STAD29: Statistics for the Life and Social Sciences

Lecture notes

Section 1

Multidimensional scaling

Multidimensional Scaling

- Have distances between individuals.
- Want to draw a picture (map) in 2 dimensions showing individuals so that distances (or order of distances) as close together as possible. (Or maybe 3 with `rgl`.)
- If want to preserve actual distances, called *metric multidimensional scaling* (in R, `cmdscale`).
- If only want to preserve order of distances, called *non-metric multidimensional scaling* (in R, `isoMDS` in package `MASS`).
- Metric scaling has solution that can be worked out exactly.
- Non-metric only has iterative solution.
- Assess quality of fit, see whether use of resulting map is reasonable. (Try something obviously 3-dimensional and assess its failure.)

Packages

The usual, plus some new stuff:

```
library(MASS)
library(tidyverse)
library(ggrepel)
library(ggmap)
library(shapes)
```

Metric scaling: European cities

CSV file `europe.csv` contains road distances (in km) between 16 European cities. Can we reproduce a map of Europe from these distances?

Read in data:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/europe.csv"
europe <- read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   City = col_character(),
##   Amsterdam = col_double(),
##   Athens = col_double(),
##   Barcelona = col_double(),
##   Berlin = col_double(),
##   Cologne = col_double(),
##   Copenhagen = col_double(),
##   Edinburgh = col_double(),
##   Geneva = col_double(),
##   London = col_double(),
##   Madrid = col_double(),
##   Marseille = col_double(),
##   Munich = col_double(),
```

The data

```
## # A tibble: 16 x 17
##    City  Amsterdam Athens Barcelona Berlin Cologne Copenhagen
##    <chr>     <dbl>  <dbl>     <dbl>  <dbl>   <dbl>      <dbl>
##  1 Amst…         0   3082      1639    649     280        904
##  2 Athe…      3082      0      3312   2552    2562       3414
##  3 Barc…      1639   3312         0   1899    1539       2230
##  4 Berl…       649   2552      1899      0     575        743
##  5 Colo…       280   2562      1539    575       0        730
##  6 Cope…       904   3414      2230    743     730          0
##  7 Edin…      1180   3768      2181   1727    1206       1864
##  8 Gene…      1014   2692       758   1141     765       1531
##  9 Lond…       494   3099      1512   1059     538       1196
## 10 Madr…      1782   3940       628   2527    1776       2597
## 11 Mars…      1323   2997       515   1584    1208       1914
## 12 Muni…       875   2210      1349    604     592       1204
## 13 Paris       515   3140      1125   1094     508       1329
## 14 Prag…       973   2198      1679    354     659       1033
## 15 Rome       1835   2551      1471   1573    1586       2352
## 16 Vien…      1196   1886      1989    666     915       1345
## # … with 10 more variables: Edinburgh <dbl>, Geneva <dbl>,
## #   London <dbl>, Madrid <dbl>, Marseille <dbl>, Munich <dbl>,
## #   Paris <dbl>, Prague <dbl>, Rome <dbl>, Vienna <dbl>
```

Multidimensional scaling

- Create distance object first using all but first column of `europe`. `europe` has distances in it already, so make into `dist` with `as.dist`.
- Then run multidimensional scaling and look at result: xxx

```
europe %>% select(-City) %>% as.dist() -> europe.d
europe.scale <- cmdscale(europe.d)
head(europe.scale)
```

```
##                    [,1]      [,2]
## Amsterdam   -348.162277  528.2657
## Athens      2528.610410 -509.5208
## Barcelona   -695.970779 -984.6093
## Berlin       384.178025  634.5239
## Cologne        5.153446  356.7230
## Copenhagen  -187.104072 1142.5926
```

- This is a `matrix` of x and y coordinates.

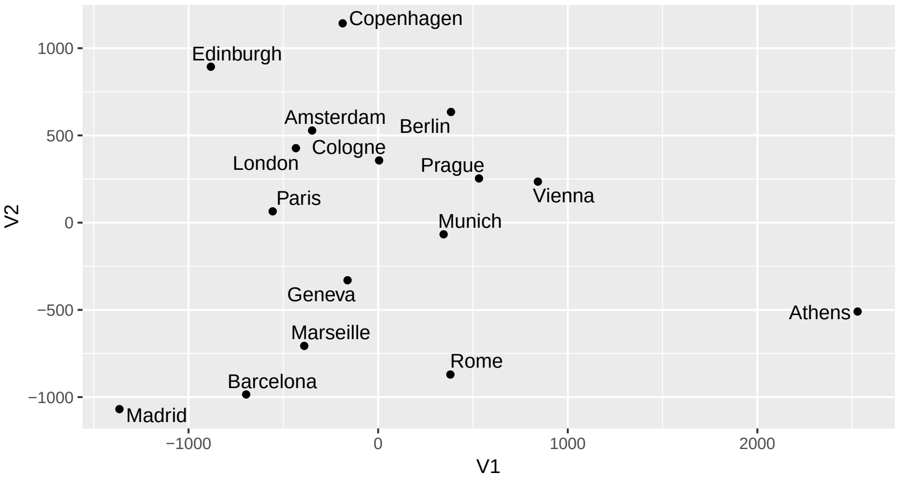
As a data frame; make picture

We know how to plot data frames, so make one first. xxx

```
europe.scale %>%
  as_tibble() %>%
  mutate(city = europe$City) -> europe_coord
g <- ggplot(europe_coord, aes(x = V1, y = V2, label = city)) +
  geom_point() + geom_text_repel()
```

The map xxx

```
g
```


Copenhagen
Edinburgh
Amsterdam
Berlin
Cologne
London
Prague
Vienna
Paris
Munich
Geneva
Athens
Marseille
Rome
Barcelona
Madrid
1000
500
0
−500
−1000
V2
−1000
0
1000
2000
V1

Making a function

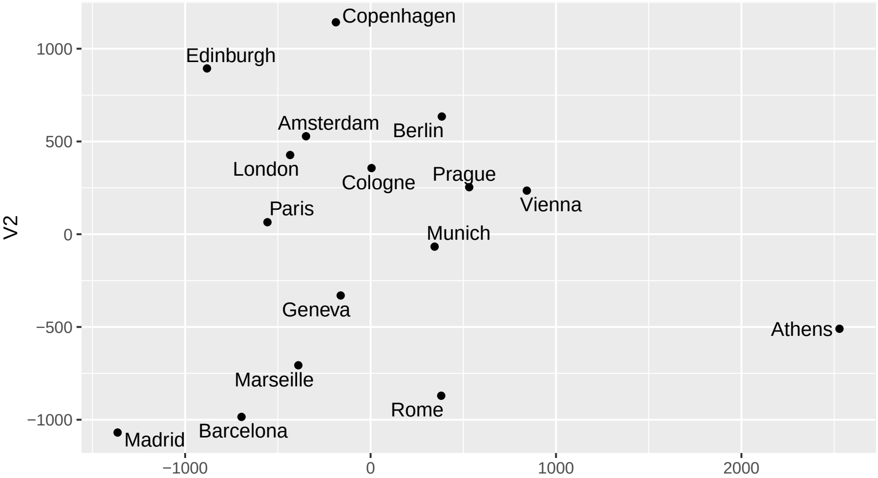
- Idea: given input distance matrix (as stored in a CSV file), output a map (like the one on the previous page). xxx

```
mds_map <- function(filename) {
  x <- read_csv(filename)
  dist <- x %>%
    select_if(is.numeric) %>%
    as.dist()
  x.scale <- cmdscale(dist) # this is a matrix
  x_coord <- x.scale %>%
    as_tibble() %>%
    mutate(place = row.names(x.scale))
  ggplot(x_coord, aes(x = V1, y = V2, label = place)) +
    geom_point() + geom_text_repel() +
    coord_fixed()
}
```

- Use `select_if` to pick out all the numerical columns (no text), whichever they are.
- `x.scale` is matrix with no column headers. Turn into data frame, acquires headers V1 and V2

Does it work?

```
mds_map("europe.csv")
```



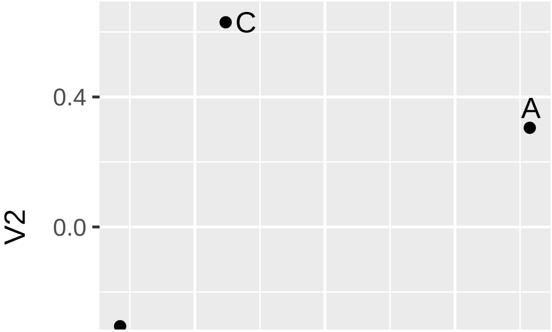
A square xxx

The data, in ‘square.csv‘:

“‘ x,A ,B ,C ,D A,0 ,1 ,1 ,1.4 B,1 ,0 ,1.4,1
C,1 ,1.4,0 ,1 D,1.4,1 ,1 ,0 “‘

\begin{minipage}[t][][b]{0.48} * The map (on right):

```
mds_map("square.csv")
```



Drawing a map of the real Europe

- Works with package ggmap.
- First find latitudes and longitudes of our cities, called *geocoding*: xxx

```
latlong <- geocode(europe$City)
latlong <- bind_cols(city = europe$City, latlong)
latlong %>% slice(1:6)
```

xxx

```
## # A tibble: 6 x 3
##   city          lon   lat
##   <chr>       <dbl> <dbl>
## 1 Amsterdam    4.90  52.4
## 2 Athens      23.7   38.0
## 3 Barcelona    2.17  41.4
## 4 Berlin      13.4   52.5
## 5 Cologne      6.96  50.9
## 6 Copenhagen 12.6   55.7
```

Making the map

- Get a map of Europe from Google Maps (specify what you want a map of any way you can in Google Maps). This one centres the map on the city shown and zooms it so all the cities appear (I had to experiment):

```
map <- get_map("Memmingen DE", zoom = 5)
```

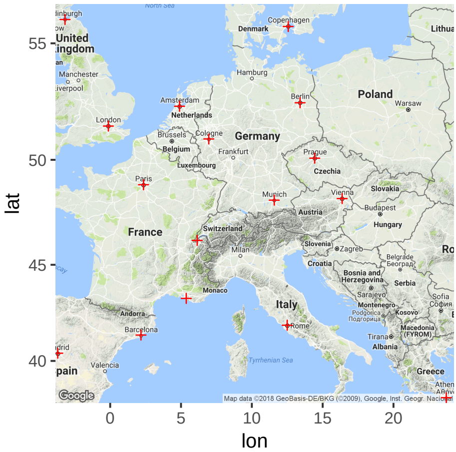
- Plot the map with ggmap. This is ggplot, so add anything to it that you would add to a ggplot, such as cities we want to show:

```
g2 <- ggmap(map) +
  geom_point(
    data = latlong, aes(x = lon, y = lat),
    shape = 3, colour = "red"
  )
```

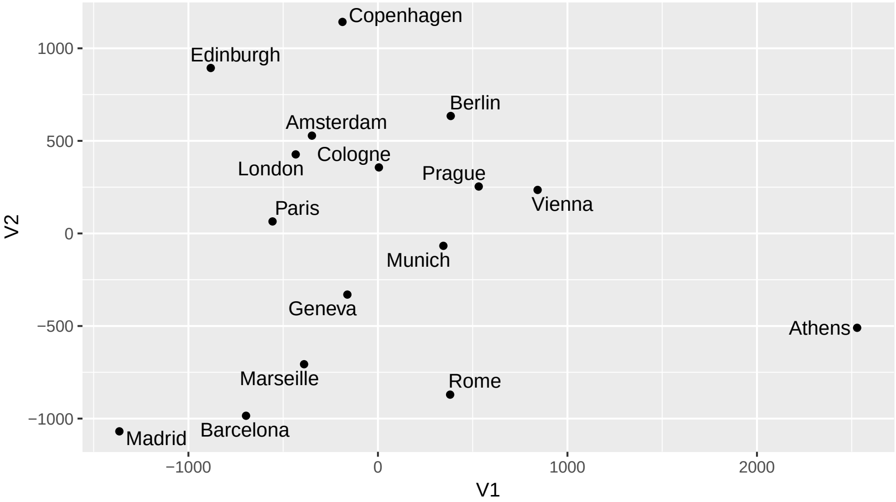
- We don't have a default data frame or aes for our geom_point, so have to specify one.

The real Europe with our cities

```
g2
```



Compare our scaling map



Comments

- North-south not quite right: Edinburgh and Copenhagen on same latitude, also Amsterdam and Berlin; Athens should be south of Rome.
- Rotating clockwise by about 45 degrees should fix that.
- General point: MDS only uses distances, so answer can be “off” by rotation (as here) or reflection (flipping over, say exchanging west and east while leaving north and south same).

Exploring the map by plotting in 3 dimensions

- Package `rgl` makes 3D plots.
- We have to fake up a 3rd dimension (by setting all its values to 1).
- Try this code:

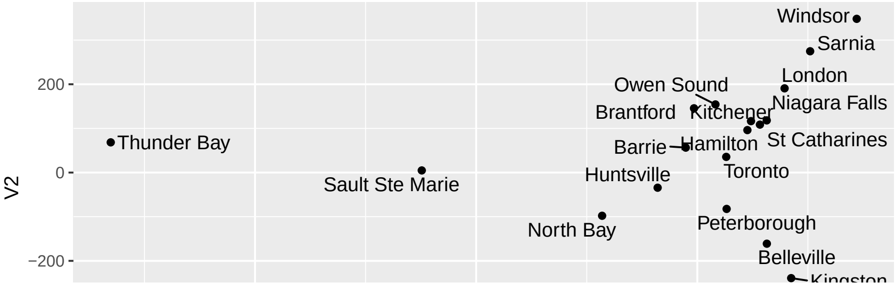
```
library(rgl)
es.2 <- cbind(europe.scale, 1)
plot3d(es.2, zlim = c(-1000, 1000))
text3d(es.2, text = d$city)
```

- Opens a graphics window with the cities plotted and named.
- Click and hold left mouse button to rotate plot. "Rotate away" 3rd dimension to get a possible map (that preserves distances).

Ontario, the same way

…using our function: xxx

```
url <-
  "http://www.utsc.utoronto.ca/~butler/d29/ontario-road-distar
g <- mds_map(url)
g
```



Removing points

- Messy: have to find which rows and columns contain those cities, then remove just those rows and columns.
- Better:
- "tidy" the distance matrix
- then remove rows we don't need
- then "untidy" it again
- save into .csv file
- Illustrate with square data first (easier to see).

Square data

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/square.csv"
square <- read_csv(my_url)
square
```

```
## # A tibble: 4 x 5
##   x         A     B     C     D
##   <chr> <dbl> <dbl> <dbl> <dbl>
## 1 A       0     1     1     1.4
## 2 B       1     0     1.4   1
## 3 C       1     1.4   0     1
## 4 D       1.4   1     1     0
```

Make tidy xxx

```
square %>% gather(point, distance, -x)
```

```
## # A tibble: 16 x 3
##    x     point distance
##    <chr> <chr>    <dbl>
##  1 A     A          0
##  2 B     A          1
##  3 C     A          1
##  4 D     A          1.4
##  5 A     B          1
##  6 B     B          0
##  7 C     B          1.4
##  8 D     B          1
##  9 A     C          1
## 10 B     C          1.4
## 11 C     C          0
## 12 D     C          1
## 13 A     D          1.4
## 14 B     D          1
## 15 C     D          1
## 16 D     D          0
```

Remove all references to point C

In column `x` or `point`: xxx

```
square %>%
  gather(point, distance, -1) %>%
  filter(x != "C", point != "C")
```

```
## # A tibble: 9 x 3
##   x     point distance
##   <chr> <chr>    <dbl>
## 1 A     A          0
## 2 B     A          1
## 3 D     A          1.4
## 4 A     B          1
## 5 B     B          0
## 6 D     B          1
## 7 A     D          1.4
## 8 B     D          1
## 9 D     D          0
```

Put back as distance matrix

and save as .csv when we are happy:

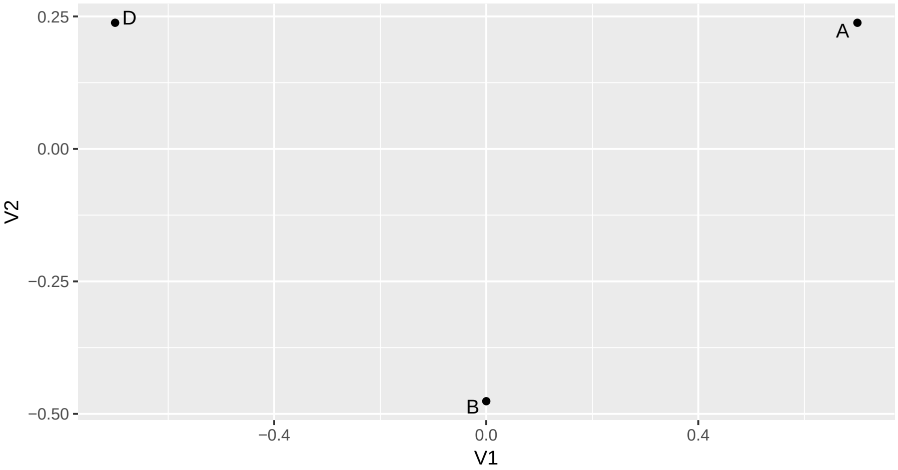
```
noc <- square %>%
  gather(point, distance, -1) %>%
  filter(x != "C", point != "C") %>%
  spread(point, distance)
noc
```

```
## # A tibble: 3 x 4
##   x         A     B     D
##   <chr> <dbl> <dbl> <dbl>
## 1 A       0       1   1.4
## 2 B       1       0   1
## 3 D       1.4     1   0
```

```
noc %>% write_csv("no-c.csv")
```

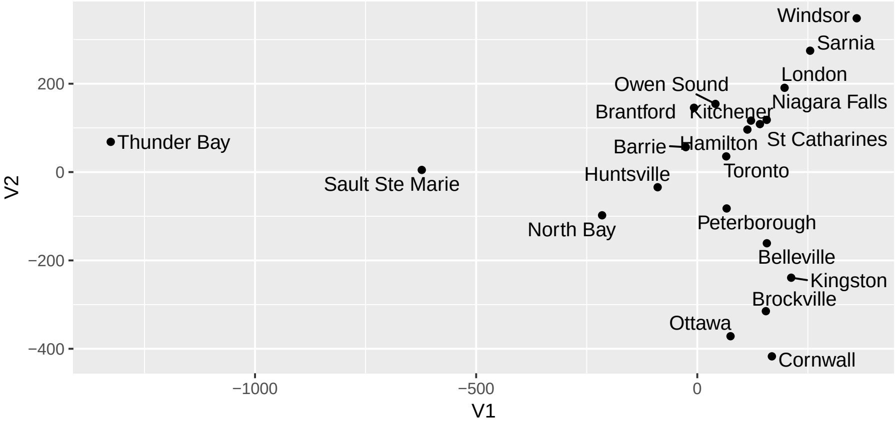
Make map of square-without-C

```
mds_map("no-c.csv")
```



Back to Ontario

```
g
```

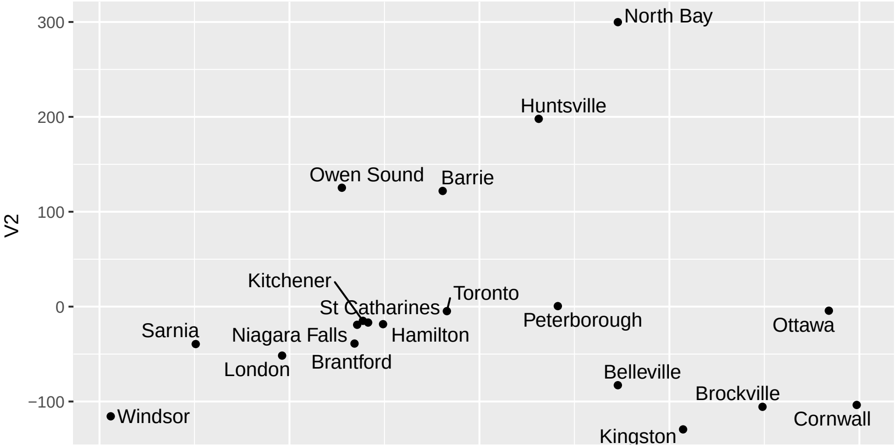


Tidy, remove, untidy xxx

```
my_url <-
  "http://www.utsc.utoronto.ca/~butler/d29/ontario-road-distances.csv"
ontario2 <- read_csv(my_url)
ontario2 %>%
  gather(city, distance, -1) %>%
  filter(
    city != "Thunder Bay",
    place != "Thunder Bay",
    city != "Sault Ste Marie",
    place != "Sault Ste Marie"
  ) %>%
  spread(place, distance) %>%
  write_csv("southern-ontario.csv")
```

Map of Southern Ontario

```r
g <- mds_map("southern-ontario.csv")
g
```



What about that cluster of points?

- Plot looks generally good, but what about that cluster of points?
- "Zoom in" on area between -150 and -100 on x axis, -50 to 0 on y axis.
- Code below overrides the `coord_fixed` we had before. xxx

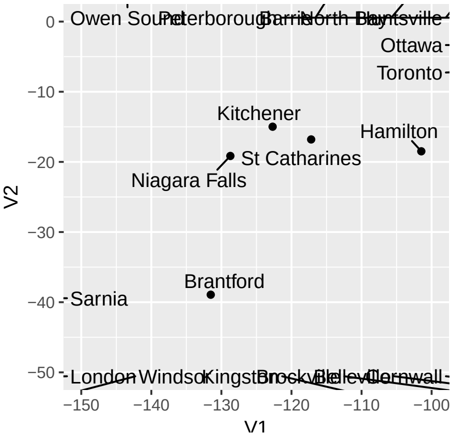
```
g2 <- g + coord_fixed(xlim = c(-150, -100), ylim = c(-50, 0))
```

```
## Coordinate system already present. Adding new coordinate system,
```

Zoomed-in plot

Ignore the arrows to points off the map:

```
g2
```



Does that make sense?

- Get a Google map of the area, with the points labelled.
- First geocode the cities of interest: xxx

```
cities <- c(
  "Kitchener ON", "Hamilton ON", "Niagara Falls ON",
  "St Catharines ON", "Brantford ON"
)
latlong <- geocode(cities)
latlong <- bind_cols(city = cities, latlong) %>% print()
```

```
## # A tibble: 5 x 3
##   city               lon   lat
##   <chr>            <dbl> <dbl>
## 1 Kitchener ON     -80.5  43.5
## 2 Hamilton ON      -79.9  43.3
## 3 Niagara Falls ON -79.1  43.1
## 4 St Catharines ON -79.2  43.2
## 5 Brantford ON     -80.3  43.1
```

- Get a Google map of the area (experiment with zoom):

Making the Google map

Plot the map, plus the cities, plus labels for the cities:

```
ggmap(map) +
  geom_point(
    data = latlong,
    aes(x = lon, y = lat),
    shape = 3, colour = "red"
  ) +
  geom_text_repel(
    data = latlong,
    aes(label = city)
  ) -> gmap
```

\begin{frame}[frame]{The `mds` map and Google map}

“‘r g2 “‘

“‘r gmap “‘

![plot of chunk unnamed-chunk-

![plot of chunk unnamed-chunk-

Quality of fit

- Read in “southern Ontario” data set from file:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/southern-or
ontario2 <- read_csv(my_url)
```

- Calling cmdscale with eig=T gives more info: xxx

```
ontario2.2 <- ontario2 %>%
  select_if(is.numeric) %>%
  cmdscale(eig = T)
names(ontario2.2)
```

```
## [1] "points" "eig"    "x"      "ac"     "GOF"
```

```
ontario2.2$GOF
```

```
## [1] 0.8381590 0.8914059
```

```
ontario2.3 <- ontario2 %>%
  select_if(is.numeric) %>%
  cmdscale(3, eig = T)
ontario2.3$GOF
```

Comments

- Coordinates now in `points`.
- `GOF` is R-squared-like measure saying how well map distances match real ones. Higher is better.
- For Ontario road distances, `GOF` better for 3 dimensions than 2, presumably to accommodate St Catharines and Niagara Falls?

3-dimensional coordinates, cities attached xxx

```
ontario2.3$points %>%
  as_tibble() %>%
  mutate(city = ontario2$x)
```

```
## # A tibble: 19 x 4
##         V1       V2      V3 city
##      <dbl>    <dbl>   <dbl> <chr>
##  1  -38.7  122.       4.17 Barrie
##  2  146.   -82.8      1.53 Belleville
##  3 -132.   -38.9     14.1  Brantford
##  4  298.  -106.      -7.74 Brockville
##  5  397.  -104.     -22.0  Cornwall
##  6 -101.   -18.5     30.0  Hamilton
##  7   62.4  198.     -14.0  Huntsville
##  8  214.  -129.      10.8  Kingston
##  9 -123.   -15.0     -6.44 Kitchener
## 10 -208.   -51.6    -36.5  London
## 11 -129.   -19.1    155.   Niagara Falls
## 12  146.   300.     -25.4  North Bay
## 13  368.    -4.30   -47.2  Ottawa
## 14 -145.   125.     -16.0  Owen Sound
## 15   82.5    0.551   -6.92 Peterborough
## 16 -299.   -39.4    -72.5  Sarnia
```

RGL code for 3 dimensions

```
library(rgl)
plot3d(ontario.3)
text3d(ontario.3, text = d2$city)
```

\begin{frame}[fragile]{Comparing MDS solution with “reality”: Procrustes rotation}

- How to tell that an MDS map makes a good correspondence with “what should be”?
- Problem: MDS map might be rotated/scaled/reflected from reality.
- How to find rotation/scaling/reflection that best matches reality?
- Answer: **Procrustes rotation**.
- In R: `procOPA` in package `shapes`.

"True" coordinates

- Get latitudes and longitudes of cities by geocoding, as before. Glue "ON" onto city names to make sure we get right ones: xxx

```
lookup <- str_c(ontario2$x, " ON")
latlong <- geocode(lookup)
latlong <- bind_cols(city = ontario2$x, latlong) %>% print(n = 4)
```

xxx

```
## # A tibble: 19 x 3
##   city          lon   lat
##   <chr>       <dbl> <dbl>
## 1 Barrie      -79.7  44.4
## 2 Belleville -77.4  44.2
## 3 Brantford  -80.3  43.1
## 4 Brockville -75.7  44.6
## # … with 15 more rows
```

- Not (x, y) coordinates: one degree of latitude is always 110.25 km, but one degree of longitude is only that at the equator (less than that as

"True" coordinates part 2

- Make coordinates by multiplying by cosine of "typical" latitude.
- Find mean latitude:

```
m <- mean(latlong$lat)
m
```

```
## [1] 44.01851
```

- Turn into radians and find its cosine:

```
mult <- cos(m * pi / 180)
mult
```

```
## [1] 0.7191153
```

- Create "true" coords by multiplying the longitudes by that. This needs to be R `matrix`, not data frame: xxx

```
truecoord <- with(latlong, cbind(V1 = lon * mult, V2 = lat))
```

Using procOPA

- Feed 2 things into `procOPA`: first, “true” coordinates, second MDS coordinates.
- Get out:
- `(centred and scaled) first set of coordinates `Ahat``
- (centred and scaled) second set of coordinates `Bhat`
- sum of squared differences between two sets of coordinates `OSS`
- Rotation matrix `R`
- `Ahat` and `Bhat` coordinates supposed to match as well as possible. xxx

```
ontario.pro <- procOPA(
  truecoord,
  ontario2.2$points
)
names(ontario.pro)
```

Make data frames of output, glue together

- Two sets of coordinates, Ahat are actual, Bhat are from MDS. xxx

```
A <- ontario.pro$Ahat %>%
  as_tibble() %>%
  mutate(which = "actual", city = ontario2$x)
B <- ontario.pro$Bhat %>%
  as_tibble() %>%
  mutate(which = "MDS", city = ontario2$x)
dp <- bind_rows(A, B)
dp %>% sample_n(6)
```

```
## # A tibble: 6 x 4
##       V1     V2 which  city
##    <dbl>  <dbl> <chr>  <chr>
## 1  1.87   0.213 actual Kingston
## 2  0.556  0.298 MDS    Peterborough
## 3  2.45   0.571 actual Brockville
## 4 -0.848 -0.879 actual Brantford
## 5  2.39   0.348 MDS    Brockville
## 6 -0.284  1.56  MDS    Huntsville
```

procOPA, part 2: plotting

Procrustes rotation plot

- Strategy: plot all the locations, and colour them by whether they were the true location (red) or the MDS one (blue), which is in `which`. Label each location with the city name in the appropriate colour.
- I realized it was actually easy to join the two instances of a city by a line (in green, here, 3rd line) by setting `group=city`: xxx

```
g_opa <- ggplot(dp, aes(
  x = V1, y = V2, colour = which,
  label = city
)) + geom_point() +
  geom_line(aes(group = city), colour = "green") +
  geom_text_repel(size = 2)
```

- On plot, look to see whether points that are same city are joined by a short green line (good) or a long one (bad).

The maps

```
## Error in FUN(X[[i]], ...): object 'V1' not found
```

Comments

- True locations red, MDS locations blue
- Most things in roughly right place (esp. relative to other things)
- Extreme cities off by a bit, but OK relative to neighbours.
- St Catharines, Niagara Falls off by most.
- Sarnia, Windsor also off noticeably.
- These four cities had largest "third dimension" in 3D representation `ontario2.3`.

Rotation matrix

Shows how MDS map needs to be rotated to get best match with actual coordinates:

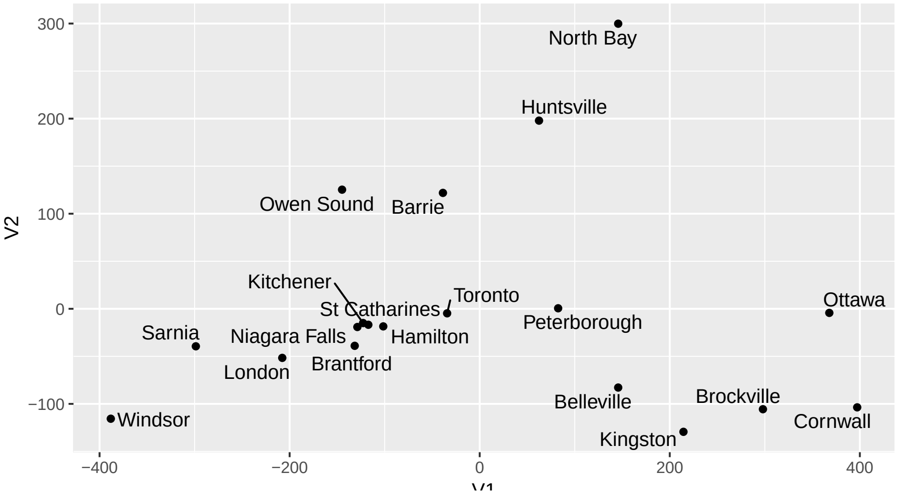
```
ontario.pro$R
```

```
##              [,1]      [,2]
## [1,]  0.8845749 0.4663981
## [2,] -0.4663981 0.8845749
```

Rotation angle θ such that $\cos\theta = 0.885$, $\sin\theta = 0.466$: $\theta = 23$ degrees (counterclockwise). $ %$ %$

Is that right? Look at MDS map again

```
g
```



A cube

```
a-----b
|\    |\
| c---- d
| |   | |
e-|---f |
\|    \|
g-----h
```

Cube has side length 1, so distance across diagonal on same face is $\sqrt{2} \simeq 1.4$ and "long" diagonal of cube is $\sqrt{3} \simeq 1.7$.

Try MDS on this obviously 3-dimensional data.

Cube data as distances xxx

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/cube.txt"
cube <- read_delim(my_url, " ")
cube
```

```
## # A tibble: 8 x 9
##   x     `  a` `  b` `  c` `  d` `  e` `   f` ` g`  `  h`
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr>  <chr> <chr>
## 1 a     "  0" " NA" " NA" " NA" " NA" "  NA" <NA>  " NA"
## 2 b     "  1" "  0" " NA" " NA" " NA" "  NA" <NA>  " NA"
## 3 c     "  1" "  1" "  0" " NA" " NA" "  NA" <NA>  " NA"
## 4 d     1.4   "  1" "  1" "  0" " NA" "  NA" <NA>  " NA"
## 5 e     "  1" 1.4   1.4   1.7   "  0" "  NA" <NA>  " NA"
## 6 f     1.4   "  1" 1.7   1.4   "  1" "   0" <NA>  " NA"
## 7 g     1.4   1.7   "  1" 1.4   "  1" " 1.4" " 0"  " NA"
## 8 h     1.7   1.4   1.4   "  1" 1.4   "   1" " 1"  "  0"
```

Making `dist` object

```
cube.d <- cube %>% select(-1) %>% as.dist()
```

```
## Warning in storage.mode(m) <- "numeric": NAs introduced by
```

```
cube.d
```

```
##         a   b   c   d   e    f   g
##   b   1.0
##   c   1.0 1.0
##   d   1.4 1.0 1.0
##   e   1.0 1.4 1.4 1.7
##    f  1.4 1.0 1.7 1.4 1.0
##  g    1.4 1.7 1.0 1.4 1.0  1.4
##   h   1.7 1.4 1.4 1.0 1.4  1.0 1.0
```

MDS and plotting commands

- By default in 2 dimensions; save the extra stuff for later:

```
cube.2 <- cube.d %>% cmdscale(eig = T)
```

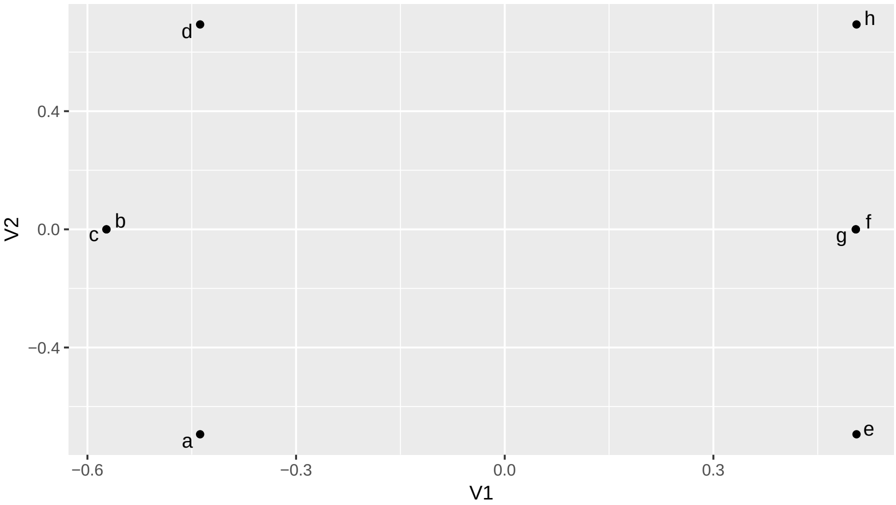
- Make data frame to plot, remembering the points to plot are in `points` now:

```
d <- cube.2$points %>%
  as_tibble() %>%
  mutate(corners = cube$x)
```

- Plot points labelled by our names for the corners:

```
g <- ggplot(d, aes(x = V1, y = V2, label = corners)) +
  geom_point() + geom_text_repel()
```

The “cube”



2 and 3 dimensions

```
cube.3 <- cube.d %>% cmdscale(3, eig = T)
cube.2$GOF
```

```
## [1] 0.639293 0.664332
```

```
cube.3$GOF
```

```
## [1] 0.9143532 0.9501654
```

- Really need 3rd dimension to represent cube.

Non-metric scaling

- Sometimes distances not meaningful *as distances*
- Only order matters: closest should be closest, farthest farthest on map, but how much further doesn't matter.
- Non-metric scaling, aims to minimize **stress**, measure of lack of fit.
- Example: languages. Make map based on "similarity" of number names, without requiring that 1 is "eight times better" than 8.

The languages

- Recall language data (from cluster analysis): 1–10, measure dissimilarity between two languages by how many number names *differ* xxx in first letter:

xxx

```
my_url <- "http://www.utsc.utoronto.ca/~butler/d29/languages.txt"
number.d <- read_table(my_url)
number.d
```

```
## # A tibble: 11 x 12
##    la       en    no    dk    nl    de    fr    es
##    <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 en        0     2     2     7     6     6     6
##  2 no        2     0     1     5     4     6     6
##  3 dk        2     1     0     6     5     6     5
##  4 nl        7     5     6     0     5     9     9
##  5 de        6     4     5     5     0     7     7
##  6 fr        6     6     6     9     7     0     2
##  7 es        6     6     5     9     7     2     0
##  8 it        6     6     5     9     7     1     1
##  9 pl        7     7     6    10     8     5     3
```

Non-metric scaling

- Turn language dissimilarities into `dist` object
- Run through `isoMDS` from `MASS` package; works like `cmdscale`.
- Map only reproduces *relative* xxx closeness of languages. xxx

```
d <- number.d %>%
  select_if(is.numeric) %>%
  as.dist()
number.nm <- d %>% isoMDS()
```

```
## initial  value 12.404671
## iter   5 value 5.933653
## iter  10 value 5.300747
## final  value 5.265236
## converged
```

```
names(number.nm)
```

```
## [1] "points" "stress"
```

Results

- Stress is very low (5%, good):

```
number.nm$stress
```

```
## [1] 5.265236
```

$ %$ %$

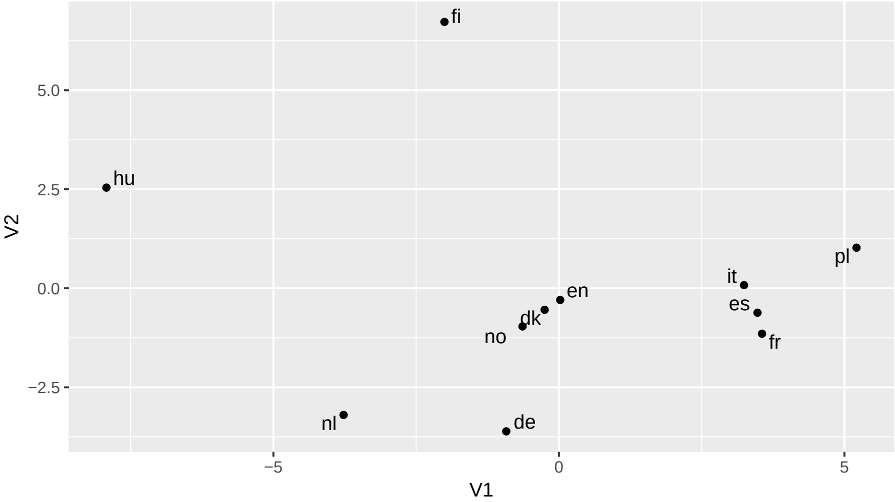
- Familiar process: make a data frame to plot. Use name `dd` for data frame this time since used `d` for distance object:

```
dd <- number.nm$points %>%
  as_tibble() %>%
  mutate(lang = number.d$la)
```

- Make plot:

```
g <- ggplot(dd, aes(x = V1, y = V2, label = lang)) +
  geom_point() + geom_text_repel()
```

The languages map



Comments

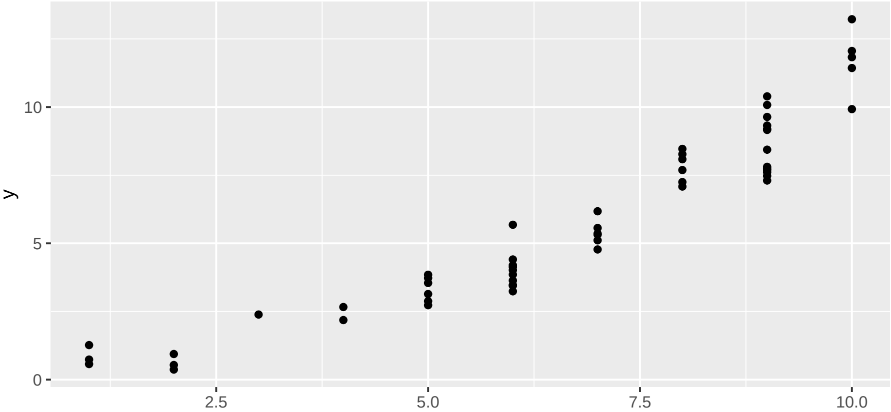
- Tight clusters: Italian-Spanish-French, English-Danish-Norwegian.
- Dutch and German close to English group.
- Polish close to French group.
- Hungarian, Finnish distant from everything else and each other!
- Similar conclusions as from the cluster analysis.

Shepard diagram

- Stress for languages data was 5.3%, very low.
- How do observed dissimilarities and map distances correspond?
- For low stress, expect larger dissimilarity to go with larger map distance, almost all the time.
- Not necessarily a linear trend since non-metric MDS works with *order* of values.
- Actual dissimilarity on x-axis; map distances on y-axis.

Shepard diagram for languages

```
Shepard(d, number.nm$points) %>%
  as_tibble() %>%
  ggplot(aes(x = x, y = y)) + geom_point()
```



Cube, revisited xxx

```
cube.d <- cube %>% select(-x) %>% as.dist(cube)
```

```
## Warning in storage.mode(m) <- "numeric": NAs introduced
## by coercion
```

```
cube.2 <- isoMDS(cube.d, trace = F)
cube.2$stress
```

```
## [1] 17.97392
```

```
cube.3 <- isoMDS(cube.d, k = 3, trace = F)
cube.3$stress
```

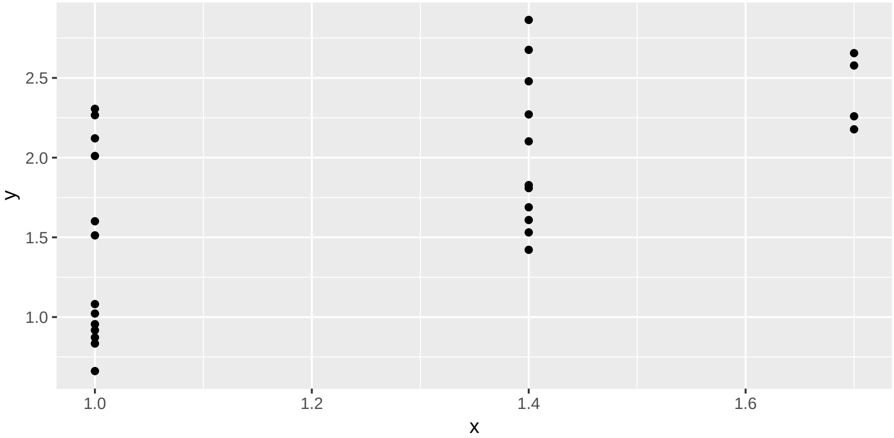
```
## [1] 0.007819523
```

- Stress is 18% for 2 dimensions, basically 0% for 3.
- Three dimensions correct, two dimensions bad.
- Shepard diagrams for these: xxx

```
cube2.sh <- Shepard(cube.d, cube.2$points)
```

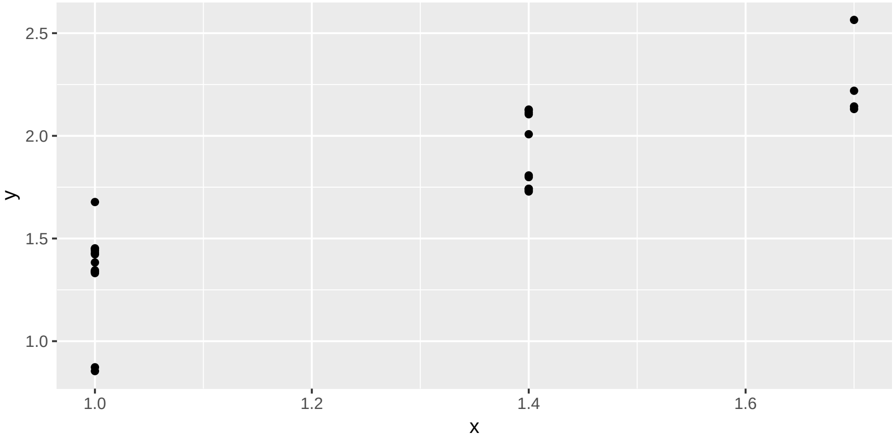
Shepard diagram for 2-dimensional cube

```
g2
```



Shepard diagram for 3-dimensional cube

```
g3
```



Guidelines for stress values, in %

Smaller is better:

Stress value	Interpretation
Less than 5	Excellent: no prospect of misinterpretation (rarely achieved)
5–10	Good: most distances reproduced well, small prospect of false inferences
10–20	Fair: usable, but some distances misleading.
More than 20	Poor: may be dangerous to interpret

- Languages: stress in "good" range.
- Cube: xxx
  - 2 dimensions "fair", almost "poor";
  - 3 dimensions, "excellent".