Basic Usage of **NetworkDistance** Package

Kisung You

1. Load

Surely, the first thing we are always bound to do is to load the package,

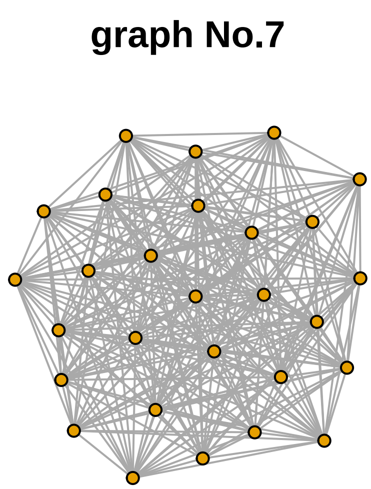
```
library(NetworkDistance)
#> **------------------------------------------------------------**
#> ** NetworkDistance - Distance Measures for Networks
#> **
#> ** Version    : 0.3.6       (2025)
#> ** Maintainer : Kisung You  (kisung.you@outlook.com)
#> **
#> ** Please share any bugs or suggestions to the maintainer.
#> **------------------------------------------------------------**
```

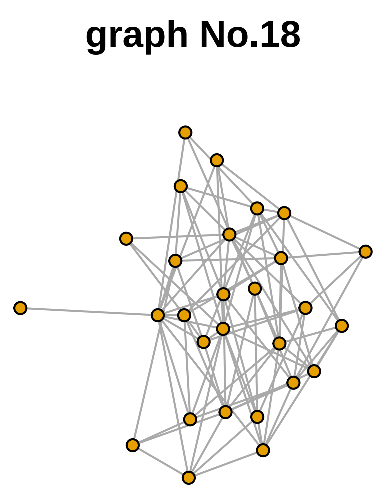
2. Computing Distances

Suppose you have N network objects represented as square adjacency matrices. All the functions in the package require your data to be in a form of `list` whose elements are your adjacency matrices. Let's load example data `graph20`.

```
data(graph20)      # use `help(graph20)' to see more details.
typeof(graph20)    # needs to be a list
#> [1] "list"
```

Before proceeding any further, since we have two types of graphs - densely and sparsely connected with $p = 0.8$ and $p = 0.2$ - we know that the distance matrix should show block-like pattern. Below is two example graphs from the dataset.

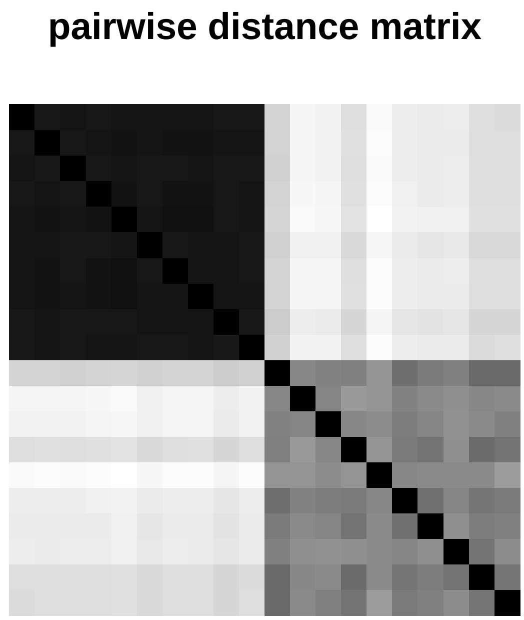




Once you have your data in such a form, all you've got is to run a single-line code to acquire distance numerics, resulting in either a `dist` class object or a square matrix. For example, let's compute *graph diffusion distance* by Hammond et al. (2013) on our example set.

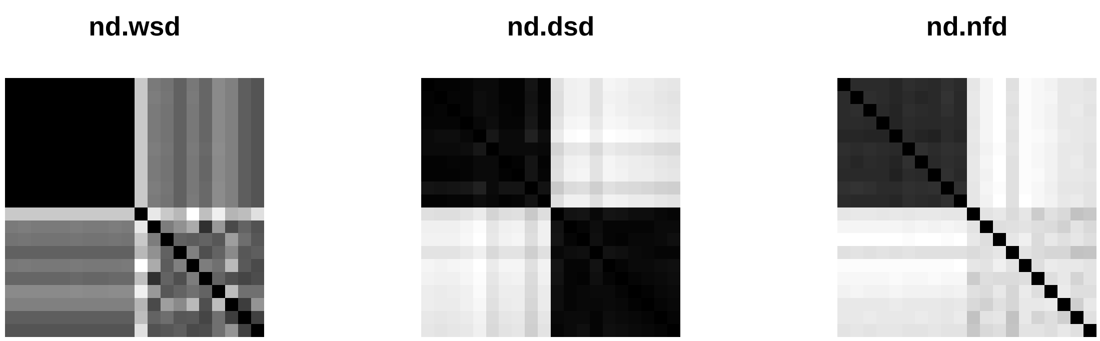
```r
dist.gdd <- nd.gdd(graph20)  # return as a 'dist' object
```

and you can see the discriminating pattern from the distance matrix `dist.gdd$D` with black represents 0 and white represents the largest positive number, indicating large deviation from 0.



Finally, let's compare different methods as well.

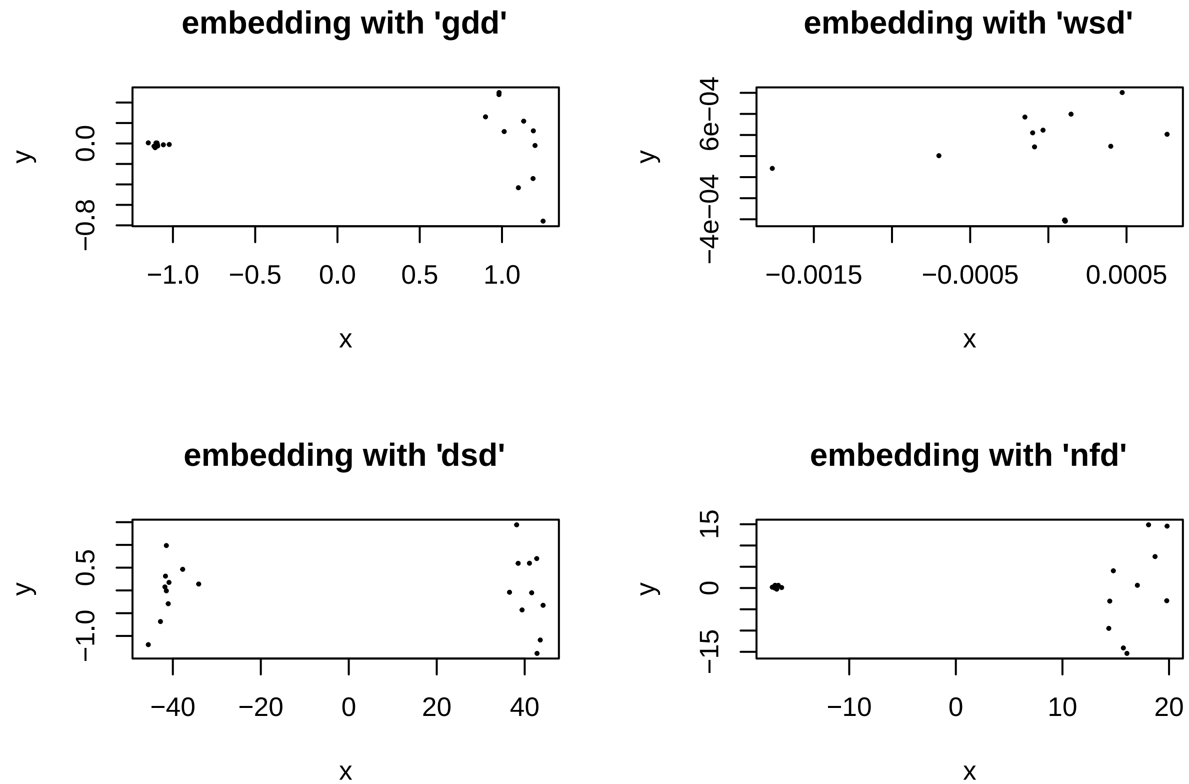
```r
dist.wsd <- nd.wsd(graph20)              # spectrum-weighted distance
dist.dsd <- nd.dsd(graph20, type="SLap") # discrete spectral measure
dist.nfd <- nd.nfd(graph20)              # network flow distance
```



3. One Application : Embedding Networks, Not Network Embedding

Our interest is focused on dealing with a collection of networks, **not** a single network. Therefore, the example we cover here is to **embed** multiple networks, not an embedding of single network and its nodes as points. We will use multidimensional scaling to embed 20 graphs we did before.

```r
gdd2 = stats::cmdscale(dist.gdd$D, k=2)  # 2-d embedding from 'gdd' distance
wsd2 = stats::cmdscale(dist.wsd$D, k=2)  #                    'wsd'
dsd2 = stats::cmdscale(dist.dsd$D, k=2)  #                    'dsd'
nfd2 = stats::cmdscale(dist.nfd$D, k=2)  #                    'nfd'
```



From the figure above, we can see that different measures/metrics reveal a variety of topological or network features. This necessitates the very existence of a package like ours to provide a set of tools for diverse perspectives on the space networks.