Spectrum

Christopher R John

2019-10-07

Spectrum is a fast adaptive spectral clustering method for single or multi-view data (John et al., 2019). Spectrum uses a density-aware kernel that increases similarities between points that share common nearest neighbours. It uses a recently developed tensor product graph data integration and diffusion procedure for integrating different data sources and reducing noise (Shu et al., 2016). Spectrum contains the classical eigengap and multimodality gap heuristics for determining the number of clusters (K). Spectrum includes an ultra-fast mode for clustering massive single-view datasets and it can impute missing data in multi-view analyses. The method is sufficiently flexible so it can generalise well to a wide range of Gaussian and non-Gaussian structures with automatic selection of K.

Contents

1. Data types and requirements

- Data must be in a data frame (single-view) or list of data frames (multi-view)
- Data must have points (samples to cluster) as columns and rows as features
- Data should be normalised appropiately so the points are comparable
- Data should be transformed appropiately so different features are comparable
- Z-score normalisation is not necessary
- Data must be on a continuous or binary scale
- Multi-view data must have the same number of points in each view and column IDs must be in the same order
- For ultra-fast mode this is only for single-view data
- Multi-view data imputation requires at least one view with data

2. Quick start parameter settings

- For most users, we recommend the default settings which runs the main algorithm (method=1)

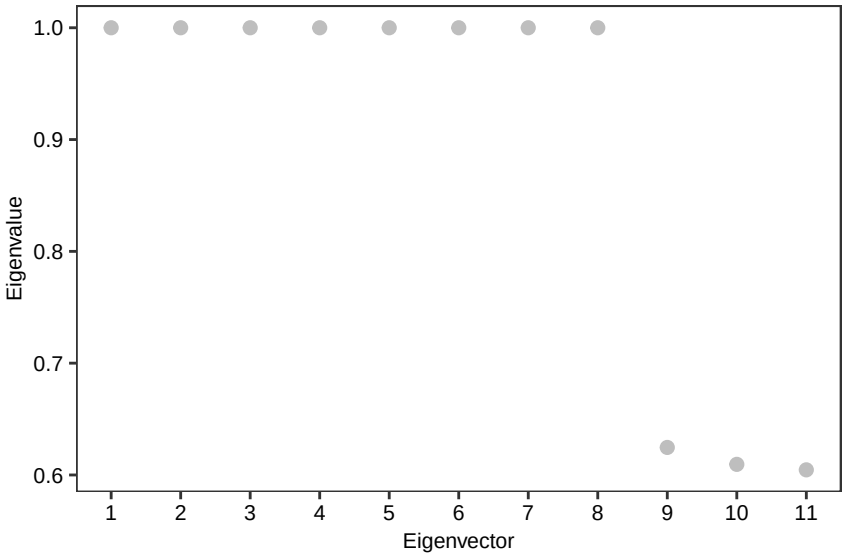
- For non-Gaussian structures, such as complex shapes, use method=2
- For spectral clustering without finding K automatically, use method=3 and set the fixk parameter
- For more than 10,000 points, Spectrum can be set in ultra-fast mode, see Section 10
- For clustering a range of K, see Section 5

3. Single-view clustering: Gaussian blobs

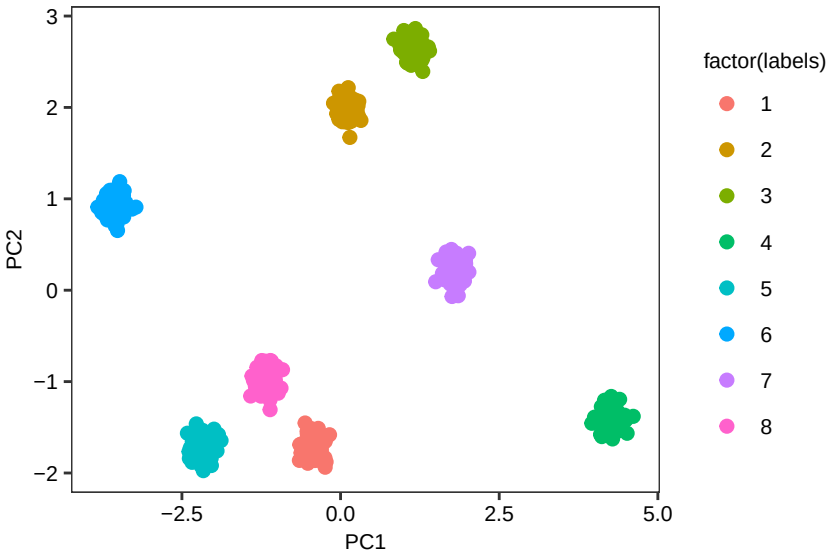
Here we cluster a simulated dataset consisting of several Gaussian blobs. This could represent a number of real world problems, for example, clustering a single omic data platform (RNA-seq, miRNA-seq, protein, or single cell RNA-seq). Method 1 is set as the default when using Spectrum which uses the eigengap method to find K. We recommend this for most Gaussian clustering tasks. Method 2 which uses the multimodality gap method which can detect K for non-Gaussian structures as well.

The first plot will show the eigenvalues, where the greatest gap is used to decide K, the second plot shows PCA results. Spectrum can also run t-SNE or UMAP on the similarity matrix as an alternative data visualisation technique (type ?Spectrum).

```
library(Spectrum)
test1 <- Spectrum(blobs,showpca=TRUE,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 1
#> method: 1
#> kernel: density
#> calculating similarity matrix 1
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvalues to select K...
#> optimal K: 8
#> doing GMM clustering...
#> done.
```



```
#> finished.
```



Spectrum generates a number of outputs for the user including the cluster each point is within in the 'assignments' vector contained in the results list (see below code). Use a '$' sign to access the results contained in the list's elements and for more information, see ?Spectrum. Cluster assignments will be in the same order as the input data.

```
names(test1)
#> [1] "assignments"          "eigenvector_analysis" "K"
#> [4] "similarity_matrix"    "eigensystem"
```

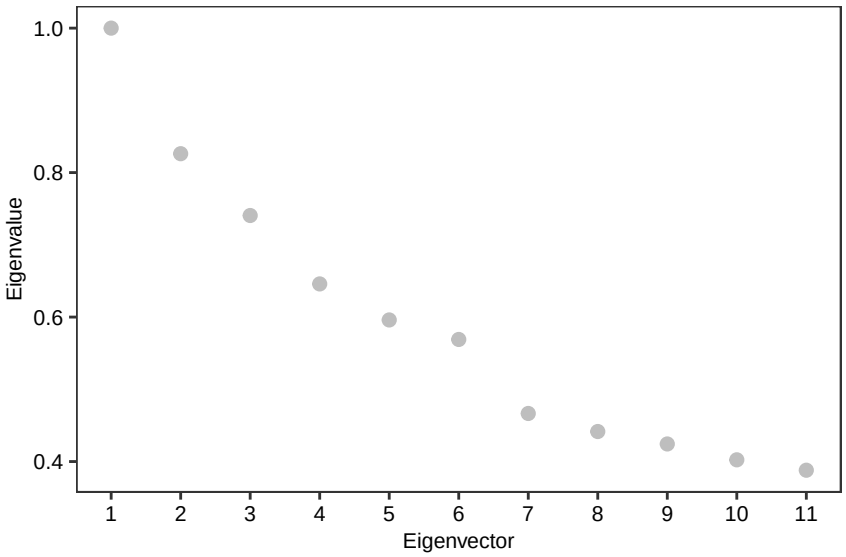
4. Single-view clustering: Brain cancer RNA-seq

Here we cluster a brain cancer RNA-seq dataset with 150 points again using the eigengap method. The point size has been reduced because of the CRAN package guidelines.

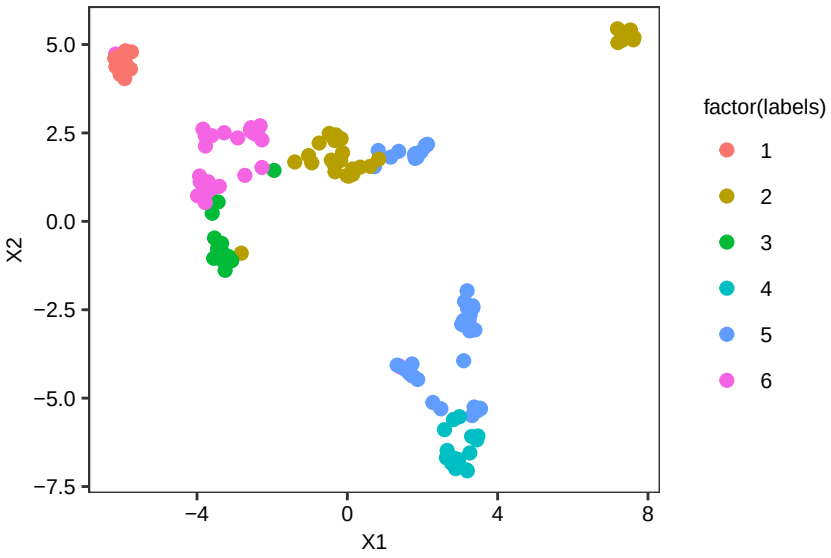
The first plot will show the eigenvalues where the greatest gap in used to decide K, the second plot shows the results from running UMAP on the diffused similarity matrix.

```
library(Spectrum)
RNAseq <- brain[[1]]
test2 <- Spectrum(RNAseq,showdimred=TRUE,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 1
#> method: 1
#> kernel: density
#> calculating similarity matrix 1
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvalues to select K...
#> optimal K: 6
```

```
#> doing GMM clustering...
#> done.
#> running UMAP on similarity matrix...
```




```
#> done.
#> finished.
```



5. Single-view clustering: Brain cancer RNA-seq clustering a range of K

There will be some cases where the user would like Spectrum to return clustering assignments for each K in a range so the results can be compared with other variables such as, histology, survival time, etc. There is an extra parameter that can be used for this.

```
library(Spectrum)
RNAseq <- brain[[1]]
test3 <- Spectrum(RNAseq,showres=FALSE,runrange=TRUE,krangemax=10)
#> ***Spectrum***
#> detected views: 1
```

```
#> method: 1
#> kernel: density
#> calculating similarity matrix 1
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvalues to select K...
#> optimal K: 6
#> clustered.
#> clustered.
#> clustered.
#> clustered.
#> clustered.
#> clustered.
#> clustered.
#> clustered.
#> clustered.
#> finished.
```

Then the point assignments for each K can be extracted as follows. For the desired K replace the number with this K in the below code, this extracts the assigments from the appropiate element of the list.

```
head(test3[[2]]$assignments)
#> [1] 2 1 2 2 1 1
```

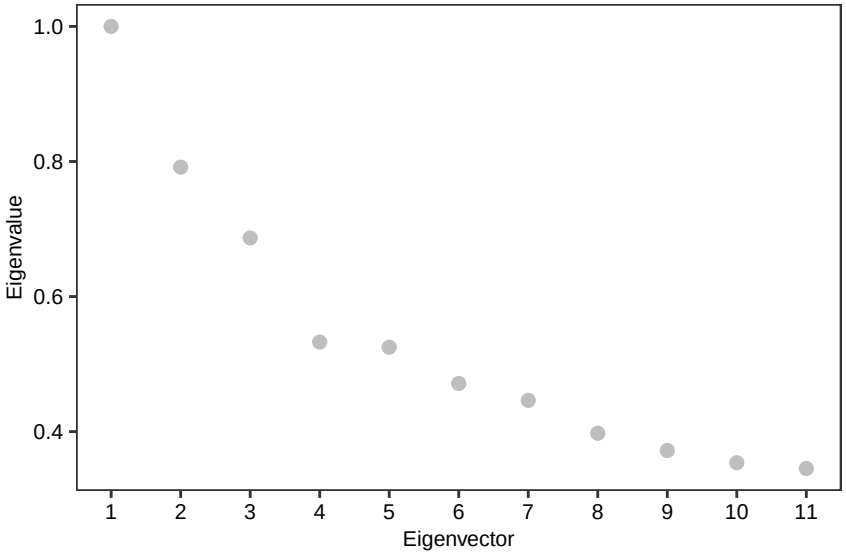
6. Multi-view clustering: Brain cancer multi-omics

Here we cluster multi-omic cancer data with three different platforms (or views): mRNA, miRNA, and protein expression data. This example uses Spectrum's tensor product graph data integration method to combine heterogeneous data sources.

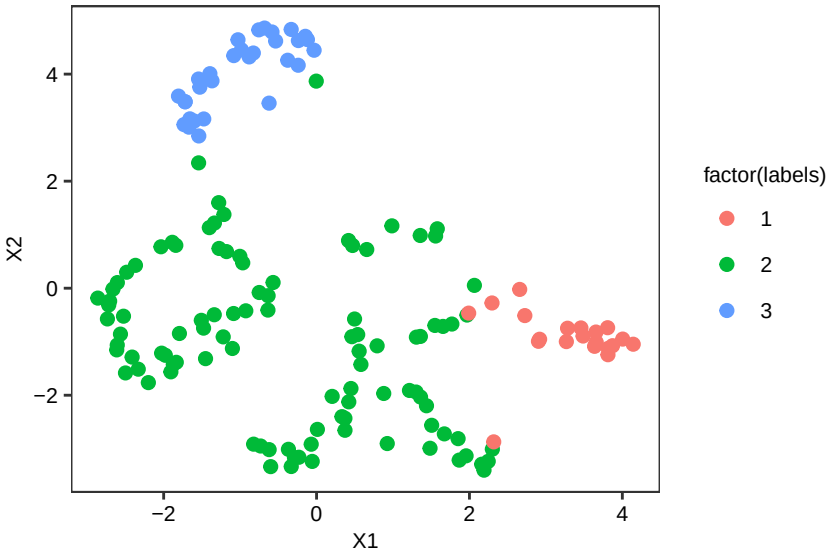
The first plot will show the eigenvalues where the greatest gap in used to decide K, the second plot shows the results from running UMAP on the combined similarity matrix.

```
library(Spectrum)
test4 <- Spectrum(brain,showdimred=TRUE,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 3
#> method: 1
#> kernel: density
#> calculating similarity matrix 1
#> done.
#> calculating similarity matrix 2
#> done.
#> calculating similarity matrix 3
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
```

```
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvalues to select K...
#> optimal K: 3
#> doing GMM clustering...
#> done.
#> running UMAP on similarity matrix...
```




```
#> done.
#> finished.
```

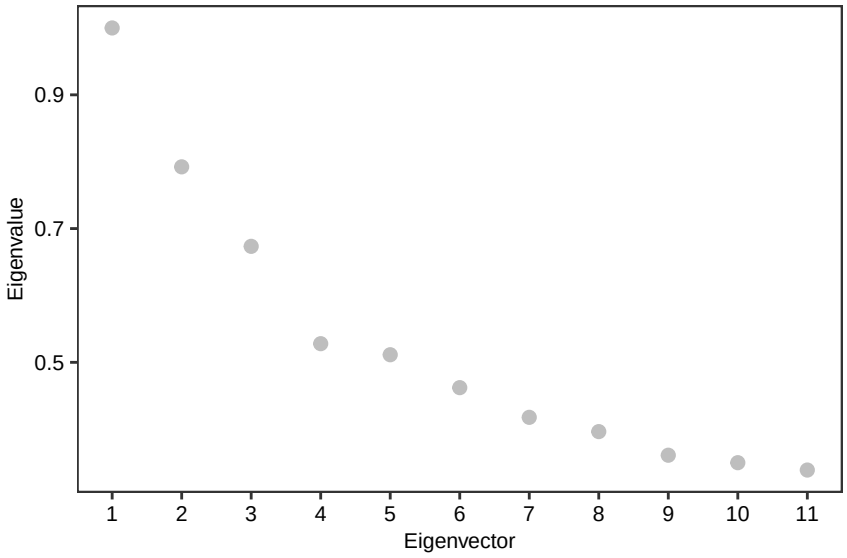


7. Multi-view clustering: Brain cancer multi-omics with missing data

Same as last multi-omic cluster analysis, but we will impute missing data. This works by mean imputation at the similarity matrix level as previously described (Rappoport et al., 2018).

We need to set missing=TRUE to run this code. Column IDs for points are required for this method to work and at least one view with measurements is required.

```
library(Spectrum)
brain1 <- brain[[1]]
brain2 <- brain[[2]]
brain3 <- brain[[3]]
brain1 <- brain1[,-5:-10]
brain_m <- list(brain1,brain2,brain3)
test4 <- Spectrum(brain_m,missing=TRUE,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 3
#> method: 1
#> kernel: density
#> calculating similarity matrix 1
#> done.
#> calculating similarity matrix 2
#> done.
#> calculating similarity matrix 3
#> done.
#> imputing missing data...
#> imputed.
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvalues to select K...
#> optimal K: 3
#> doing GMM clustering...
#> done.
#> finished.
```

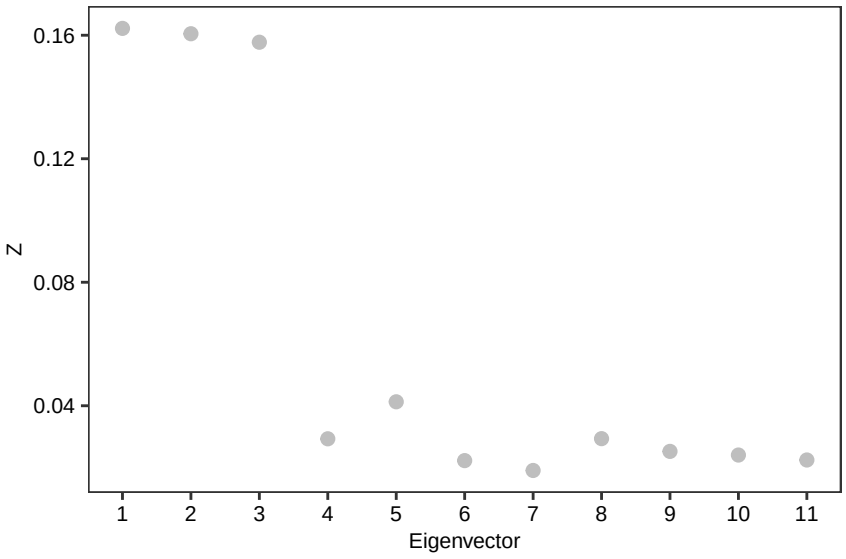


8. Single-view clustering: Non-Gaussian data, 3 circles

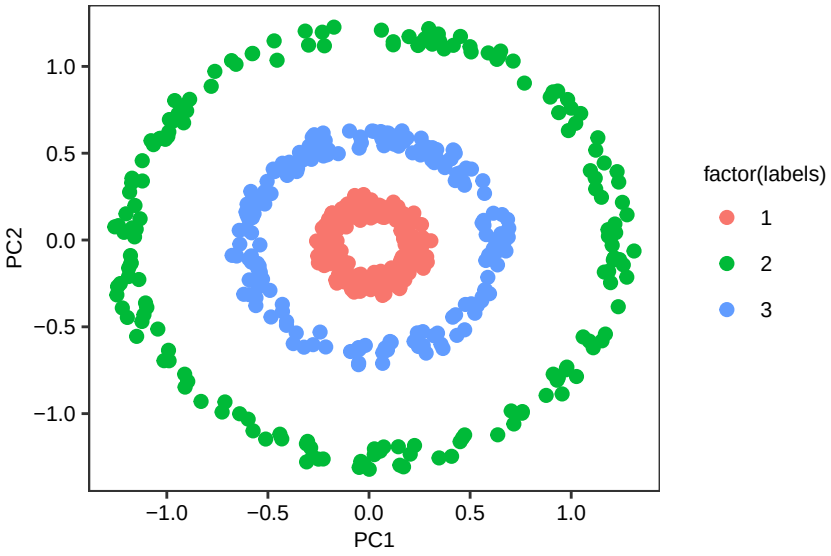
For analysing non-Gaussian structures and determining K automatically, it is much better to use our method that examines the multimodality of the eigenvectors and searches for the last substantial drop. This method, by default, also tunes the kernel by examining the multimodality gaps, although this is not always necessary. The method can handle Gaussian clusters too, multimodality is quantified using the well known dip-test statistic (Hartigan et al., 1985).

In the plot, we have the dip test statistics (Z) which measure the multimodality of the eigenvectors. When the multimodality is higher, the eigenvectors are more informative and represent blocks of the data's similarity matrix. Thus, when there is a big gap this may correspond to the optimal K as we have no more blocks to find. However, Spectrum has its own greedy algorithm to search for 'the last substantial gap' instead of the greatest gap. This is because searching for the greatest gap sometimes gets stuck in local minima without including all informative eigenvectors. The parameters for the search can be adjusted with the 'thresh' and 'frac' parameters. The last plot shown here is PCA to visualise the data, run just on the input data for the user.

```
library(Spectrum)
test5 <- Spectrum(circles,showpca=TRUE,method=2,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 1
#> method: 2
#> kernel: density
#> calculating similarity matrix 1
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvector distributions to select K...
#> finding informative eigenvectors...
#> done.
#> optimal K: 3
#> doing GMM clustering...
#> done.
```




```
#> finished.
```



9. Single-view clustering: Non-Gaussian data, spirals

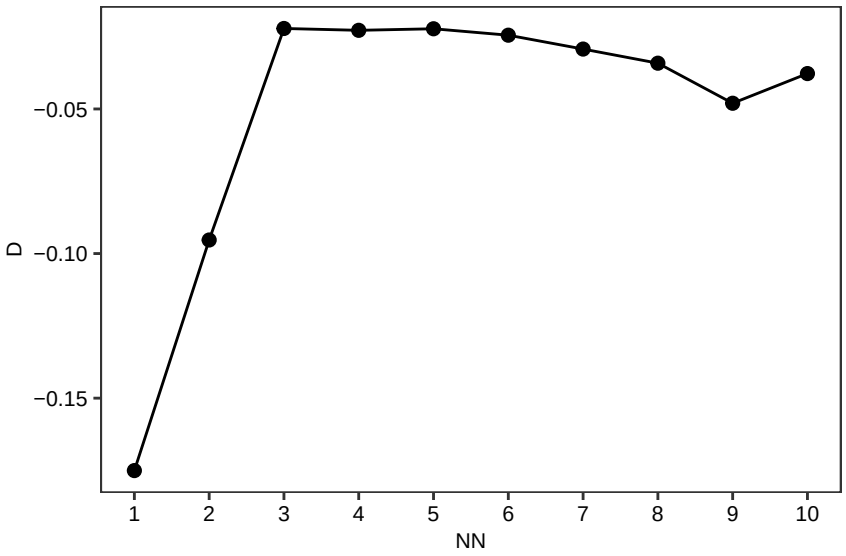
Same as the last example, but for the spirals dataset.

In this example, kernel tuning is required to detect the optimal K. Kernel tuning using our method is best saved for datasets containing unusual shapes (e.g spirals, letters, etc) that are not well recognised with the standard kernel parameters. It does add significant computational time.

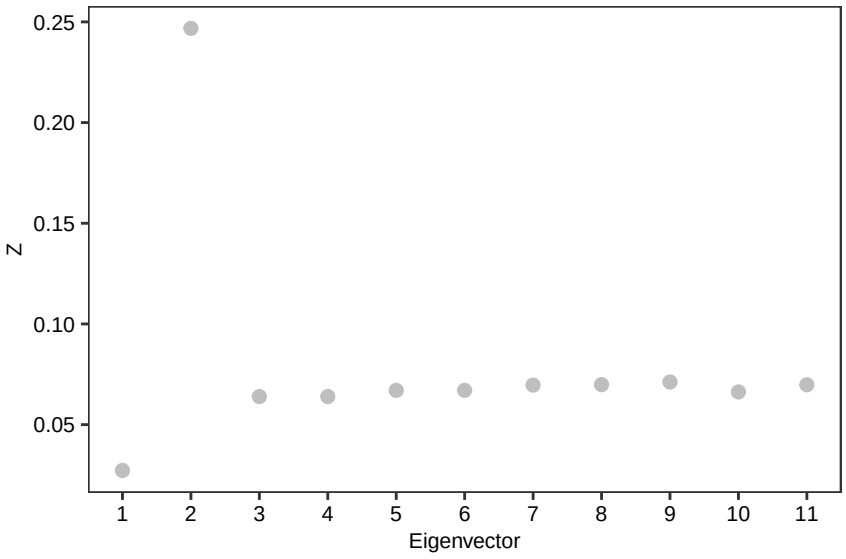
The kernel tuning plot shows the results from tuning the kernel's nearest neighbour parameter (NN). D refers to the greatest difference in dip-test statistics between any consecutive eigenvectors of the graph Laplacian for that value of NN. Spectrum automatically searches for a minimum (corresponding to the maximum drop in multimodality) to find the optimal kernel.

```
library(Spectrum)
test6 <- Spectrum(spirals,showpca=TRUE,method=2,tunekernel=TRUE,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 1
```

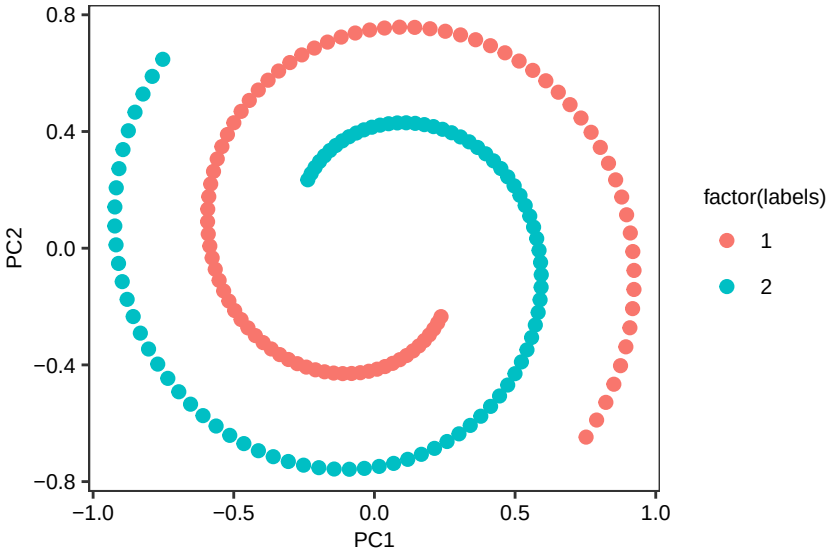
```
#> method: 2
#> kernel: density
#> calculating similarity matrix 1
#> finding optimal NN kernel parameter by examining eigenvector distributions
#> tuning kernel NN parameter: 1
#> tuning kernel NN parameter: 2
#> tuning kernel NN parameter: 3
#> tuning kernel NN parameter: 4
#> tuning kernel NN parameter: 5
#> tuning kernel NN parameter: 6
#> tuning kernel NN parameter: 7
#> tuning kernel NN parameter: 8
#> tuning kernel NN parameter: 9
#> tuning kernel NN parameter: 10
#> optimal NN:1
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvector distributions to select K...
#> finding informative eigenvectors...
#> done.
```




```
#> optimal K: 2
#> doing GMM clustering...
#> done.
```




```
#> finished.
```



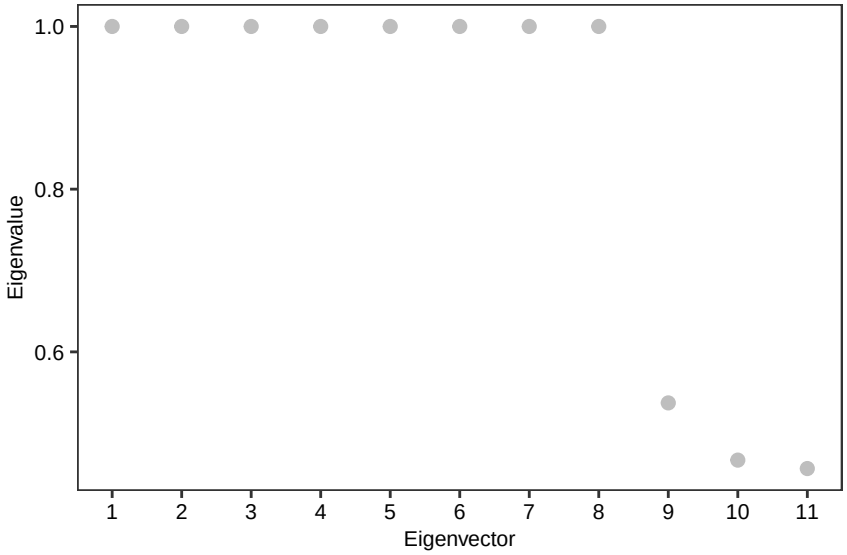
10. Ultra-fast single-view clustering: Gaussian blobs II

To enable clustering of very high numbers of points (e.g. 10,000-100,000+) on a single core of a Desktop computer, Spectrum implements the Fast Approximate Spectral Clustering (FASP) method (Yan et al., 2009). FASP computes k centroids and then uses these for clustering, after running Spectrum will then assign the original data to clusters via their centroids. This option is recommended for very large datasets, the loss in accuracy is usually marginal, see Yan et al. (2009) for further details.

To perform this method the user needs to set the FASPk parameter to a suitable number for their data, this depends on the data to some degree. For example, if we had 50,000 points, we might set FASPk to 1,000, to reduce the clustering data input size by 50x.

```
library(Spectrum)
test7 <- Spectrum(blobs,FASP=TRUE,FASPk=300,fontsize=8,dotsize=2)
#> ***Spectrum***
#> detected views: 1
```

```
#> method: 1
#> kernel: density
#> running with FASP data compression
#> calculating similarity matrix 1
#> done.
#> combining similarity matrices if > 1 and making kNN graph...
#> done.
#> diffusing on tensor product graph...
#> done.
#> calculating graph laplacian (L)...
#> getting eigensystem of L...
#> done.
#> examining eigenvalues to select K...
#> optimal K: 8
#> doing GMM clustering...
#> done.
#> finished.
```



Looking at the results, we have found the same K as before (8), but with a reduced point size and runtime. Observe the output below, where there is a new item in the list for the original point assignments.

```
names(test7)
#> [1] "allsample_assignments" "centroid_assignments"  "eigenvector_analysis"
#> [4] "K"                     "similarity_matrix"     "eigensystem"
```

```
head(test7[[1]])
#> c1s1 c2s1 c3s1 c4s1 c5s1 c6s1
#>    2    4    8    1    3    6
```

11. Advanced operation: Ng spectral clustering

It is possible to run a series of spectral clustering modules using Spectrum for certain workflows. This is the Ng spectral clustering algorithm with a heuristic to find the optimal sigma for the Ng kernel and GMM for clustering the eigenvectors. The eigengap is used for estimating the optimal K.

The Ng kernel uses a global sigma that must be tuned to the data or estimated in some manner, newer kernels

such as those provided elsewhere in the package will not require a tuning or estimation step as they adapt to the data automatically. We have provided this kernel primarily because it is a historically important example.

```
library(Spectrum)
s <- sigma_finder(blobs)
#> estimated sigma:  0.0755706132483631
s1 <- ng_kernel(blobs,sigma=s)
e1 <- estimate_k(s1,showplots=FALSE)
#> egap optimal K: 8
r <- cluster_similarity(s1,k=8,clusteralg='GMM')
#> Ng spectral clustering.
#> Gaussian Mixture Modelling clustering.
```

12. Advanced operation: Customised data integration

Another option with Spectrum is integrating different data sources using different kernels. For instance, those from other packages could be used instead in the below code.

Below we provide a minimal example with two kernels internal to Spectrum, run on the same data, then integrated for a joint clustering solution.

```
library(Spectrum)
s1 <- CNN_kernel(blobs)
s2 <- CNN_kernel(blobs)
klist <- list(s1,s2)
x <- integrate_similarity_matrices(klist)
e1 <- estimate_k(x,showplots=FALSE)
#> egap optimal K: 8
r <- cluster_similarity(x,k=8,clusteralg='GMM')
#> Ng spectral clustering.
#> Gaussian Mixture Modelling clustering.
```

13. Parameter settings

Generally speaking, Spectrum is set up to handle a wide range of data automatically with its self-tuning kernel on the default settings. However, we provide the following advice in more specific circumstances.

- For a lot of noisy and not noisy Gaussian data, e.g. RNA-seq, protein arrays, etc, we have a preference for method 1 which is the eigengap.
- For very high numbers of points; switch FASP mode on and set 'FASPk' to e.g. 500-10,000. The 'FASPk' parameter should be high enough to get a good representation of the data.
- Although in our experiments we found our adaptive density aware kernel superior to the classic Zelnik-Manor et al. (2005) self tuning kernel, the user might want to experiment on their data with the Zelnik-Manor kernel (stsc) also included in Spectrum.
- The kernel parameters N and NN2 can be experimented with which control the number of nearest neighbours used when calculating the local sigma or density parameter. Lower values will prefer local structures and higher values global structures. A sensible range for N is 2-7 and NN2 is 5-15. While the default parameters have been tested with a wide range of real and simulated data and generalise well, it may be preferable to do some dataset specific optimisation by changing these parameters and observing the variables of interest.
- For data containing non-Gaussian clusters like circles and spirals; use method 2, which can automatically detect K for these structures. In some cases, the greedy search parameters, 'thresh' and 'frac', which

define the cut-offs for searching for the last substantial gap in multimodality, may require manually changing for the data type. Kernel tuning for method 2 is optional, it can help for complex structures, but for a large dataset it may be too time consuming. Especially for omic data we suggest not to use this mode.

14. Code for heatmaps

For users that want to make a heatmap from Spectrum's cluster assignment output, the following code will help. It needs to be ensured the input data's columns are ordered according to the clustering assignments. Then, when the user makes the heatmap with points as columns, they should not cluster the columns because this has been done already. The assignments can be added as an annotation track to visualise the clusters identity.

```
## 1. run my clustering algorithm yielding assignments in vector, e.g. 1,2,2,1,2,2...
## 2. reorder data according to assignments
ind <- sort(as.vector(test2$assignments),index.return=TRUE)
datax <- RNAseq[,ind$ix] ## order the original data
#annonx <- meta[ind$ix,] ## order the meta data
#annonx$cluster <- ind$x ## add the cluster to the meta data
## 3. do heatmap
# insert your favourite heatmap function
```

15. Closing comments

Spectrum is an unique assembly of spectral clustering techniques designed to serve the R community. Included are both innovations, such as the adaptive density-aware kernel and the multimodality gap procedure, as well as implementations of other state-of-the-art methods in the spectral clustering field. Please see our Bioinformatics manuscript for further details of the main method Spectrum uses.

We recommend Spectrum be used in conjuction with dimensionality reduction methods to confirm the structure and taking into account our recommendations in the parameter settings section of the vignette.

16. References

Hartigan, John A., and Pamela M. Hartigan. "The dip test of unimodality." The annals of Statistics 13.1 (1985): 70-84.

Christopher R John, David Watson, Michael R Barnes, Costantino Pitzalis, Myles J Lewis, Spectrum: fast density-aware spectral clustering for single and multi-omic data, Bioinformatics, btz704, https://doi.org/10.1093/bioinformatics/btz704

Yan, Donghui, Ling Huang, and Michael I. Jordan. "Fast approximate spectral clustering." Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009.

Rappoport, Nimrod, and Ron Shamir. "NEMO: Cancer subtyping by integration of partial multi-omic data." bioRxiv (2018): 415224.

Shu, Le, and Longin Jan Latecki. "Integration of single-view graphs with diffusion of tensor product graphs for multi-view spectral clustering." Asian Conference on Machine Learning. 2016.

Zelnik-Manor, Lihi, and Pietro Perona. "Self-tuning spectral clustering." Advances in neural information processing systems. 2005.