

Spectral clustering - raport

Przemysław Kaleta

January 25, 2019

Użyteczne funkcje

Poniżej kod do wczytywania danych i rysowania wykresów. Kod, którego użyłem do zapisania wyników można znaleźć w pliku `raport.R`.

```
library(mclust)
library(dendextend)
library(genie)
library(dbSCAN)
library(stringi)

source("spectral.R")

read_data <- function(benchmark, dataset){
  matrix_file_name <- paste(dataset, ".data.gz", sep="")
  labels_file_name <- paste(dataset, ".labels0.gz", sep="")
  matrix_path <- file.path("../benchmarks", benchmark, matrix_file_name)
  labels_path <- file.path("../benchmarks", benchmark, labels_file_name)
  X <- as.matrix(read.table(matrix_path))
  Y <- as.matrix(read.table(labels_path))
  return(list(X=X, Y=Y))
}

plot_data <- function(X, Y, title=""){
  plot(X[, 1], X[, 2], col=unlist(Y), pch=20)
  title(title)
}
```

Analiza wyników

```
result <- read.csv("results.csv")
```

Który algorytm jest najlepszy?

```
result %>% select(algorithm, FM, AR) %>%
  group_by(algorithm) %>% summarise(mean_FM = mean(FM), mean_AR = mean(AR)) %>%
  arrange(desc(mean_FM))
```

```
## # A tibble: 9 x 3
##   algorithm      mean_FM mean_AR
##   <fct>          <dbl>   <dbl>
## 1 genie          0.867    0.809
## 2 hclust_average 0.777    0.582
## 3 hclust_centroid 0.767    0.531
## 4 hclust_complete 0.734    0.521
```

```
## 5 kmeans          0.729  0.553
## 6 hclust_mcquitty  0.720  0.510
## 7 spectral        0.716  0.546
## 8 hclust_median    0.702  0.483
## 9 hdbscan          0.504  0.376
```

Cieszymy się patrząc na te wyniki, bo widzimy że spectral clustering nie jest na końcu. Nie odstaje tak bardzo od metod funkcji `hclust`. Jedynie `genie` znacząco wyprzedza je wszystkie, zarówno dla miary FM jak i (zwłaszcza) AR.

Czy `genie` rzeczywiście jest najlepszy?

Nie ufamy za bardzo tym wynikom. Któryś z algorytmów musiał mieć największy wynik FM, ale być może było to całkiem przypadkowe. Dlatego sprawdzimy czy możemy mówić o statystycznej istotności tego wyniku.

```
genie_result <- unlist(result %>% filter(algorithm == "genie") %>% select(FM))
hclust_result <- unlist(result %>% filter(algorithm == "hclust_average") %>% select(FM))

wilcox.test(genie_result, hclust_result, alternative = "greater")
```

```
##
## Wilcoxon rank sum test with continuity correction
##
## data:  genie_result and hclust_result
## W = 1217, p-value = 0.005612
## alternative hypothesis: true location shift is greater than 0
```

Wynik, który otrzymujemy jest przybliżony. `Genie` okazuje się najlepszy i to mimo (być może) nie do końca poprawnej metodologii - porównywaliśmy go z najlepszym z pozostałych algorytmów.

Sprawdźmy wynik porównań wszystkich algorytmów. Musimy tutaj uważać na problem wielokrotnego testowania, ale twórcy funkcji `pairwise.wilcox.test` pomyśleli o tym. Co ciekawe radzą sobie z tym metodą Holma, ale mamy też do wyboru inne metody.

```
pairwise.wilcox.test(result$FM, result$algorithm)
```

```
##
## Pairwise comparisons using Wilcoxon rank sum test
##
## data:  result$FM and result$algorithm
##
##          genie  hclust_average hclust_centroid hclust_complete
## hclust_average 0.2469 -          -              -
## hclust_centroid 0.1024 1.0000 -              -
## hclust_complete 0.0066 1.0000 1.0000 -
## hclust_mcquitty 0.0062 1.0000 1.0000 1.0000
## hclust_median  0.0016 1.0000 1.0000 1.0000
## hdbscan         1.6e-05 0.0095 0.0142 0.0807
## kmeans          0.0067 1.0000 1.0000 1.0000
## spectral        0.0045 1.0000 1.0000 1.0000
##          hclust_mcquitty hclust_median hdbscan kmeans
## hclust_average -          -              -          -
## hclust_centroid -          -              -          -
## hclust_complete -          -              -          -
## hclust_mcquitty -          -              -          -
## hclust_median  1.0000 -          -          -
```

```
## hdbscan          0.1032          0.2064          -          -
## kmeans           1.0000          1.0000          0.0945          -
## spectral         1.0000          1.0000          0.1061          1.0000
##
## P value adjustment method: holm
```

Testując wielokrotnie nie ma podstaw do odrzucenia hipotezy że nasz algorytm jest tak samo dobry jak hclust_average, ale wydaje mi się, że nie ma się czym cieszyć. Nie mamy podstaw do powiedzenia, że wyniki hdbscanu są gorsze niż hclust_complete, a gołym okiem widać że gorsze są.

Okazuje się jednak, że nawet robiąc test jednokrotny nie ma podstaw do odrzucenia hipotezy, że nasz algorytm jest tak samo dobry jak hclust_average.

```
spectral_result <- unlist(result %>% filter(algorithm == "spectral") %>% select(FM))
hclust_result <- unlist(result %>% filter(algorithm == "hclust_average") %>% select(FM))

wilcox.test(spectral_result, hclust_result, alternative = "less")
```

```
##
## Wilcoxon rank sum test with continuity correction
##
## data: spectral_result and hclust_result
## W = 783.5, p-value = 0.1124
## alternative hypothesis: true location shift is less than 0
```

Na koniec porównamy nasz algorytm z hdbscanem.

```
spectral_result <- unlist(result %>% filter(algorithm == "spectral") %>% select(FM))
hdbscan_result <- unlist(result %>% filter(algorithm == "hdbscan") %>% select(FM))

wilcox.test(spectral_result, hdbscan_result, alternative = "greater")
```

```
##
## Wilcoxon rank sum test with continuity correction
##
## data: spectral_result and hdbscan_result
## W = 1254.5, p-value = 0.002211
## alternative hypothesis: true location shift is greater than 0
```

Należy jednak pamiętać, że hdbscan nie wie o prawdziwej liczbie skupień - jesteśmy zatem nie fair w stosunku do niego.

Jakie zbiory sprawiały najwięcej problemów analizie skupień?

```
result %>% select(benchmark, dataset, FM, AR) %>%
  group_by(benchmark, dataset) %>%
  summarize(FM_mean=mean(FM), AR_mean=mean(AR)) %>%
  arrange(FM_mean)
```

```
## # A tibble: 43 x 4
## # Groups:   benchmark [5]
## benchmark dataset FM_mean AR_mean
## <fct> <fct> <dbl> <dbl>
## 1 wut z1 0.471 0.185
## 2 wut cross 0.475 0.140
## 3 sipu s4 0.507 0.444
```

```
## 4 sipu      spiral      0.537  0.290
## 5 graves    zigzag      0.566  0.268
## 6 sipu      s3          0.567  0.515
## 7 graves    fuzzyx      0.601  0.471
## 8 sipu      pathbased   0.607  0.392
## 9 wut       x2          0.611  0.296
## 10 graves   line        0.614  0.116
## # ... with 33 more rows
```

Wygląda na to, że studenci WUTu byli dość złośliwi dobierając zbiory danych.

Analiza działania spectral clustering

Jak parametr M wpływa na działanie algorytmu?

Zbadamy teraz jak wybór hiperparametru M wpływa na działanie algorytmu. Posłużymy się w tym celu poniższą funkcją:

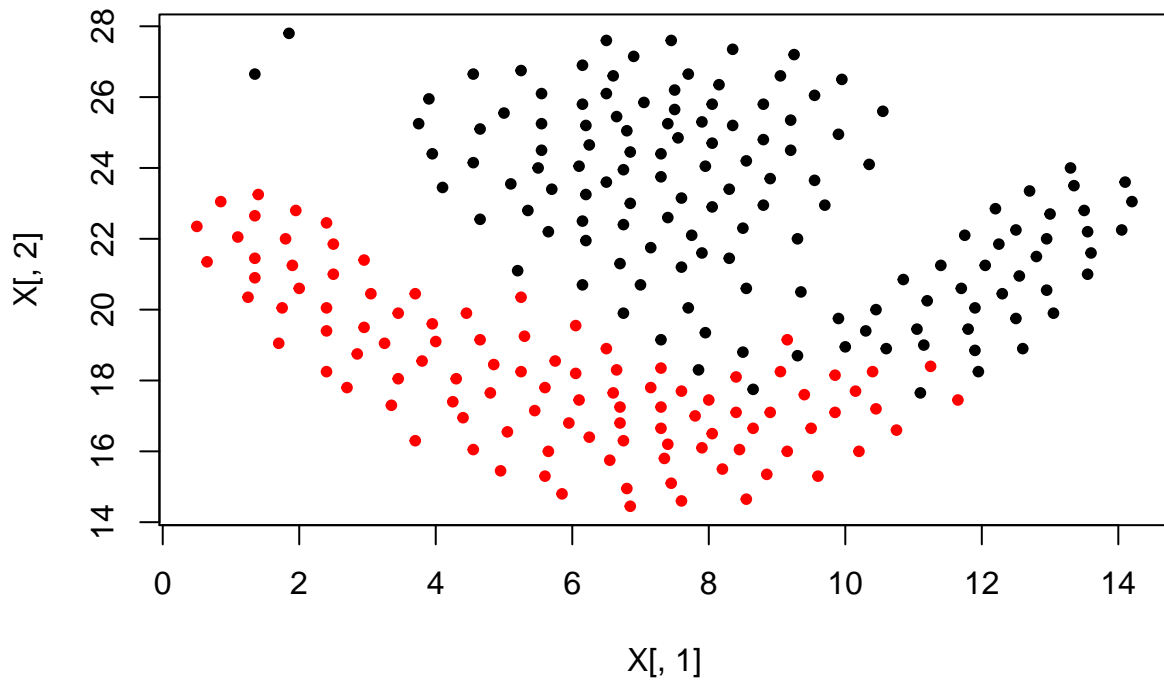
```
test_spectral_single <- function(benchmark, dataset, M=20, k=NULL, scale=FALSE, plot=TRUE){
  data <- read_data(benchmark, dataset)
  X <- data$X
  if(scale){
    X <- scale(X)
  }
  Y <- data$Y
  if(is.null(k)){
    k = length(unique(unlist(Y)))
  }
  set.seed(42) # because kmeans in spectral clustering randomly initializes centers
  Y_pred <- spectral_clustering(X, k, M)
  if(plot){
    plot_data(X, Y_pred, paste(paste(benchmark, dataset, sep="/"), ": spectral ", "M=", M, sep=""))
  }
  print(paste("FM:", FM_index(Y, Y_pred), " AR:", adjustedRandIndex(Y, Y_pred), sep=" "))
}
```

Spójrzmy teraz jak zadziała nasz algorytm dla różnych M. Zwróćmy uwagę, że wyeliminowaliśmy losowość wynikającą z losowego przyporządkowywania początkowych wartości dla centrowych punktów w algorytmie kmeans.

```
Ms <- c(2, 10, 20, 50)
benchmark <- "sipu"; dataset <- "flame"
for(m in Ms){
  print(m)
  test_spectral_single(benchmark, dataset, M=m)
}
```

```
## [1] 2
```

sipu/flame: spectral M=2

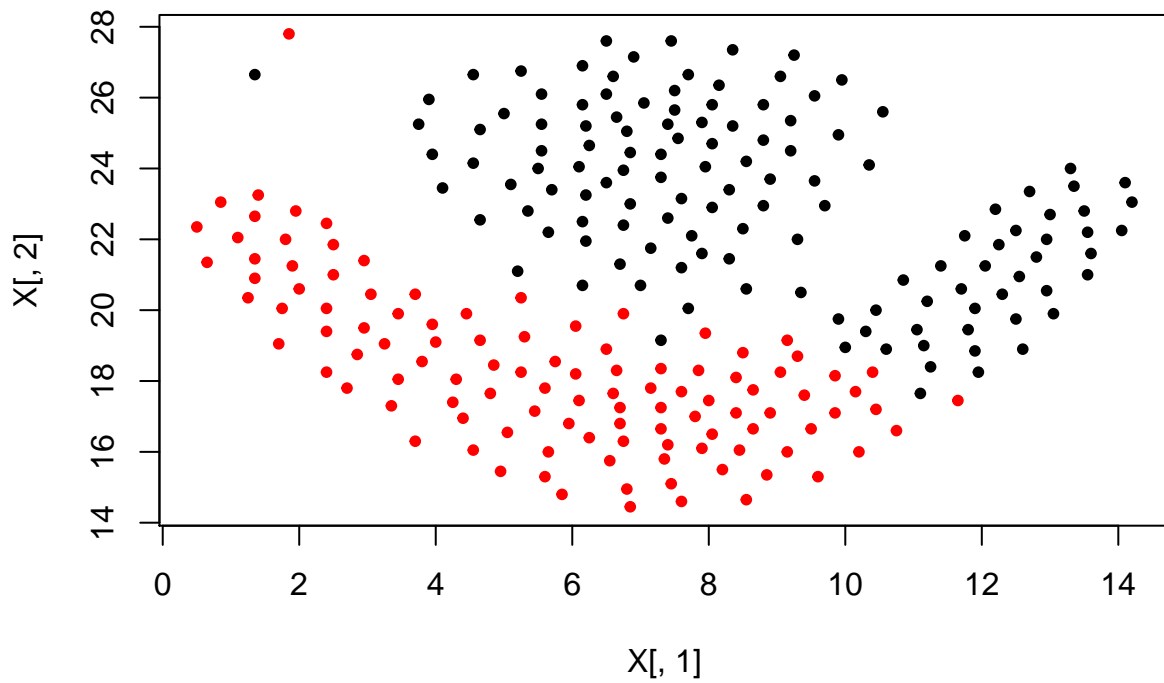


```

## [1] "FM: 0.687308991818585  AR: 0.346709333039244"
## [1] 10

```

sipu/flame: spectral M=10

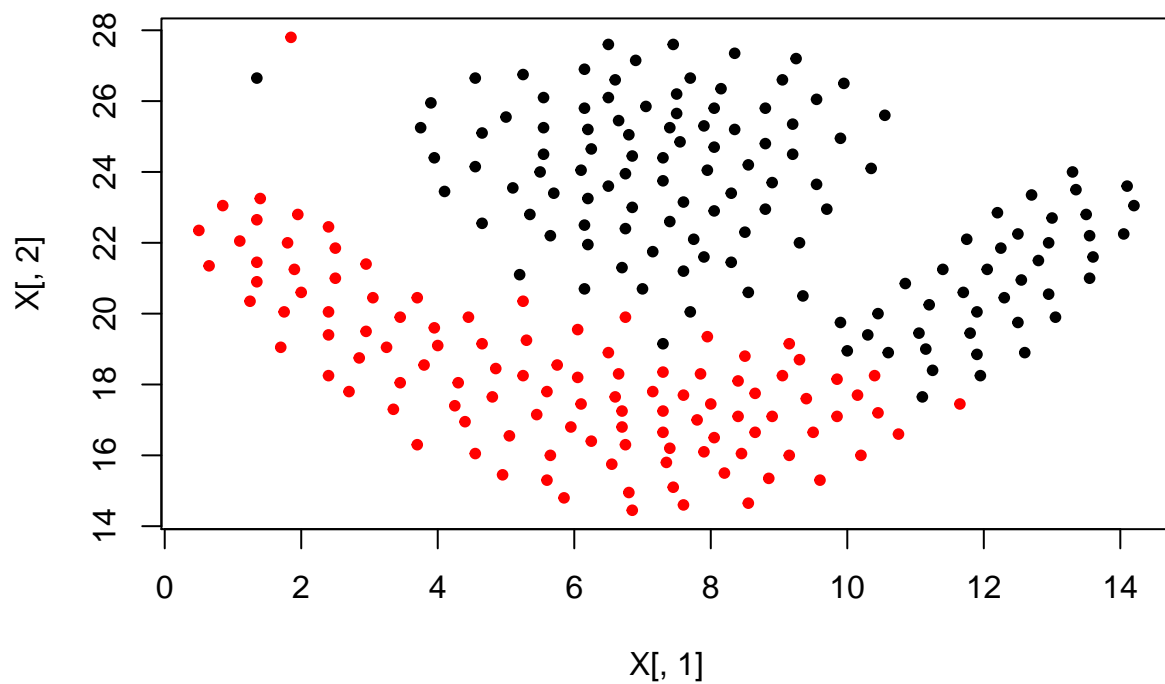


```

## [1] "FM: 0.705422032154052  AR: 0.387952806505857"
## [1] 20

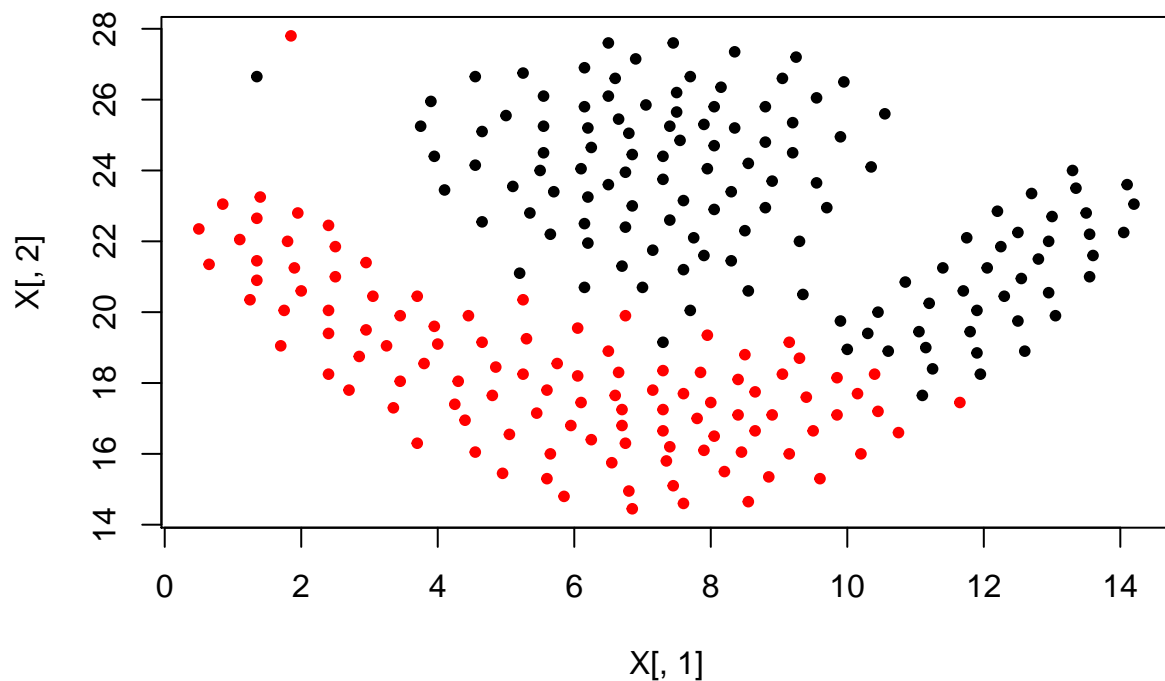
```

sipt/flame: spectral M=20



```
## [1] "FM: 0.705422032154052  AR: 0.387952806505857"
## [1] 50
```

sipt/flame: spectral M=50



```
## [1] "FM: 0.705422032154052  AR: 0.387952806505857"
```

Jak widzimy dla małego M algorytm działa trochę gorzej, natomiast dla wyższych zwiększanie M nie zmienia

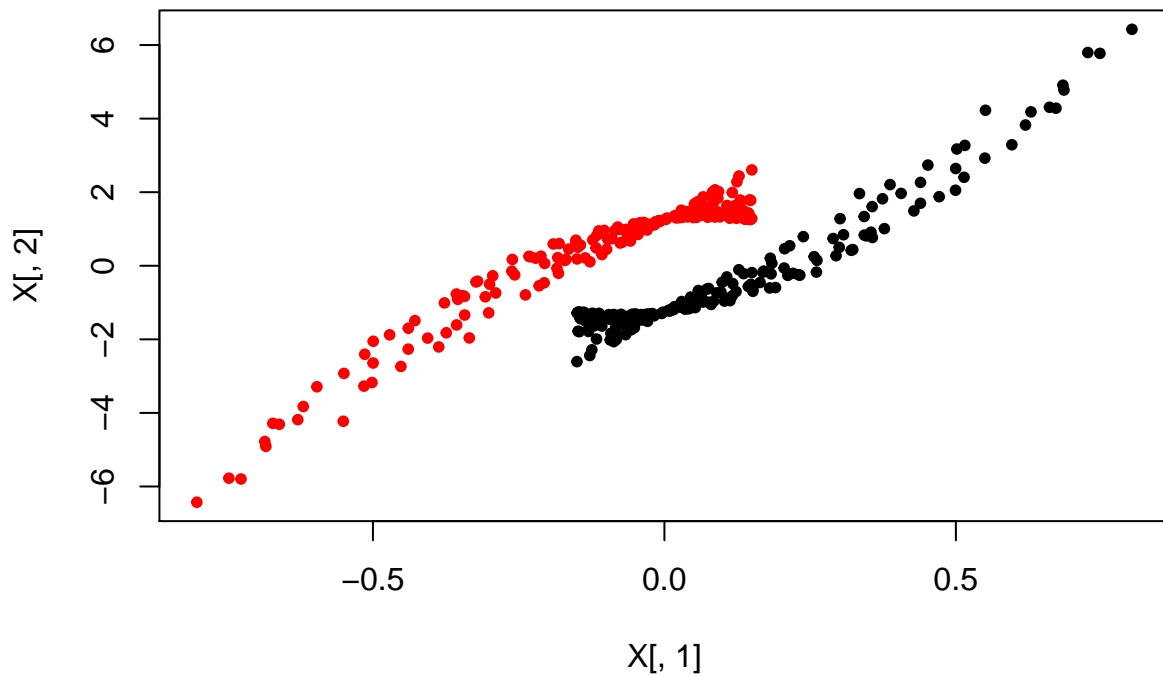
tego jak algorytm przyporządkowuje punkty.

Sprawdźmy jeszcze na to samo dla innych danych:

```
Ms <- c(2, 10, 20, 50)
benchmark <- "wut"; dataset <- "twosplashes"
for(m in Ms){
  print(m)
  test_spectral_single(benchmark, dataset, M=m)
}
```

```
## [1] 2
```

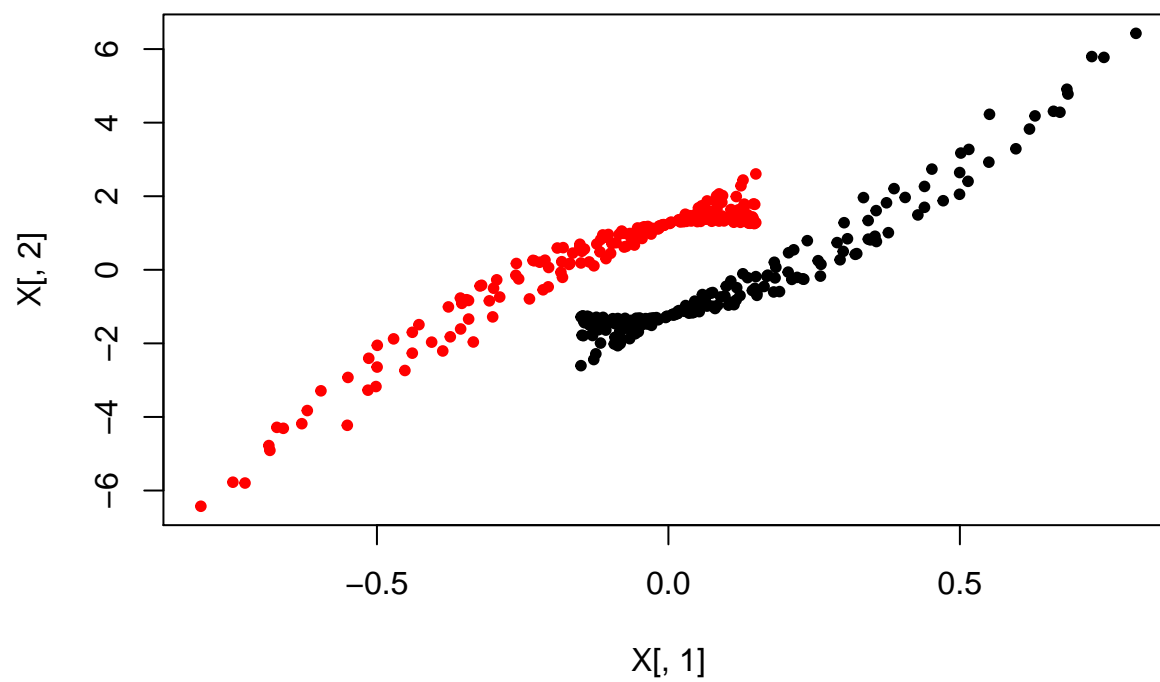
wut/twosplashes: spectral M=2



```
## [1] "FM: 1  AR: 1"
```

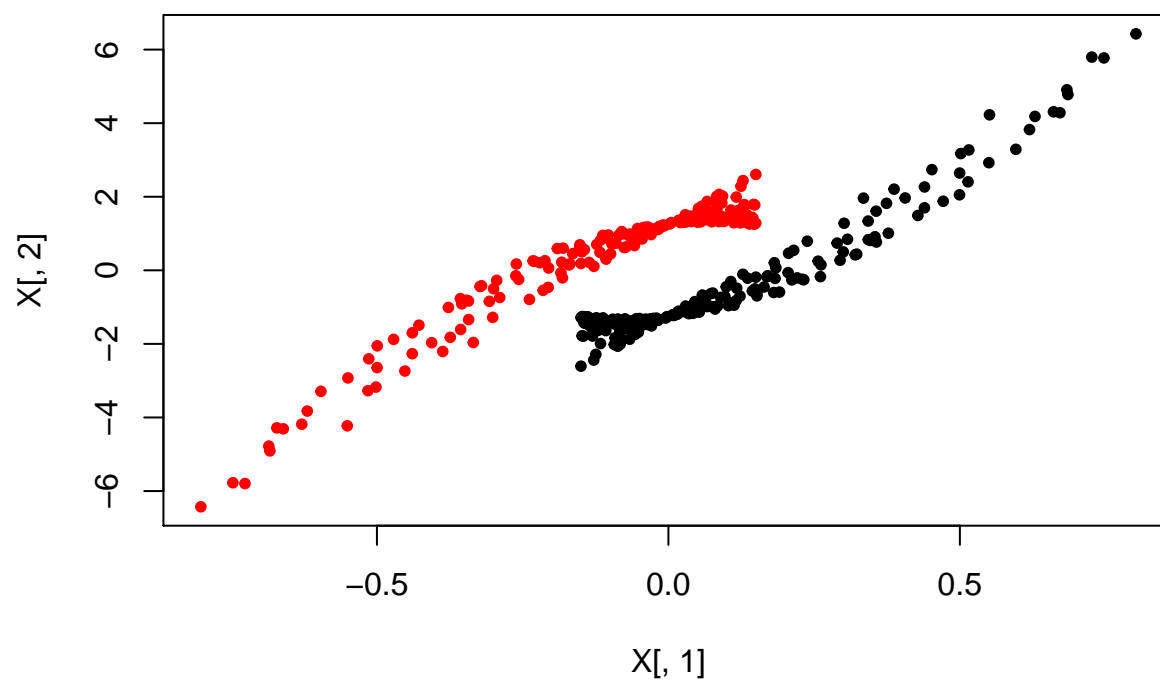
```
## [1] 10
```

wut/twoslashes: spectral M=10



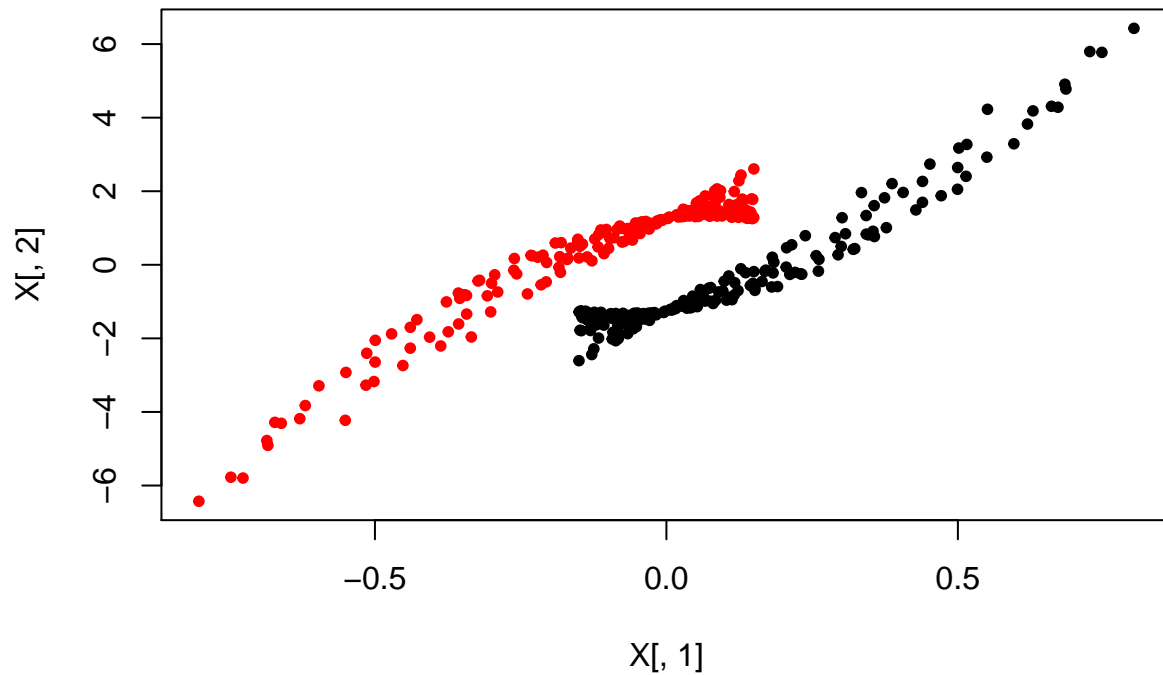
```
## [1] "FM: 1  AR: 1"  
## [1] 20
```

wut/twoslashes: spectral M=20



```
## [1] "FM: 1  AR: 1"  
## [1] 50
```


wut/twosplashes: spectral M=50



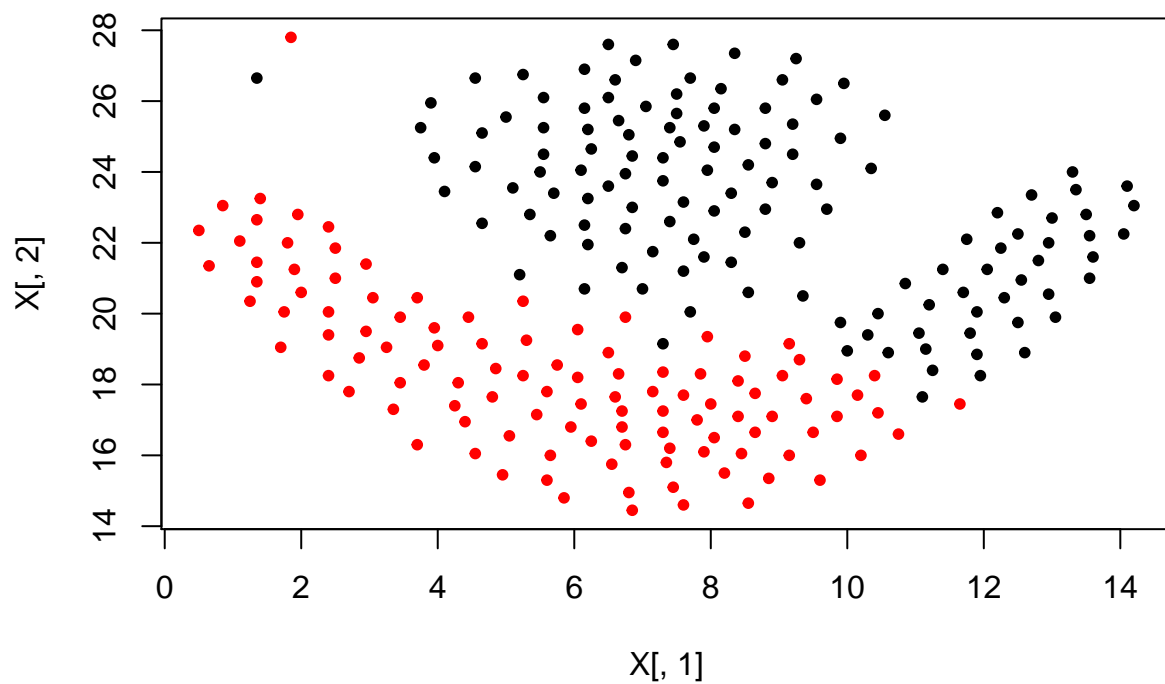
```
## [1] "FM: 1  AR: 1"
```

Wyniki potwierdzają naszą tezę, że działanie algorytmu nie zmienia się wraz ze zmianą M o ile M nie jest zbyt małe. Dlatego jako domyślny parametr można przyjąć na przykład $M=10$.

Jak skalowanie wpływa na wyniki?

```
benchmark <- "sipu"; dataset <- "flame"  
test_spectral_single(benchmark, dataset, scale=FALSE)
```

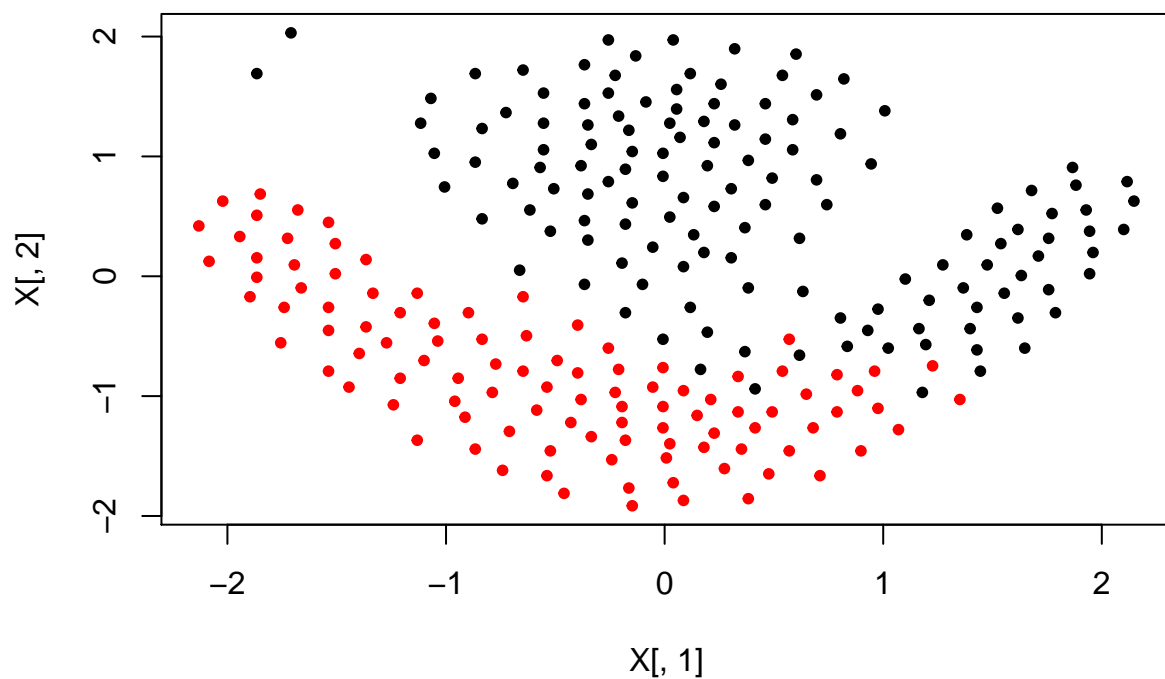
siyu/flame: spectral M=20



```
## [1] "FM: 0.705422032154052  AR: 0.387952806505857"
```

```
test_spectral_single(benchmark, dataset, scale=TRUE)
```

siyu/flame: spectral M=20

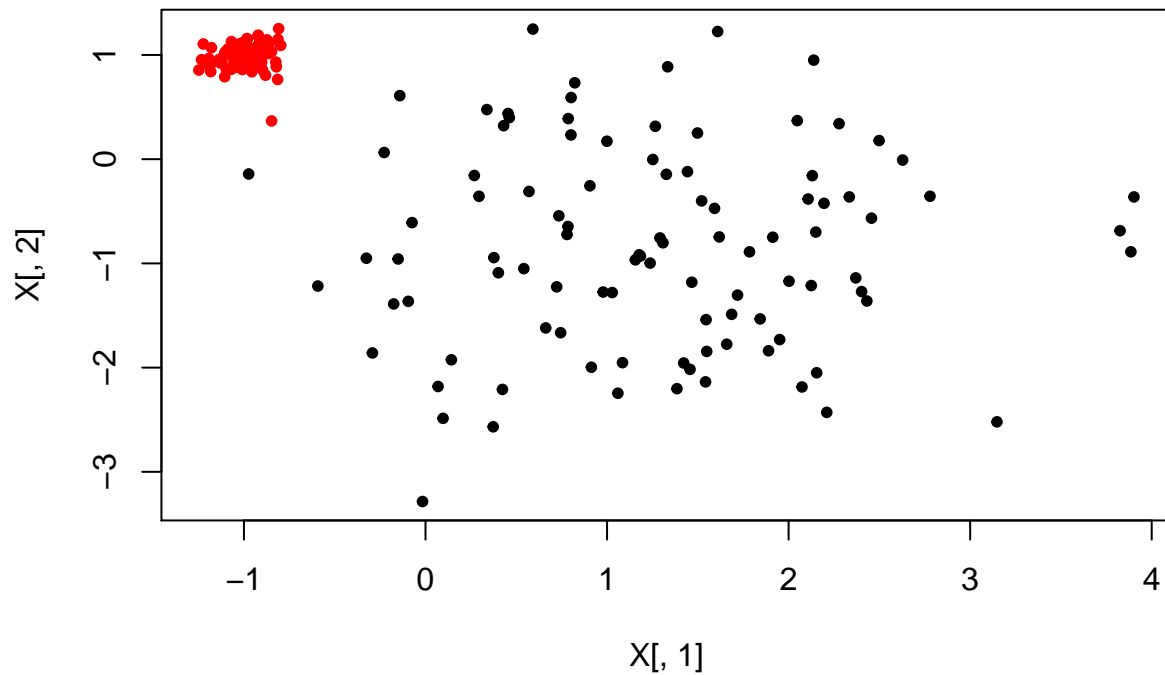


```
## [1] "FM: 0.687308991818585  AR: 0.346709333039244"
```

W tym przypadku skalowanie trochę pogorszyło działanie algorytmu. Natomiast dla innych zbiorów działanie może być różne:

```
test_spectral_single("graves", "dense", M=20)
```

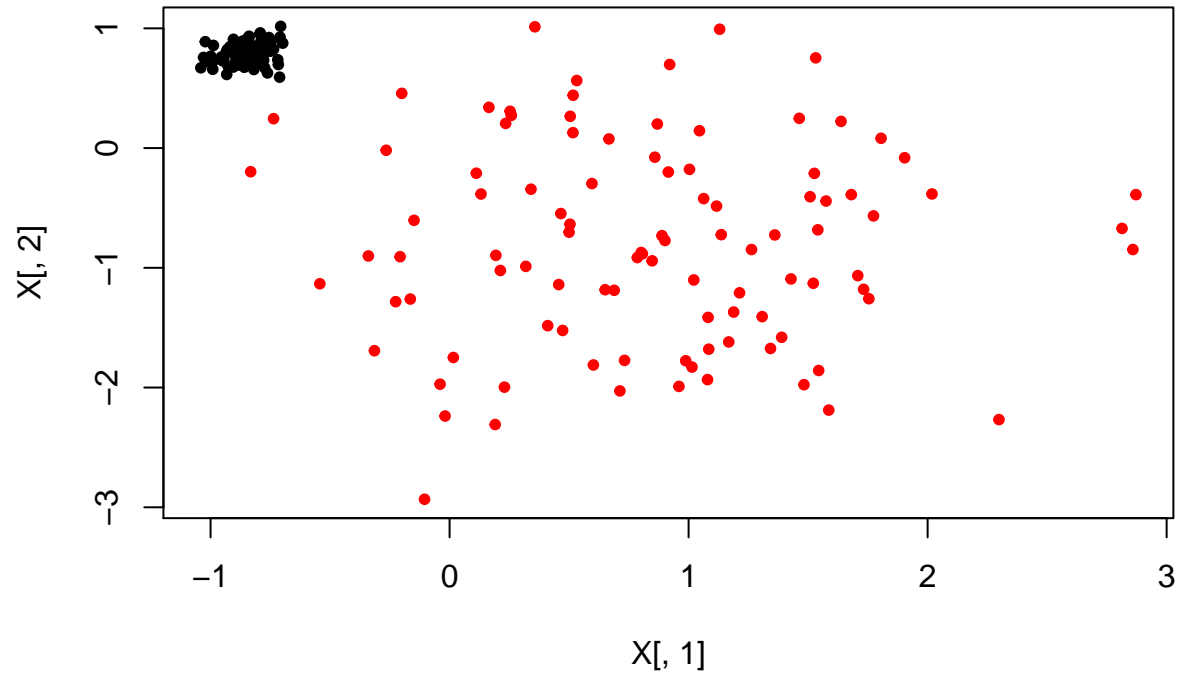
graves/dense: spectral M=20



```
## [1] "FM: 0.98995000378756  AR: 0.979999505050755"
```

```
test_spectral_single("graves", "dense", M=20, scale=TRUE)
```

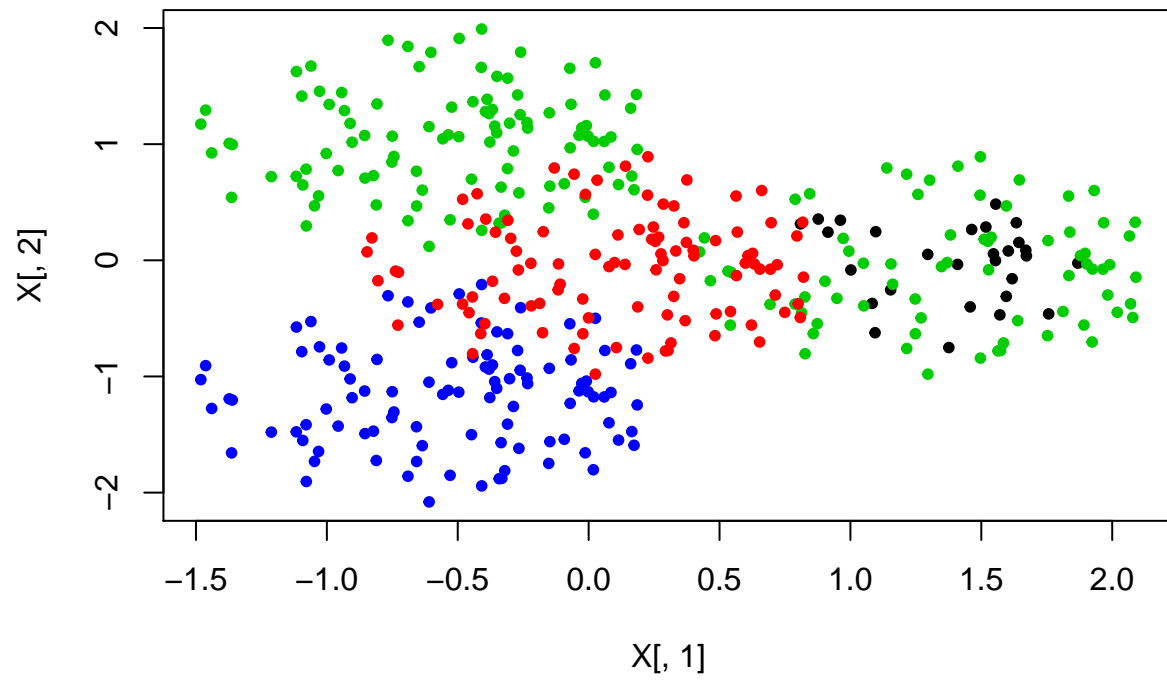
graves/dense: spectral M=20



```
## [1] "FM: 1 AR: 1"
```

```
test_spectral_single("fcps", "tetra", M=20)
```

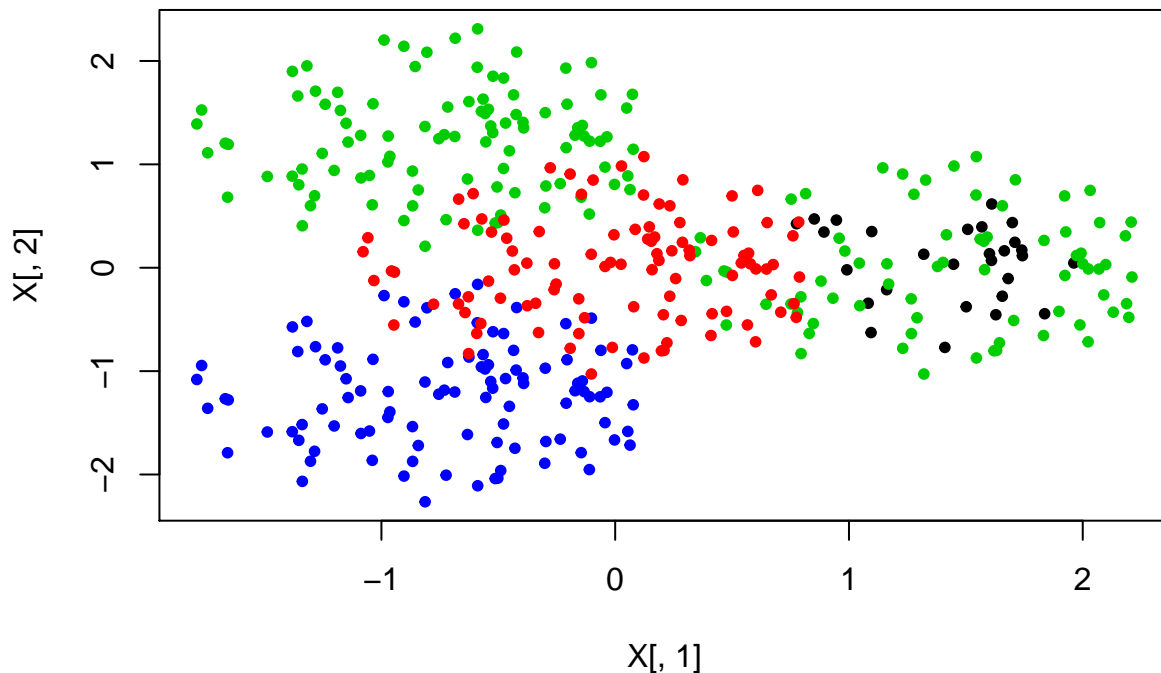
fcps/tetra: spectral M=20



```
## [1] "FM: 0.799587265813454 AR: 0.715431933672105"
```

```
test_spectral_single("fcps", "tetra", M=20, scale=TRUE)
```

fcps/tetra: spectral M=20



```
## [1] "FM: 0.799587265813454  AR: 0.715431933672105"
```

Wynika stąd, że skalowanie nie ma wielkiego wpływu na jakość naszego algorytmu. Już podczas testów zauważyliśmy, że algorytm nie przywiązuje dużej uwagi do bliskości punktów i był w stanie łączyć punkty odległe do siebie. Zatem i skalowanie niewiele zmienia w jego działaniu. Intuicja jest taka, że w tej metodzie wykorzystujemy tylko indeksy najbliższych punktów, a zatem konkretne wielkości nie mają takiego znaczenia.

Wnioski

Nasz algorytm `spectral clustering` działa całkiem nieźle nawet w porównaniu do różnych algorytmów R-owych. Charakteryzuje się tym, że w przeciwieństwie do na przykład `kmeans` nie przywiązuje uwagi do skupisk które są blisko siebie i udaje mu się wychwycić “podobieństwa” między punktami leżącymi daleko od siebie. Zaobserwowaliśmy, że hiperparametr `M` nie ma dużego wpływu na jakość wykonywanej analizy skupień. Ogólnie `spectral clustering` nie działa najszybciej, dlatego zaimplementowaliśmy go w Rcpp, co trochę poprawiło czas wykonywania analizy skupień.