Assignment 7

Christian Diangco

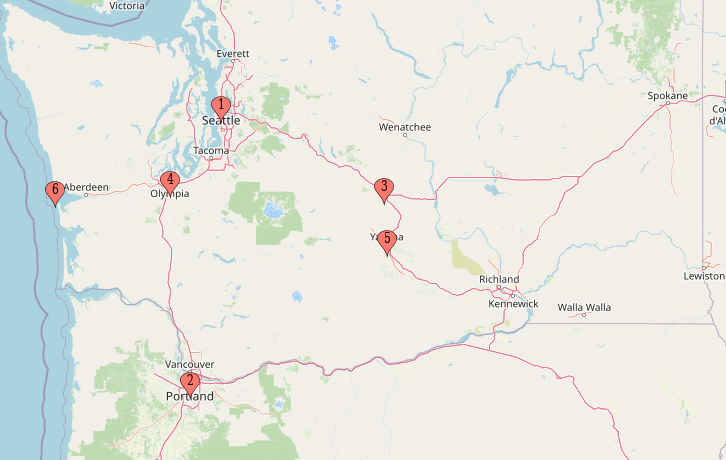
Math 381 A

We will be applying multi-dimensional scaling (MDS) to a set of 6 cities that are within 500 km of each other.

I’ve decided to looking at these 6 cities:

1. Seattle, WA
2. Portland, OR
3. Ellensburg, WA
4. Olympia, WA
5. Yakima, WA
6. Ocean Shores, WA

The map below has each city marked on it with its corresponding number.

(From mapcustomizer.com)

We can create a 6x6 matrix *D1* that contains the distances between each pair of cities, where entry (*i, j*) corresponds to the distance between city *i* and city *j*.

I will be filling *D1* with the lengths of the shortest routes by car between each city pair, obtained via Google Maps.

The following R code will be used to create *D1*:

# This matrix contains the distances between each city to one another (within 500 km).

D1 <- matrix(

c(0, 290, 173, 98, 229, 212,

290, 0, 360, 185, 303, 268,

173, 360, 0, 238, 58, 352,

98, 185, 238, 0, 295, 116,

229, 303, 58, 295, 0, 365,

212, 268, 352, 116, 365, 0

),

nrow=6,

ncol=6,

byrow=TRUE

)

Now that we have *D1*, we can use it to create a 1-dimensional model with this command:

cmdscale(D1, k=1, eig=TRUE)

Which gives us the following output:

$points

[,1]

[1,] -15.52599

[2,] 115.53555

[3,] -177.48811

[4,] 87.24804

[5,] -175.08387

[6,] 165.31437

$eig

[1] 1.106870e+05 5.007616e+04 1.010031e+04 1.639822e+03 1.273293e-11 -1.148759e+04

$x

NULL

$ac

[1] 0

$GOF

[1] 0.6015895 0.6416515

From $points, we see that we have 6 points, one for each city, in a range of about -177 to 165.

We can plot them using this code:

model1 <- cmdscale(D1, k=1)

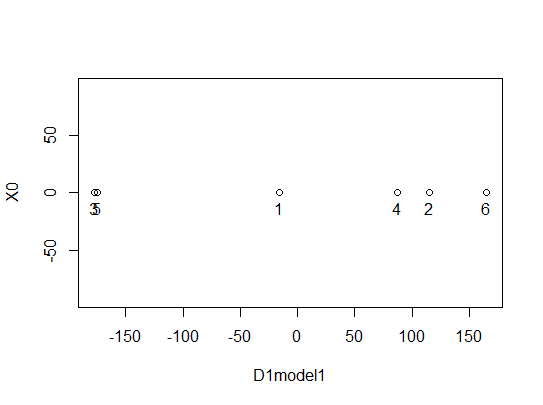
mylabels <- c(1, 2, 3, 4, 5, 6)

D1plot1 <- data.frame(model1, 0)

plot(D1plot1)

text(D1plot1, labels=mylabels, pos=1)

Which gives us the following plot:



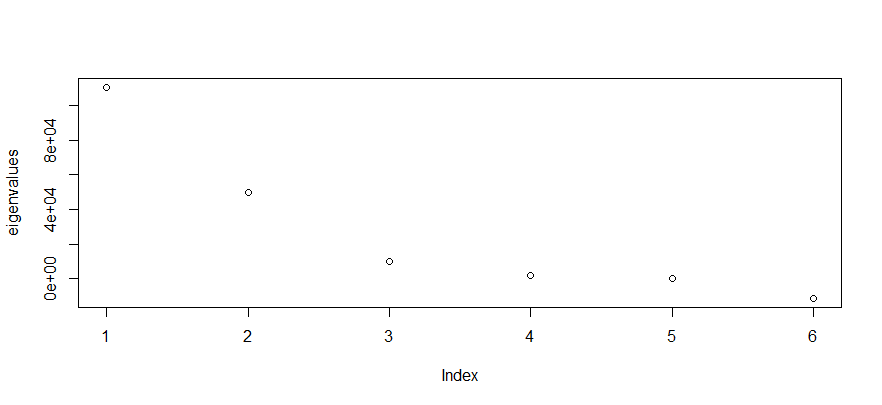
Each point represents a city and is labeled with the number that corresponds to that city.

The two points that are very close together are Ellensburg (left) and Yakima (right).

From $eig, we can obtain the eigenvalues of *D1* and plot them using this code:

eigenvalues <- cmdscale(D1, k=1, eig=TRUE)$eig

plot(eigenvalues)

Which creates this plot:

The first 4 eigenvalues are much larger than the last 2 (easier to see from output values), which suggests that a 4-dimensional model would work best.

However, it doesn’t make sense to look at any dimension higher than 3, since the dimensions of the model are actual physical dimensions.

The 6th eigenvalue is a large negative number, which indicates that our distances are non-euclidean.

This makes sense since the distances are measured along roads between the cities.

$GOF gives us goodness of fit values of 0.6015895 and 0.6416515.

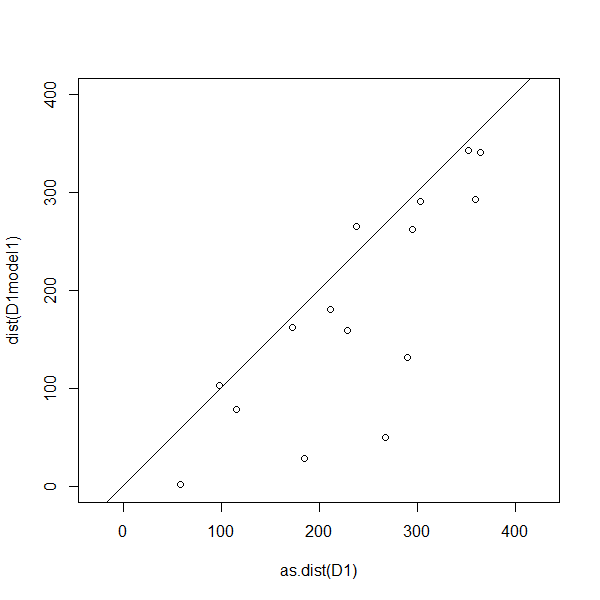
We will now compare the distances of our 1D model vs our original distances.

To do this we will make a scatter plot, with our original distances as the x-values and our 1D model distances as the y-values.

Using this code, we can create the plot:

plot(as.dist(D1), dist(D1model1), asp=1, xlim=c(0,400), ylim=c(0,400))

abline(0,1)



The line on the plot is *y* = *x*.

The better a model is, the closer the scatter plot is to *y* = *x*.

We can see that the points deviate a lot from the line, meaning the 1D model is not very good.

We can calculate the error of our 1D model by subtracting the matrix of model distances from the matrix of original distances and taking the absolute value.

We can do this using this code:

abs(D1 - as.matrix(dist(D1model1$points)))

Which outputs this matrix:

1 2 3 4 5 6

1 0.000000 158.93846 11.037879 4.774026 69.44211 31.159639

2 158.938462 0.00000 66.976341 156.712488 12.38058 218.221177

3 11.037879 66.97634 0.000000 26.736147 55.59577 9.197518

4 4.774026 156.71249 26.736147 0.000000 32.66809 37.933665

5 69.442114 12.38058 55.595765 32.668088 0.00000 24.601752

6 31.159639 218.22118 9.197518 37.933665 24.60175 0.000000

The largest error is 218.221 at (6, 2) and (2, 6), corresponding to Portland (2) and Ocean Shores (6).

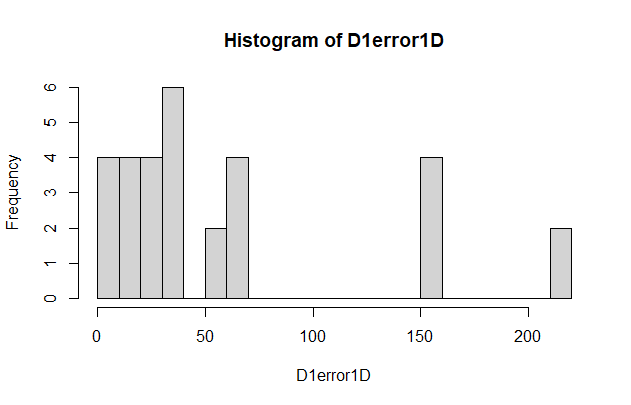
The average error is 50.910.

Below is a histogram of the error values using this code:

D1error3D <- abs(D1 - as.matrix(dist(D1model3$points)))

D1error3D[D1error3D == 0] <- NA

hist(D1error3D, breaks=30)



We can see that most of the error values are within the range of about 0 - 70, with a few that are much higher.

***D1* 2-dimensional model**

Now let’s use the same cities, but with a 2-dimensional model.

We can create the 2D model like so:

cmdscale(D1, k=2, eig=TRUE)

Which outputs:

[,1] [,2]

[1,] -15.52599 86.36358

[2,] 115.53555 -170.88761

[3,] -177.48811 38.09058

[4,] 87.24804 34.20926

[5,] -175.08387 -67.09662

[6,] 165.31437 79.32082

.

.

.

$GOF

[1] 0.8737561 0.9319425

We can see that by increasing dimensions, our goodness of fit values have also increased.

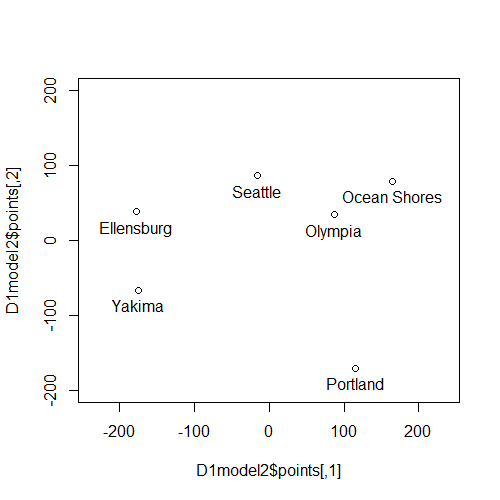
We can plot our 2D model using this code:

D1model2 <- cmdscale(D1, k=2, eig=TRUE)

mylabels <- c("Seattle", "Portland", "Ellensburg", "Olympia", "Yakima", "Ocean Shores")

plot(D1model2$points, asp=1, xlim=c(-200,200), ylim=c(-200,200))

text(D1model2$points, labels=mylabels, pos=1)



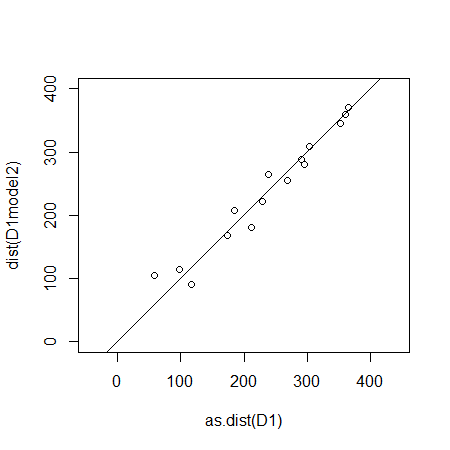
Interestingly, Ocean Shores and Olympia appear to be closest together, rather than Ellensburg and Yakima.

Also, Portland is the city furthest from all the others instead of Seattle.

We will now make a scatter plot with our original distances as the x-values and our 2D model distances as the y-values using this code:

plot(as.dist(D1), dist(D1model2), asp=1, xlim=c(0,400), ylim=c(0,400))

abline(0,1)



We can see that the points are closer to *y* = *x* for the 2D model than the 1D model, though there is still clearly some error.

We obtain the error matrix using:

abs(D1 - as.matrix(dist(D1model2$points)))

1 2 3 4 5 6

1 0.000000 1.28682403 3.99700860 17.25005 7.620797 31.022552

2 1.286824 0.00000000 0.09064262 22.03842 5.597178 12.887895

3 3.997009 0.09064262 0.00000000 26.76460 47.214670 6.726957

4 17.250049 22.03841646 26.76459765 0.00000 13.786713 25.836783

5 7.620797 5.59717816 47.21467034 13.78671 0.000000 5.552335

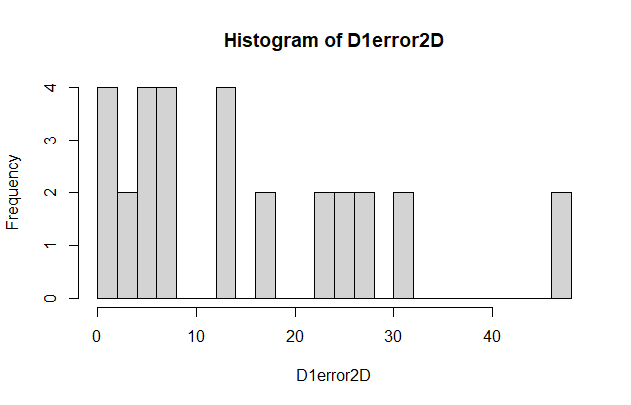
6 31.022552 12.88789466 6.72695660 25.83678 5.552335 0.000000

The largest error value is 47.215 at (5, 3) and (3, 5) which corresponds to Ellensburg (3) and Yakima (5).

The mean error is 12.649.

Both of these values are lower than for the 1D model.

Here is a histogram of the error values:



We can see that the error values are now less spread out than before and we have fewer high outlier values.

***D1* 3-dimensional**

Now let’s use a 3D model for *D1*.

We create the 3D model using:

cmdscale(D1, k=3, eig=TRUE)

Which outputs:

$points

[,1] [,2] [,3]

[1,] -15.52599 86.36358 -39.91155

[2,] 115.53555 -170.88761 -17.43040

[3,] -177.48811 38.09058 -10.27344

[4,] 87.24804 34.20926 -42.39198

[5,] -175.08387 -67.09662 43.82053

[6,] 165.31437 79.32082 66.18684

.

.

.

$GOF

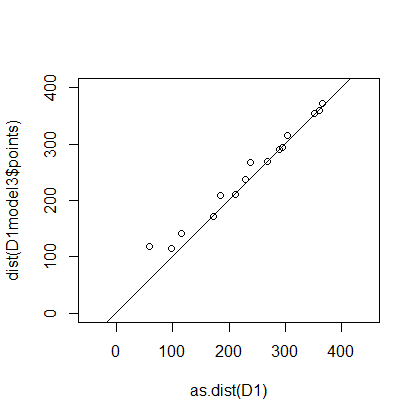
[1] 0.9286518 0.9904940

The goodness of fit values have again increased from 2D going to 3D.

We will now make a scatter plot with our original distances as the x-values and our 3D model distances as the y-values using this code:

plot(as.dist(D1), dist(D1model3$points), asp=1, xlim=c(0,400), ylim=c(0,400))

abline(0,1)



We can see that while not perfect, this is the closest the points have come to *y* = *x* thus far.

We obtain the error matrix using:

abs(D1 - as.matrix(dist(D1model3$points)))

1 2 3 4 5 6

1 0.0000000 0.41287994 1.41786704 17.2767378 7.6850494 2.2150994

2 0.4128799 0.00000000 0.01949008 23.5377337 11.6170606 0.4660659

3 1.4178670 0.01949008 0.00000000 28.7056287 60.3058962 1.6377373

4 17.2767378 23.53773366 28.70562875 0.0000000 0.8682104 25.1338552

5 7.6850494 11.61706060 60.30589624 0.8682104 0.0000000 6.2267295

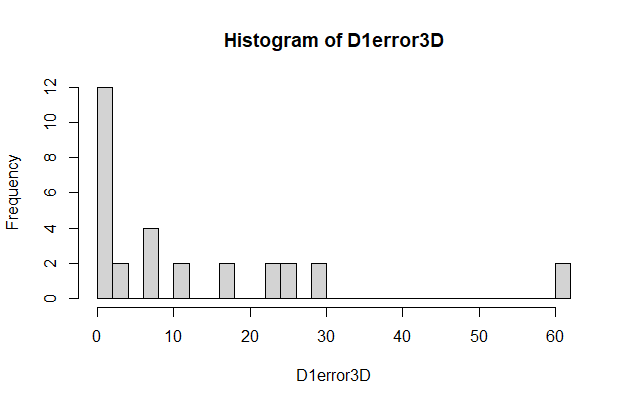
6 2.2150994 0.46606591 1.63773735 25.1338552 6.2267295 0.0000000

The largest error value is 60.0359 at (3, 4) and (4, 3) which corresponds to Ellensburg (3) and Olympia (4).

The mean error is 10.418.

Interestingly, the max error value is higher than for the 2D model, but the mean is lower.

Here is a histogram of the error values:



We see that we have much more error values that are close to 0, strengthening the conclusion that the 3D model is the best we’ve seen so far.

**Global Cities**

Let’s now apply MDS to 6 cities that are spread out around the world.

The cities we will look at are:

1. Edmonton, Alberta, Canada
2. Asuncion, Paraguay
3. Juba, South Sudan
4. Warsaw, Poland
5. Wuhan, China
6. Sydney, Australia

The map below has each city marked with its corresponding number.

We will make another 6x6 matrix similar to *D1* but using the distances between these new cities.

The distances between these cities are direct flight distances obtained from Google Maps.

We will denote this matrix as *D2*, which we can make using this code:

# This matrix contains the distance between each city to one another (around the world).

D2 <- matrix(

c(0, 10270, 12742, 7515, 9582, 13289,

10270, 0, 10146, 11464, 19010, 12763,

12742, 10146, 0, 5357, 9032, 13018,

7515, 11464, 5357, 0, 7574, 15585,

9582, 19010, 9032, 7574, 0, 8142,

13289, 12763, 13018, 15585, 8142, 0

),

nrow=6,

ncol=6,

byrow=TRUE

)

We will analyze 1D, 2D, and 3D models for *D2* just like how we did for *D1*.

We can create the 1D model using:

cmdscale(D2, k=1, eig=TRUE)

Which outputs:

$points

[,1]

[1,] -1115.0389

[2,] -9857.5225

[3,] -743.2999

[4,] -338.2135

[5,] 8835.3754

[6,] 3218.6994

$eig

[1] 1.875048e+08 1.372848e+08 8.148380e+07 5.215406e-08

[5] -1.241410e+07 -6.091762e+07

$x

NULL

$ac

[1] 0

$GOF

[1] 0.3909566 0.4615237

We will plot this model using this code:

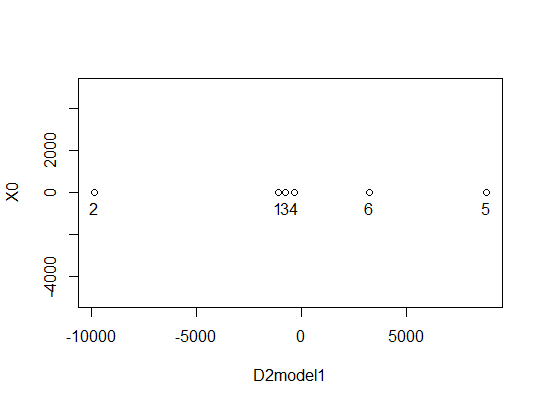
D2model1 <- cmdscale(D2, k=1)

mylabels <- c(1, 2, 3 ,4 ,5, 6)

D2plot1 <- data.frame(D2model1, 0)

plot(D2plot1, asp=1)

text(D2plot1, labels=mylabels, pos=1)



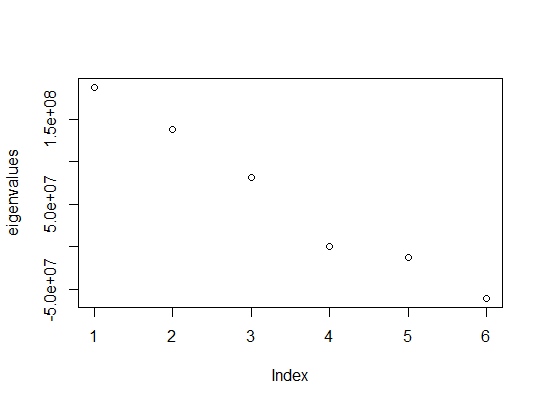
We see that Edmonton (1), Juba (3), and Warsaw (4) are very close together.

Asuncion (2) seems to be isolated from the other cities.

Now let’s plot the eigenvalues of *D2* using this code:

eigenvalues <- cmdscale(D1, k=1, eig=TRUE)$eig

plot(eigenvalues)



Similar to D1, the eigenvalues show that a 4D model would work best.

The 5th and 6th eigenvalues are both large negative values, which means our distances are again non-euclidean.

This makes sense as the distances are measured around a globe.

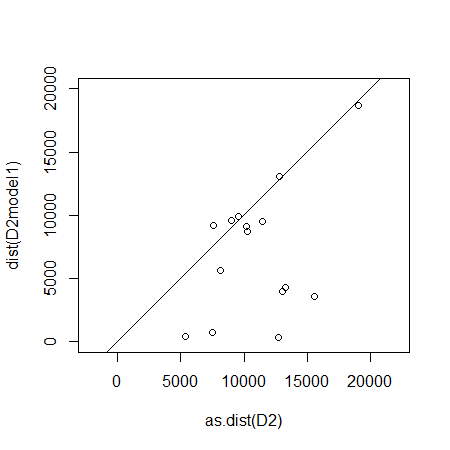
We get goodness of fit values of 0.3909566 and 0.4615237, which are worse than those for the 1D model of *D1*.

This implies that *D2* does not model as well with a 1D model as *D1*, which makes sense as the distances for *D2* are more spread out in terms of length and direction.

We will now make a scatter plot with our original distances as the x-values and our 1D model distances as the y-values using this code:

plot(as.dist(D2), dist(D2model1), asp=1, xlim=c(0,20000), ylim=c(0,20000))

abline(0,1)



We see that the most of the points deviate greatly from *y* = *x*.

We obtain the error matrix using:

abs(D2 - as.matrix(dist(D2model1$points)))

1 2 3 4 5 6

1 0.0000 1527.5164 12370.2610 6738.175 368.4143 8955.2617

2 1527.5164 0.0000 1031.7774 1944.691 317.1020 313.2219

3 12370.2610 1031.7774 0.0000 4951.914 546.6754 9056.0007

4 6738.1746 1944.6910 4951.9136 0.000 1599.5890 12028.0871

5 368.4143 317.1020 546.6754 1599.589 0.0000 2525.3239

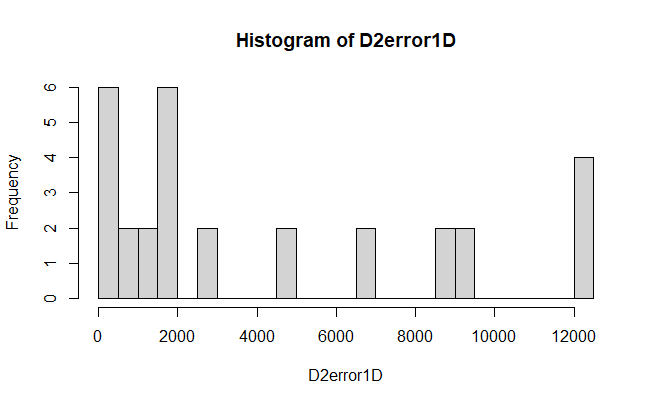
6 8955.2617 313.2219 9056.0007 12028.087 2525.3239 0.0000

The largest error value is 12370.26 at entry (3, 1) and (1, 3) which corresponds to Edmonton (1) and Juba (3).

The mean error value is 3570.778.

Both of these values are significantly larger than those for the 1D model of *D1*, further supporting that *D2* does not model as well in 1D.

Here is a histogram of the error values:



We see a lot of errors fall in 0 to 2,000, but also a pretty even spread of values from about 3.000 to 1,000.

There are also multiple errors above 12,000.

***D2* 2-dimensional model**

Let’s now look at the 2D model for *D2*.

We create the model using:

cmdscale(D2, k=2, eig=TRUE)

Which outputs:

$points

[,1] [,2]

[1,] -1115.0389 -1577.7110

[2,] -9857.5225 2621.6345

[3,] -743.2999 -2616.7128

[4,] -338.2135 -6361.9001

[5,] 8835.3754 -988.5654

[6,] 3218.6994 8923.2549

.

.

.

$GOF

[1] 0.6772022 0.7994361

As expected, the goodness of fit values have increased going from 1D to 2D.

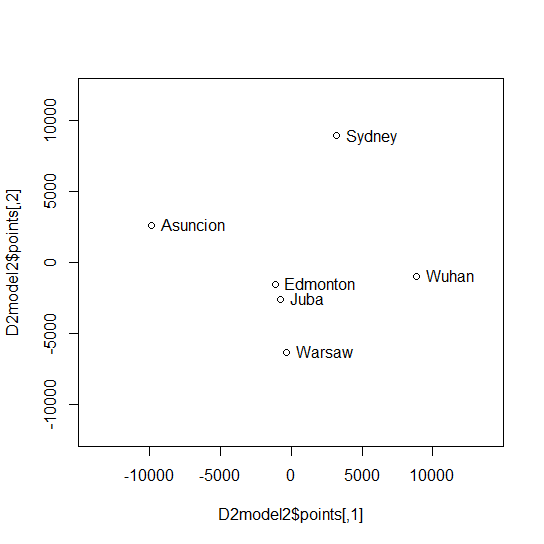
We will plot the 2D model using this code:

D2model2 <- cmdscale(D2, k=2, eig=TRUE)

mylabels <- c("Edmonton", "Asuncion", "Juba", "Warsaw", "Wuhan", "Sydney")

plot(D2model2$points, asp=1, xlim=c(-12000,12000), ylim=c(-12000,12000))

text(D2model2$points, labels=mylabels, pos=4)



Similar to the 1D model, Edmonton and Juba are very close together again, but Warsaw is further away from them this time.

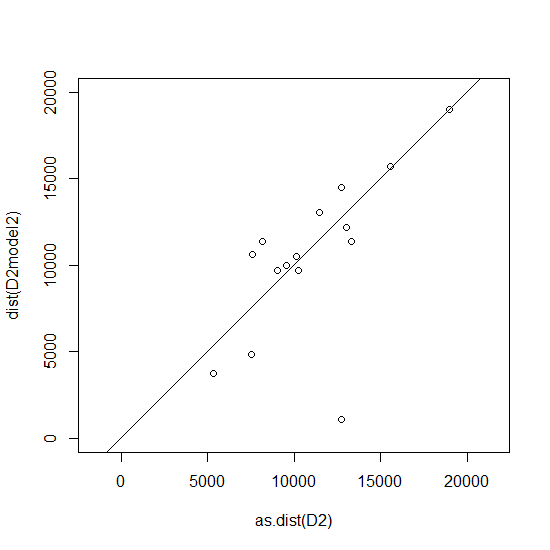
It also appears that Sydney is the city that is most separated from the others, rather than Asuncion.

We will now make a scatter plot with our original distances as the x-values and our 2D model distances as the y-values using this code:

D2model2 <- cmdscale(D2, k=2)

plot(as.dist(D2), dist(D2model2), asp=1, xlim=c(0,20000), ylim=c(0,20000))

abline(0,1)



As expected, we see that the points are closer to *y* = *x* for the 2D model than the 1D model.

We obtain the error matrix using this code:

abs(D2 - as.matrix(dist(D2model2$points)))

1 2 3 4 5 6

1 0.0000 571.26180 11638.4990 2668.1533 385.84019 1928.9132

2 571.2618 0.00000 366.3421 1624.9701 28.32917 1752.4400

3 11638.4990 366.34207 0.0000 1589.9690 684.06328 816.8405

4 2668.1533 1624.97005 1589.9690 0.0000 3057.43735 108.5525

5 385.8402 28.32917 684.0633 3057.4373 0.00000 3250.5954

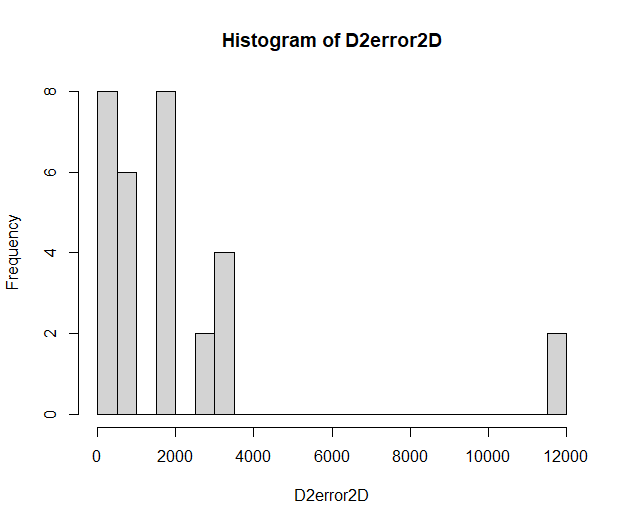
6 1928.9132 1752.44004 816.8405 108.5525 3250.59544 0.0000

The largest error value is 11638.5 at entry (1, 3) and (3, 1) which corresponds to Edmonton (1) and Juba (3), the same cities with the largest error for the 1D model.

The mean error value is 1692.9.

Both of these values are lower than those from the 1D model.

Here is a histogram of the error values:



We see that most of the error values are from 0 to about 3,800, with some around 12,000,

Which is a better distribution than that of the 1D model.

***D2* 3-dimensional model**

Finally, let’s look at the 3D model for *D2*.

We create the 3D model using:

cmdscale(D2, k=3, eig=TRUE)

Which outputs:

$points

[,1] [,2] [,3]

[1,] -1115.0389 -1577.7110 6764.9792

[2,] -9857.5225 2621.6345 -297.5585

[3,] -743.2999 -2616.7128 -5935.1040

[4,] -338.2135 -6361.9001 -187.6738

[5,] 8835.3754 -988.5654 221.5351

[6,] 3218.6994 8923.2549 -566.1780

.

.

.

$GOF

[1] 0.8470998 1.0000000

We see that the goodness of fit values have again increased going from 2D to 3D.

The first goodness of fit value is lower than that of the 3D model for *D1*, but the second value is greater.

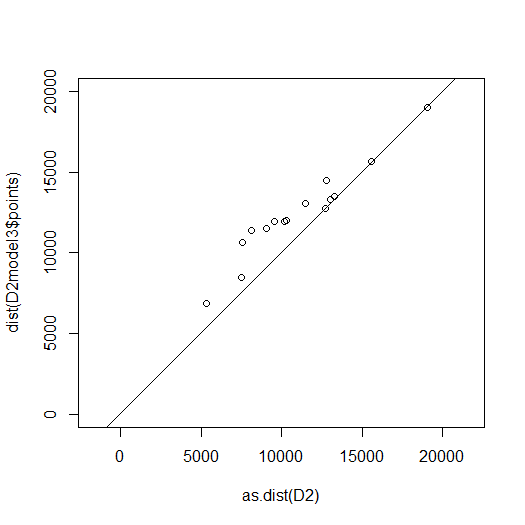
I believe this means that *D2* does not model as well in 3D as *D1*.

We will now make a scatter plot with our original distances as the x-values and our 3D model distances as the y-values using this code:

D1error3D <- abs(D1 - as.matrix(dist(D1model3$points)))

D1error3D[D1error3D == 0] <- NA

hist(D1error3D, breaks=30)



We see that the points have gotten closer to *y* = *x* going from 2D to 3D.

However, we also see that the points are not as close as they are for the 3D model of *D1*, which strengthens the conclusion that *D2* does not model as well in 3D.

We obtain the error matrix using this code:

abs(D2 - as.matrix(dist(D2model3$points)))

1 2 3 4 5 6

1 0.000000 1727.70648 5.934285 960.3351 2341.69481 231.2603

2 1727.706481 0.00000 1782.589846 1625.4313 35.40459 1754.9253

3 5.934285 1782.58985 0.000000 1514.9341 2470.43851 312.1785

4 960.335138 1625.43130 1514.934057 0.0000 3065.30975 113.1163

5 2341.694812 35.40459 2470.438514 3065.3098 0.00000 3277.7952

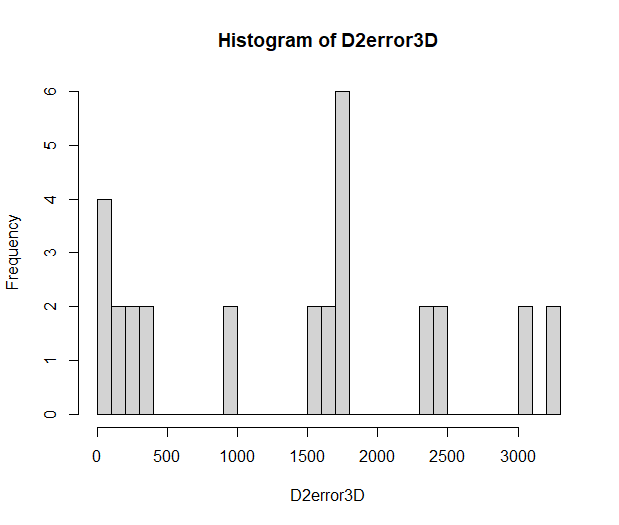
6 231.260272 1754.92533 312.178503 113.1163 3277.79522 0.0000

The max error value is 3277.795 at entry (5, 6) and (6, 5) which corresponds to Wuhan (5) and Sydney (6).

The mean error value is 1178.836.

Both of these values are lower than they were for the 2D model.

Here is a histogram of the error values:



We see that the error values are decently spread out from 0 to about 3500, with a lot around 1700.