Multidimensional Scaling (chapter 15)

In multidimensional scaling, you represent distances between multidimensional objects using a smaller number of dimensions, typically two or three.

You can then plot the objects onto this reduced dimensional space. The idea is similar to only plotting the first two principle components, except that instead of working with linear combinations of the original variables, you work with distances between the observations.

Multidimensional scaling

For some problems, you can work with either principle components or multidimensional scaling and might get similar results. However, with multidimensional scaling, because you are working with distances, you only need to be able to define the distances between two observations. This can be useful, for example if you have missing data.

Suppose you have observation vectors $\mathbf{y}_i$, $i = 1, \ldots, n$ but there is a high degree of missing data so that for a given i, y_{ij} might only be defined for some $j \in \{1, \ldots, p\}$. Then the distance between vectors $\mathbf{y}_i$ and $\mathbf{y}_j$ can be defined as

$$d(\mathbf{y}_i, \mathbf{y}_j) = \sqrt{\sum_{k: y_{ik}, y_{jk} \text{ are not missing}} (y_{ik} - y_{jk})^2}$$

Multidimensional scaling

The above distances would be defined as long as there is at least one $k \in \{1, \ldots, p\}$ such that y_{ik} and y_{jk} are both not missing.

As an example, you might want to make something like a PCA of countries based on economic profiles, but some economic indicators might not be available for some countries. Still, pairwise distances between countries can be defined based on the economic indicators that are not missing for both countries.

Even if two observations have no variables in common, you could use triangle inequality arguments to get bounds on the distances. I.e., if countries A and C have no variables in common, but A has variables in common with B, and B has variables in common with C, then $d(A, C) \geq d(A, B) + d(B, C)$.

Multidimensional scaling (MDS)

For multidimensional scaling, we can imagine that there is a true set of distances, $\{\delta ij\}$, based on p dimensions, that are reduced to a lower dimensional set of distances, $\{d_{ij}\}$. As an example, knowing the true distances between cities on Earth requires taking into account that the cities are on an imperfect sphere, in three dimensions, and we might approximate this with a two-dimensional map of latitude and longitude. The distances based on two-dimensional Euclidean distances might be good enough for some purposes, especially if the cities are close together.

Often distances are more abstract, such as the economic distances between countries, distances between politicians based on voting records, genetic distances between populations, etc. So in multidimensional scaling, we project distances based on p dimensions into distances based on $k < p$ distances. Typically, the user chooses the value of k, but often MDS is used for graphical purposes, so that $k = 2$ is often used, even this results in substantial distortion of the distances.

Multidimensional scaling

For metric multidimensional scaling, in which we have true distances between objects based on p dimensions, we have a distance matrix $\mathbf{D} = (\delta_{ij})$. The matrix $\mathbf{D}$ is $n \times n$ since there are n observations.

The goal is to find a set of points in k dimensions (often $k = 2$), such that the distances in these k dimensions are d_{ij} and $d_{ij} \approx \delta_{ij}$.

An analogy is to suppose you have a list of flight distances)presumably based on geodesics, which depend on the three-dimensional shape of the Earth) between cities. For example,

$$d(ABQ, LAX) = 664, \quad d(ABQ, SEA) = 1184, \quad d(LAX, SEA) = 959$$

based on these data only, can you construct a triangle in $\mathbb{R}^2$ with the three points that have these distances?

Multidimensional scaling

To construct the points, we do the following:

1. Construct $\mathbf{A} = (a_{ij}) = (-\frac{1}{2}\delta_{ij}^2)$.
2. Construct $\mathbf{B} = (b_{ij})$ where

$$b_{ij} = a_{ij} - \overline{a}_{i\cdot} - \overline{a}_{\cdot j} + \overline{a}_{\cdot\cdot}$$

$$\mathbf{B} = \left(\mathbf{I} - \frac{1}{n}\mathbf{J}\right)\mathbf{A}\left(\mathbf{I} - \frac{1}{n}\mathbf{J}\right)$$

3. Use the spectral decomposition of $\mathbf{B}$ to write

$$\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}' = \mathbf{V}_1\mathbf{\Lambda}_1\mathbf{V}_1 = \mathbf{V}_1\mathbf{\Lambda}_1^{1/2}\mathbf{\Lambda}_1^{1/2}\mathbf{V}_1 = \mathbf{Z}\mathbf{Z}'$$

where $\mathbf{V}$ is the matrix of eigenvectors of $\mathbf{B}$ and $\mathbf{\Lambda}$ is the diagonal matrix of the eigenvalues. The matrices $\mathbf{V}_1$ and $\mathbf{\Lambda}_1$ correspond to the eigenvectors and eigenvalues where the eigenvalues are positive.

Multidimensional scaling

If only the first $q < k$ eigenvalues of $\mathbf{B}$ are positive, then a q-dimensional configuration of points can satisfy $d_{ij} = \delta_{ij}$. If $k < q$, then the k-dimensional multidimensional scaling cannot recover the original distances, but we seek a configuration of points that minimizes the distortion in the distances.

To continue the method for finding the best points...

Multidimensional scaling

4. The rows of $\mathbf{Z}$, $\mathbf{z}_1', \ldots, \mathbf{z}_n'$ satisfy

$$d_{ij}^2 = (\mathbf{z}_i - \mathbf{z}_j)'(\mathbf{z}_i - \mathbf{z}_j)$$

5. Since typically $q > k$ only the first k eigenvectors and eigenvalues are used to find the new coordinates of the n points. The matrix $\mathbf{Z}$ is $n \times q$, so just the first k columns can be used to plot the new points in $\mathbb{R}^k$.

Multidimensional scaling

We'll use the three-point example of the airline distances between ABQ, LAX, and SEA airports. For this example,

$$\mathbf{D} = \begin{pmatrix} 0 & 664 & 1184 \\ 664 & 0 & 959 \\ 1184 & 959 & 0 \end{pmatrix}$$

$$\mathbf{A} = -\begin{pmatrix} 0 & 664^2 & 1184^2 \\ 664^2 & 0 & 959^2 \\ 1184^2 & 959^2 & 0 \end{pmatrix}$$

$$\mathbf{B} = \left(\mathbf{I} - \frac{1}{3}\mathbf{J}\right)\mathbf{A}\left(\mathbf{I} - \frac{1}{3}\mathbf{J}\right)$$

Multidimensional scaling in R

```
> D <- c(0,664,1184,664,0,959,1184,959,0)
> D <- matrix(D,ncol=3)
> D
     [,1] [,2] [,3]
[1,]    0  664 1184
[2,]  664    0  959
[3,] 1184  959    0
> A <- -(1/2)*D^2
> A
        [,1]       [,2]       [,3]
[1,]       0 -220448.0 -700928.0
[2,] -220448       0.0 -459840.5
[3,] -700928 -459840.5       0.0
```

Multidimensional scaling in R

```
> I3 <- diag(3)
> I3
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> J <- matrix(rep(1,9),ncol=3)
> J
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
```

Multidimensional scaling in R

Note that the third eigenvalue is close to 0, suggesting that the data is nearly two-dimensional.

```
> B <- (I3 - (1/3)*J) %*% A %*% (I3 - (1/3)*J)
> B
             [,1]          [,2]       [,3]
[1,]  307313.667     6503.167 -313816.8
[2,]    6503.167   146588.667 -153091.8
[3,] -313816.833 -153091.833   466908.7
> V <- eigen(B)
> V
$values
[1] 7.378113e+05 1.829997e+05 1.205708e-12

$vectors
           [,1]       [,2]      [,3]
[1,] -0.5779379  0.5767620 0.5773503
```

Multidimensional scaling in R

Eigenvectors are in the columns of `V$vectors`.

```
> B %*% V$vectors[,1]
           [,1]
[1,] -426409.1
[2,] -155325.2
[3,]  581734.4
> V$vectors[,1] * V$values[1]
[1] -426409.1 -155325.2  581734.4
```

Multidimensional scaling in R

The three dimensional coordinates are

```
> V$vectors %*% diag(sqrt(V$values))
          [,1]      [,2]         [,3]
[1,] -496.425  246.7299 6.33958e-07
[2,] -180.830 -337.4750 6.33958e-07
[3,]  677.255   90.7451 6.33958e-07
```

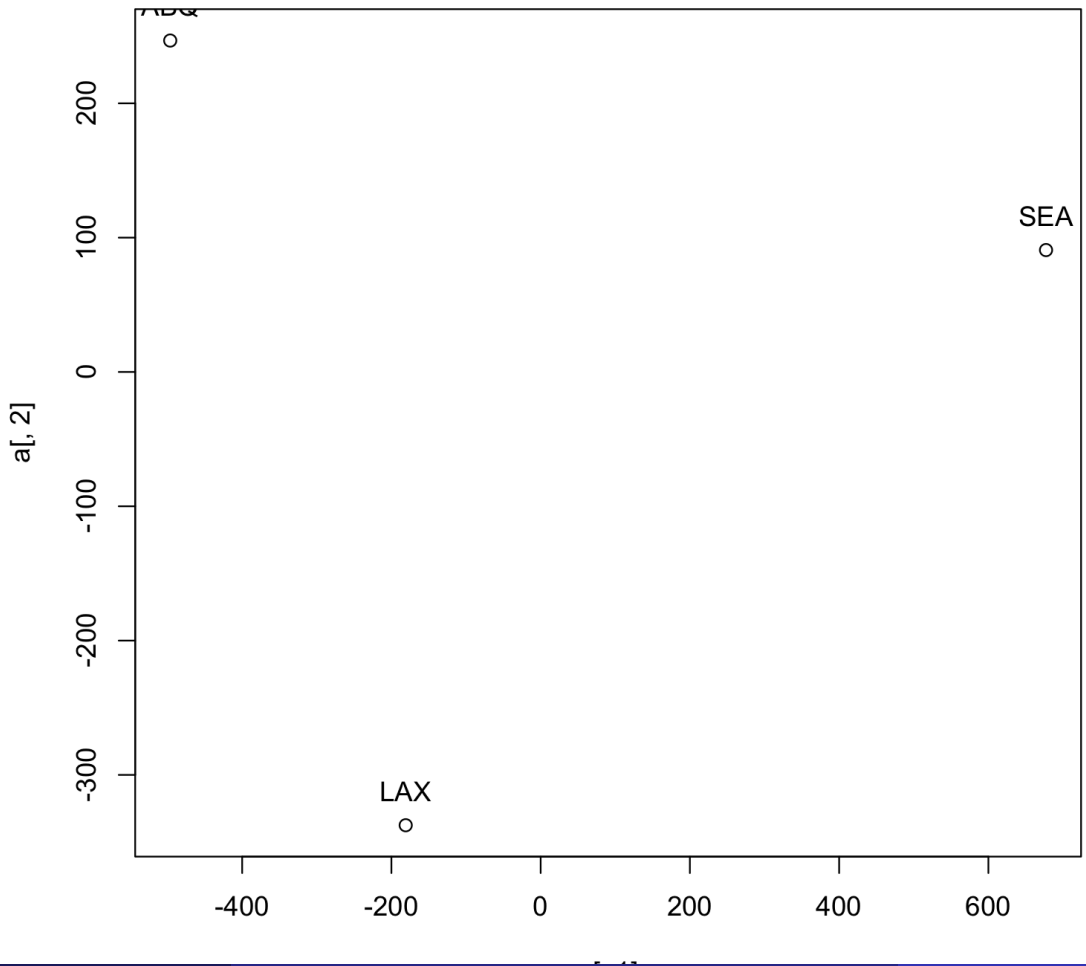
Multidimensional scaling in R

Since the third coordinate is the same in all cases, it can be ignored, and we get 2-dimensional coordinates for the three cities.

```
> V$vectors %*% diag(sqrt(V$values))
          [,1]       [,2]         [,3]
[1,] -496.425  246.7299 6.33958e-07
[2,] -180.830 -337.4750 6.33958e-07
[3,]  677.255   90.7451 6.33958e-07
> Z <- V$vectors %*% diag(sqrt(V$values))
> plot(a[,1],a[,2])
> sqrt(sum((a[1,]-a[2,])^2)
+ )
[1] 664
> sqrt(sum((a[1,]-a[2,])^2))
[1] 664
> sqrt(sum((a[1,]-a[3,])^2))
[1] 1184
```

Plotting MDS

The plot of the coordinates isn't very intuititve:

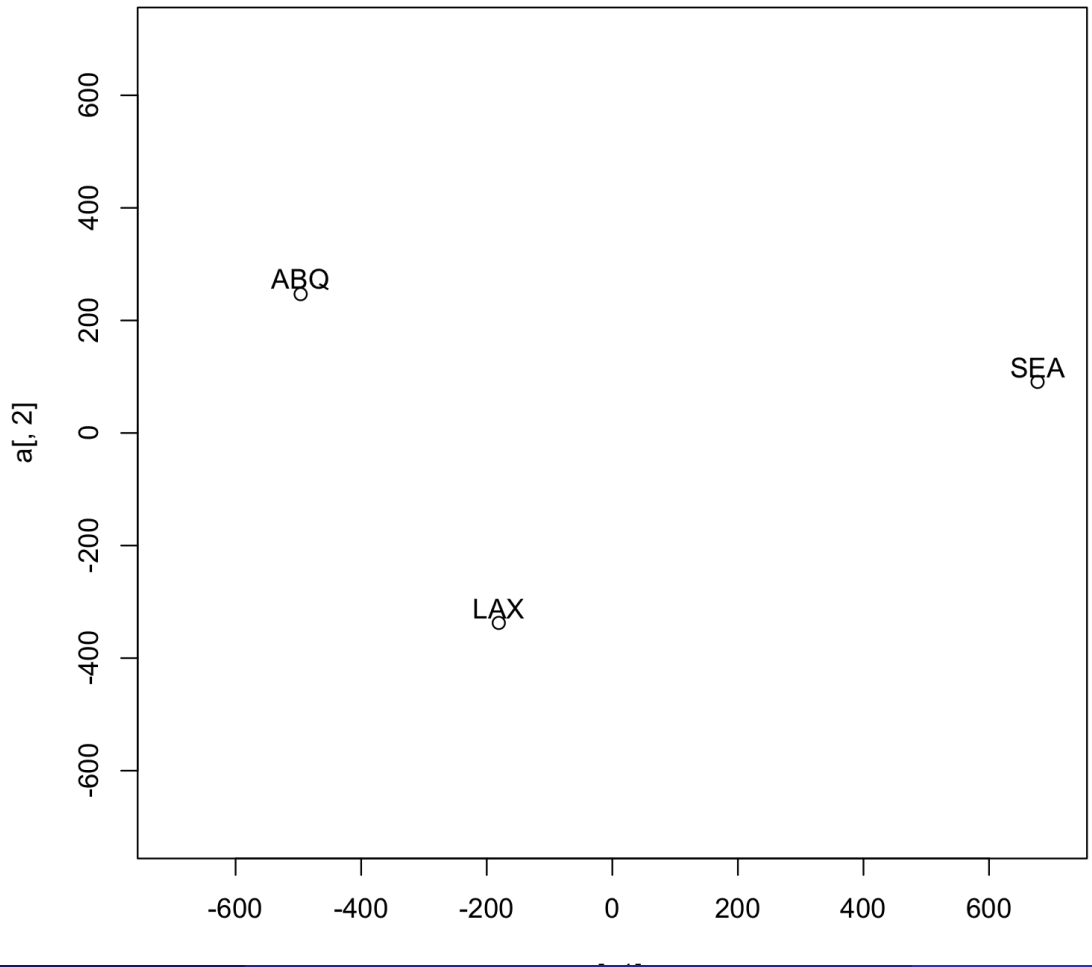


Multidimensional scaling in R

```
> plot(a[,1],a[,2],xlim=c(-700,700),ylim=c(-700,700))
> text(a[,1],a[,2]+25,c("ABQ","LAX","SEA"))
> # or try
> plot(-a[,1],a[,2],xlim=c(-700,700),ylim=c(-700,700))
> text(-a[,1],a[,2]+25,c("ABQ","LAX","SEA"))
```

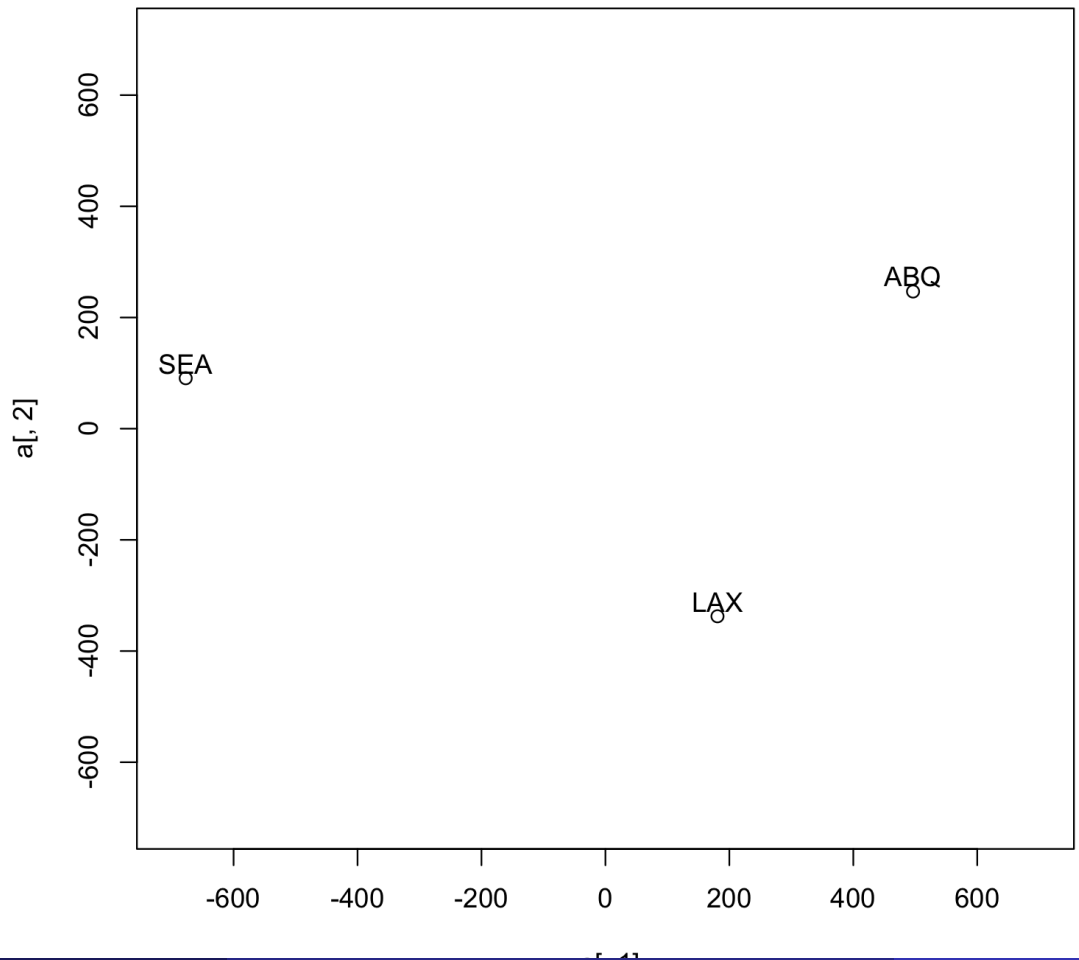
Plotting MDS

It helps to make the plotting area square.



Plotting MDS

It helps to make the plotting area square.



Plotting MDS

The rotation of the axes and the mirror image aspect isn't unique. The distances are equally good whether you take a mirror image and whether you rotate. If you have an intuitive idea of how the points should map in terms of left-right or up-down, then it is ok to multiple coordinates by -1 to make a mirror image, and also ok to rotate points around the origin. This is similar to principle components – we care about the relative positions of the points, not the exact coordinates.

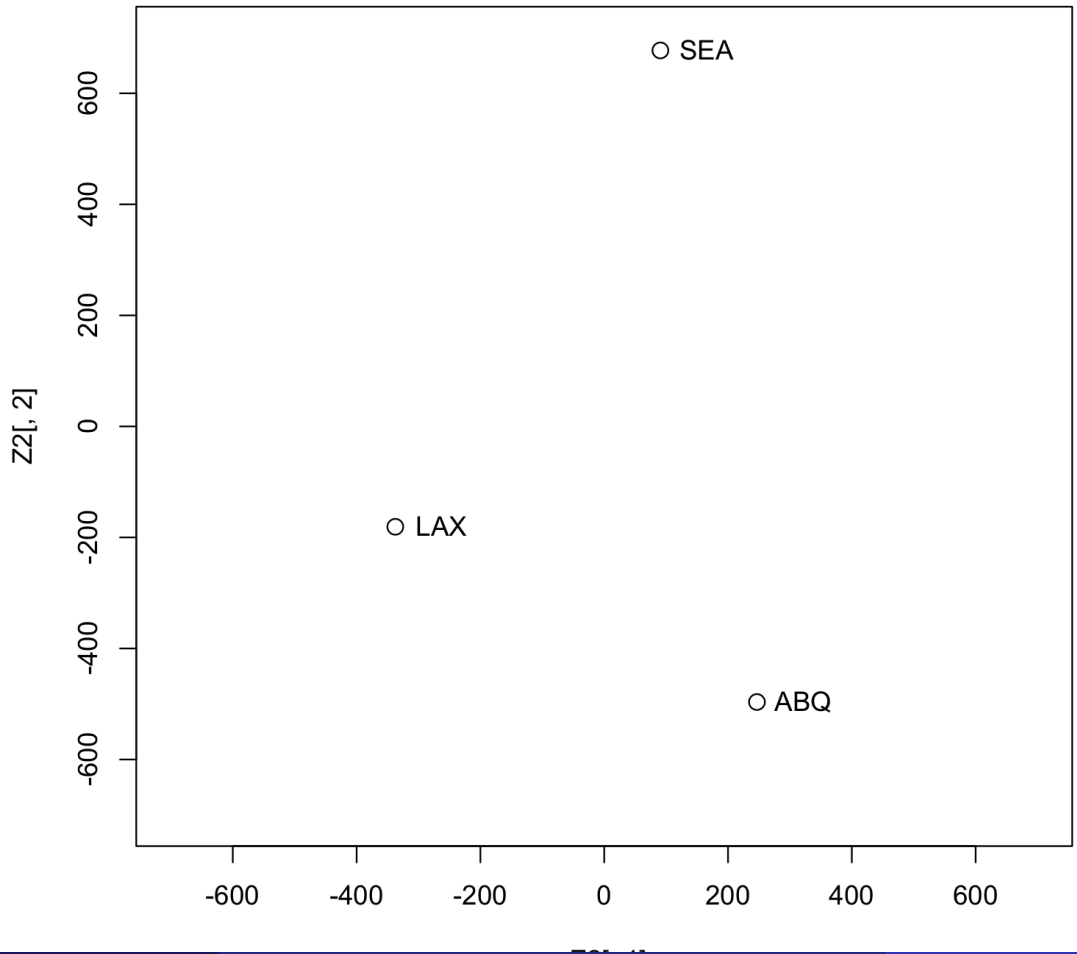
For this example, if we multiply the first coordinate by -1, then it looks like rotating about 90 degrees clockwise will generate a plot that resembles the geographic positions better. Alternatively, we could rotate the original data 90 degrees counter clockwise and then take a mirror image.

Multidimensional scaling in R

```
> theta <- pi/2
> R <- c(cos(theta),-sin(theta),sin(theta),cos(theta))
> R <- matrix(R,ncol=2,byrow=T)
> Z <- a[,1:2]
> Z2 <- t(R %*% t(Z2))
> plot(-Z2[,1],Z2[,2],cex=1.3,ylim=c(-700,700),xlim=c(-700,700
> text(-Z2[,1]+75,Z2[,2],c("ABQ","LAX","SEA"))
```

Plotting MDS

It helps to make the plotting area square.



Multidimensional scaling in R

This appears to have over-rotated the points a bit. If they are rotated a little bit clockwise again, then we should get something that appears about right. You can experiment with different values of theta until you get the plot you want.

Later this week, we'll try making a set of points like this correspond to an actual map and try to find the optimal rotation relative to an actual map to make them correspond as closely as possible. This can be done with a single calculation (instead of trial and error with different values of theta) using Procrustes rotations to minimize the sum of squared distances between the true points and points based on MDS.

Multidimensional scaling

To choose the dimension k to use, one statistic that is used is called the STRESS statistic. This measures the sum of squared differences between distances based on p dimensions versus distances based on $k < p$ dimensions and scales this value by the sum of squared distances based on p dimensions.

$$STRESS = \frac{\sum_{i<j}(d_{ij} - \delta_{ij})^2}{\sum_{i<j}\delta_{ij}^2}$$

The STRESS statistic is then plotted as a function of k. This can be used to justify the choice of $k = 2$ or $k = 3$, or if the plots don't look good, can help explain why $k = 2$ is not sufficient to capture the distances in the original data.

Plotting MDS

It helps to make the plotting area square.

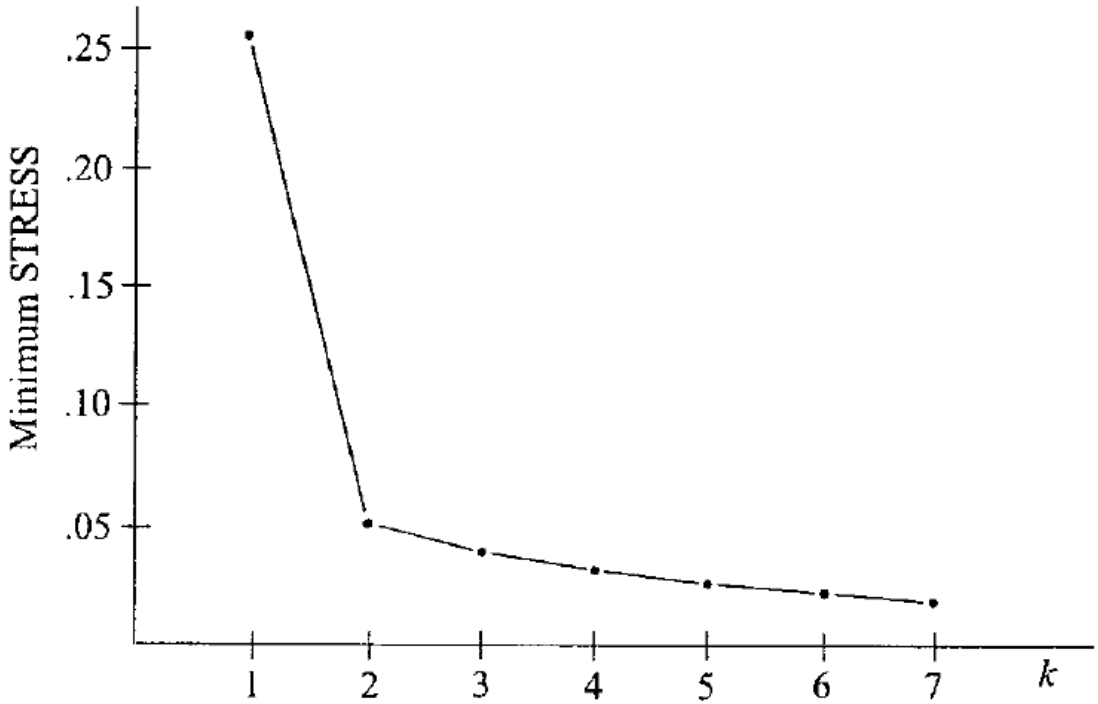


Figure 15.4. Ideal plot of minimum STRESS versus k.

Multidimensional scaling in R

MDS is built into R.

```
> MDS
$points
          [,1]       [,2]
[1,]   496.425 -246.7299
[2,]   180.830  337.4750
[3,] -677.255   -90.7451

$eig
[1] 7.37811e+05 1.83000e+05 4.36557e-11
```

This essentially agrees with what we got before but the new points are rotated compared with what we had. The `cmdscale` command also requires $k < n$, so the function returns an error for $k = 3$ even though the theory works out.

MDS: voting example

Table 15.2. Dissimilarity Matrix for Voting Records of 15 Congressmen

	R_1	R_2	D_1	D_2	R_3	R_4	R_5	D_3	D_4	D_5	D_6	R_6	R_7	R_8	D_7
R_1	0	8	15	15	10	9	7	15	16	14	15	16	7	11	13
R_2	8	0	17	12	13	13	12	16	17	15	16	17	13	12	16
D_1	15	17	0	9	16	12	15	5	5	6	5	4	11	10	7
D_2	15	12	9	0	14	12	13	10	8	8	8	6	15	10	7
R_3	10	13	16	14	0	8	9	13	14	12	12	12	10	11	11
R_4	9	13	12	12	8	0	7	12	11	10	9	10	6	6	10
R_5	7	12	15	13	9	7	0	17	16	15	14	15	10	11	13
D_3	15	16	5	10	13	12	17	0	4	5	5	3	12	7	6
D_4	16	17	5	8	14	11	16	4	0	3	2	1	13	7	5
D_5	14	15	6	8	12	10	15	5	3	0	1	2	11	4	6
D_6	15	16	5	8	12	9	14	5	2	1	0	1	12	5	5
R_6	16	17	4	6	12	10	15	3	1	2	1	0	12	6	4
R_7	7	13	11	15	10	6	10	12	13	11	12	12	0	9	13
R_8	11	12	10	10	11	6	11	7	7	4	5	6	9	0	9
D_7	13	16	7	7	11	10	13	6	5	6	5	4	13	9	0

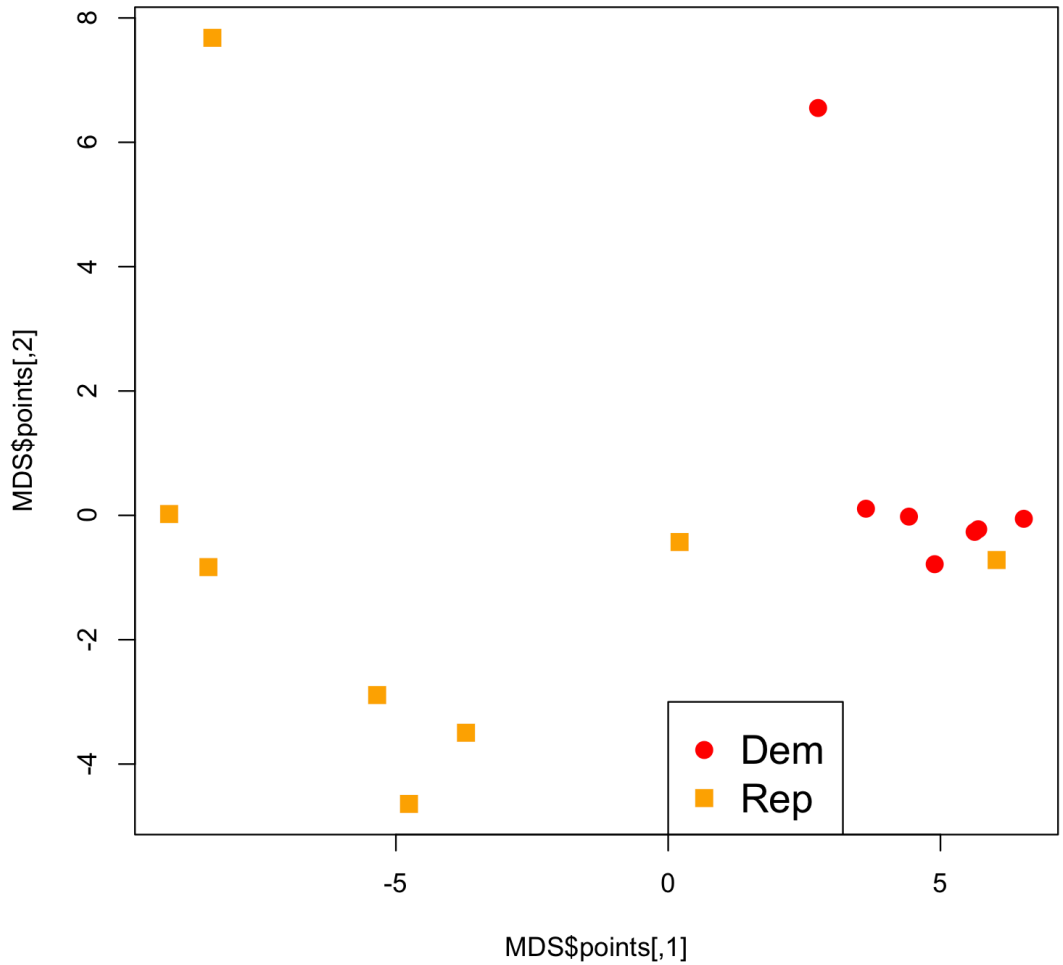
Multidimensional scaling in R

```
> x <- read.table("vote.txt")
> D <- x[,2:16]
> x
   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16
1  R1  0  8 15 15 10  9  7 15  16  14  15  16   7  11  13
2  R2  8  0 17 12 13 13 12 16  17  15  16  17  13  12  16
3  D1 15 17  0  9 16 12 15  5   5   6   5   4  11  10   7
4  D2 15 12  9  0 14 12 13 10   8   8   8   6  15  10   7
5  R3 10 13 16 14  0  8  9 13  14  12  12  12  10  11  11
6  R4  9 13 12 12  8  0  7 12  11  10   9  10   6   6  10
7  R5  7 12 15 13  9  7  0 17  16  15  14  15  10  11  13
8  D3 15 16  5 10 13 12 17  0   4   5   5   3  12   7   6
9  D4 16 17  5  8 14 11 16  4   0   3   2   1  13   7   5
10 D5 14 15  6  8 12 10 15  5   3   0   1   2  11   4   6
11 D6 15 16  5  8 12  9 14  5   2   1   0   1  12   5   5
12 R6 16 17  4  6 12 10 15  3   1   2   1   0  12   6   4
13 R7  7 13 11 15 10  6 10 12  13  11  12  12   0   9  13
14 R8 11 12 10 10 11  6 11  7   7   4   5   6   9   0   9
```

Multidimensional scaling in R

```
> MDS <- cmdscale(D,k=2,eig=T)
> mypch=c(15,15,16,16,15,15,15,16,16,16,16,15,15,15,16)
> col=c("orange","orange","red","red","orange","orange",
"orange","red","red","red","red","orange","orange","orange","red")
> plot(MDS$points,type="n")
> points(MDS$points[,1],MDS$points[,2],col=col,
pch=mypch,cex=1.5)
>  legend(0,-3,legend=c("Dem","Rep"),
col=c("red","orange"),pch=c(16,15),cex=1.5)
```

MDS: voting example



MDS: voting example

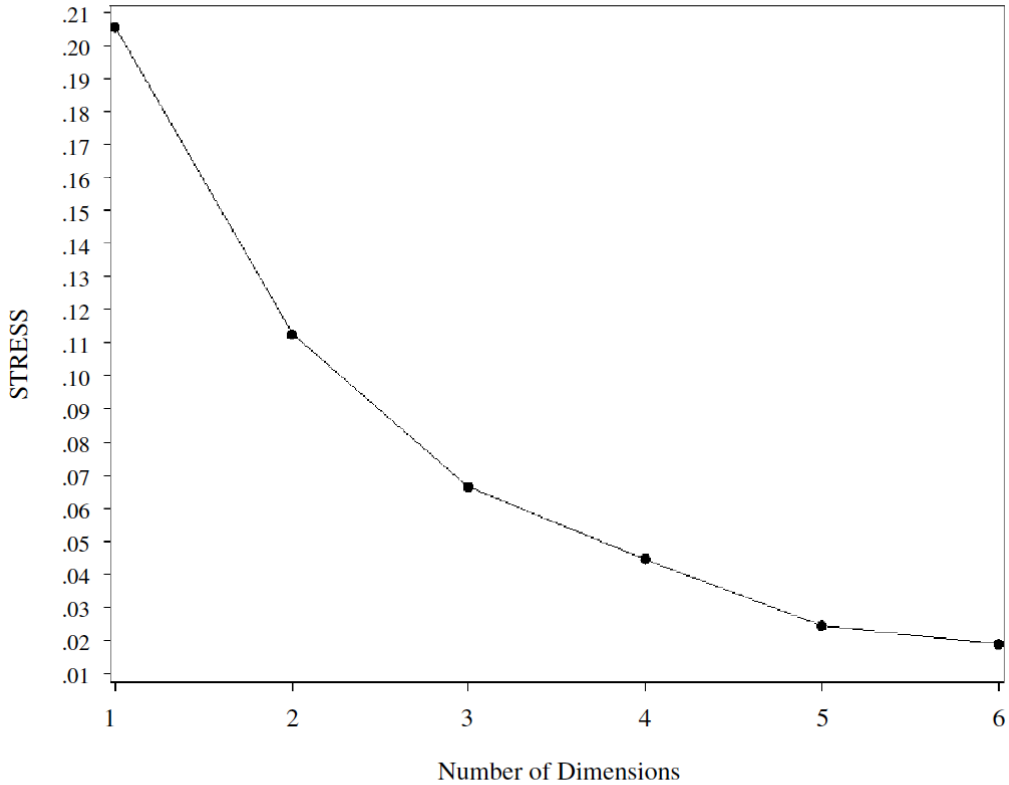


Figure 15.5. Plot of STRESS for each value of k for the voting data in Table 15.2.

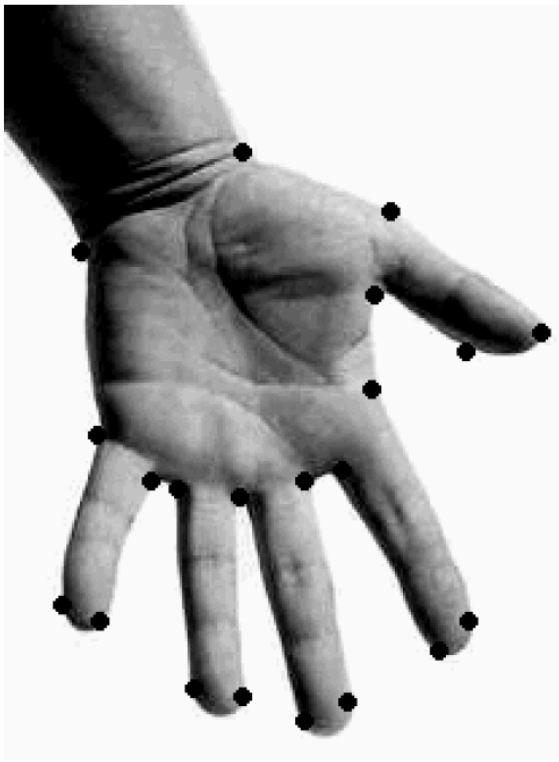
Comparing shapes

An example where you might need something like PROC IML is if you have to rotate your data, which can be accomplished through matrix multiplication of your original data. This comes up in statistics when you want to compare to sets of points (usually either in 2D or 3D, but could be higher-dimensional), and the points are not oriented the same way or scaled the same way.

This can come up particularly in shape analysis, where you want to determine whether two shapes are roughly equivalent, or you want to compare two photographs taken from slightly different positions. In addition to statistical testing, sometimes you just want to best visualize the difference between two sets of points, and this is best accomplished by lining up the points as nearly as possible.

Comparing shapes

As an example, consider two photographs of hands.



Comparing shapes

We have reference points on the hands, and we want to line up the reference points as closely as possible by rotating the images, rescaling if necessary (suppose you have photos that are cropped or zoomed in and still want to compare the shapes). In general we might also allow mirror images. In this case, we assume the points are two-dimensional so that they each have just an x and y coordinate. This would typically be the case for analyzing photographic images, although in general you can imagine have three dimensional data as well.

Comparing shapes

Applications for this are widespread. In medical imaging, you might take an x-ray of a patient overtime to compare how their spine is changing with osteoporosis. The x-rays won't be taken at identical distances, angles and so forth, so you need to align the images by stretching and rotating.

Other examples will include MRI scans of the brains, where you might want to either compare the same individual at different time points, the left versus the right hemisphere to look for asymmetries, or two separate individuals to see how closely aligned two brains are. Here we want to ignore the fact that one brain might be slightly larger than the other.
If eyes or fingerprints are used for ID, again it will be easiest to compare two images by rotation and rescaling.

Comparing shapes

If you have satellite images of regions on earth, you might want to measure things like habitat loss. Successive photos of the same region won't be exactly the same, so you might try to align two photos using certain geographical reference points. Once the photos are aligned, you can use differences in the area that is green as a measure of vegetation loss, for example.

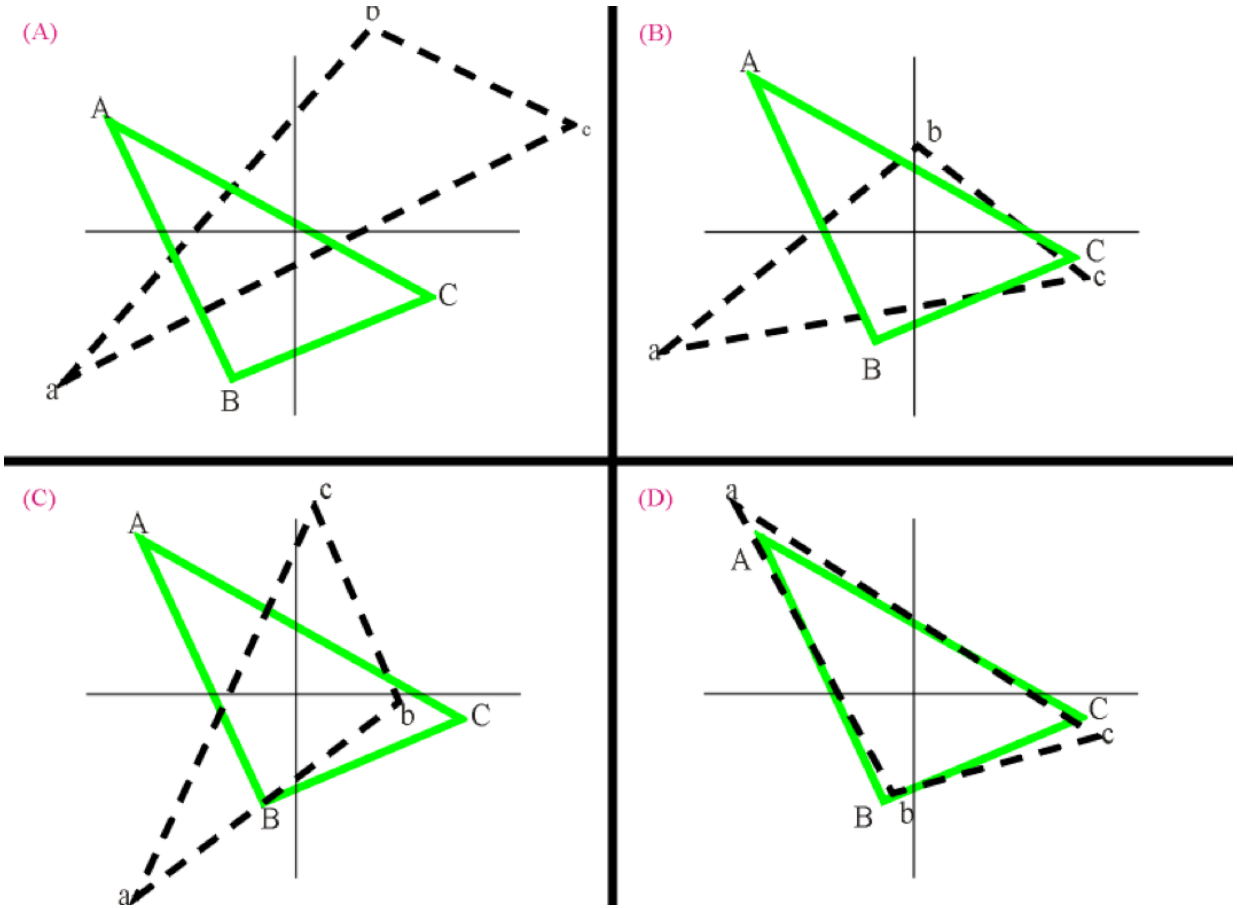
"The name Procrustes refers to a bandit from Greek mythology who made his victims fit his bed either by stretching their limbs or cutting them off." (Wikipedia)

Procrustes illustrations

You can find some interesting illustrations online....

Procrustes illustrations

Procrustes illustrations



http://jackson.eeb.utoronto.ca/procrustes-analysis/

Comparing shapes

Back to the hand example. Here we have two sets of coordinates. We might call them

$$X = \{(x_{11}\ , x_{12}), (x_{21}, x_{22}), \ldots, (x_{n1}, x_{n2})\}$$

and

$$Y = \{(y_{11}, y_{12}), \ldots, (y_{n1}, y_{n2})\}$$

How should we align the points? If we use the distances between corresponding points, we can minimize the distance between points over all possible angles of rotation, rescalings, and reflections. To deal with reflections, it might help to center the points so that $(\overline{x}_1, \overline{x}_2) = 0$ and $(\overline{y}_1, \overline{y}_2) = 0$. For many problems (like with satellite photographs or the same patient over time), reflections won't matter.

Comparing shapes

The distance between two individual points $x_i = (x_{i1}, x_{i2})$ and $y_i = (y_{i1}, y_{i2})$ is naturally defined as

$$d(x_i, y_i) = \sqrt{(x_{i1} - y_{i1})^2 + (x_{i2} - y_{i2})^2}$$

This is the Euclidean distance between two points in the plane. We might define the overall squared distance from the set of points X to the set of points Y as

$$d^2(X, Y) = \sum_{i=1}^{n} d^2(x_i, y_i)$$

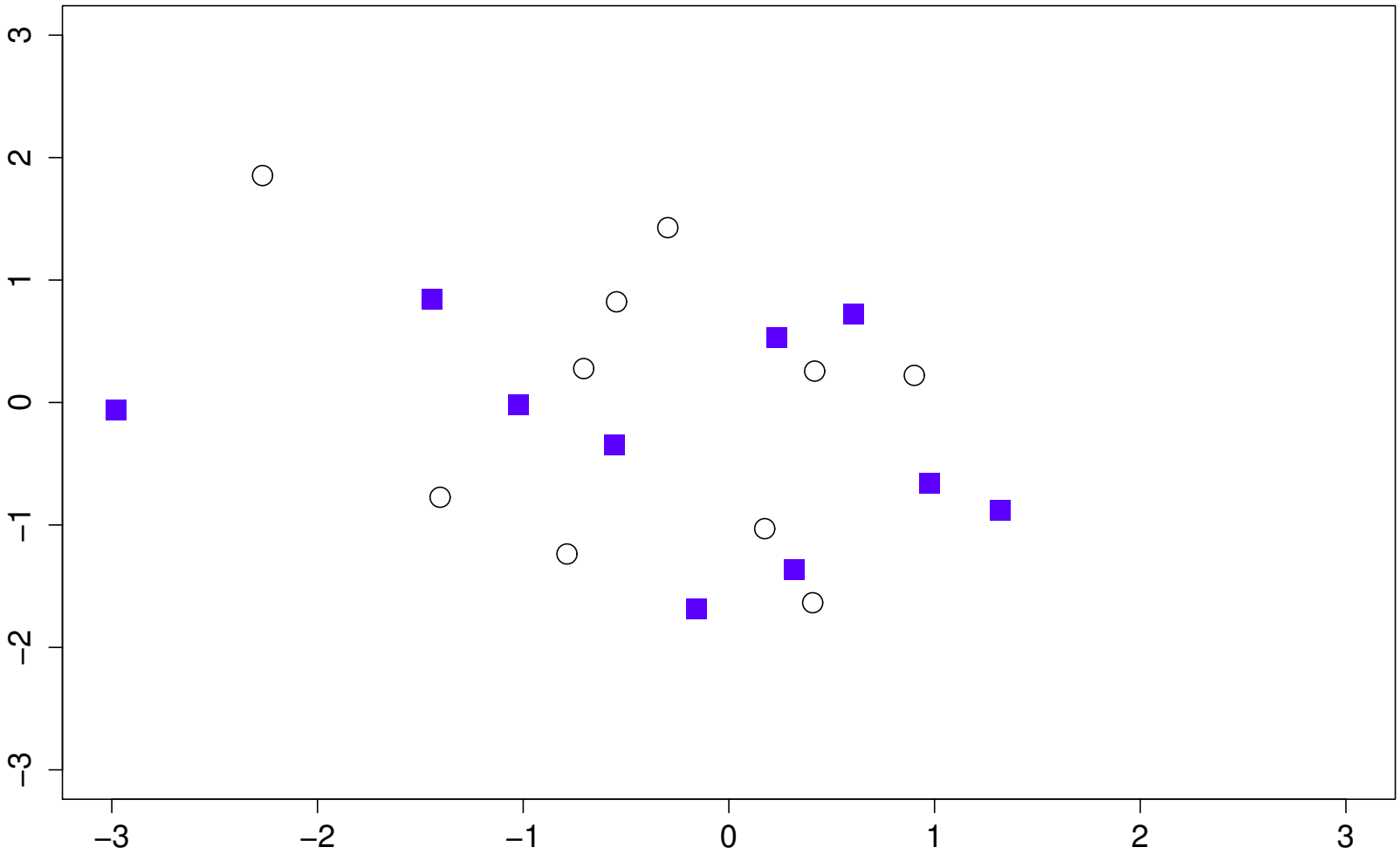
Comparing shapes

We can then minimize the sum of squared distances between points (this is equivalent to minimizing the distances–why?). This is fairly similar as a criterion to what we do in regression, so hopefully doesn't seem too weird. In other words, we want to minimize

$$d^2(x_i, y_i) = (x_{i1} - y'_{i1})^2 + (x_{i2} - y'_{i2})^2$$

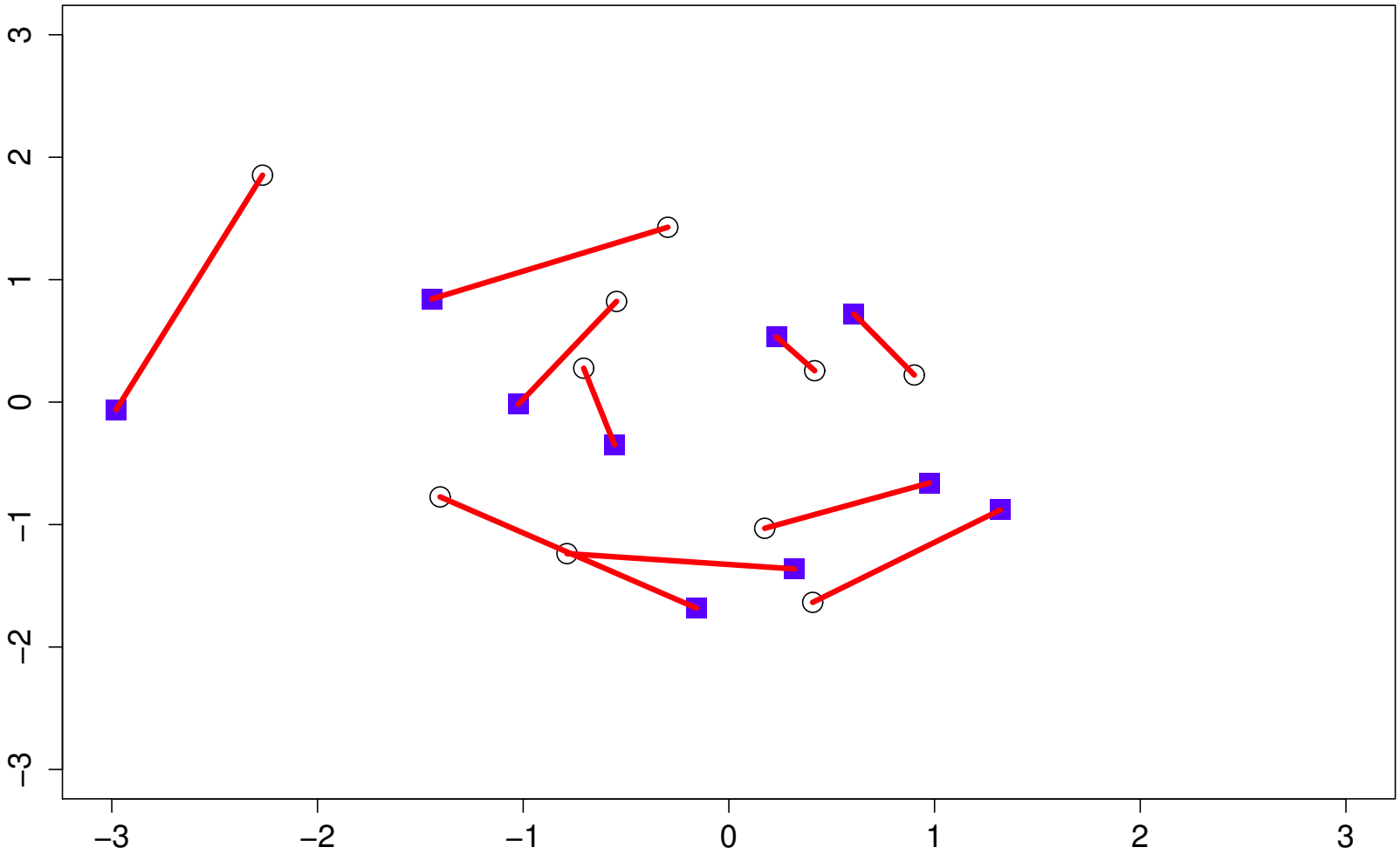
over all choices of θ and c.
To do this, we need to write y'_i as a function of y_i, θ, and a scaling factor c.

Example



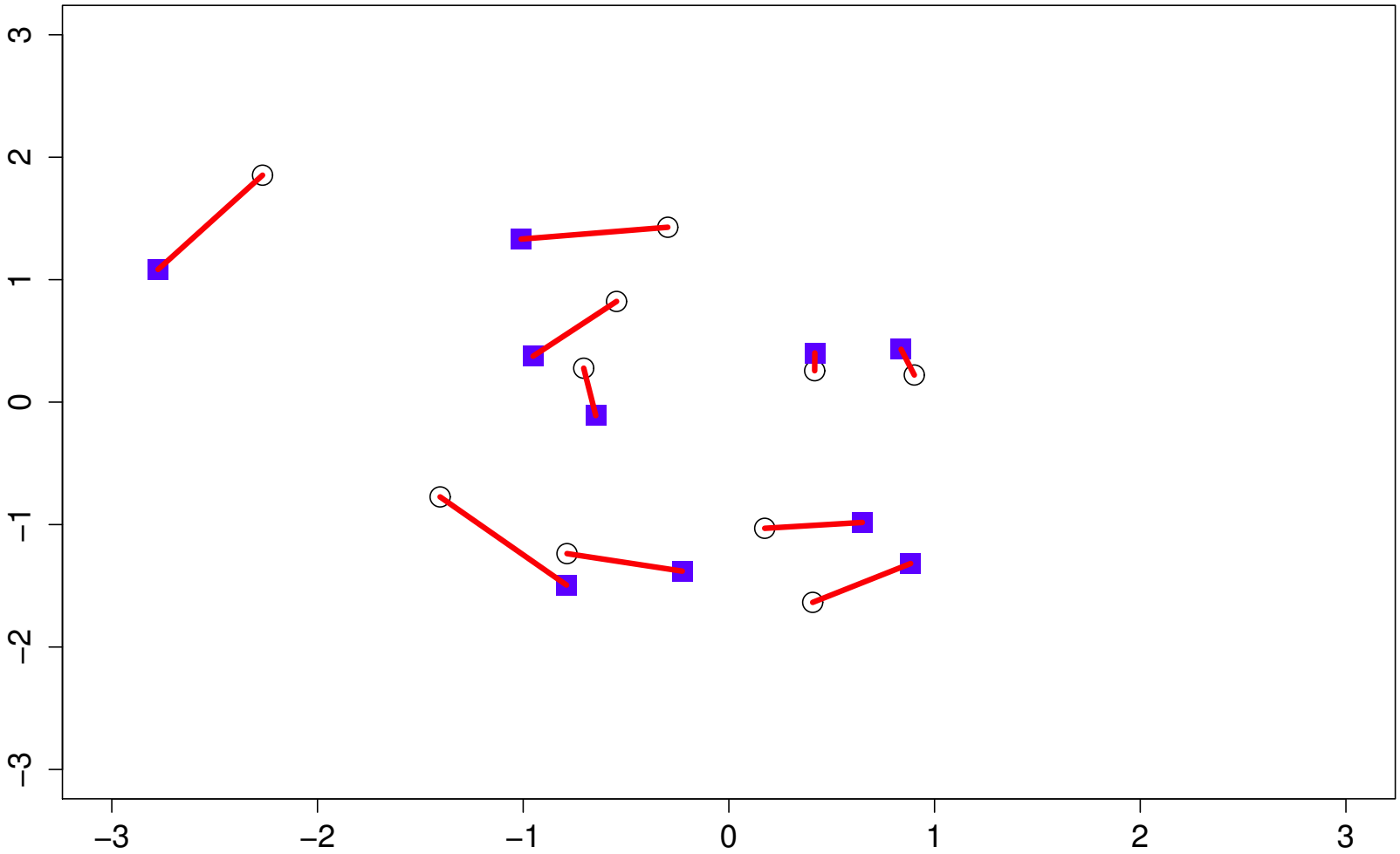
Example

Lines connecting corresponding points.

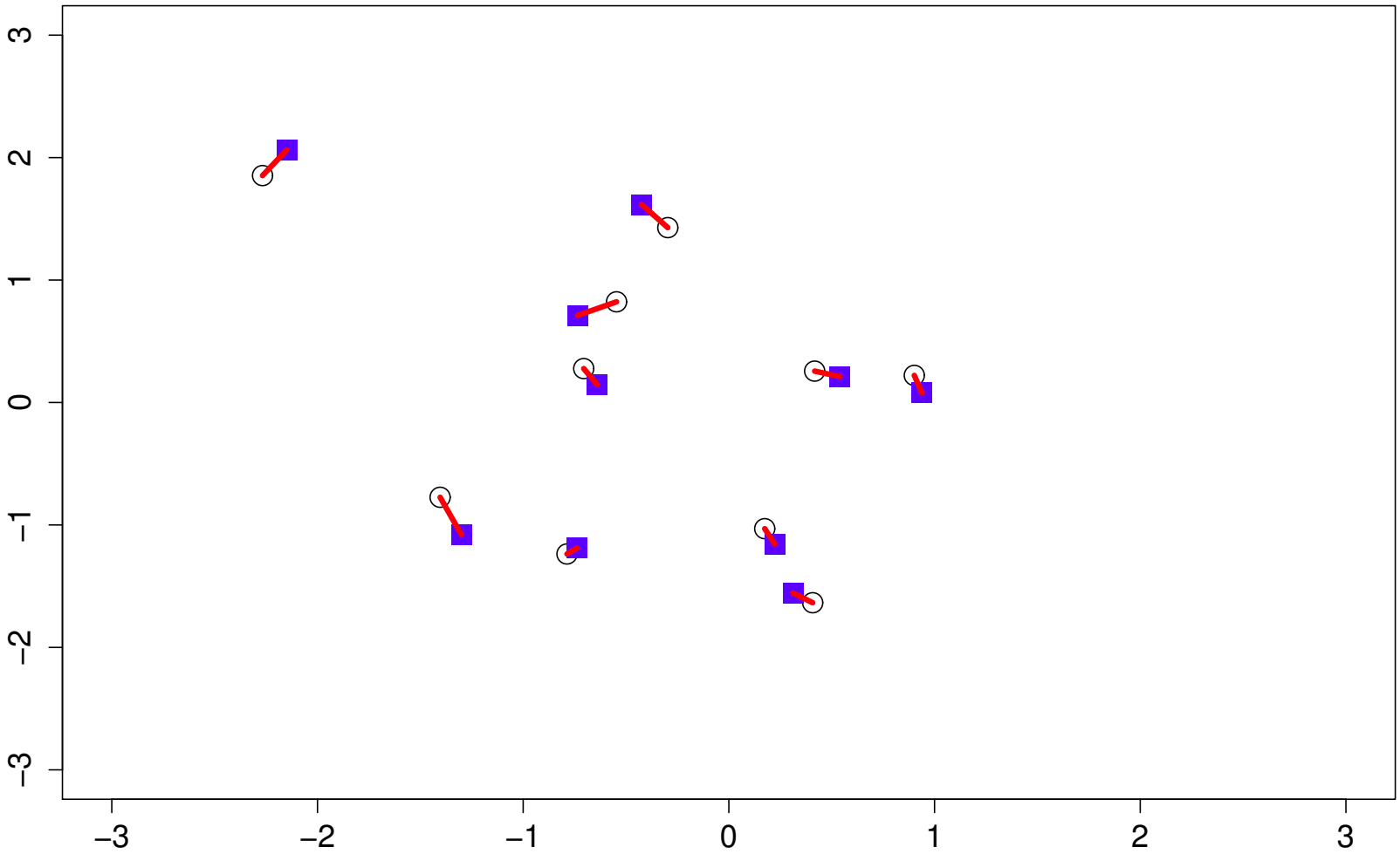


Example

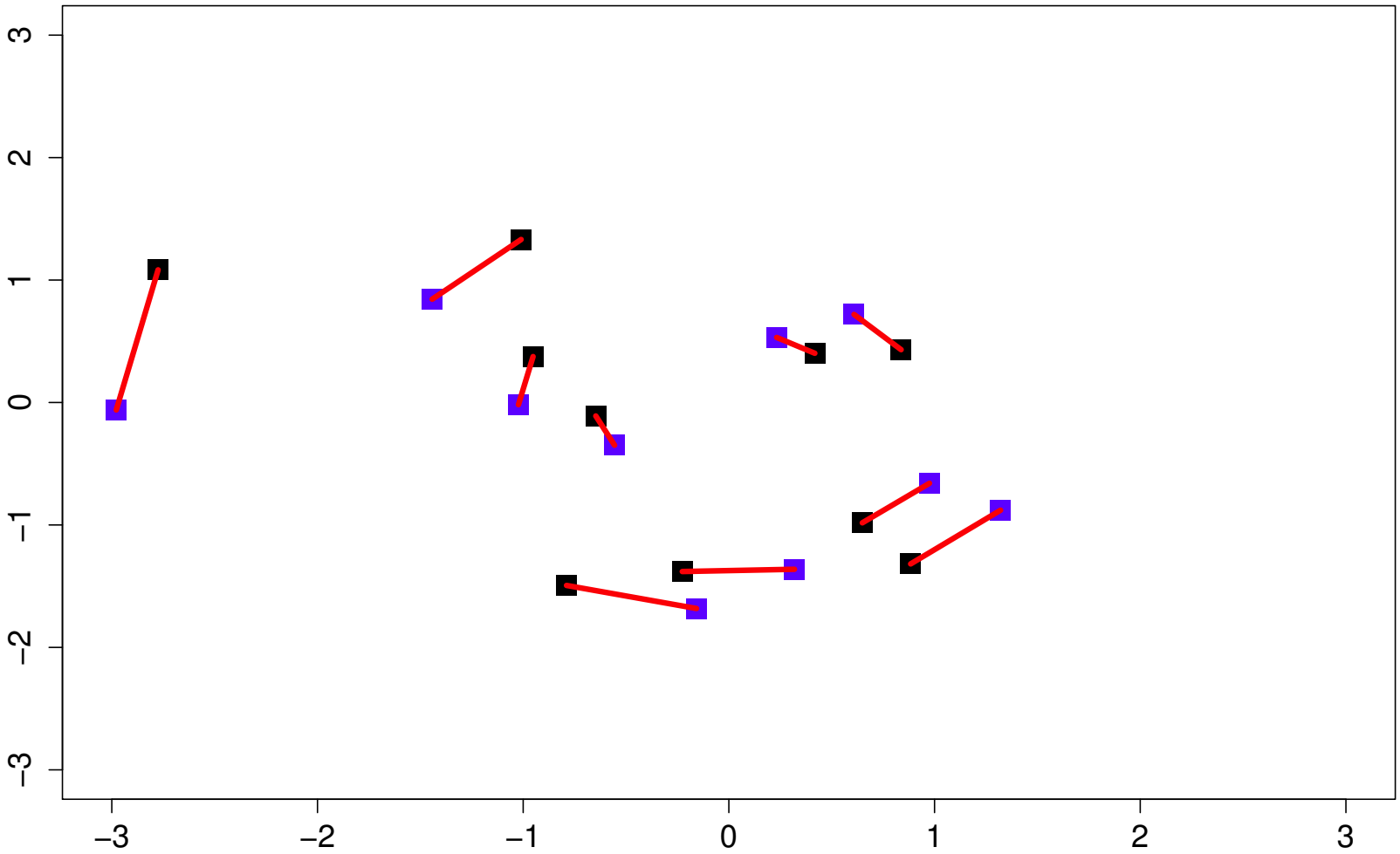
Rotating $\pi/8$ radians $= 22.5$ degrees clockwise, we get...



Example



Example



Comparing shapes

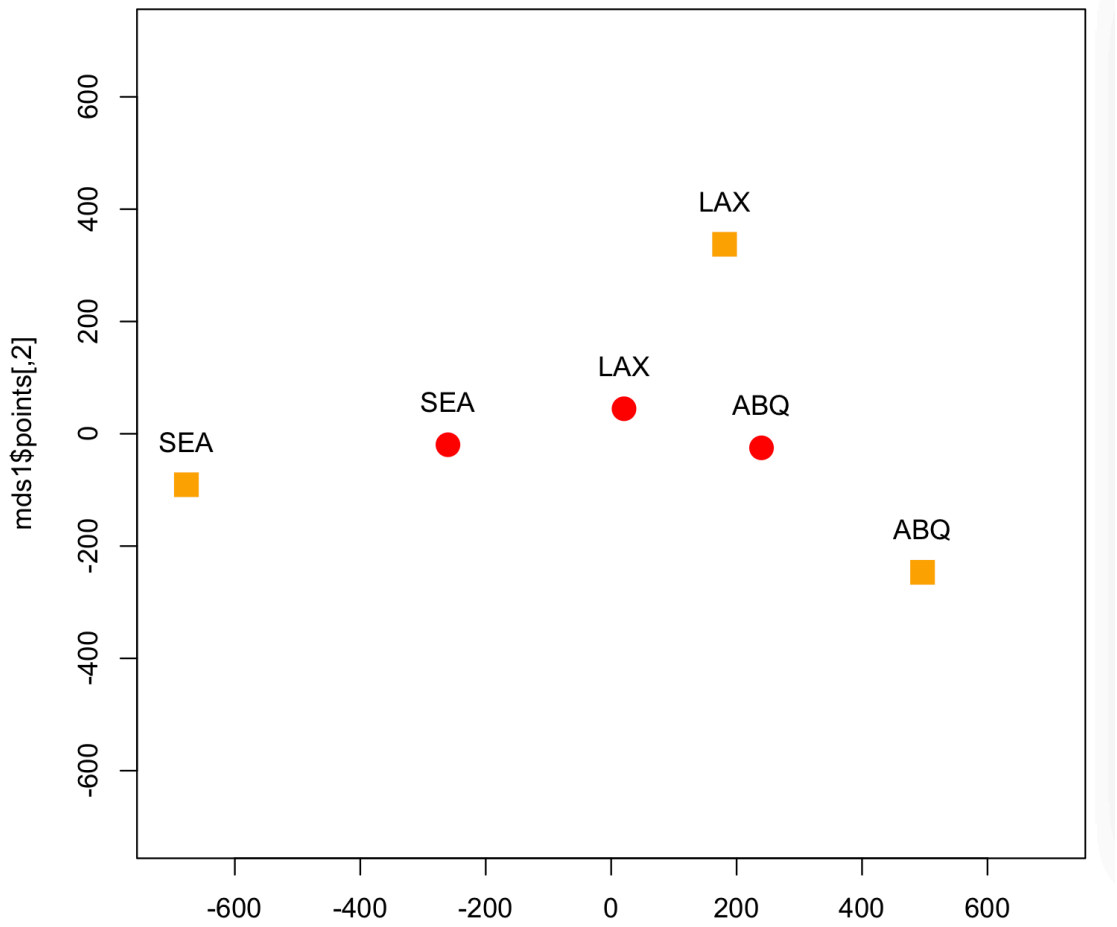
As an example, we'll do a multidimensional scaling of distances between *ABQ*, *LAX*, and *SEA* as measured by airline prices. These distances are (using the cheapest available prices on a website and rounding)

$$d(ABQ, LAX) = 230, \quad d(ABQ, SEA) = 500, \quad d(LAX, SEA) = 288$$

We'll plot the two MDS plots together on the same plot. Since the numbers happen to be on a similar scale (distances and dollars both being in the hundreds), the points will show up on the same plot in spite of the units being different. Alternatively, you could use z-scores for both distances and dollars.

```
> D1 <- c(0,664,1184,664,0,959,1184,959,0)
> D1 <- matrix(D1,ncol=3)
> mds1 <- cmdscale(D1,k=2,eig=T)
> D2 <- c(0,230,500,230,0,288,500,288,0)
> D2 <- matrix(D2,ncol=3)
> mds2 <- cmdscale(D2,k=2,eig=T)
> plot(mds1$points,ylim=c(-700,700),xlim=c(-700,700),cex=1.5,c
> points(mds2$points,cex=1.5,col="orange",pch=16)
> text(mds1$points[,1],mds1$points[,2]+75,c("ABQ","LAX","SEA")
> text(mds2$points[,1],mds2$points[,2]+75,c("ABQ","LAX","SEA")
```

How similar are these triangles? (red=distance,orange=dollars)



Procrustes rotation

If we have two data sets, $\mathbf{Y}$ and $\mathbf{X}$, we can imagine transforming one data set to closely match the second data set.

The idea is to find a rotation matrix $\mathbf{A}$ (which might also cause reflections), a scaling factor ρ and possibly a translation vector $\mathbf{b}$ such that the transformation

$$f(\mathbf{x}_i) = \rho \mathbf{A} \mathbf{x}_i + \mathbf{b}$$

minimizes

$$d(f(\mathbf{X}), \mathbf{Y}) = \sum_{i=1}^{n} (\mathbf{y}_i - f(\mathbf{x}_i))'(\mathbf{y}_i - f(\mathbf{x}_i))$$

the sum of squared distances.

Procrustes rotation

Let $\mathbf{X}_0$ be a matrix where each row is the k-dimensional centroid of $\mathbf{X}$ and similarly $\mathbf{Y}_0$ has rows consisting of the centroids of $\mathbf{Y}$. Then let $\mathbf{X}_c = \mathbf{X} - \mathbf{X}_0$ and $\mathbf{Y}_c = \mathbf{Y} - \mathbf{Y}_0$. These are centered versions of the data sets, so that when plotted, there centroids will be at the origin.
Once we've computed these, the solution to the Procrustes problem is

Procrustes rotation

$$\mathbf{A} = \mathbf{V}\mathbf{U}'$$
$$\rho = \text{tr}(\Lambda)/\text{tr}(\mathbf{X}_c'\mathbf{X}_c)$$
$$\mathbf{b} = \mathbf{y}_0 + \rho\mathbf{A}'\mathbf{x}_0$$

where $\mathbf{VU}'$ comes from the singular value decomposition of $\mathbf{C} = \mathbf{Y}_c'\mathbf{X}_c = \mathbf{U}\mathbf{\Lambda}\mathbf{V}'$ and where $\mathbf{y}_0$ and $\mathbf{x}_0$ are the centroids of $\mathbf{Y}$ and $\mathbf{X}$. The distance between the transformed datasets is the Procrustes distance:

$$d(f(\mathbf{X}), \mathbf{Y}) = \text{tr}(\mathbf{Y}_c'\mathbf{Y}_c) - [\text{tr}(\Lambda])]^2/\text{tr}(\mathbf{X}_c'\mathbf{X}_c)$$

Procrustes rotation

```
> y0 <- colMeans(mds1$points)
> x0 <- colMeans(mds2$points)
> # centroids are approximately 0
> X0 <- matrix(c(x0,x0,x0),ncol=2,byrow=T)
> SVD <- svd(t(mds2$points) %*% mds1$points)
> XX <- t(mds2$points-X0) %*% (mds2$points-X0)
> XX[1,1]+XX[2,2] # trace
[1] 128614.7
> sum(eigen(t(mds2$points-X0) %*% (mds1$points-Y0))$values)
[1] 321971
>  321971/128614.7
[1] 2.503376 # scaling factor
```

Procrustes rotation

```
> plot(mds1$points,pch=15,col="orange",cex=2,ylim=c(-700,700),
xlim=c(-700,700))
> points(mds2$points,pch=16,col="red",cex=2)
> text(mds1$points[,1],mds1$points[,2]+75,c("ABQ","LAX","SEA"))
> text(mds2$points[,1],mds2$points[,2]+75,c("ABQ","LAX","SEA"))
> points(A %*% mds2$points,col="blue",cex=2,pch=17)
Error in A %*% mds2$points : non-conformable arguments
> points(t(A %*% t(mds2$points)),col="blue",cex=2,pch=17)
> points(2.503376*t(A %*% t(mds2$points)),col="black",cex=2,pch=17)
> legend(-600,-300,c("distance","dollar","rotated dollar",
"scaled and rotated dollar"),pch=c(15,16,17,17),col=
c("orange","red","blue","black"),cex=1.5)
```

How similar are these triangles?