Spectral Clustering Notes

Much of this uses information from A Tutorial on Spectral Clustering by Ulrike von Luxburg and Proximity in Statistical Machine Learning by Michael Trosset.

The Ratio Cut Problem

Ratio Cut for $k = 2$

It can be shown that in the relaxed case for $k = 2$, minimizing:

$$W(k) = \sum_{i=1}^{k}(x_i - m_1)^2 + \sum_{i=k+1}^{n}(x_i - m_2)^2$$

where m_1 and m_2 are k-means centers, as perscribed by Luxburg results in the same clustering as by assigning clusters by minimizing

$$R(k) = \sum_{i=1}^{k}\left(x_i + \sqrt{\frac{n-k}{k}}\right)^2 + \sum_{i=k+1}^{n}\left(x_i - \sqrt{\frac{k}{n-k}}\right)^2$$

in $\mathbb{R}^1$ and where x_is are ordered, i.e., $x_i \leq \cdots \leq x_n$. We also constrain the problem to $\sum_i^n x_i = 0$ and $\sum_i^n x_i^2 = n$.

The k-means centers in $\mathbb{R}$ are

$$m_1 = \frac{1}{k}\sum_{i=1}^{k} x_i$$

$$m_2 = \frac{1}{n-k}\sum_{i=k+1}^{n} x_i$$

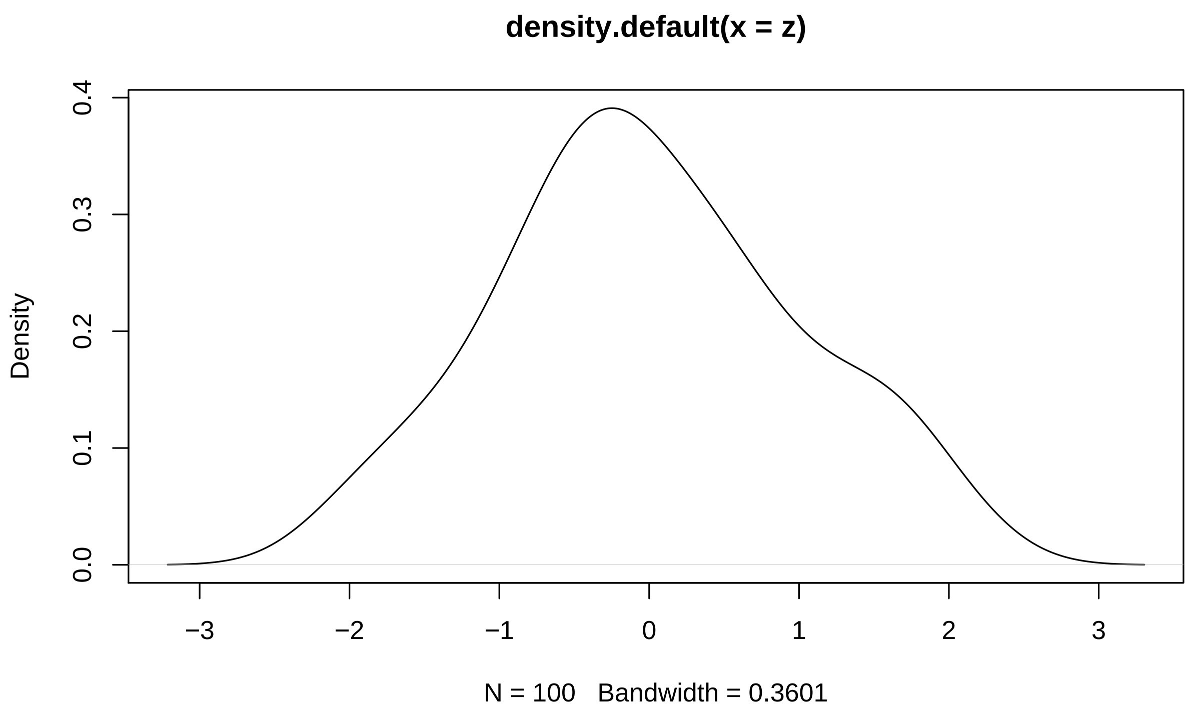
Note that m_1 and m_2 are functions of k.

Numerical results

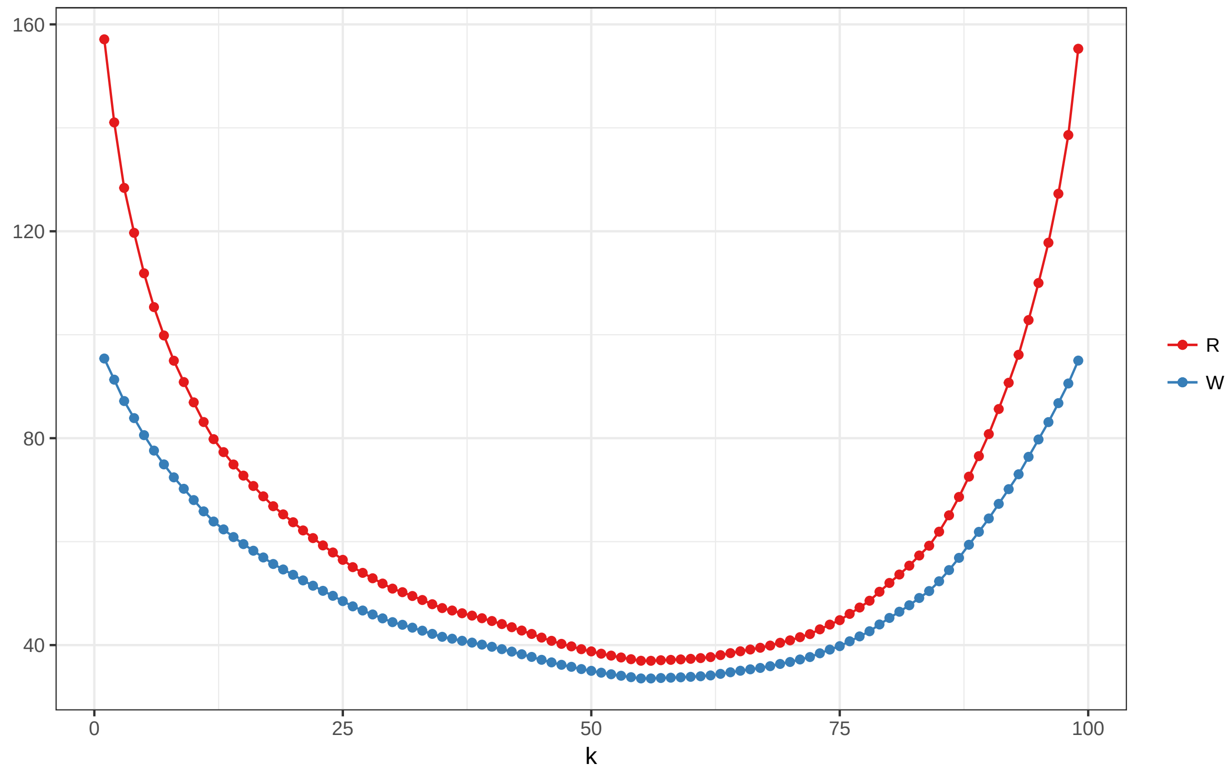
Using an arbitrary vector $\overrightarrow{x} \in \mathbb{R}^n$ such that $\sum_i x_i = 0$ and $|\overrightarrow{x}|_2^2 = n$:

```
# packages
library(ggplot2)
import::from(magrittr, `%>%`, `%<>%`)
theme_set(theme_bw())
import::from(psych, tr)
library(qgraph)

# --- functions --- #

normalize <- function(x) {
```

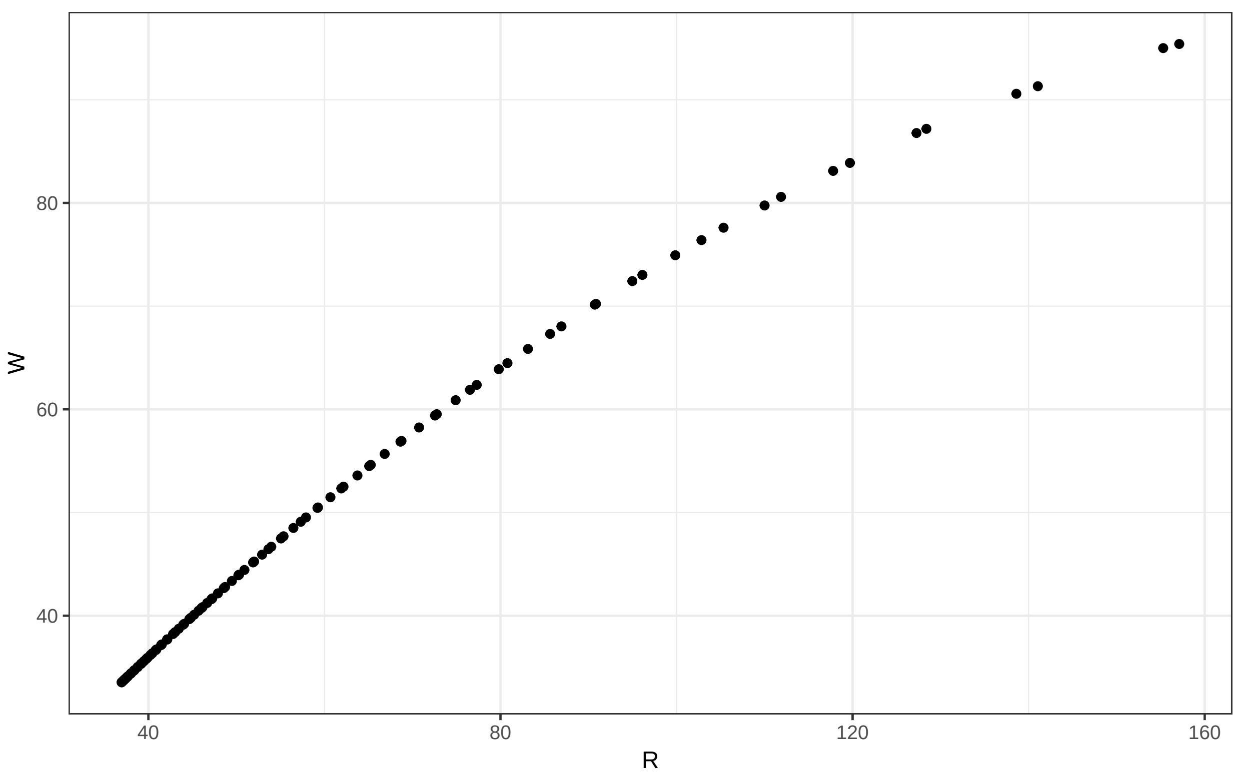
```r
  y <- x - mean(x)
  z <- y / sqrt(mean(y ** 2))
  return(z)
}

k.means <- function(x) {
  x <- sort(x)
  n <- length(x)

  sapply(seq(n - 1), function(k) {
    m1 <- 1 / k * sum(x[seq(k)])
    m2 <- 1 / (n - k) * sum(x[seq(k + 1, n)])

    W <- sum((x[seq(k)] - m1)^2) + sum((x[seq(k + 1, n)] - m2)^2)
    return(W)
  })
}

ratio.cut <- function(x) {
  x <- sort(x)
  n <- length(x)

  sapply(seq(n - 1), function(k) {
    Ac <- sqrt((n - k) / k)
    A <- sqrt(k / (n - k))

    R <- sum((x[seq(k)] + Ac)^2) + sum((x[seq(k + 1, n)] - A)^2)
    return(R)
  })
}

# generate random data
n <- 100
k <- 4
x <- c(rnorm(n / k, -1),
       rnorm(n / k, 0),
       rnorm(n / k, 1),
       rnorm(n / k, 3))
z <- normalize(x)
plot(density(z))
```




```r
# compute W and R
W <- k.means(z)
R <- ratio.cut(z)

# visualizations
k <- seq(n - 1)

ggplot() +
  geom_point(aes(x = k, y = W, colour = 'W')) +
  geom_line(aes(x = k, y = W, colour = 'W')) +
  geom_point(aes(x = k, y = R, colour = 'R')) +
  geom_line(aes(x = k, y = R, colour = 'R')) +
  scale_colour_brewer(palette = 'Set1') +
  labs(x = 'k', y = NULL, colour = NULL)
```




```r
ggplot() +
  geom_point(aes(x = R, y = W))
```



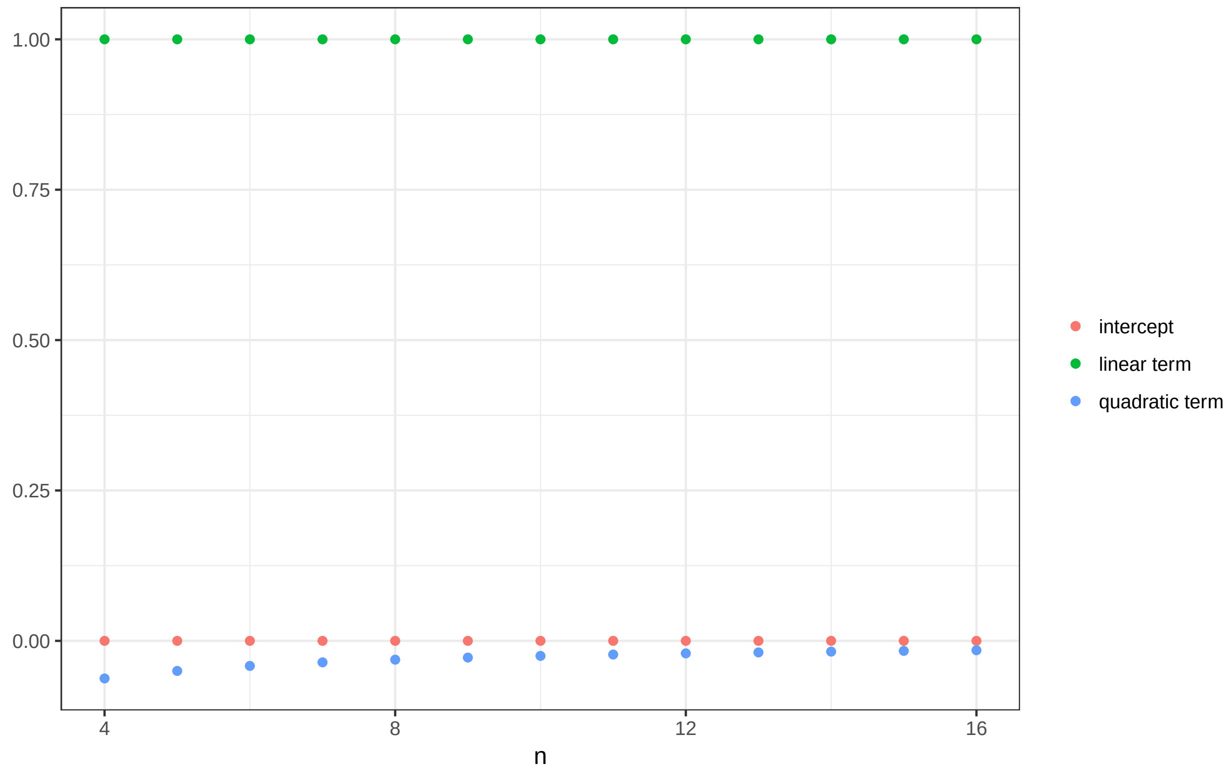
It appears that there is a definite relationship between W and R. Using quadratic regression:

```
summary(lm(W ~ R + I(R ** 2)))
```

```
Call:
lm(formula = W ~ R + I(R^2))

Residuals:
       Min         1Q     Median         3Q        Max 
-6.104e-14 -3.515e-15  5.300e-17  3.618e-15  8.500e-14 

Coefficients:
              Estimate Std. Error    t value Pr(>|t|)    
(Intercept) -1.143e-14  7.614e-15 -1.501e+00    0.137    
R            1.000e+00  2.101e-16  4.759e+15   <2e-16 ***
I(R^2)      -2.500e-03  1.226e-18 -2.040e+15   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.176e-14 on 96 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:      1 
F-statistic: 1.031e+32 on 2 and 96 DF,  p-value: < 2.2e-16
```

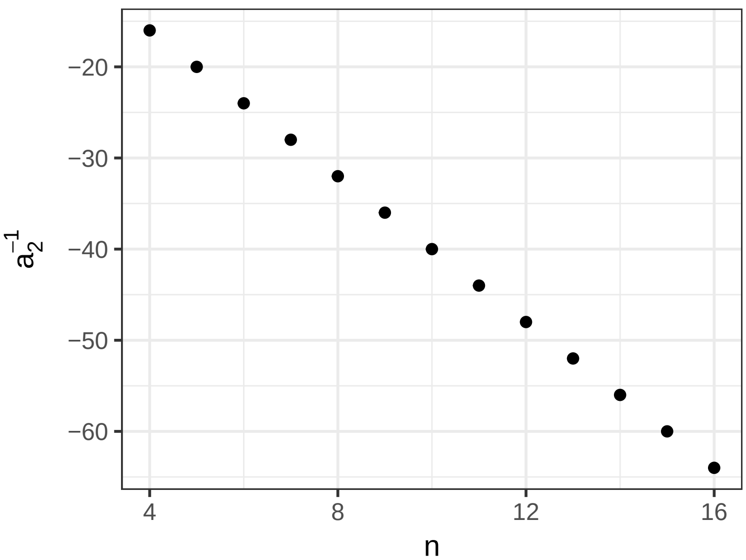
… we get a perfect fit. We can try several values of the data size n:

```
N <- 2 ** 4  # number of obs to try
coefs.df <- lapply(seq(4, N), function(n) {
  x <- normalize(rnorm(n))  # generate data

  # compute results
  W <- k.means(x)
  R <- ratio.cut(x)

  # compute coefs for quadratic equation
  quad.coefs <- lm(W ~ R + I(R ** 2))$coefficients
  a0 <- quad.coefs['(Intercept)']
  a1 <- quad.coefs['R']
  a2 <- quad.coefs['I(R^2)']

  # compile into data frame
  dplyr::data_frame(n, a0, a1, a2)
}) %>% 
  dplyr::bind_rows()

ggplot(coefs.df) + 
  geom_point(aes(x = n, y = a0, colour = 'intercept')) + 
  geom_point(aes(x = n, y = a1, colour = 'linear term')) + 
  geom_point(aes(x = n, y = a2, colour = 'quadratic term')) + 
  labs(colour = NULL, y = NULL)
```



The intercept term stays at 0 and the linear term stays at 1. Looking closer at the quadratic term:

```
ggplot(coefs.df) +
  geom_point(aes(x = n, y = a2 ** -1)) +
  labs(y = expression(a[2]^-1))
```



Then we arrive at the result $W = R - \frac{1}{4n}R^2$.

Analytic result

We can show:

$$W(k) = R(k) - \frac{(R(k))^2}{4n}$$

or $W(R) = R - \frac{1}{4n}R^2$. This function is strictly increasing for $R(k) \leq 2n$. Expanding $R(k)$, we get:

$$R(k) = 2n + 2\sqrt{\frac{n-k}{k}}\sum_{i=1}^{k} x_i - 2\sqrt{\frac{k}{n-k}}\sum_{i=k+1}^{n} x_i$$

It can be shown that $R(k)$ is maximized at the endpoints.

Note that since $\sum_i^n x_i = 0$, $\sum_i^{k<n} x_i \leq 0$, $x_n \geq 0$, and $x_i \leq 0$.

k can range from 1 to $n-1$. If $k = n-1$, we get $R(n-1) = 2n + 2\sqrt{\frac{1}{k}}\sum_i^{n-1} x_i - 2\sqrt{n-1}x_n$. The second term is ≤ 0 and the third term is ≥ 0, so we get $R \leq 2n$. On the other hand, if $k = 1$, $R(1) = 2n + 2\sqrt{n-1}x_1 - \frac{2}{\sqrt{n-1}}\sum_{i=2}^n x_i = 2n + 2\sqrt{n-1}x_1 - \frac{2}{\sqrt{n-1}}(-x_1) = 2n + x_1\big(2\sqrt{n-1} + \frac{2}{\sqrt{n-1}}\big) \leq 2n$.

Expanding $W(k)$, we get:

$$\begin{aligned}
W(k) &= \sum_{i=1}^{k} x_i^2 - 2m_1\sum_{i=1}^{k} x_i + km_1^2 + \sum_{i=k+1}^{n} x_i^2 - 2m_2\sum_{i=k+1}^{n} x_i + m_2^2(n-k) \\
&= n - \frac{2}{k}\left(\sum_{i=1}^{k} x_i\right)^2 + \frac{1}{k}\left(\sum_{i=1}^{k} x_i\right)^2 - \frac{2}{n-k}\left(\sum_{i=k+1}^{n} x_i\right)^2 + \frac{1}{n-k}\left(\sum_{i=k+1}^{n} x_i\right)^2 \\
&= n - \frac{1}{k}\left(\sum_{i=1}^{k} x_i\right)^2 - \frac{1}{n-k}\left(\sum_{i=k+1}^{n} x_i\right)^2 \\
&= n - km_1^2 - (n-k)m_2^2
\end{aligned}$$

Since $\sum_i^n x_i = 0$, $km_1 + (n-k)m_2 = 0$, or, $-nm_2 = k(m_1 - m_2)$. Then

$$\begin{aligned}
W(k) &= n - km_1^2 - (n-k)m_2^2 \\
&= n - km_1^2 - nm_2^2 + km_2^2 \\
&= n - km_1^2 + (nm_2)m_2 + km_2^2 \\
&= n - km_1 + k(m_1 - m_2)m_2 + km_2 2 \\
&= n + k(-m_1^2 + m_1m_2 - m_2^2 + m_2^2) \\
&= n + km_1(m_2 - m_1)
\end{aligned}$$

Using the relationship $-nm_2 = k(m_1 - m_2) \implies m_2 - m_1 = \frac{nm_2}{k}$ from before, we can again rewrite $W(k)$:

$$\begin{aligned}
W(k) &= n - (n-k)m_2\frac{nm_2}{k} \\
&= n\frac{(n-k)n}{k}m_2^2
\end{aligned}$$

And we use the same relationship again: $m_2 - m_1 = \frac{n}{k}m_2 \implies m_2 = (m_2 - m_1)\frac{k}{n}$.

Then we can finally write $W(k)$ as:

$$W(k) = n - \frac{(n-k)n}{k}\frac{k^2}{n^2}(m_1 - m_2)^2$$
$$\boxed{W(k) = n - \frac{(n-k)k}{n}(m_1 - m_2)^2}$$

On the other hand, if we expand $R(k)$:

$$\begin{aligned} R(k) &= \sum_1^k x_i^2 + 2\sqrt{\frac{n-k}{k}}\sum_1^k x_i + n - k + \sum_{k+1}^n x_i^2 - 2\sqrt{\frac{k}{n-k}}\sum_{k+1}^n x_i + k \\ &= 2n + 2\sqrt{k(n-k)}m_1 - 2\sqrt{k(n-k)}m_2 \end{aligned}$$

If we expand and simplify $-\frac{(R(k))^2}{4n}$, we get:

$$-\frac{(R(k))^2}{4n} = -n - \frac{k(n-k)}{n}m_1^2 - \frac{k(n-k)}{n}m_2^2 - 2m_1\sqrt{k(n-k)} + 2m_2\sqrt{k(n-k)} + \frac{2k(n-k)}{n}m_1 m_2$$

Then noting that some terms cancel each other out, $R(k) - \frac{(R(k))^2}{4n}$:

$$\begin{aligned} n - \frac{k(n-k)}{n}m_1^2 - \frac{k(n-k)}{n}m_2^2 + 2\frac{k(n-k)}{n}m_1 m_2 \\ = n - \frac{k(n-k)}{n}(m_1^2 + m_2^2 - 2m_1 m_2) \\ \boxed{= n - \frac{k(n-k)}{n}(m_1 - m_2)^2} \end{aligned}$$

Which is exactly the same as our expression for $W(k)$.

Arbitrary k

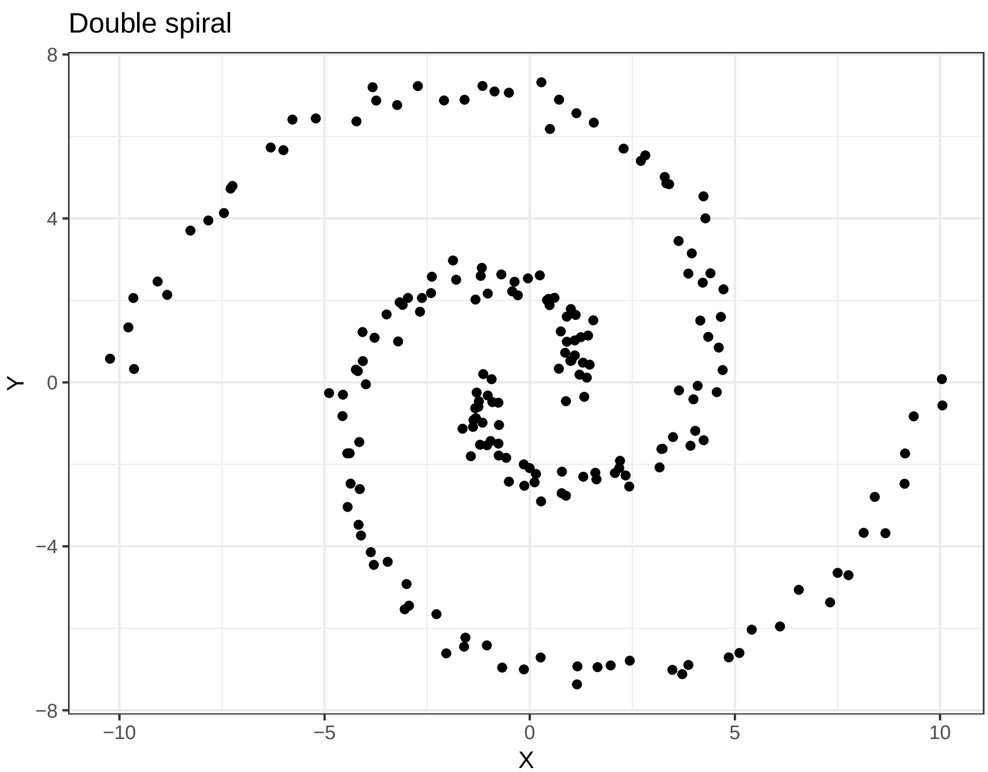
For arbitrary k, the methods as prescribed by Luxburg involve embedding the graph to $\mathbb{R}^k$ and then performing k-means clustering. In this case, we cannot perform the same sort of analysis since there is no way to "order" the x_i's.

Numerical experiment for $k = 2$ and $\mathbb{R}^2$

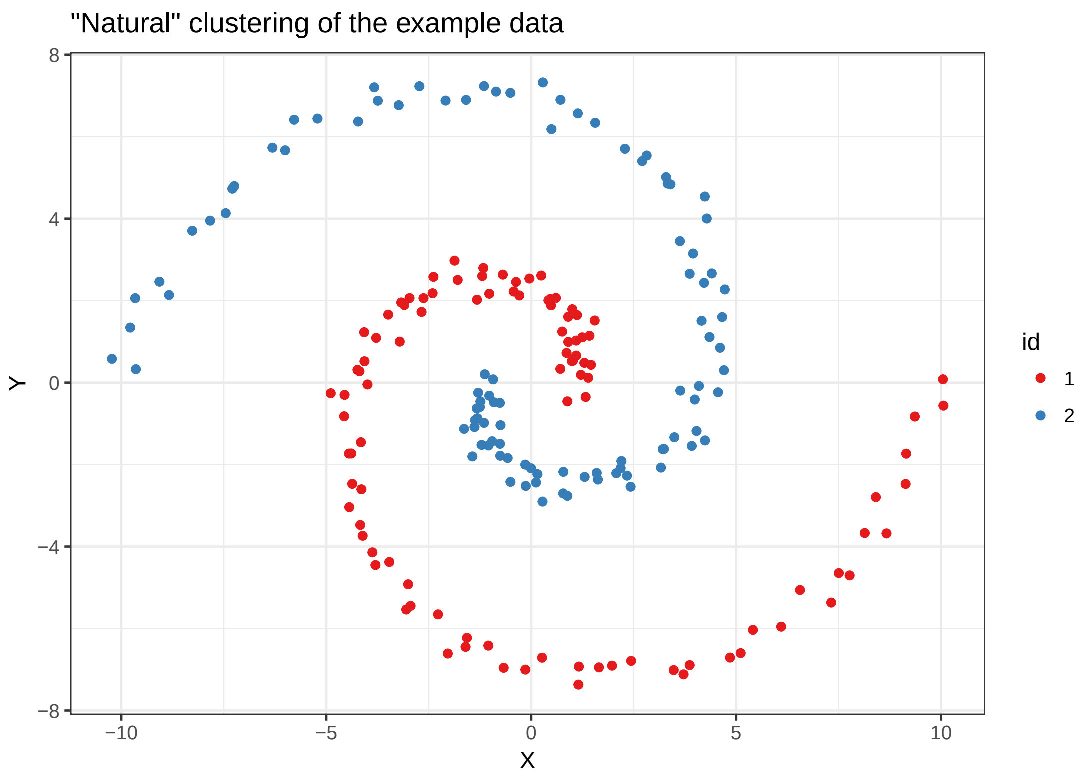
Double spiral

We will generate a "double spiral" to be partitioned into two clusters. This is an example that k-means would fail but is fairly distinguishable visually or intuitively. After generating the spiral, we will construct a k-nearest neighbors graph.

```r
# borrow some functions from S675
source('http://pages.iu.edu/~mtrosset/Courses/675/manifold.r')

# parameters
set.seed(112358)
s <- 2 ** 5
eps <- 2 ** -2
k <- 10  # for constructing the knn graph
K <- 2  # number of clusters
cols2 <- colorRampPalette(c('blue', 'white', 'red'))(256)
rad.max <- 10
ang.max <- 2 * pi
angles <- seq(0, ang.max, length.out = 100)
radii <- seq(1, sqrt(rad.max), length.out = 100) ** 2
N <- 100  # number of times to try k-means clustering

# data
spiral.df <- dplyr::data_frame(X = radii * cos(angles),
                               Y = radii * sin(angles))
spiral.df <- dplyr::data_frame(X = radii * cos(angles),
                               Y = radii * sin(angles))
neg.spiral.df <- dplyr::mutate(spiral.df,
                               X = -X, Y = -Y,
                               id = '2')
spiral.df %<>%
  dplyr::mutate(id = '1') %>%
  dplyr::bind_rows(neg.spiral.df) %>%
  dplyr::mutate(X = X + rnorm(n = n(), sd = eps),
                Y = Y + rnorm(n = n(), sd = eps))

n <- nrow(spiral.df)  # number of vertices

# viz
ggplot(spiral.df) +
  geom_point(aes(x = X, y = Y)) +
  coord_fixed() +
  labs(title = 'Double spiral')
```

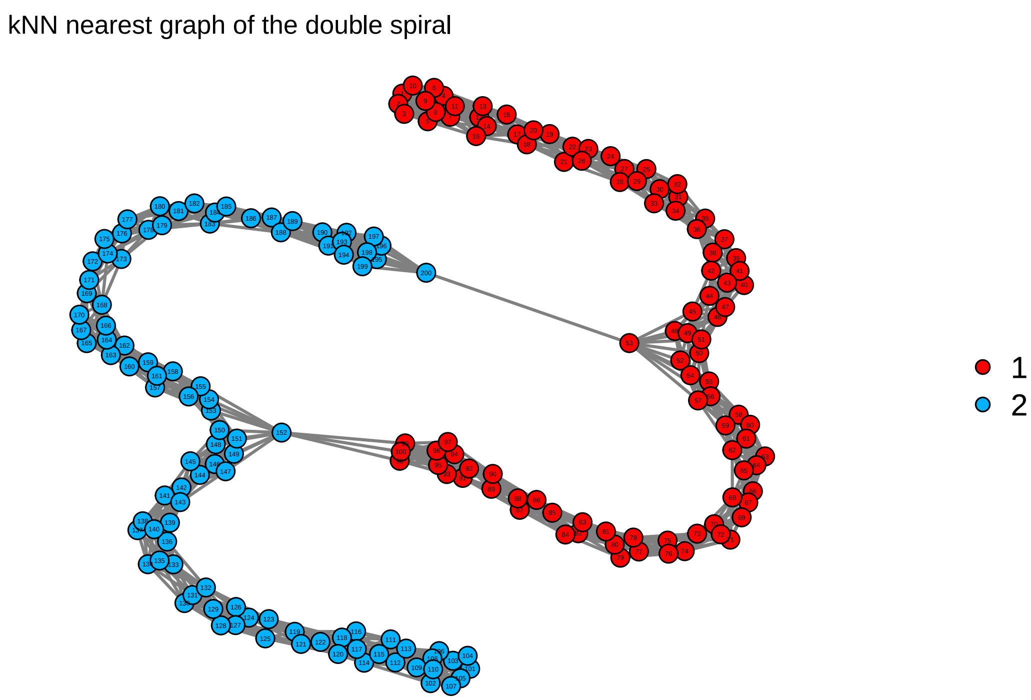



```
ggplot(spiral.df) +
  geom_point(aes(x = X, y = Y, colour = id)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1') +
  labs(title = '"Natural" clustering of the example data')
```



Then we will construct a(n) 10-nearest-neighbors graph:

```
# construct W
W <- spiral.df %>%
  dplyr::select(X, Y) %>%
  as.matrix() %>%
  mds.edm1() %>%
  graph.knn(k) %>%
  graph.adj()

# viz
qgraph(W, groups = spiral.df$id,
       title = 'kNN nearest graph of the double spiral',
       layout = 'spring')
```



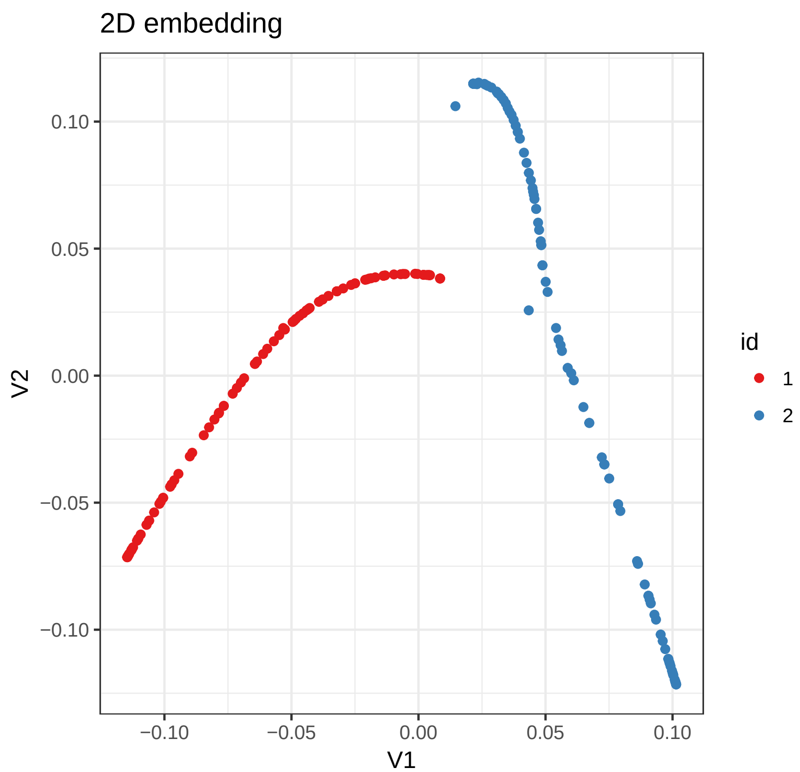
Here it is pretty obvious how we should cut the graph.

Then proceed with the clustering method as described by Luxburg:

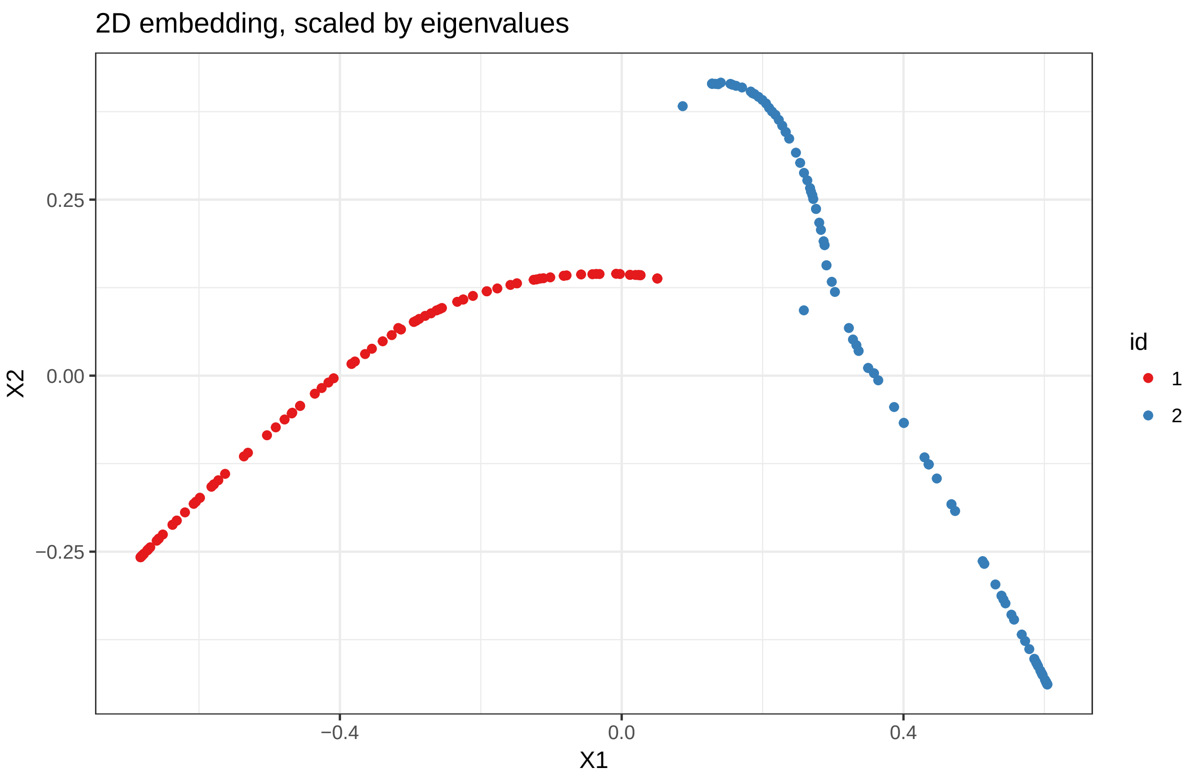
```r
# construct L
L <- graph.laplacian(W)

# eigendecomposition
L.eigen <- eigen(L)

two.eigen.df <- L.eigen$vectors[, seq(n - 1, n - K)] %>%
  as.data.frame() %>%
  dplyr::mutate(id = spiral.df$id) %>%
  dplyr::mutate(X1 = V1 / sqrt(L.eigen$values[n - 1]),
                X2 = V2 / sqrt(L.eigen$values[n - 2]))
```

Then we can take a look at the projection to $\mathbb{R}^2$:

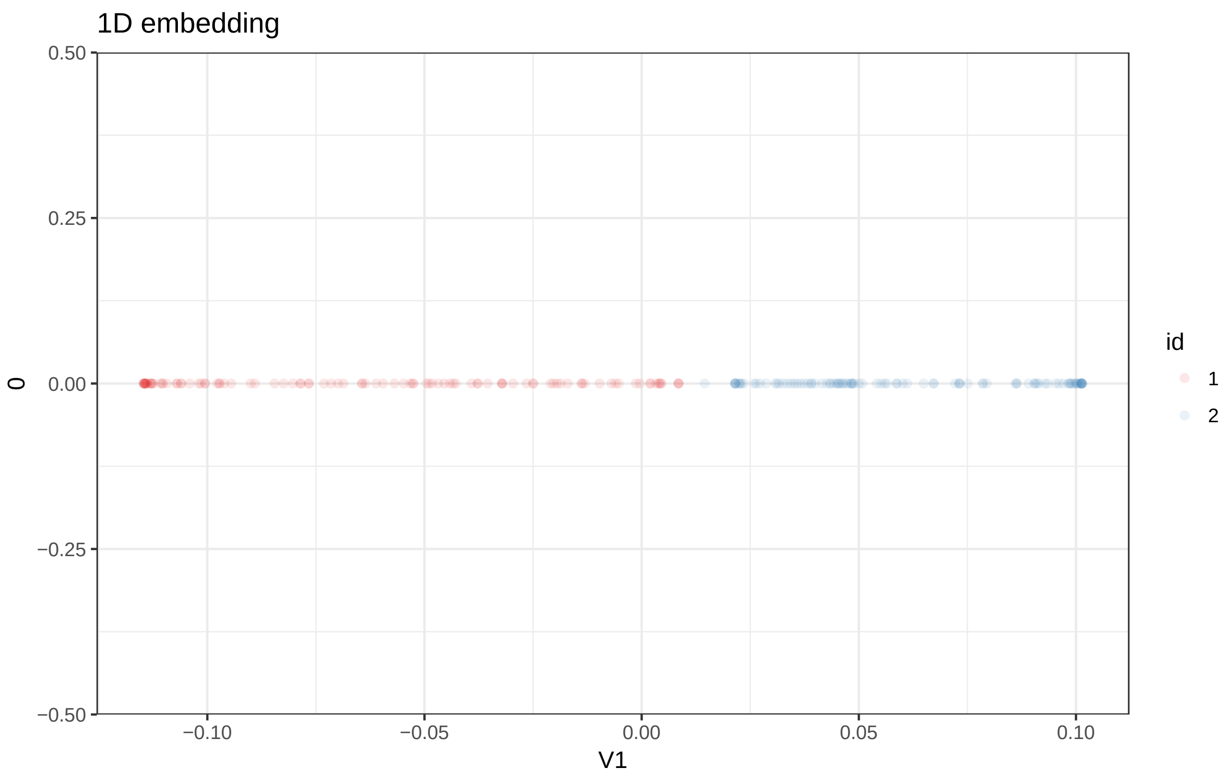
```r
ggplot(two.eigen.df) +
  geom_point(aes(x = V1, y = V2, colour = id)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1') +
  labs(title = '2D embedding')
```




```r
ggplot(two.eigen.df) +
  geom_point(aes(x = X1, y = X2, colour = id)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1') +
  labs(title = '2D embedding, scaled by eigenvalues')
```




```r
ggplot(two.eigen.df) +
  geom_point(aes(x = V1, y = 0, colour = id),
             alpha = .1) +
  scale_colour_brewer(palette = 'Set1') +
  labs(title = '1D embedding')
```



Next, we can take a look at how the three methods correspond to our original labels:

```
# run each clustering method N times and pick the best one
multiple.clusterings <- function(input.df, columns, clusters = 2, iter = 100) {
  # run the k-means algorithm many times
  clusterings <- lapply(seq(iter), function(i) {
    input.df %>%
      dplyr::select(columns) %>%
      dist() %>%
      kmeans(clusters)
  })

  # see which time was best
  tot.withinss <- sapply(clusterings, function(clustering) {
    clustering$tot.withinss
  })
  best.clust.ind <- which.min(tot.withinss)[1]

  return(clusterings[[best.clust.ind]])
}

unscaled.clustering <- multiple.clusterings(two.eigen.df, c('V1', 'V2'), K, N)
scaled.clustering <- multiple.clusterings(two.eigen.df, c('X1', 'X2'), K, N)
one.d.clustering <- multiple.clusterings(two.eigen.df, 'X1', K, N)

spiral.df %<>%
  dplyr::mutate(unscaled = as.character(unscaled.clustering$cluster),
                scaled = as.character(scaled.clustering$cluster),
                one.d = as.character(one.d.clustering$cluster))
```

```r
table(spiral.df$id, spiral.df$unscaled)
```

```
      1   2
  1 100   0
  2  60  40
```

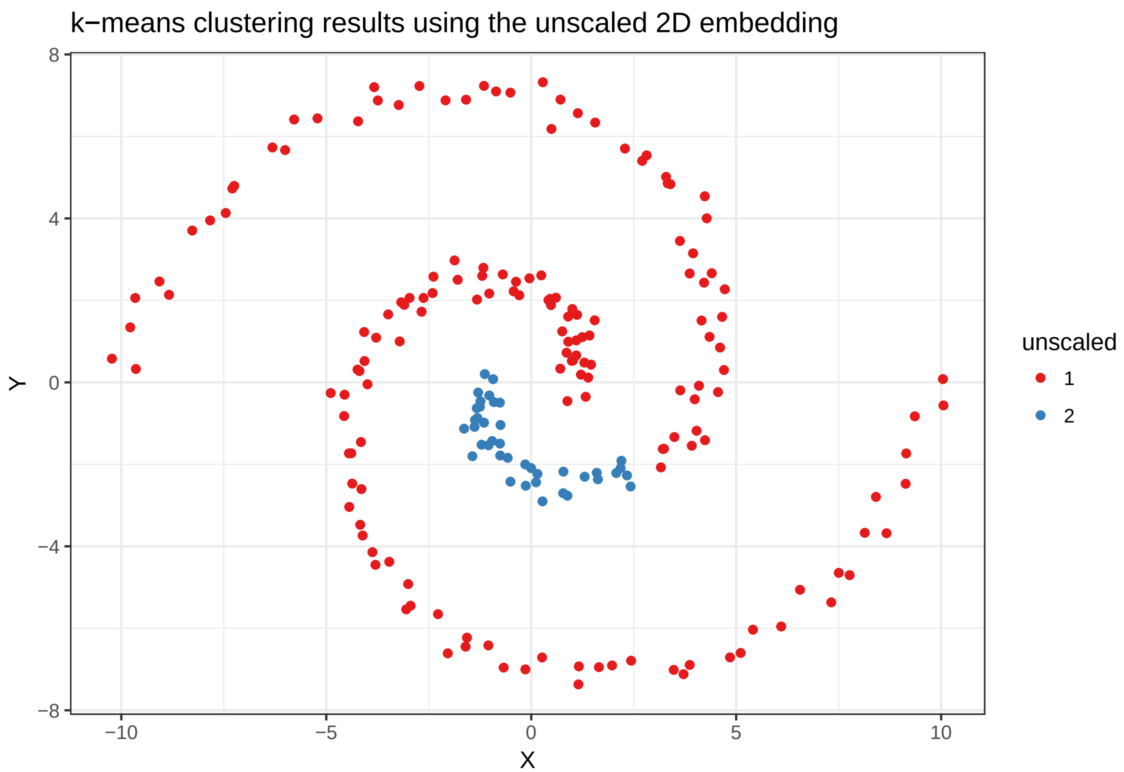
```r
table(spiral.df$id, spiral.df$scaled)
```

```
      1   2
  1  80  20
  2   0 100
```

```r
table(spiral.df$id, spiral.df$one.d)
```

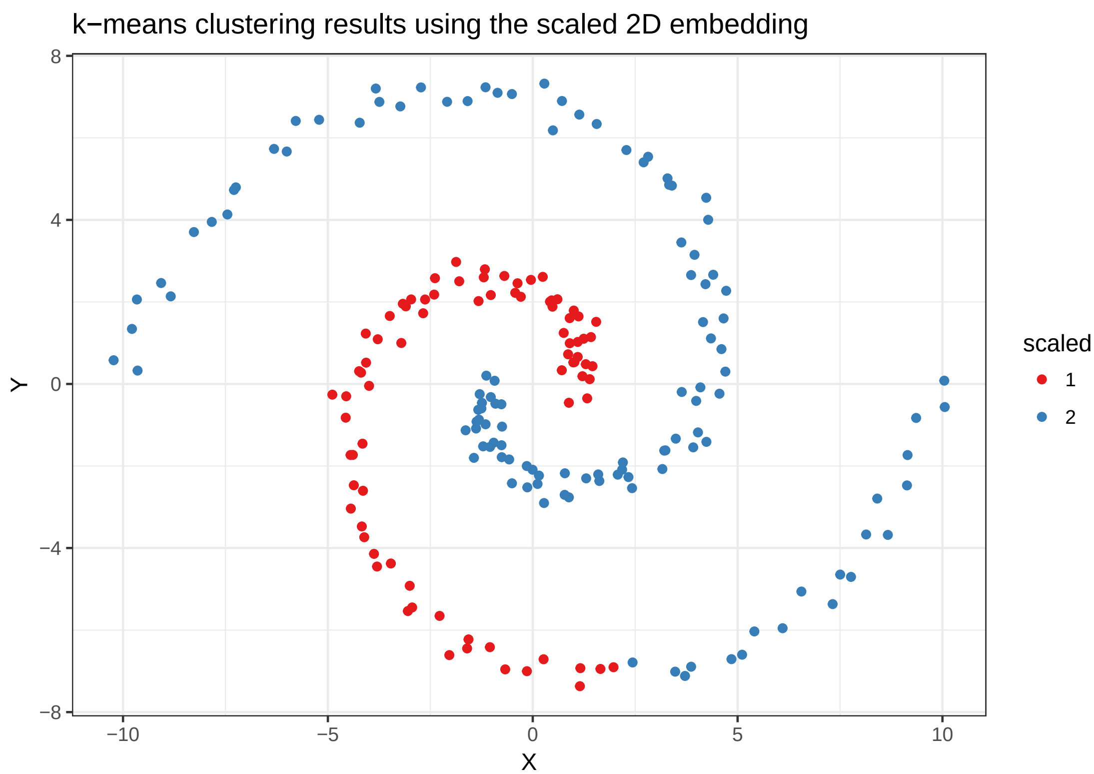
```
      1   2
  1  19  81
  2 100   0
```

```r
ggplot(spiral.df) +
  geom_point(aes(x = X, y = Y, colour = unscaled)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1') +
  labs(title = 'k-means clustering results using the unscaled 2D embedding')
```

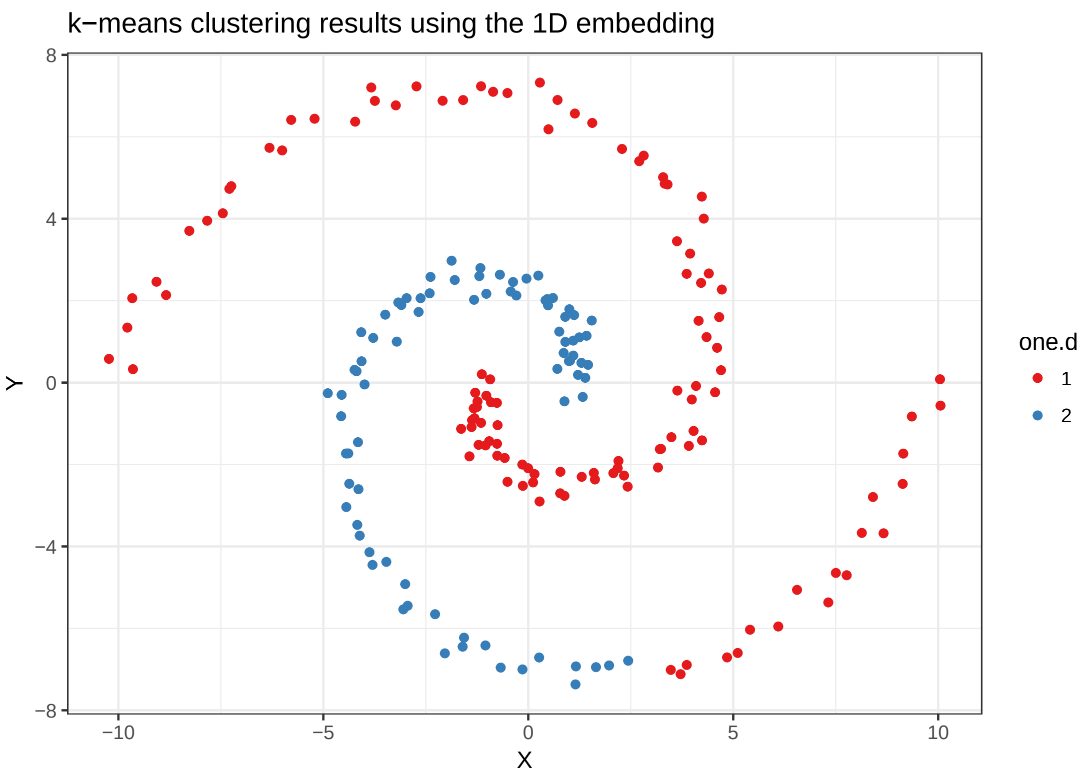



```r
ggplot(spiral.df) +
  geom_point(aes(x = X, y = Y, colour = scaled)) +
  coord_fixed() +
```

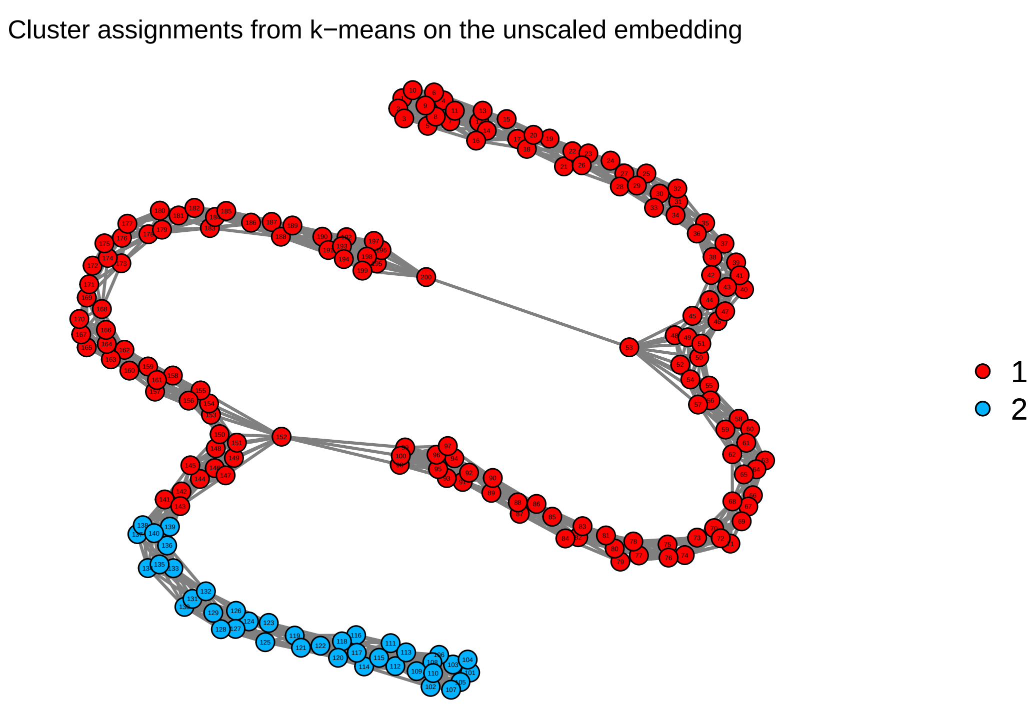
```
  scale_colour_brewer(palette = 'Set1') +
  labs(title = 'k-means clustering results using the scaled 2D embedding')
```




```
ggplot(spiral.df) +
  geom_point(aes(x = X, y = Y, colour = one.d)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1') +
  labs(title = 'k-means clustering results using the 1D embedding')
```

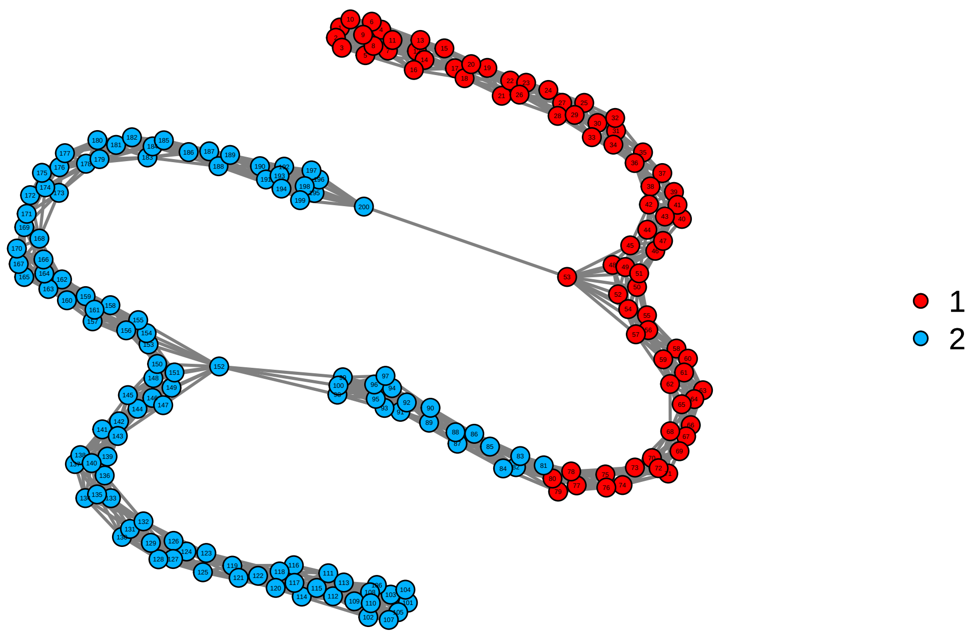



```r
qgraph(W, groups = spiral.df$unscaled,
       title = 'Cluster assignments from k-means on the unscaled embedding')
```

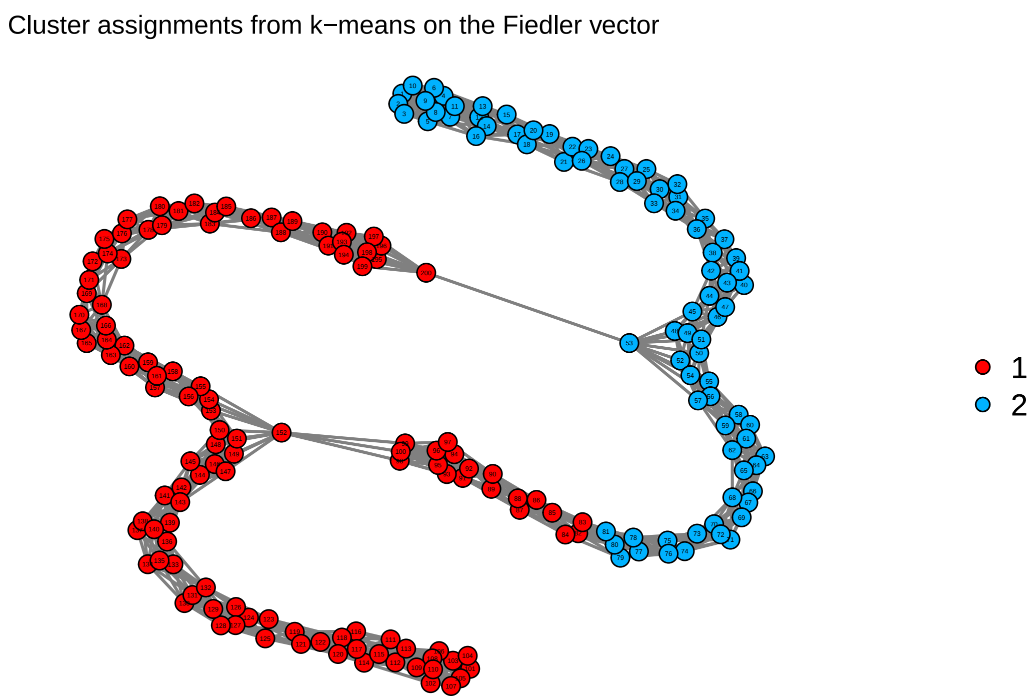


```r
qgraph(W, groups = spiral.df$scaled,
       title = 'Cluster assignments from k-means on the scaled embedding')
```

Cluster assignments from k−means on the scaled embedding




```r
qgraph(W, groups = spiral.df$one.d,
       title = 'Cluster assignments from k-means on the Fiedler vector')
```



In a way, we have “unraveled” the spirals, resulting in two curves. Eye-balling this representation gives us two obvious clusters, but k-means fails to separate them the way we would expect since they are not spherical clusters. Projection to the unscaled space seems to perform especially poorly, and the plots of the columns of H_{approx} reveal why.

```
two.eigen.df %<>%
  dplyr::mutate(unscaled.clust = as.character(unscaled.clustering$cluster),
                scaled.clust = as.character(scaled.clustering$cluster),
                one.d.clust = as.character(one.d.clustering$cluster))

clust.list <- list('unscaled.clust', 'scaled.clust', 'one.d.clust')

clust.plots <- lapply(clust.list, function(i) {
  ggplot(two.eigen.df) +
    geom_point(aes_string(x = 'V1', y = 'V2', colour = i)) +
    scale_colour_brewer(palette = 'Set1') +
    coord_fixed() +
    guides(colour = FALSE) +
    labs(x = NULL, y = NULL)
})

.gridarrange <- function(...) gridExtra::grid.arrange(..., ncol = 3)
do.call(.gridarrange, clust.plots)
```

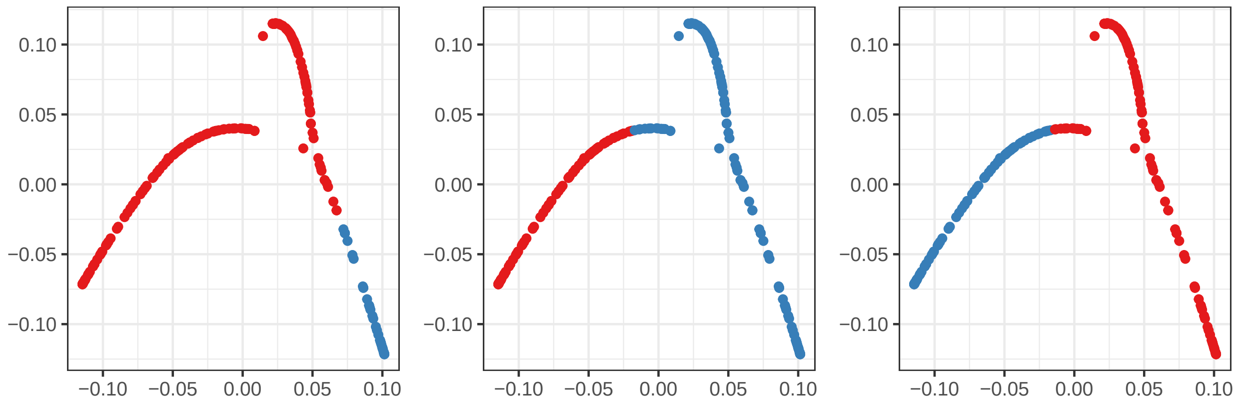
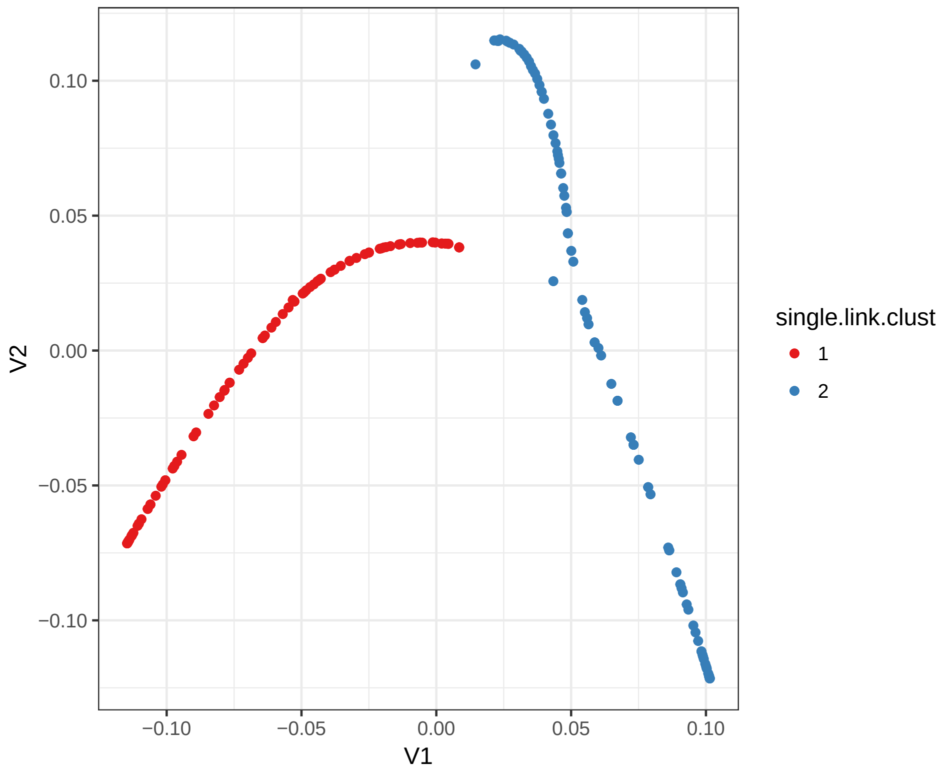


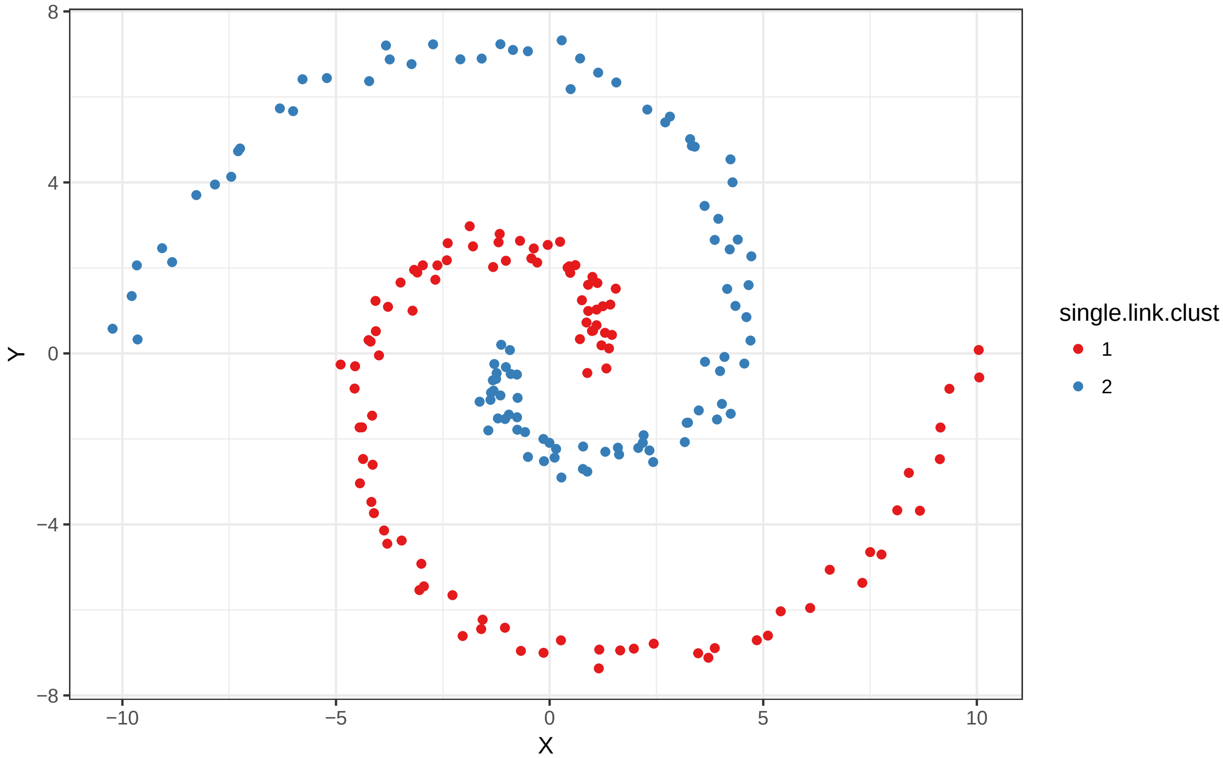
Figure 1: 2D embedding annotated by cluster assignments. From left to right: k-means on the unscaled embedding, k-means on the scaled embedding, and k-means on the 1D embedding.

In fact, it almost looks as if another clustering method is more appropriate here, in particular, single-linkage.

```
single.link.clust <- two.eigen.df %>%
  dplyr::select(V1, V2) %>%
  dist() %>%
  hclust(method = 'single') %>%
  cutree(K)

two.eigen.df %<>%
  dplyr::mutate(single.link.clust = as.character(single.link.clust))
spiral.df %<>%
  dplyr::mutate(single.link.clust = as.character(single.link.clust))

ggplot(two.eigen.df) +
  geom_point(aes(x = V1, y = V2, colour = single.link.clust)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1')
```




```
ggplot(spiral.df) +
  geom_point(aes(x = X, y = Y, colour = single.link.clust)) +
  coord_fixed() +
  scale_colour_brewer(palette = 'Set1')
```




```
table(spiral.df$id, spiral.df$single.link.clust)
```

```
      1   2
  1 100   0
  2   0 100
```

We should also check that this clustering is actually better according to the metrics from the Luxburg tutorial. We try this two ways:

1. H_{approx} should be as close of an approximation to H as possible. Here, H_{approx} is the k-dimensional embedding via spectral decomposition while H defines the actual best clustering. In this comparison, we fix H_{approx} and construct H according to the resulting clustering of H_{approx} (e.g., via k-means or single-linkage), then compare the constructed H to H_{approx} via the Frobenius norm.
2. The exact clustering should minimize $\mathrm{Tr}(H^T LH)$ where H is defined by the particular clustering. If single-linkage is indeed better than k-means in this case, then the H constructed from single-linkage clustering should result in a lower metric than the one constructed from k-means clustering.

```
# construct Happrox
H.approx <- two.eigen.df %>%
  dplyr::select(V1, V2) %>%
  as.matrix()

#' @title construct H function
#' @description H is constructed based on a vector that designates the clusters
#' @param clustering (numeric) A vector of cluster assignments
#' @return (matrix) H based on the cluster assignments
construct.H <- function(clustering) {
  # this function is limited to nonempty clusters
  # e.g., if there are 3 clusters, they must be assigned as 1, 2, 3
```

```r
  clusters <- unique(clustering)
  if (length(clusters) != max(clustering)) {
    stop(simpleError('there are empty clusters'))
  }
  if (min(clustering) < 1) {
    stop(simpleError('cluster indexing starts at 1'))
  }

  # find |A_k|
  cluster.sizes <- sapply(clusters, function(i) {
    length(clustering[clustering == i])
  })

  # construct H
  H <- sapply(clustering, function(i) {
    h <- rep(0, length(clusters))
    h[i] <- 1 / sqrt(cluster.sizes[i])
    return(h)
  }) %>%
    t()

  return(H)
}

# construct H for k-means (unscaled)
H.kmeans <- construct.H(unscaled.clustering$cluster)

# construct H for single linkage
H.singlelink <- construct.H(single.link.clust)

# frobenius norm comparison
norm(H.approx - H.kmeans, type = 'F')
```

```
[1] 2.400969
```

```r
norm(H.approx - H.singlelink, type = 'F')
```

```
[1] 2.257204
```

```r
# making sure ordering doesn't matter
# this could get messy for large k
norm(H.approx - H.kmeans[, 2:1], type = 'F')
```

```
[1] 1.497036
```

```r
norm(H.approx - H.singlelink[, 2:1], type = 'F')
```

```
[1] 1.704416
```

```r
# minimizing trace comparison
# here order doesn't matter
tr(t(H.kmeans) %*% L %*% H.kmeans)
```

```
[1] 0.46875
```

```r
tr(t(H.singlelink) %*% L %*% H.singlelink)
```

```
[1] 0.08
```

```r
# trace for Happrox
tr(t(H.approx) %*% L %*% H.approx)
```

```
[1] 0.1049757
```

Here, we can see that single linkage produces a H matrix that is closer to H_{approx} and results in a better clustering metric.

For completeness, let's check whether single-linkage produced a better clustering according to the k-means metric:

```r
# function for computing total within SS
within.ss <- function(X, clustering) {
  clusters <- unique(clustering)
  sapply(clusters, function(i) {
    X.clust <- X[clustering == i, ]
    sum(dist(X.clust) ** 2) / 2 / nrow(X.clust)
  }) %>%
    sum()
}

# total within-ss from k-means
within.ss(dplyr::select(two.eigen.df, V1, V2), unscaled.clustering$cluster)
```

```
[1] 0.5689028
```

```r
# total within-ss from single linkage
within.ss(dplyr::select(two.eigen.df, V1, V2), single.link.clust)
```

```
[1] 0.6209851
```

And indeed, the k-means clustering result resulted in a better k-means metric. This conclusively tells us that k-means on the embedding is not the best clustering method for this example.

However, this may be a very particular case. From the embedding, we can see that while there is a very obvious clustering, the clusters are not ellipsoids, which is what k-means is good at. Perhaps in most graph embeddings of this type, we do end up with ellpsoid clusters, and we just happened to get long strands in this example.

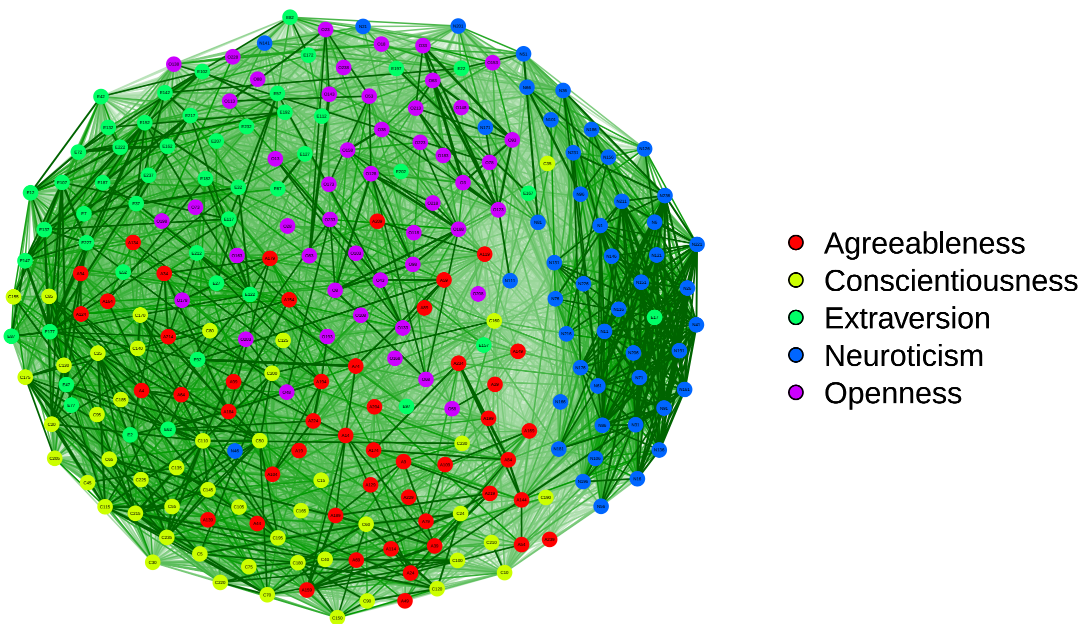
Numerical experiment with $k = 5$

For this example, we will look at the "Big 5" data from psychology. In this dataset, 500 students took a 240-question personality quiz. Each question corresponds to one of 5 personality types and is rated on a 5-point scale. We construct a graph using the questions as objects/vertices and the edges as empirical similarities based on how the students responded.

```r
# load the data
data(big5)
data(big5groups)

n <- min(dim(big5))  # number of vertices
K <- length(big5groups)  # number of clusters
group.names <- names(big5groups)
N <- 2 ** 10  # number of times to try k-means

# reformat the groups into a vector
# this is just so we can do easier comparisons with the clusters
```

```r
group.assign <- rep(NA, n)
for (i in seq(n)) {
  for (j in group.names) {
    if (i %in% big5groups[[j]]) {
      group.assign[i] <- j
    }
  }
}

# similarities
# just use correlation per the qgraph example
# and adjust so they're all positive
W <- (cor(big5) + 1) / 2
```

```r
qgraph(W, groups = group.assign, minimum = .5,
       borders = FALSE, layout = 'spring')
```



And then we proceed by constructing the Laplacian, choosing the "first" 5 eigenvectors as our embedding, and performing k-means clustering with 5 clusters.

```r
# function to create the embedding matrix H
spect.embed <- function(sim.graph, dims = 2) {
  L <- graph.laplacian(sim.graph)
  L.eigen <- eigen(L)
  L.eigen$vectors[, seq(nrow(L) - 1, nrow(L) - dims)] %>%
    as.data.frame() %>%
    return()
}
```

```r
embed.df <- spect.embed(W, K)
unscaled.clustering <- multiple.clusterings(embed.df,
                                            paste0('V', seq(K)),
                                            K, N)

table(unscaled.clustering$cluster, group.assign)
```

```
   group.assign
    Agreeableness Conscientiousness Extraversion Neuroticism Openness
  1             0                 1            0           1        0
  2            40                39           41           5       47
  3             0                 0            0           2        0
  4             1                 2            2           8        0
  5             7                 6            5          32        1
```

We can see that there isn't much correspondence between the personality types and the clusters. Unfortunately this problem is hard to visualize due to its dimensionality, but we can still compare some clustering metrics.

First, let's check whether the original groupings would result in a better k-means clustering metric (total within sum of squares) than the one that k-means found:

```r
# total within-ss from k-means
within.ss(embed.df, unscaled.clustering$cluster)
```

```
[1] 1.407175
```

```r
# total within-ss using original group assignments
within.ss(embed.df, group.assign)
```

```
[1] 1.898069
```

Next, we can compute $\mathrm{Tr}(H^T L H)$ for the k-means result and then for the original group assignments:

```r
H.kmeans <- construct.H(unscaled.clustering$cluster)
H.original <- construct.H(as.numeric(as.factor(group.assign)))

L <- graph.laplacian(W)

tr(t(H.kmeans) %*% L %*% H.kmeans)
```

```
[1] 3138.471
```

```r
tr(t(H.original) %*% L %*% H.original)
```

```
[1] 478.3982
```

We can also try single-linkage again and see if that does any better than k-means:

```r
single.link.clust <- embed.df %>%
  dist() %>%
  hclust(method = 'single') %>%
  cutree(K)

H.singlelink <- construct.H(single.link.clust)

tr(t(H.singlelink) %*% L %*% H.singlelink)
```

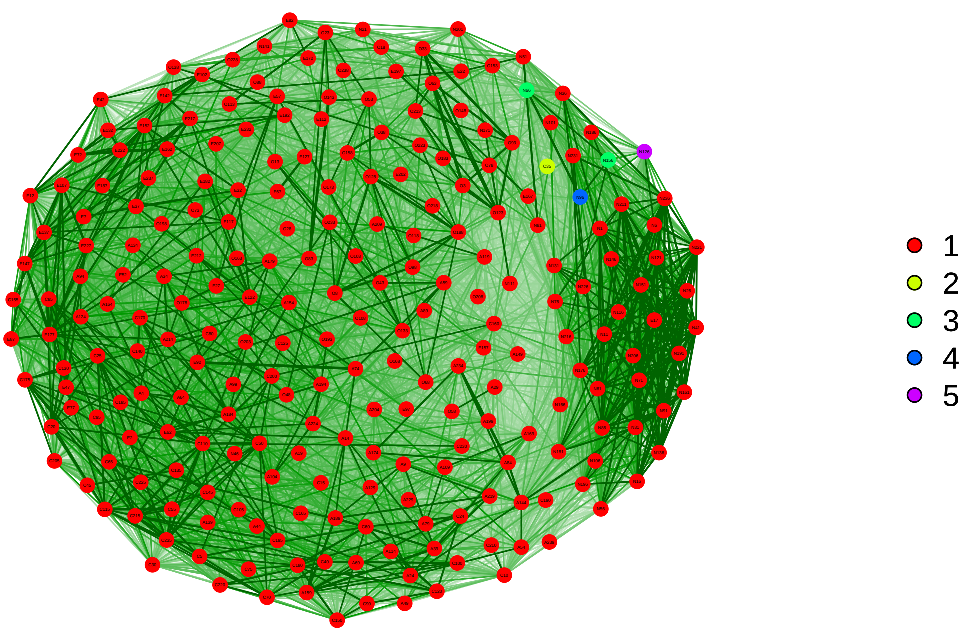
```
[1] 462.5282
```

```r
table(group.assign, single.link.clust)
```

```
                     single.link.clust
group.assign           1  2  3  4  5
  Agreeableness       48  0  0  0  0
  Conscientiousness 47  1  0  0  0
  Extraversion        48  0  0  0  0
  Neuroticism         44  0  2  1  1
  Openness            48  0  0  0  0
```

Strangely, while single linkage does not produce a clustering that looks anything like the original group assignments, it results in a better clustering according to the ratio cut metric. We can try visualizing it:

```r
qgraph(W, groups = as.character(single.link.clust), minimum = .5,
       borders = FALSE, layout = 'spring')
```



Finally, we can look at the embedding, H_{approx}:

```r
Happrox <- as.matrix(embed.df)
tr(t(Happrox) %*% L %*% Happrox)
```

```
[1] 573.1686
```

And again, this value is higher than what we got with single linkage, which is not expected, since this is the solution to the unconstrained optimization problem, whereas the various H's constructed from cluster assignments are solutions to the constrained optimization problem.

We can try this again, except in $\mathbb{R}^4$:

```r
# embed into R4
embed.r4.df <- spect.embed(W, K - 1)
```

```r
# k-means clustering with k=5
unscaled.clustering.r4 <- multiple.clusterings(embed.df,
                                               paste0('V', seq(K - 1)),
                                               K, N)

# compare to original group assignments
table(unscaled.clustering.r4$cluster, group.assign)
```

```
   group.assign
    Agreeableness Conscientiousness Extraversion Neuroticism Openness
  1             0                 0            0           2        0
  2            42                45           44           6       48
  3             1                 0            2           9        0
  4             0                 1            0           1        0
  5             5                 2            2          30        0
```

```r
table(unscaled.clustering$cluster, unscaled.clustering.r4$cluster)
```

```

      1   2   3   4   5
  1   0   0   0   2   0
  2   0 172   0   0   0
  3   2   0   0   0   0
  4   0   1   8   0   4
  5   0  12   4   0  35
```

```r
# ratio cut metric
H.kmeans.r4 <- construct.H(unscaled.clustering.r4$cluster)
tr(t(H.kmeans.r4) %*% L %*% H.kmeans.r4)
```

```
[1] 2172.212
```

```r
# try single-link clustering
single.link.clust.r4 <- embed.r4.df %>%
  dist() %>%
  hclust(method = 'single') %>%
  cutree(K)
H.singlelink.r4 <- construct.H(single.link.clust.r4)
tr(t(H.singlelink.r4) %*% L %*% H.singlelink.r4)
```

```
[1] 462.5282
```

```r
table(group.assign, single.link.clust.r4)
```

```
                   single.link.clust.r4
group.assign         1  2  3  4  5
  Agreeableness     48  0  0  0  0
  Conscientiousness 47  1  0  0  0
  Extraversion      48  0  0  0  0
  Neuroticism       44  0  2  1  1
  Openness          48  0  0  0  0
```

```r
table(single.link.clust, single.link.clust.r4)
```

```
                 single.link.clust.r4
single.link.clust   1   2   3   4   5
                1 235   0   0   0   0
                2   0   1   0   0   0
```

```
3   0   0   2   0   0
4   0   0   0   1   0
5   0   0   0   0   1
```

According to the ratio cut metric, embedding in $\mathbb{R}^4$ is equivalent or better than embedding in $\mathbb{R}^5$ in this particular case.

Another look at the relaxed optimization problem

In both the double spiral and Big 5 data, we noticed that H constructed from single linkage clustering resulted in a lower ratio cut metric $\text{Tr}(H^T L H)$ compared to H_{approx}, which is constructed from the eigenvectors of L. This is unexpected, since the former is a (possible) solution to a *constrained* optimization problem whereas the latter is the solution to an *unconstrained* optimization problem.

Another look at the theorem mentioned in Section 5.2.2. of Luxburg's paper says that the solution to

$$\min_{H \in \mathbb{R}^{n \times k}} \{\text{Tr}(H^T L H)\}$$

is given by combining the first k eigenvectors of L, including the one corresponding to the zero eigenvalue. Furthermore,

$$\min_{H \in \mathbb{R}^{n \times k}} \{\text{Tr}(H^T L H)\} = \lambda_0 + \lambda_1 + \cdots + \lambda_{k-1}$$

where $\lambda_0 = 0$ in our particular case.

Finally, we know that the eigenvector that corresponds to $\lambda_0 = 0$ is $h_0 = \frac{1}{\sqrt{n}} \mathbb{1}$, so when embedding, we can ignore this outright. Then instead of embedding the graph in $\mathbb{R}^k$, we should actually embed in what is effectively $\mathbb{R}^{k-1}$.

This has some interesting implications to one of our criteria in judging how good a particular clustering is. Recall that we said that the matrix constructed from a particular choice of clustering, H_{clust}, should be close to H_{approx}, so we came up with

$$||H_{clust} - H_{approx}||_F^2$$

as a metric for how good a particular clustering is. However, before, we used $H_{approx} = \begin{bmatrix} h_1 & h_2 & \cdots & h_k \end{bmatrix}$ instead of $H_{approx} = \begin{bmatrix} h_0 & h_2 & \cdots & h_{k-1} \end{bmatrix}$. In the following numerical example, we show using the double spiral that the "natural" clustering results in a smaller $||H_{clust} - H_{approx}||_F^2$ metric, and we construct a nonsensical clustering resulting in a H_{clust} that is closer to the former H_{approx} than the natural clustering.

```r
# construct W
W <- spiral.df %>%
  dplyr::select(X, Y) %>%
  as.matrix() %>%
  mds.edm1() %>%
  graph.knn(k) %>%
  graph.adj()

# construct L
L <- graph.laplacian(W)

# decompose L
L.eigen <- eigen(L)
```

```r
# H_approx using h0 and h1
H.01 <- L.eigen$vectors[, 200:199]

# H_approx using h1 and h2
H.12 <- L.eigen$vectors[, 199:198]

# show that H.01 results in a lower ratio cut metric (which should be obvious)
tr(t(H.01) %*% L %*% H.01)
```

```
[1] 0.02817946
```

```r
tr(t(H.12) %*% L %*% H.12)
```

```
[1] 0.1049757
```

```r
# the natural clustering from the spirals
H.spiral <- construct.H(c(rep(1, 100), rep(2, 100)))

# show that H.spiral is closer to H.01 than H.12
min(c(norm(H.spiral - H.01, type = 'F'),
      norm(H.spiral[, 2:1] - H.01, type = 'F')))
```

```
[1] 1.166761
```

```r
min(c(norm(H.spiral - H.12, type = 'F'),
      norm(H.spiral[, 2:1] - H.12, type = 'F')))
```

```
[1] 1.704416
```

```r
# construct H.clust that is closer to H.12
H.clust <- sapply(seq(nrow(W)), function(i) {
  x <- rep(0, 2)
  clust.assign <- which.max(H.12[i, ])
  x[clust.assign] <- 1
  return(x)
}) %>%
  t()
norm.const <- apply(H.clust, 2, function(x) sqrt(sum(x)))
H.clust <- H.clust / norm.const

# check that H.12 is closer to H.clust than H.spiral
min(c(norm(H.clust - H.12, type = 'F'),
      norm(H.clust[, 2:1] - H.12, type = 'F')))
```

```
[1] 1.648077
```