# Lecture 14a

# Multi-Dimensional Scaling – Part II

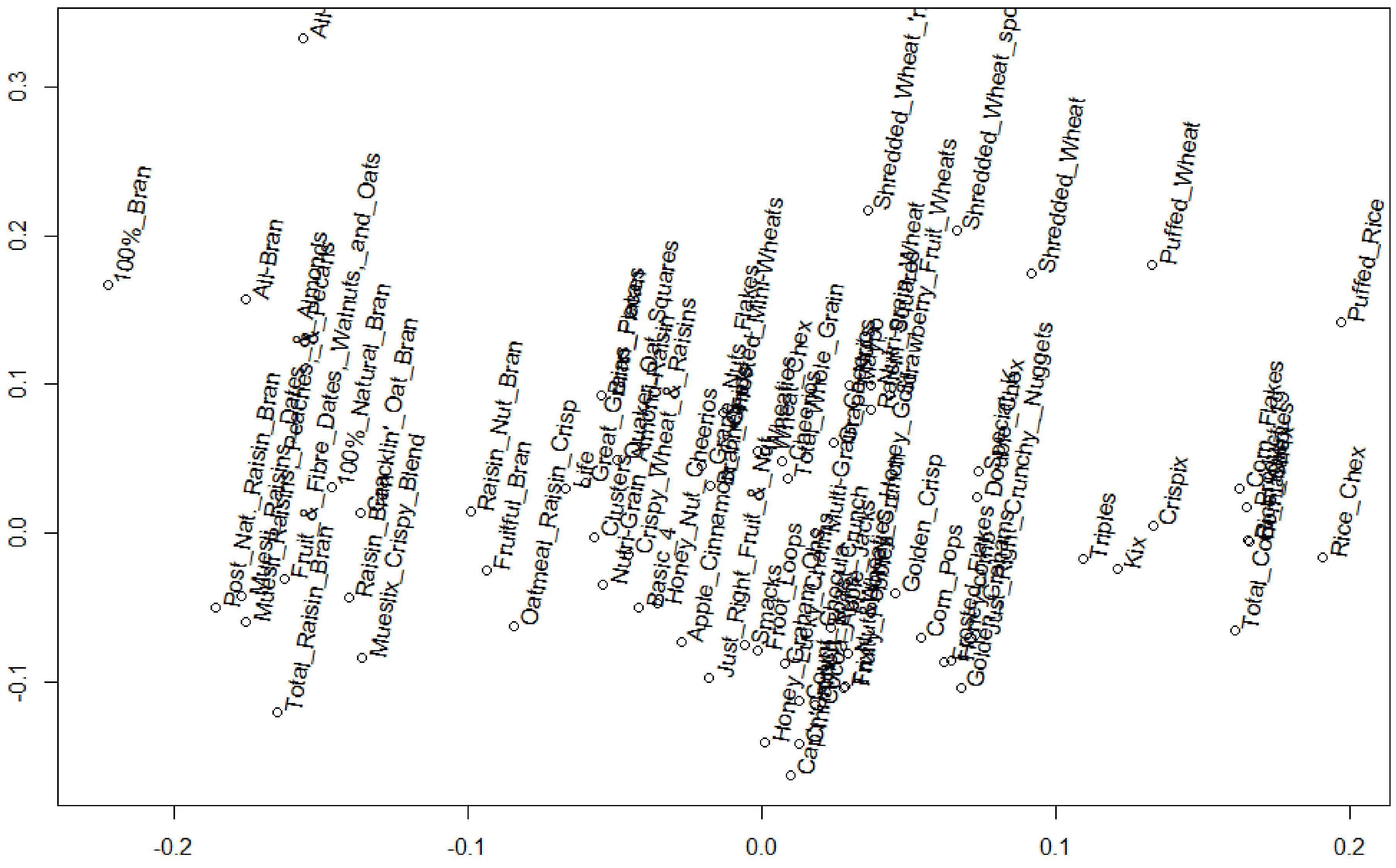
Breitzman 12/5/2018

## Introduction

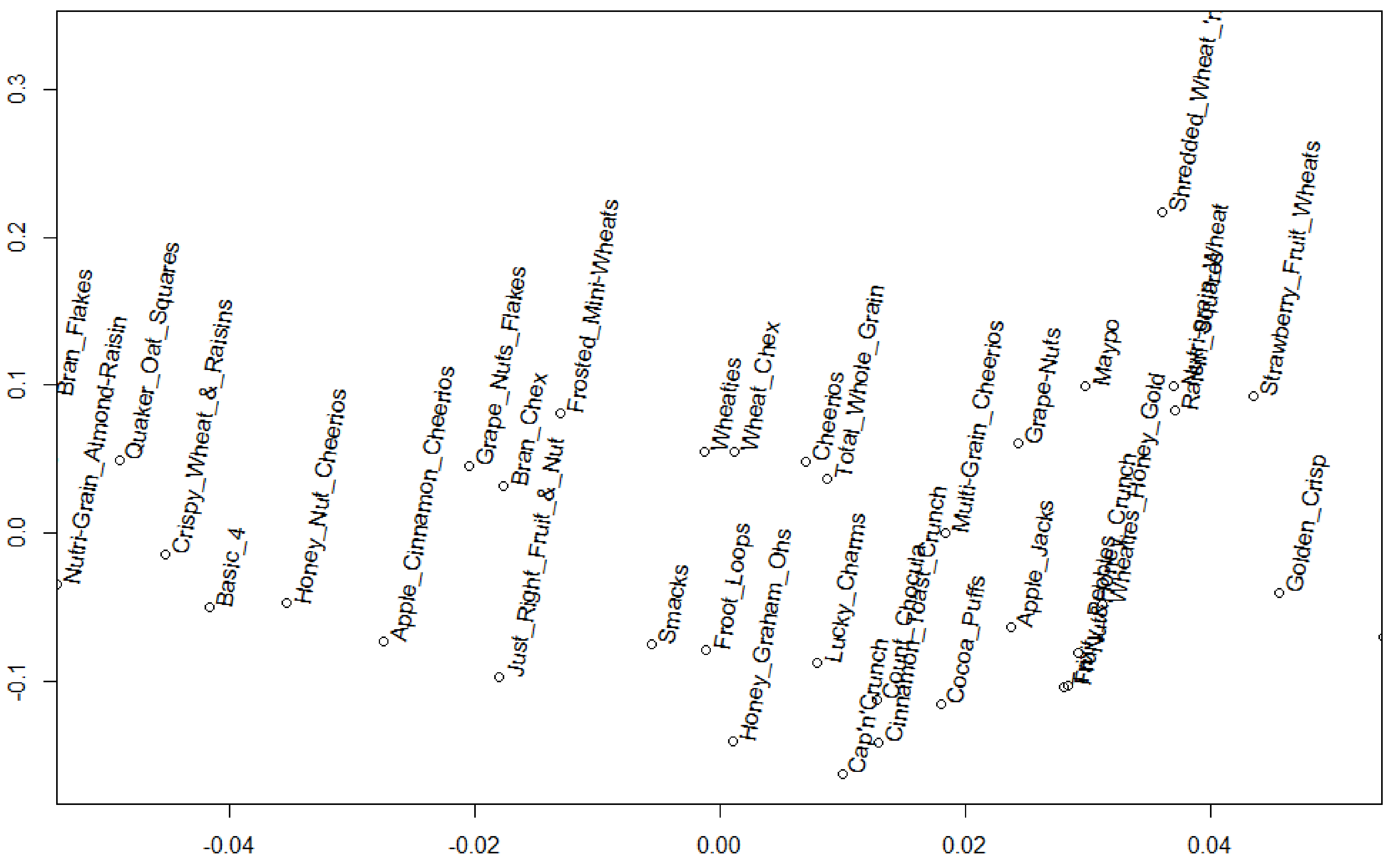
- A couple of follow-up items to last week

- Recall: Multidimensional scaling takes a distance matrix and attempts to plot it in fewere dimensions while preserving pairwise distances as best as it can

# We Showed this Plot Last Week



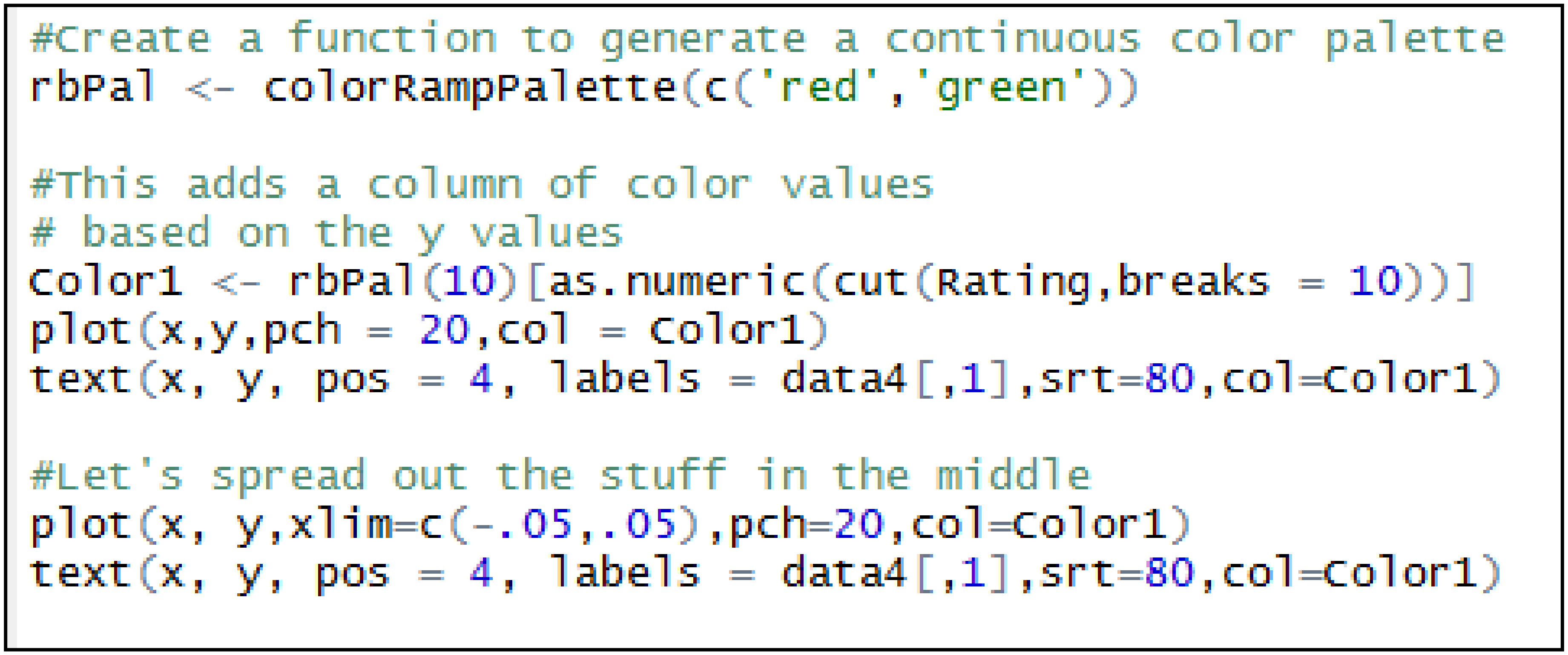
# And This Zoom in to the middle



## Color

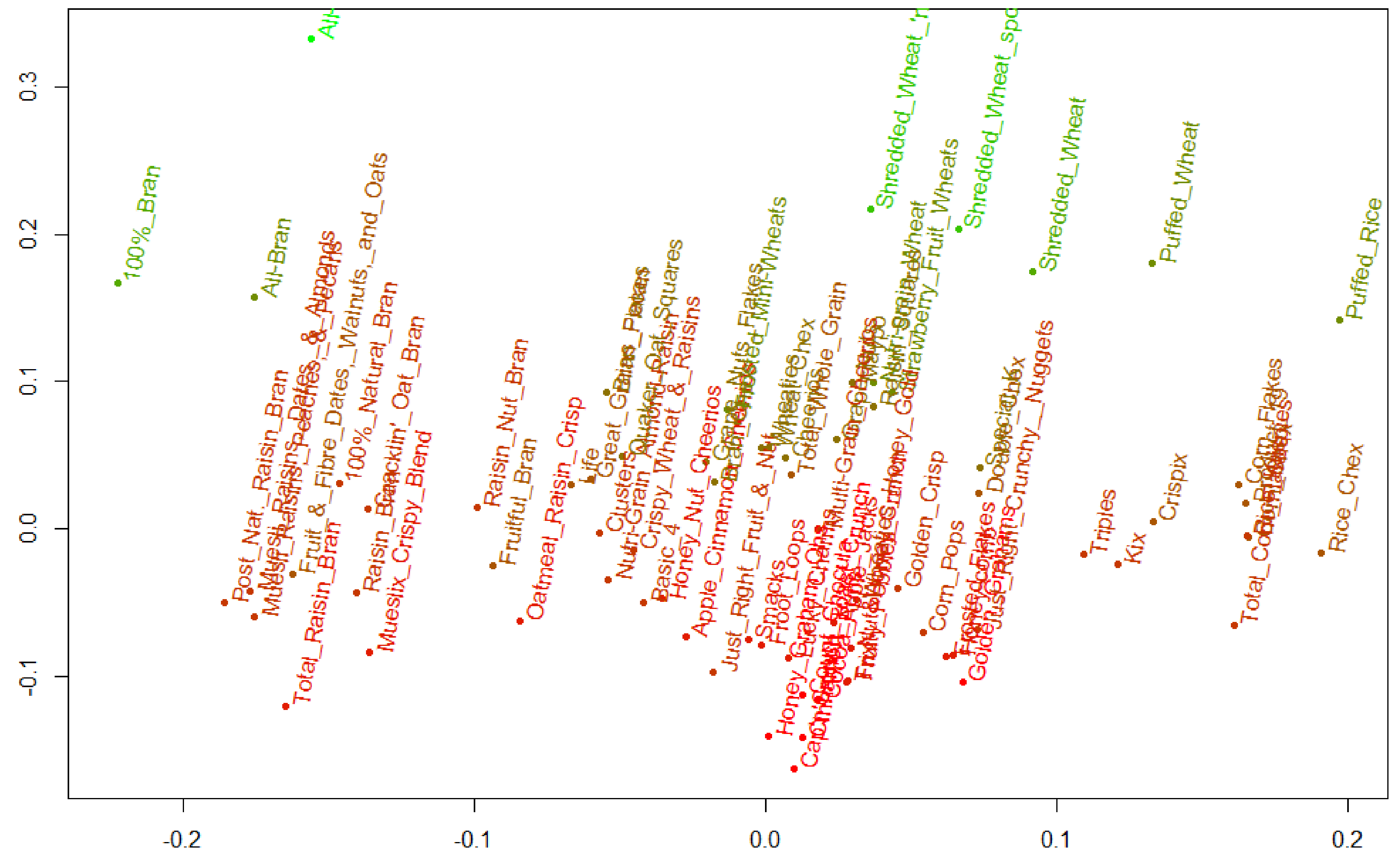
- It occurred to me later, that those plots would be better if we colored them based on rating
- Then we could see whether there was any hope of creating a model to predict rating
- Here’s how…

## Color (II)

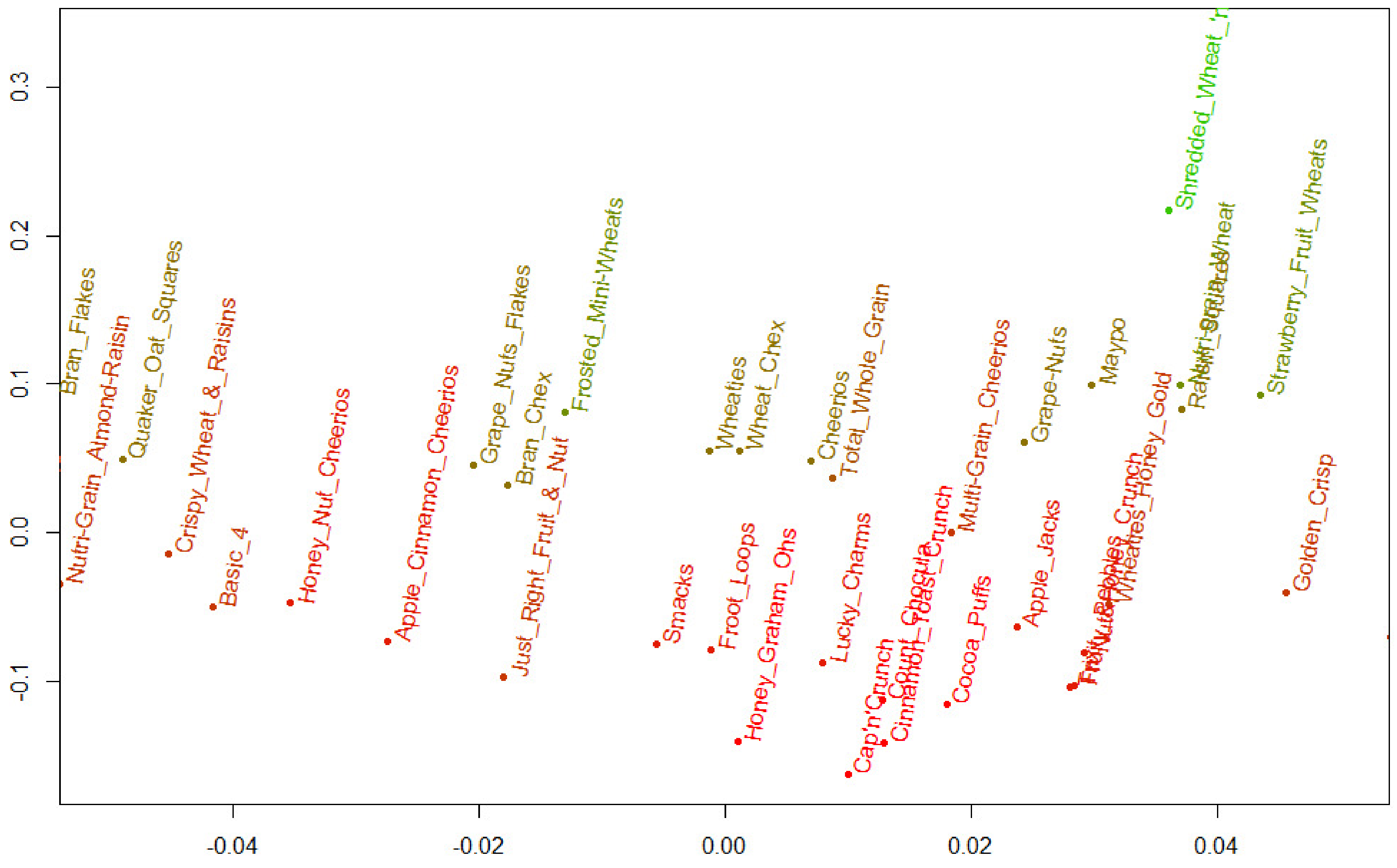


```
#Create a function to generate a continuous color palette
rbPal <- colorRampPalette(c('red','green'))

#This adds a column of color values
# based on the y values
Color1 <- rbPal(10)[as.numeric(cut(Rating,breaks = 10))]
plot(x,y,pch = 20,col = Color1)
text(x, y, pos = 4, labels = data4[,1],srt=80,col=Color1)

#Let's spread out the stuff in the middle
plot(x, y,xlim=c(-.05,.05),pch=20,col=Color1)
text(x, y, pos = 4, labels = data4[,1],srt=80,col=Color1)
```

# Color (III)


100%_Bran
All-Bran
Post_Nat._Raisin_Bran
Total_Raisin_Bran
Fruit_&_Fibre_Dates,_Walnuts,_and_Oats
100%_Natural_Bran
Raisin_Bran
Mueslix_Crispy_Blend
Raisin_Nut_Bran
Fruitful_Bran
Oatmeal_Raisin_Crisp
Life
Great_Grains_Pecan
Clusters
Nutri-Grain_Almond-Raisin
Crispy_Wheat_&_Raisins
Quaker_Oat_Squares
Basic_4
Honey_Nut_Cheerios
Apple_Cinnamon_Cheerios
Just_Right_Fruit_&_Nut
Smacks
Froot_Loops
Wheaties
Total_Whole_Grain
Golden_Crisp
Corn_Pops
Frosted_Flakes
Just_Right_Crunchy__Nuggets
Shredded_Wheat
Strawberry_Fruit_Wheats
Puffed_Wheat
Puffed_Rice
Triples
Kix
Crispix
Corn_Flakes
Total_Corn_Flakes
Rice_Chex
-0.2
-0.1
0.0
0.1
0.2
0.3

# Zoom-In


Bran_Flakes
Nutri-Grain_Almond-Raisin
Quaker_Oat_Squares
Crispy_Wheat_&_Raisins
Basic_4
Honey_Nut_Cheerios
Apple_Cinnamon_Cheerios
Grape_Nuts_Flakes
Bran_Chex
Just_Right_Fruit_&_Nut
Frosted_Mini-Wheats
Smacks
Froot_Loops
Honey_Graham_Ohs
Wheaties
Wheat_Chex
Cheerios
Total_Whole_Grain
Lucky_Charms
Cap'n'Crunch
Count_Chocula
Cinnamon_Toast_Crunch
Multi-Grain_Cheerios
Cocoa_Puffs
Apple_Jacks
Grape-Nuts
Maypo
Wheaties_Honey_Gold
Shredded_Wheat
Strawberry_Fruit_Wheats
Golden_Crisp
0.3
0.2
0.1
0.0
-0.1
-0.04
-0.02
0.00
0.02
0.04

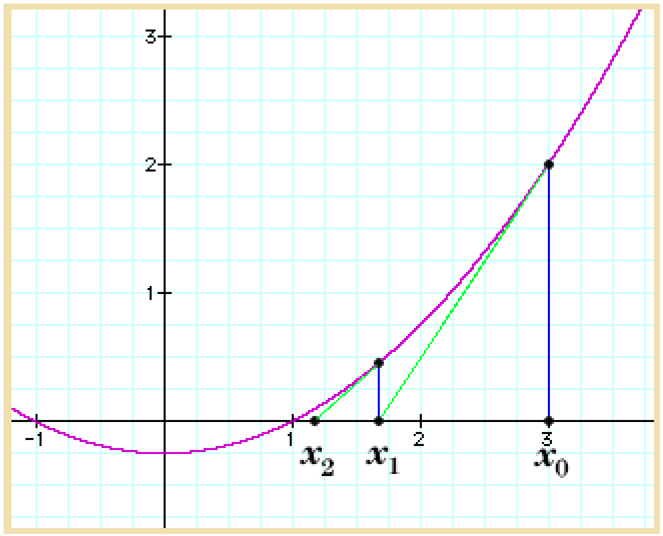
## Python-SciKit

- The other thing I forgot to mention is how to do MDS in Python
- sklearn contains function sklearn.manifold.MDS, and can be used via Orange

## Computations

- Also last week, Ron seemed kind of disappointed that we didn't do any MDS computations, so here is a short version of the Math
- Recall Newton's Method allows us to numerically find the roots of a function

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



## Computations (II)

- So we learned in Calc 1 that Newton’s method could be used to find zeros of a function as shown in previous slide
- We also learned in Calc I that to find local mins and maxs we need to find when the first derivative is 0
- It follows that if we use the method of the previous page we can find zeros of the derivative simply by making the numerator a first derivative and the denominator a second derivative
- When we do this it is called Newton Optimization instead of Newton root finding
- This tends to be faster than gradient descent which is another way to find local minimum that we’ve talked about and you will see in Machine Learning

PSYCHOMETRIKA—VOL. 29, NO. 2
JUNE, 1964

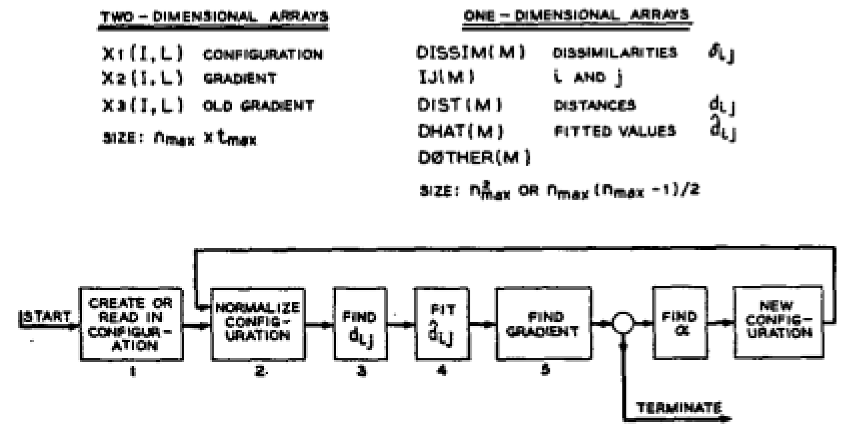
# NONMETRIC MULTIDIMENSIONAL SCALING: A NUMERICAL METHOD

J. B. KRUSKAL
BELL TELEPHONE LABORATORIES

We describe the numerical methods required in our approach to multidimensional scaling. The rationale of this approach has appeared previously.

## 7. *Programming Technique*

Since the procedure described in this paper is entirely impractical without the aid of an automatic computer, it seems desirable to describe the procedure in sufficient detail that an experienced programmer can easily program it.



## Computations (III)

- Translating the paper, here are the key steps

1. Pick Random points in the dimension desired for each of your original data points
2. Create a distance matrix for the assigned points ($\hat{D}$)
3. Compare to original (desired) distance matrix ($D$)
4. Minimize $f = \sum | D_{i,j} - \hat{D}_{i,j} |$ using Newton or Gradient Descent
5. Note it's possible for Newton to get stuck in local minimums so repeat a couple of times with new random points

## Go to Excel

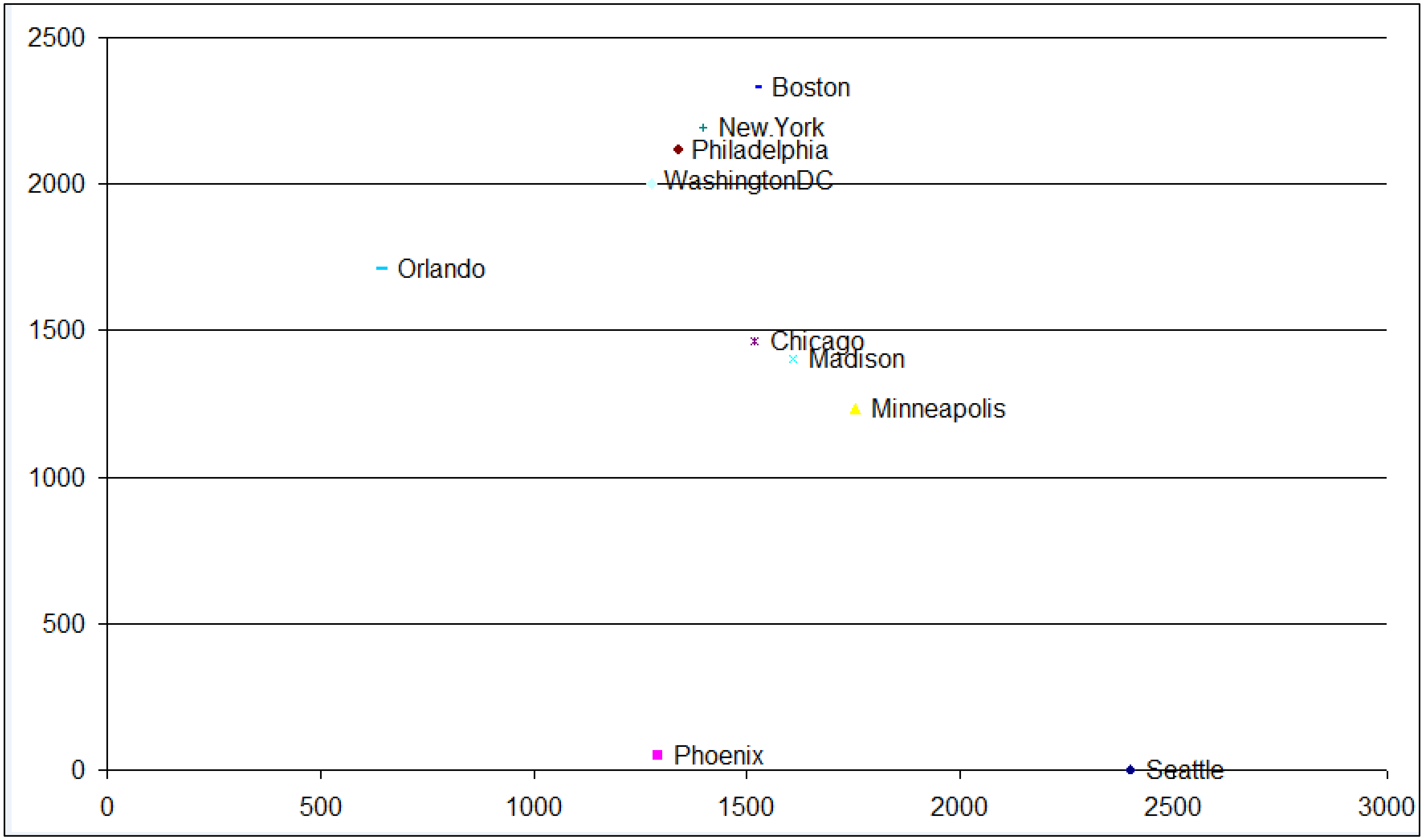
- We'll use solver, because computing Newton by hand can be a pain.
- I make my students do it exactly once in Calc I and that's enough
- Don't forget to come back here after Excel!

## From Excel (Difference Matrix)

| Difference Matrix | Seattle | Phoenix | Minneapoli | Madison | Chicago | Philadelphi | New.York | Boston | Orlando | WashingtonDC |
|---|---|---|---|---|---|---|---|---|---|---|
| Seattle | 0.00 | 0.00 | 1.63 | 6.77 | 1.23 | 1.25 | 1.95 | 1.28 | 0.00 | 4.79 |
| Phoenix | 0.00 | 0.00 | 0.64 | 0.01 | 6.67 | 1.42 | 5.19 | 0.01 | 19.41 | 0.04 |
| Minneapolis | 1.63 | 0.64 | 0.00 | 0.00 | 0.08 | 0.65 | 4.76 | 0.01 | 0.98 | 0.24 |
| Madison | 6.77 | 0.01 | 0.00 | 0.00 | 0.17 | 0.00 | 0.91 | 0.24 | 0.53 | 0.00 |
| Chicago | 1.23 | 6.67 | 0.08 | 0.17 | 0.00 | 0.13 | 0.00 | 0.09 | 2.69 | 0.06 |
| Philadelphia | 1.25 | 1.42 | 0.65 | 0.00 | 0.13 | 0.00 | 0.52 | 0.09 | 1.00 | 0.01 |
| New.York | 1.95 | 5.19 | 4.76 | 0.91 | 0.00 | 0.52 | 0.00 | 0.00 | 0.90 | 0.00 |
| Boston | 1.28 | 0.01 | 0.01 | 0.24 | 0.09 | 0.09 | 0.00 | 0.00 | 3.81 | 0.15 |
| Orlando | 0.00 | 19.41 | 0.98 | 0.53 | 2.69 | 1.00 | 0.90 | 3.81 | 0.00 | 0.00 |
| WashingtonDC | 4.79 | 0.04 | 0.24 | 0.00 | 0.06 | 0.01 | 0.00 | 0.15 | 0.00 | 0.00 |
| | | | | | | | | | | |
| | | | | | | | | | | |
| Total Error | 140.5808 | | | | | | | | | |

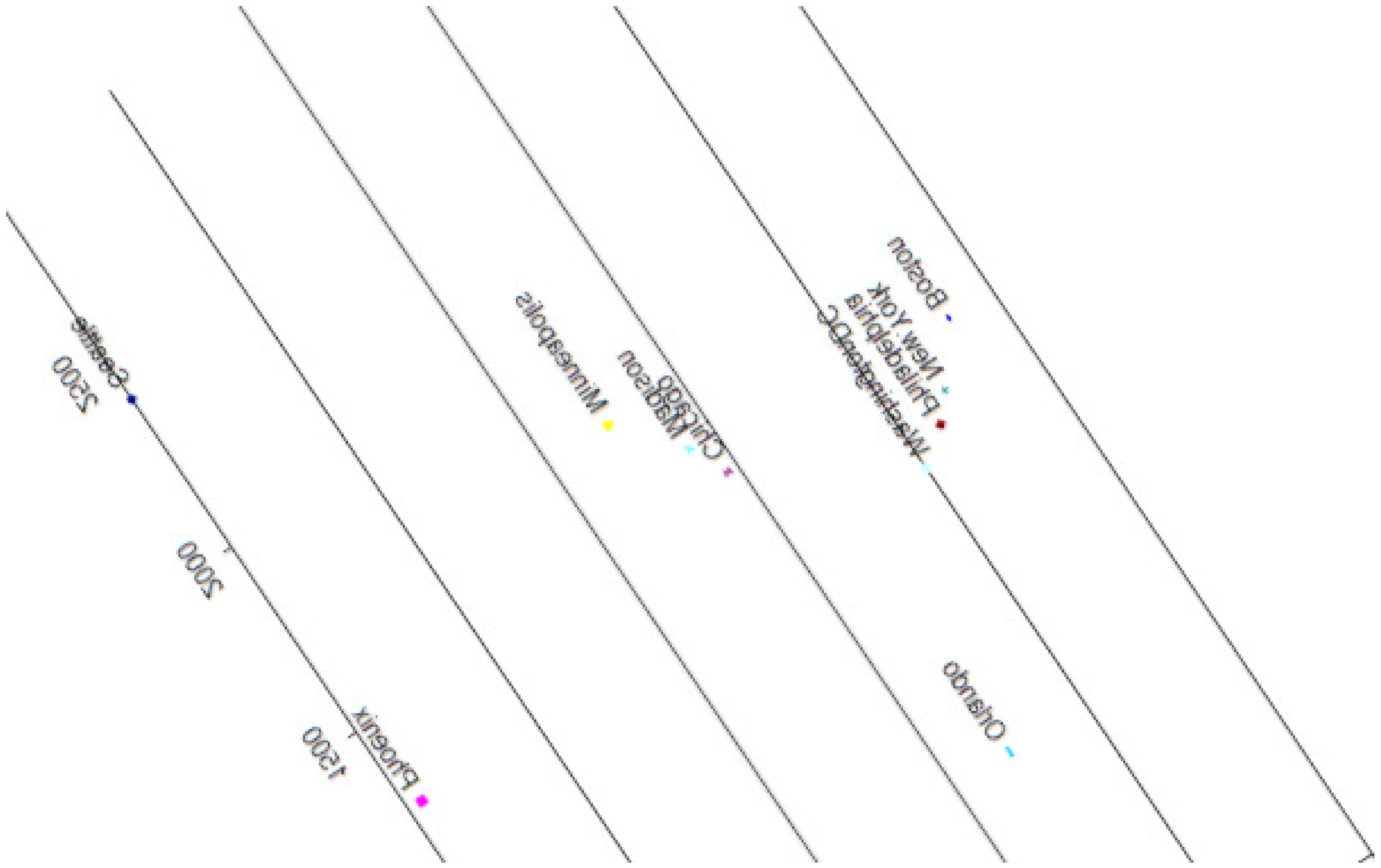
- Off by a total of 140 miles
- Most off by a few miles.  Worst is Phoenix, Orlando (19.41)
- Probably something to do with curvature of the earth

## Best Result from Solver (Error = 140 Miles)



- Actually pretty good.  Needs to be flipped and rotated to look right

## Flip and Rotate



## Summary

- Filled in some details left out of last week
- Use R or Python, but can be done in Excel or by hand if you wanted
- Note: If Error cannot be brought down no matter what, then increase dimension
- For example if we have 100 starting attributes and error is huge in 2 dimensions then maybe we can bring it down by going to 3 or 4 or 5 dimensions
- Note even if we can’t bring it down to 2 or 3 dimensions to visualize it, if we can mimic the distance matrix fewer than the original dimensions, this can sometimes be as effective as PCA