40       41       42       43
## 1 1.263239 2.054943 5.353324 5.141861 4.604046
## 2 5.017712 2.949282 7.099531 6.641340 5.350988
## 3 1.617395 3.462659 5.035857 5.113100 5.859570
## 4 1.638478 3.439454 5.056262 5.117573 5.840013
## 5 1.846190 3.646725 5.331019 5.509723 6.495437
```

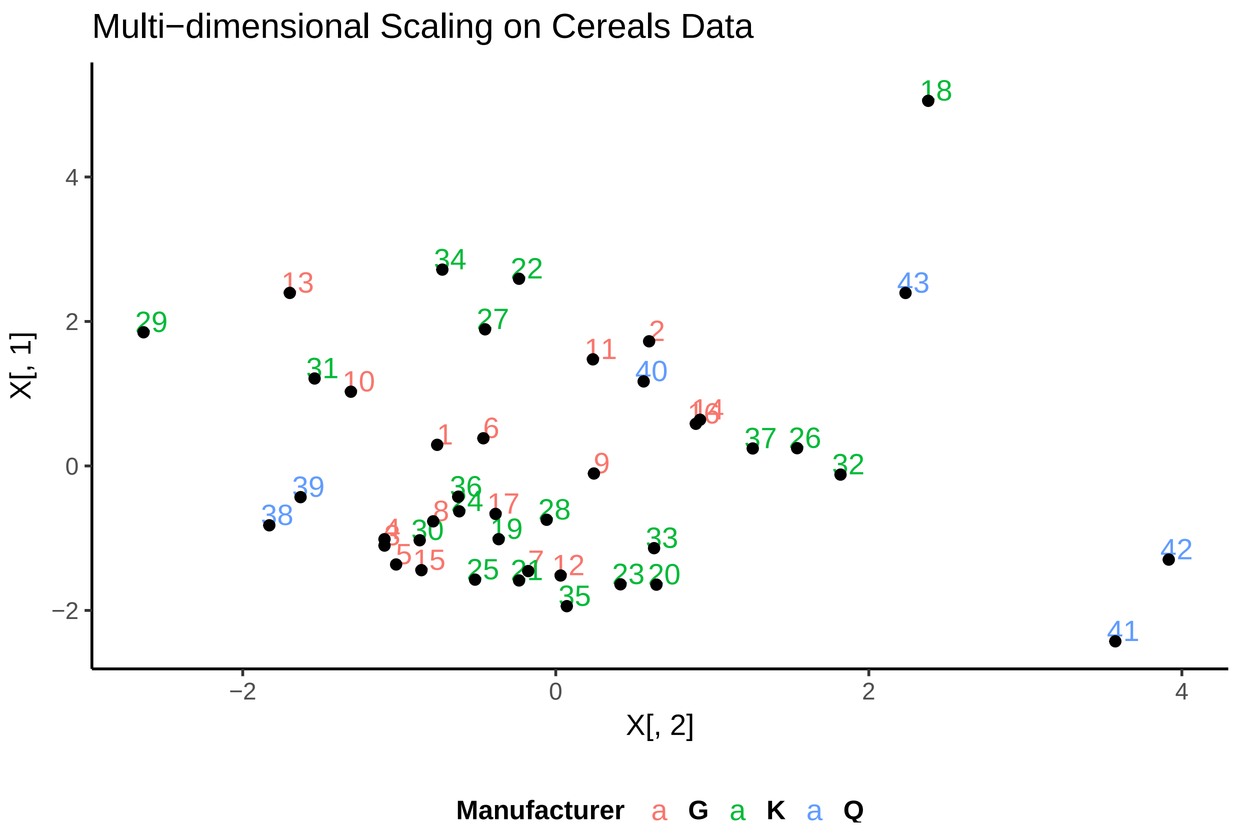
3

Now we will perform the multidimensional scaling of our distance matrix using the `cmdscale` command and store the results in our variable named as `MDS`.

```
n = nrow(distance)
MDS <- cmdscale(distance, k= 2, eig = TRUE, x.ret = TRUE)
X <- MDS$points[,1:2]
```

Now, in order to make prettier plots, we have used the `ggplot2`. In the follwoing plot of two dimensional solution of multidimensional scaling, we have used numbers to specify the observation and colour to specify the manufacturer. The tomato colour corresponds to G, green to K and skybblue to Q where G,K, Q are respective manufacturers.

```r
library(ggplot2)
g <- ggplot(data.frame(X, cereals),
            aes(X[,2], X[,1], label = rownames(cereals), group=Manufacturer))

g + geom_text(aes(color = Manufacturer) ,
              angle=0, show.legend = T, nudge_x = 0.05, nudge_y = 0.15) +
  ggtitle("Multi-dimensional Scaling on Cereals Data") +
  geom_point() + theme_classic() +
  theme(legend.position = "bottom", legend.title =
          element_text(size=10, face="bold"),
        legend.text = element_text(size=10, face="bold"))
```



# 4 According to the two dimensional solution of the multidimensional scaling, the cereals 14 and 16 seems to be most similar followed by the cereals 3 and 4.

5

According to the two dimensional solution of the multidimensional scaling problem, the cereals 41 and 18 are most distinct as confirmed by distance matrix.

6

It is possible to obtain the exact representation of our original distance matrix. In such case, we have to set the number of dimension equal to the rank of our matrix.

7

The follwoing code reports the eigen values and computes the goodness of fit of the solution

```
ev <- MDS$eig
gof <- MDS$GOF
print(round(ev,digits=4))
```

```
##  [1] 106.9979  77.9000  74.2909  36.4679  20.8866  15.0120   2.5411
##  [8]   1.9036   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
## [15]   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
## [22]   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
## [29]   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
## [36]   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
## [43]   0.0000
```

```
print(gof)
```

```
## [1] 0.5502914 0.5502914
```

We can also report the goodness of fit directly using the output of cmdscale as follows-

```
print(cmdscale(distance, 2, eig=TRUE)$GOF)
```

```
## [1] 0.5502914 0.5502914
```

or we can compute the goodness of fit manually using the two different criterion as follows-

```
print( (ev[1]+ev[2])/sum(abs(ev)) )
```

```
## [1] 0.5502914
```

```
print( (ev[1]+ev[2])/sum(ev[ev>0]))
```

```
## [1] 0.5502914
```

8. (1p) Are there any zero eigenvalues? Can you explain these?

Yes. There are almost always few number of nonzero eigenvalues accompanied by a number of zero eigenvlaues (There will always be one eigenvalue equal to zero in practice). The number of non-zero eigenvalues is equal to p.

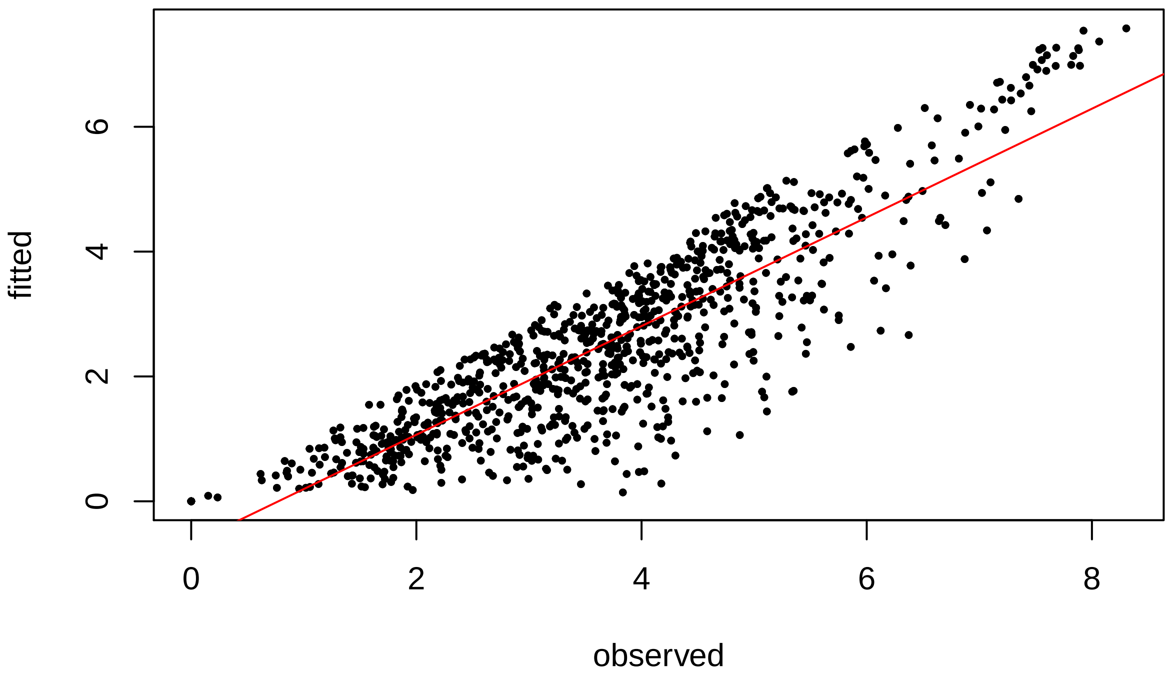
If the coordinates of n points in p dimensions be denoted by X_i, i = 1, . . . ,n. These can be collected together in a n x p matrix X . When we perform the eigendecomposition of the matrix B, if p < n then there are n-p zero eigenvalues. In fact if the points are not in "general position" the number of zero eigenvalues will be greater than n - p. For a detailed reference, we have used this source

9.

The following code computes the fitted distances.

Also, In the following plot, we have plotted the fitted distances against the observed distances. The red line correpsonds to the the regression line. We can see that the regression line goes almost linearly with respect to the fitted and observed distances. This proves that there is a strong correlation between the observed and fitted distances which was our initial objective. Since the line is almost at 45 degree, we can say that correlation is strong and hence, visuall, it gives a good goodness of fit.

```
fitted <- as.matrix(dist(X, method = "euclidean"))
fitted <- as.vector(fitted)
observed <- as.vector(distance)
reg <- lm(fitted~observed)
plot(observed, fitted,pch=19, cex=0.4)
abline(lm(fitted~observed), col="red")
```



```
print(paste("Coefficient of determination:", summary(reg)$r.squared))
```

```
## [1] "Coefficient of determination: 0.770427902994939"
```

The coefficient of determination has been estimated as 0.7704

10.

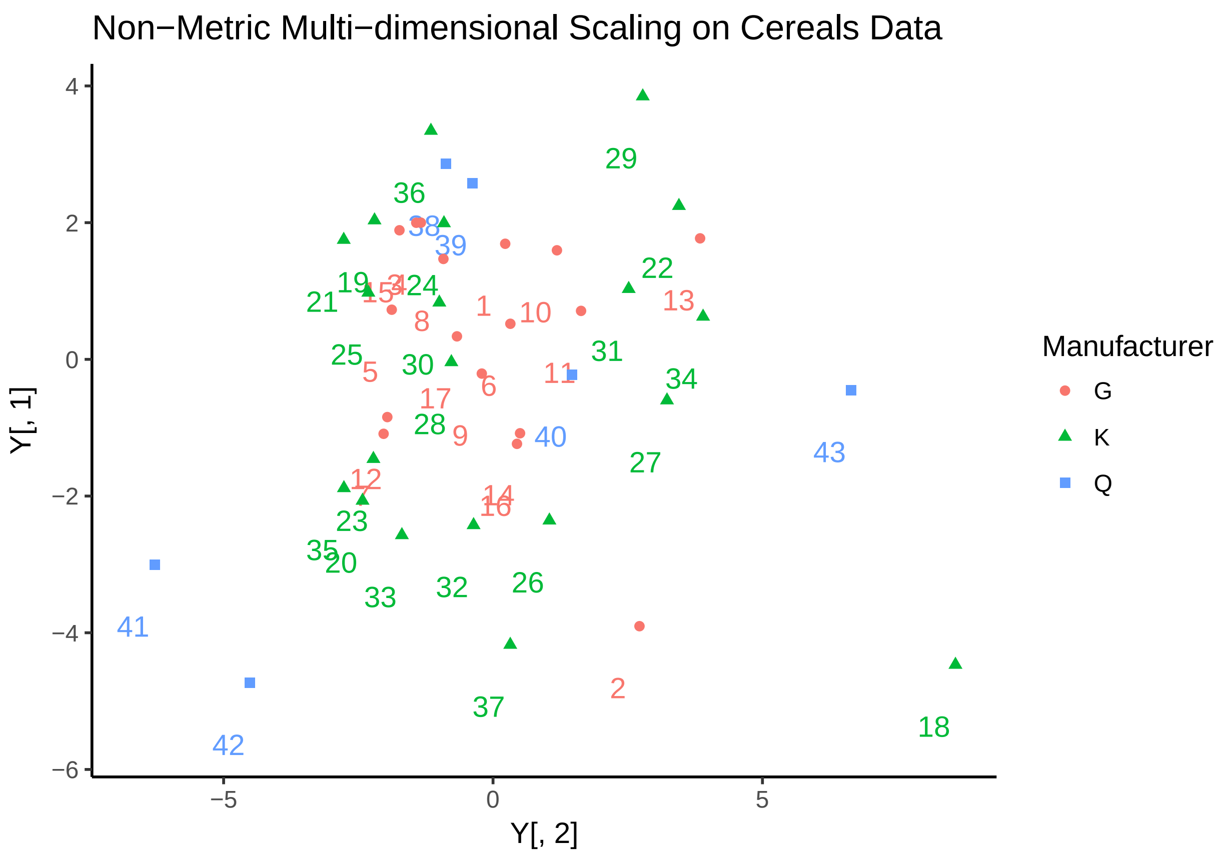
(1p) Try now non-metric MDS with the isoMDS program. Plot the two-dimensional solution, labelling the points again with the name or number of the brand, and using different symbols for different manufacturers.

```
library(MASS)
n <- nrow(distance)
init <- scale(matrix(runif(n*2),ncol=2),scale=FALSE)
nmmds.out <- isoMDS(distance, init, k=2, maxit = 100)
```

```
## initial  value 42.680221
## iter   5 value 34.284275
## iter  10 value 23.299401
## iter  15 value 19.615140
## iter  20 value 17.813695
## final  value 17.740333
## converged
```

```
Y <- nmmds.out$points
g <- ggplot(data.frame(Y, cereals), aes(Y[,2], Y[,1], label = rownames(cereals)))
```

```r
g + geom_text(aes(color = Manufacturer) , angle=0, show.legend = F, nudge_x = -0.40,
              nudge_y = -0.90) +
  ggtitle("Non-Mteric Multi-dimensional Scaling on Cereals Data") +
  geom_point(aes(shape = Manufacturer, color = Manufacturer)) + theme_classic() + ggtitle("Non-Metric Mu
```



11

We cn observe that the pair of cereals 3 and 4 are most similar according to the two dimensional solution of non-metric MDS.

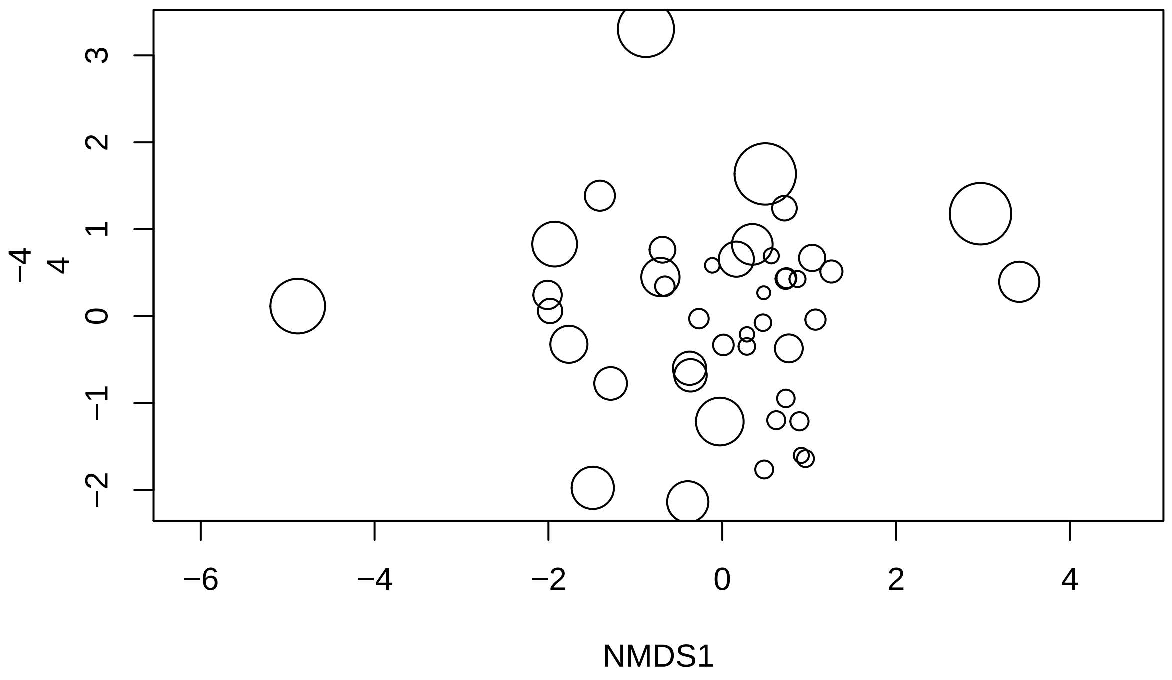
```r
library(vegan)
```

```
## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.4-6
```

```r
nmmds <- metaMDS(comm = distance, distance = "euclidean", k=2)
```

```
## Run 0 stress 0.1702258
## Run 1 stress 0.1702227
## ... New best solution
## ... Procrustes: rmse 0.0007218864  max resid 0.002887462
## ... Similar to previous best
## Run 2 stress 0.170227
## ... Procrustes: rmse 0.0006852868  max resid 0.003250339
## ... Similar to previous best
## Run 3 stress 0.2060757
## Run 4 stress 0.17212
## Run 5 stress 0.1595412
## ... New best solution
## ... Procrustes: rmse 0.116031  max resid 0.4265278
## Run 6 stress 0.1725582
## Run 7 stress 0.1676206
## Run 8 stress 0.1595464
## ... Procrustes: rmse 0.00157721  max resid 0.006751632
## ... Similar to previous best
## Run 9 stress 0.1702268
## Run 10 stress 0.1733471
## Run 11 stress 0.1595435
## ... Procrustes: rmse 0.0005330969  max resid 0.002514414
## ... Similar to previous best
## Run 12 stress 0.159547
## ... Procrustes: rmse 0.0008902341  max resid 0.004229834
## ... Similar to previous best
## Run 13 stress 0.172112
## Run 14 stress 0.1595502
## ... Procrustes: rmse 0.001349737  max resid 0.006782064
## ... Similar to previous best
## Run 15 stress 0.1702215
## Run 16 stress 0.1733465
## Run 17 stress 0.1595411
## ... New best solution
## ... Procrustes: rmse 0.000759516  max resid 0.00359329
## ... Similar to previous best
## Run 18 stress 0.1743242
## Run 19 stress 0.1856263
## Run 20 stress 0.1595479
## ... Procrustes: rmse 0.0116009  max resid 0.05657002
## *** Solution reached
```

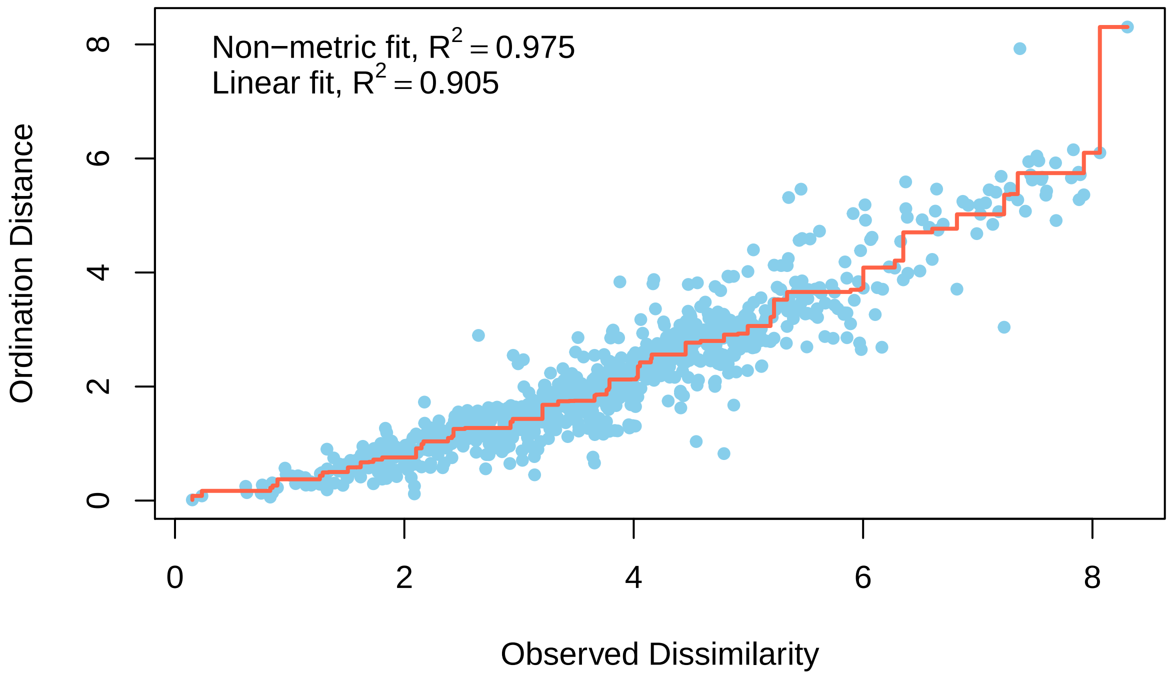
```r
gof <- goodness(nmmds)

plot(nmmds, display = "sites", type = "n", ylab=c(-4,4))
points(nmmds, display = "sites", cex = 2*gof/mean(gof))
```



Extra (Ignore)

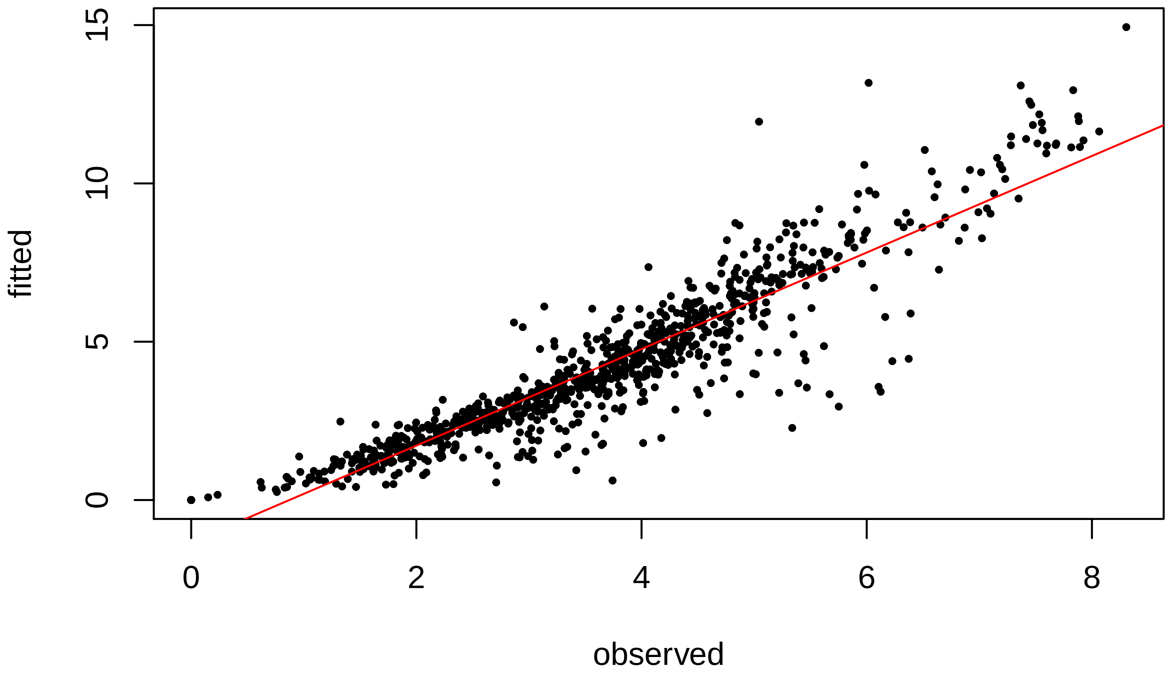
Now, in order to graphically examine the stress in case of non-metric multidimensional scaling, let us consider the following stressplot-

```r
stressplot(nmmds, pch = 19, cex=0.75, l.col = "tomato", p.col = "skyblue")
```



12

```r
fitted <- as.vector(as.matrix(dist(Y, method = "euclidean")))
observed <- as.vector(as.matrix(distance))
reg <- lm(fitted~observed)
plot(observed, fitted,pch=19, cex=0.4)
abline(lm(fitted~observed), col="red")
```



```r
print(paste("Coefficient of determination:", summary(reg)$r.squared))
```

```
## [1] "Coefficient of determination: 0.853487545142084"
```

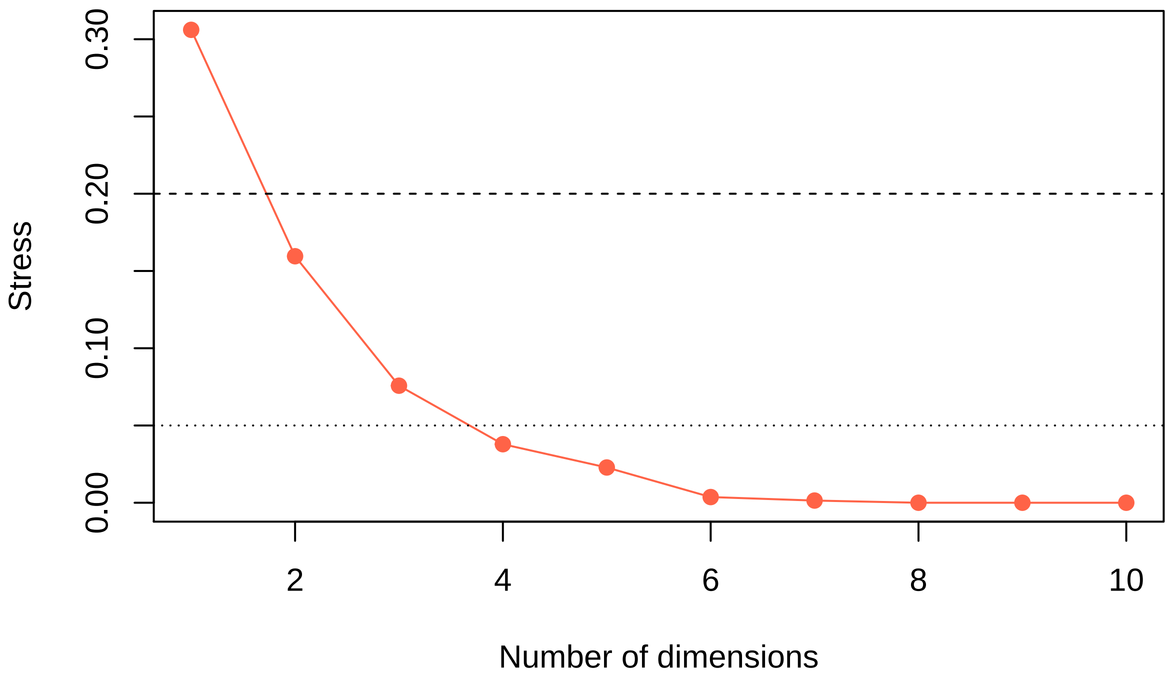
We can observe from the plot that the fitted distances are in almost perfectly linearly related with observed distances.

13. Now, we will be computing the stress for the dimensions 1, 2, 3, 4, 5, ... and explain how many dimensions are required to obtain a good fit.

```r
stress_vec <- numeric(10)
for(i in seq(10)){
  stress_vec[i] <- metaMDS(distance, distance = "euclidean", k=i)$stress
}
```

```
## Run 0 stress 0.3060661
## Run 1 stress 0.3535874
## Run 2 stress 0.3309996
## Run 3 stress 0.4200212
## Run 4 stress 0.4130431
## Run 5 stress 0.562711
## Run 6 stress 0.3660669
## Run 7 stress 0.5625056
## Run 8 stress 0.4118256
## Run 9 stress 0.4021706
## Run 10 stress 0.3634776
## Run 11 stress 0.3319585
## Run 12 stress 0.5637637
## Run 13 stress 0.3338051
## Run 14 stress 0.3997049
## Run 15 stress 0.3864545
## Run 16 stress 0.3797405
## Run 17 stress 0.374313
## Run 18 stress 0.3803012
## Run 19 stress 0.4072627
## Run 20 stress 0.4270053
## *** No convergence -- monoMDS stopping criteria:
##      5: stress ratio > sratmax
##     15: scale factor of the gradient < sfgrmin
## Run 0 stress 0.1702258
## Run 1 stress 0.1702253
## ... New best solution
## ... Procrustes: rmse 0.0004017531  max resid 0.001622586
## ... Similar to previous best
## Run 2 stress 0.1595487
## ... New best solution
## ... Procrustes: rmse 0.1164024  max resid 0.4235237
## Run 3 stress 0.1702258
## Run 4 stress 0.1595351
## ... New best solution
## ... Procrustes: rmse 0.003383909  max resid 0.01248035
## Run 5 stress 0.167384
## Run 6 stress 0.1595441
## ... Procrustes: rmse 0.002853359  max resid 0.01049966
## Run 7 stress 0.1721112
## Run 8 stress 0.1856282
## Run 9 stress 0.1759174
## Run 10 stress 0.1957593
## Run 11 stress 0.1702228
## Run 12 stress 0.1676165
## Run 13 stress 0.1676153
## Run 14 stress 0.1759089
## Run 15 stress 0.1595457
## ... Procrustes: rmse 0.003090012  max resid 0.01074032
## Run 16 stress 0.1595409
## ... Procrustes: rmse 0.00148207  max resid 0.006094186
## ... Similar to previous best
## Run 17 stress 0.170224
## Run 18 stress 0.1721092
## Run 19 stress 0.1721075
## Run 20 stress 0.1595472
## ... Procrustes: rmse 0.01078655  max resid 0.05545105
## *** Solution reached
## Run 0 stress 0.07574604
## Run 1 stress 0.07574885
## ... Procrustes: rmse 0.0004175199  max resid 0.001518188
## ... Similar to previous best
## Run 2 stress 0.07574781
## ... Procrustes: rmse 0.001009857  max resid 0.004426859
## ... Similar to previous best
## Run 3 stress 0.07574675
## ... Procrustes: rmse 0.0001667717  max resid 0.0006714758
## ... Similar to previous best
## Run 4 stress 0.07574637
## ... Procrustes: rmse 0.0008021903  max resid 0.003568407
## ... Similar to previous best
## Run 5 stress 0.07574767
## ... Procrustes: rmse 0.0002858847  max resid 0.001027239
## ... Similar to previous best
## Run 6 stress 0.1097874
## Run 7 stress 0.07574655
## ... Procrustes: rmse 0.0007892536  max resid 0.003116146
## ... Similar to previous best
## Run 8 stress 0.07574596
## ... New best solution
## ... Procrustes: rmse 0.0007050393  max resid 0.003106302
## ... Similar to previous best
## Run 9 stress 0.0757479
## ... Procrustes: rmse 0.0009581408  max resid 0.00395347
## ... Similar to previous best
## Run 10 stress 0.07574583
## ... New best solution
## ... Procrustes: rmse 0.0005359665  max resid 0.002096749
## ... Similar to previous best
## Run 11 stress 0.1097875
## Run 12 stress 0.07574675
## ... Procrustes: rmse 0.0006749021  max resid 0.00253705
## ... Similar to previous best
## Run 13 stress 0.1098061
## Run 14 stress 0.07574623
## ... Procrustes: rmse 0.0003088697  max resid 0.001526415
## ... Similar to previous best
## Run 15 stress 0.0757467
## ... Procrustes: rmse 0.0006637665  max resid 0.002455733
## ... Similar to previous best
## Run 16 stress 0.1097874
## Run 17 stress 0.07574632
## ... Procrustes: rmse 0.0006221545  max resid 0.002370143
## ... Similar to previous best
## Run 18 stress 0.07574769
## ... Procrustes: rmse 0.0004475324  max resid 0.002097178
## ... Similar to previous best
## Run 19 stress 0.07574687
## ... Procrustes: rmse 0.0007150067  max resid 0.002686308
## ... Similar to previous best
## Run 20 stress 0.07574678
## ... Procrustes: rmse 0.0006996335  max resid 0.002617732
## ... Similar to previous best
## *** Solution reached
## Run 0 stress 0.03787597
## Run 1 stress 0.03787568
## ... New best solution
## ... Procrustes: rmse 0.0005354071  max resid 0.00204673
## ... Similar to previous best
## Run 2 stress 0.03787671
## ... Procrustes: rmse 0.000601632  max resid 0.001696945
## ... Similar to previous best
## Run 3 stress 0.03787506
## ... New best solution
## ... Procrustes: rmse 0.0006789313  max resid 0.002997697
## ... Similar to previous best
## Run 4 stress 0.03787502
## ... New best solution
## ... Procrustes: rmse 0.0005625698  max resid 0.002324185
## ... Similar to previous best
## Run 5 stress 0.03787442
## ... New best solution
## ... Procrustes: rmse 0.0004252104  max resid 0.001597489
## ... Similar to previous best
## Run 6 stress 0.03787468
## ... Procrustes: rmse 0.0003052441  max resid 0.001383147
## ... Similar to previous best
## Run 7 stress 0.03787527
## ... Procrustes: rmse 0.0002906751  max resid 0.001220252
## ... Similar to previous best
## Run 8 stress 0.03787548
## ... Procrustes: rmse 0.0004704466  max resid 0.00215682
## ... Similar to previous best
## Run 9 stress 0.03787556
## ... Procrustes: rmse 0.0002253857  max resid 0.001096794
## ... Similar to previous best
## Run 10 stress 0.03787518
## ... Procrustes: rmse 0.0004850232  max resid 0.00205356
## ... Similar to previous best
## Run 11 stress 0.03787616
## ... Procrustes: rmse 0.000473906  max resid 0.002247098
## ... Similar to previous best
## Run 12 stress 0.03787519
## ... Procrustes: rmse 0.0004193552  max resid 0.002000756
## ... Similar to previous best
## Run 13 stress 0.03787534
## ... Procrustes: rmse 0.0002770147  max resid 0.001095816
## ... Similar to previous best
## Run 14 stress 0.0378753
## ... Procrustes: rmse 0.0001980678  max resid 0.000931992
## ... Similar to previous best
## Run 15 stress 0.03787528
## ... Procrustes: rmse 0.0001872772  max resid 0.0008950709
## ... Similar to previous best
## Run 16 stress 0.03787473
## ... Procrustes: rmse 0.0003901417  max resid 0.001509052
## ... Similar to previous best
## Run 17 stress 0.03787637
## ... Procrustes: rmse 0.000357798  max resid 0.001059601
## ... Similar to previous best
## Run 18 stress 0.03787633
## ... Procrustes: rmse 0.0004210455  max resid 0.001766928
## ... Similar to previous best
## Run 19 stress 0.03787551
## ... Procrustes: rmse 0.0005314471  max resid 0.002269794
## ... Similar to previous best
## Run 20 stress 0.03787492
## ... Procrustes: rmse 0.0002258482  max resid 0.001110219
## ... Similar to previous best
## *** Solution reached
## Run 0 stress 0.02300081
## Run 1 stress 0.02558896
## Run 2 stress 0.02281395
## ... New best solution
## ... Procrustes: rmse 0.01834851  max resid 0.08480776
## Run 3 stress 0.02300058
## ... Procrustes: rmse 0.01878856  max resid 0.0856236
## Run 4 stress 0.02319263
## ... Procrustes: rmse 0.009241628  max resid 0.03901368
## Run 5 stress 0.02281448
## ... Procrustes: rmse 0.001221091  max resid 0.005050362
## ... Similar to previous best
## Run 6 stress 0.02281473
## ... Procrustes: rmse 0.000497461  max resid 0.00170889
## ... Similar to previous best
## Run 7 stress 0.02553388
## Run 8 stress 0.02587657
## Run 9 stress 0.02299944
## ... Procrustes: rmse 0.01873728  max resid 0.08552143
## Run 10 stress 0.02300155
## ... Procrustes: rmse 0.01900138  max resid 0.08576487
## Run 11 stress 0.02300127
## ... Procrustes: rmse 0.01878441  max resid 0.08569651
## Run 12 stress 0.02558847
## Run 13 stress 0.02300286
## ... Procrustes: rmse 0.01913291  max resid 0.08581714
## Run 14 stress 0.02325072
## ... Procrustes: rmse 0.007425289  max resid 0.03038944
## Run 15 stress 0.02281518
## ... Procrustes: rmse 0.001316007  max resid 0.005507901
## ... Similar to previous best
## Run 16 stress 0.02567905
## Run 17 stress 0.02281506
## ... Procrustes: rmse 0.001294016  max resid 0.00567419
## ... Similar to previous best
## Run 18 stress 0.02300086
## ... Procrustes: rmse 0.01895851  max resid 0.08573411
## Run 19 stress 0.0254998
## Run 20 stress 0.02281664
## ... Procrustes: rmse 0.001460392  max resid 0.005763152
## ... Similar to previous best
## *** Solution reached
## Run 0 stress 0.003674216
## Run 1 stress 0.003682324
## ... Procrustes: rmse 0.0008501464  max resid 0.003420548
## ... Similar to previous best
## Run 2 stress 0.003679006
## ... Procrustes: rmse 0.0005791742  max resid 0.002240168
## ... Similar to previous best
## Run 3 stress 0.003784503
## ... Procrustes: rmse 0.002775195  max resid 0.01250713
## Run 4 stress 0.003682261
## ... Procrustes: rmse 0.0006770818  max resid 0.00317145
## ... Similar to previous best
## Run 5 stress 0.003680755
## ... Procrustes: rmse 0.0006884109  max resid 0.003225152
## ... Similar to previous best
## Run 6 stress 0.003715135
## ... Procrustes: rmse 0.001660756  max resid 0.007115435
## ... Similar to previous best
## Run 7 stress 0.003687075
## ... Procrustes: rmse 0.0008980935  max resid 0.004269539
## ... Similar to previous best
## Run 8 stress 0.003677222
## ... Procrustes: rmse 0.0006253625  max resid 0.002873396
## ... Similar to previous best
## Run 9 stress 0.003677738
## ... Procrustes: rmse 0.0005582431  max resid 0.00204787
## ... Similar to previous best
## Run 10 stress 0.003684762
## ... Procrustes: rmse 0.000819589  max resid 0.003998719
## ... Similar to previous best
## Run 11 stress 0.003686235
## ... Procrustes: rmse 0.0007708116  max resid 0.003636504
## ... Similar to previous best
## Run 12 stress 0.003683824
## ... Procrustes: rmse 0.0009610845  max resid 0.004160064
## ... Similar to previous best
## Run 13 stress 0.003675059
## ... Procrustes: rmse 0.0002319243  max resid 0.001129281
## ... Similar to previous best
## Run 14 stress 0.003696676
## ... Procrustes: rmse 0.001206413  max resid 0.00549787
## ... Similar to previous best
## Run 15 stress 0.003693416
## ... Procrustes: rmse 0.00108909  max resid 0.005284606
## ... Similar to previous best
## Run 16 stress 0.003677549
## ... Procrustes: rmse 0.0004310211  max resid 0.001744354
## ... Similar to previous best
## Run 17 stress 0.003679988
## ... Procrustes: rmse 0.0007481944  max resid 0.003322448
## ... Similar to previous best
## Run 18 stress 0.003699719
## ... Procrustes: rmse 0.001360096  max resid 0.006127837
## ... Similar to previous best
## Run 19 stress 0.003678925
## ... Procrustes: rmse 0.0007602166  max resid 0.003521182
## ... Similar to previous best
## Run 20 stress 0.003677698
## ... Procrustes: rmse 0.0005288027  max resid 0.001619588
## ... Similar to previous best
## *** Solution reached
## Run 0 stress 0.001410885
## Run 1 stress 0.003252081
## Run 2 stress 0.002166201
## Run 3 stress 0.002943979
## Run 4 stress 0.002261617
## Run 5 stress 0.002913921
## Run 6 stress 0.002079044
## Run 7 stress 0.002125275
## Run 8 stress 0.002069321
## Run 9 stress 0.0029469
## Run 10 stress 0.002745024
## Run 11 stress 0.002498018
## Run 12 stress 0.002336907
## Run 13 stress 0.00198462
## Run 14 stress 0.002402201
## Run 15 stress 0.002588613
## Run 16 stress 0.002808729
## Run 17 stress 0.001821153
## ... Procrustes: rmse 0.00496553  max resid 0.0123216
## Run 18 stress 0.002618327
## Run 19 stress 0.00286562
## Run 20 stress 0.001971431
## *** No convergence -- monoMDS stopping criteria:
##     20: no. of iterations >= maxit
## Run 0 stress 0
## Run 1 stress 0.00243334
## Run 2 stress 0.001636143
## Run 3 stress 0.002056024
## Run 4 stress 0.002335616
## Run 5 stress 0.003400115
## Run 6 stress 0.005234969
## Run 7 stress 0.00268336
## Run 8 stress 0.001617673
## Run 9 stress 0.002741753
## Run 10 stress 0.001745473
## Run 11 stress 0.002534003
## Run 12 stress 0.002286869
## Run 13 stress 0.002679878
## Run 14 stress 0.00206906
## Run 15 stress 0.002595453
## Run 16 stress 0.001631103
## Run 17 stress 0.002007851
## Run 18 stress 0.001995699
## Run 19 stress 0.002049601
## Run 20 stress 0.002139906
## *** No convergence -- monoMDS stopping criteria:
##     20: no. of iterations >= maxit
## Warning in metaMDS(distance, distance = "euclidean", k = i): Stress is
## (nearly) zero - you may have insufficient data

## Run 0 stress 0
## Run 1 stress 0.00141055
## Run 2 stress 0.00174403
## Run 3 stress 0.001629922
## Run 4 stress 0.001588375
## Run 5 stress 0.002174224
## Run 6 stress 0.002101639
## Run 7 stress 0.001915571
## Run 8 stress 0.002126941
## Run 9 stress 0.00194368
## Run 10 stress 0.001901484
## Run 11 stress 0.001970391
## Run 12 stress 0.001779089
## Run 13 stress 0.002675535
## Run 14 stress 0.001924934
## Run 15 stress 0.001206355
## Run 16 stress 0.003738087
## Run 17 stress 0.001693816
## Run 18 stress 0.001996237
## Run 19 stress 0.00307957
## Run 20 stress 0.001616021
## *** No convergence -- monoMDS stopping criteria:
##     20: no. of iterations >= maxit

## Warning in metaMDS(distance, distance = "euclidean", k = i): Stress is
## (nearly) zero - you may have insufficient data

## Run 0 stress 0
## Run 1 stress 0.002068087
## Run 2 stress 0.002387298
## Run 3 stress 0.001508236
## Run 4 stress 0.001260157
## Run 5 stress 0.001373797
## Run 6 stress 0.00256846
## Run 7 stress 0.001691436
## Run 8 stress 0.004636998
## Run 9 stress 0.002435967
## Run 10 stress 0.002589573
## Run 11 stress 0.002340995
## Run 12 stress 0.002106739
## Run 13 stress 0.001323927
## Run 14 stress 0.001826877
## Run 15 stress 0.002331522
## Run 16 stress 0.001657686
## Run 17 stress 0.001577194
## Run 18 stress 0.003689343
## Run 19 stress 0.001196982
## Run 20 stress 0.001193224
## *** No convergence -- monoMDS stopping criteria:
##     20: no. of iterations >= maxit

## Warning in metaMDS(distance, distance = "euclidean", k = i): Stress is
## (nearly) zero - you may have insufficient data
```

```r
plot(seq(10),stress_vec, type = 'o', ylab = "Stress", xlab = "Number of dimensions",
     col="tomato", pch=19)
abline(h=0.2, lty=2)
abline(h=0.05, lty=3)
```



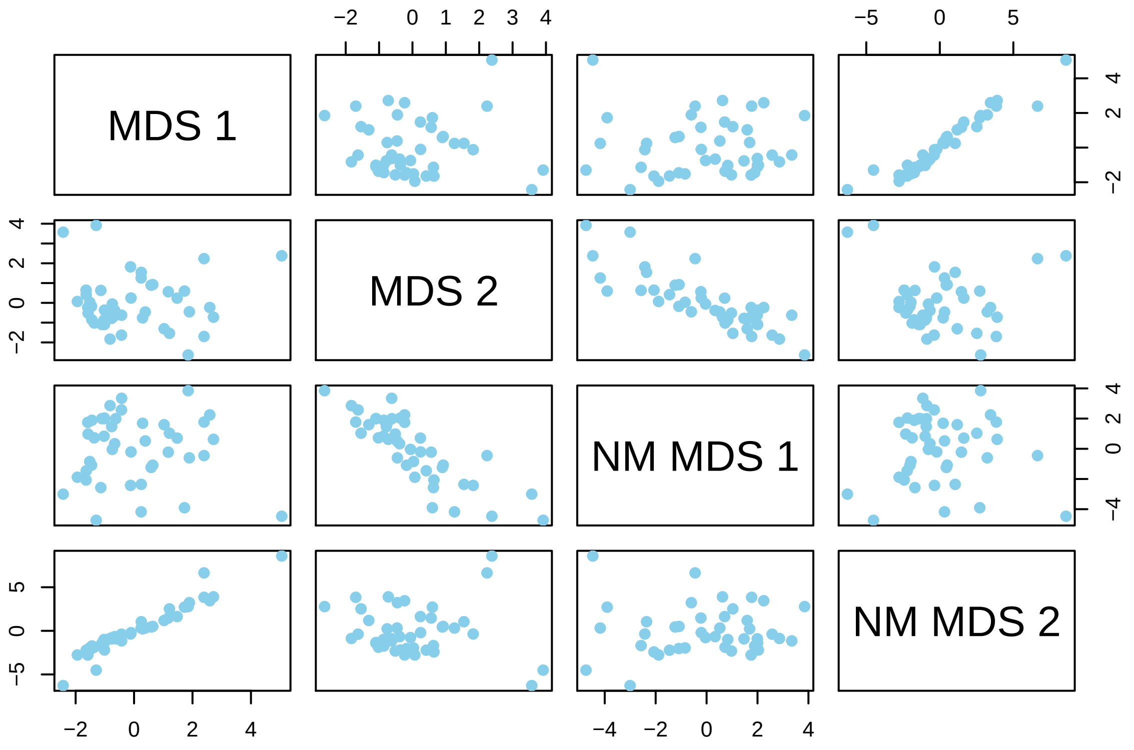
Now, in order to obtain a good fit, we must have a stress of approximately 5% or equivalently, 0.05. However, so far upto three dimensions, the stress exisitng is equal to 7% approximately. Hence, It would be in our favor to obtain a good fit if we chose four dimensions where our stress reduces to 3% approximately.

```r
print(paste("Stress Values", stress_vec, sep = ": "))
```

```
##  [1] "Stress Values: 0.306066068424548"
##  [2] "Stress Values: 0.159535115640721"
##  [3] "Stress Values: 0.075745834312033"
##  [4] "Stress Values: 0.0378744187101069"
##  [5] "Stress Values: 0.0228139496600098"
##  [6] "Stress Values: 0.00367421620284672"
##  [7] "Stress Values: 0.00141088470041126"
##  [8] "Stress Values: 0"
##  [9] "Stress Values: 0"
## [10] "Stress Values: 0"
```

14.

```r
pairs(cbind(X,Y), labels = c("MDS 1", "MDS 2", "NM MDS 1", "NM MDS 2"), col="skyblue",
      pch=19)
```



The correlation matrix of the four variables are as follows-

```
cor(cbind(X,Y))
```

```
##                [,1]          [,2]        [,3]        [,4]
## [1,]  1.000000e+00 -4.389205e-18 -0.03373288  0.95970470
## [2,] -4.389205e-18  1.000000e+00 -0.83379117 -0.08503558
## [3,] -3.373288e-02 -8.337912e-01  1.00000000  0.01679022
## [4,]  9.597047e-01 -8.503558e-02  0.01679022  1.00000000
```

15

If we will be using the non-standardized variables, then the distance matrix will not be accurate as different variables will be on different scales and it will result in an absurd distance matrix. Therefore, it is important to use the standarized variables for computing the distance matrix in our case because different variables are based on different scales and have different units.