Manifold_Warping

Description

Knowledge transfer is computationally challenging, due in part to the curse of dimensionality. Recent work on manifold learning has shown that data collected in real-world settings often have high-dimensional representations but lie on low-dimensional manifolds.
This package is designed to align two sequentially ordered high-dimensional data sets by combining traditional manifold alignment and dynamic time warping algorithms. In each iteration, it firstly aligns two data sets by graph Laplacian and then uses a dynamic time warping method to pair them. Finally, it updates the lost matrix. One can choose linear or non-linear Laplacian method by parameter mode ('linear' or 'nonlinear'). In order to compare with embedding-only algorithm, it also can show embedding by choosing mode 'embed'. The idea and theoretical formulation are from https://people.cs.umass.edu/~ccarey/pubs/ManifoldWarping.pdf.

Installation

You can use following codes to install the package.

```
## You may need following codes to install dependent packages.
library(devtools)
install_github("emanuel996/maniwarp")
```

After this, we can use this package.

```
library(maniwarp)
```

Handwritten letter example

Here we use toy example "handwritten letter 'd'" to show our main function "manifold_warping".

```
X1 = dataset1()$X1
```

```
##
## Attaching package: 'plotly'
```

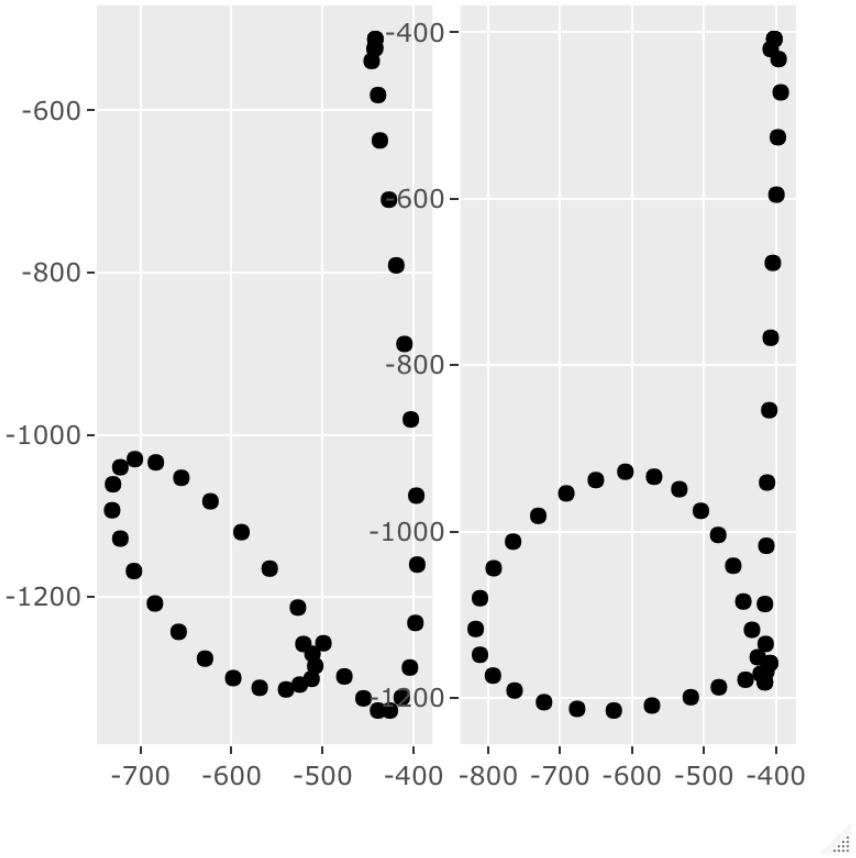
```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
X2 = dataset1()$X2
p = dataset1()$p
```

One can use packages such as "ggplot" and "plotly" for visualization.



It returns a list of 3 matrices. The first matrix is the warping path.

```
output = manifold_warping(X1, X2, mode = 'nonlinear', target_dim = 2, k = 8,
                          mu = 0.3, thresh = 0.01, max_its = 100)
```

```
## Loading required package: proxy
##
## Attaching package: 'proxy'
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
## The following object is masked from 'package:base':
##
##     as.matrix
## Loaded dtw v1.21-3. See ?dtw for help, citation("dtw") for use in publication.
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following objects are masked from 'package:pracma':
##
##     expm, lu, tril, triu
```

```
## Loading required package: tseriesChaos

## Loading required package: fields

## Loading required package: spam

## Loading required package: dotCall64

## Loading required package: grid

## Spam version 2.4-0 (2019-11-01) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following object is masked from 'package:Matrix':
##
##     det

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: maps

## See https://github.com/NCAR/Fields for
##  an extensive vignette, other supplements and source code

## Loading required package: plot3D
```

```
t(output$P)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## newY1    1    2    3    4    5    6    7    7    7     7     7     7     8
## newY2    1    2    3    3    3    3    3    4    5     6     7     8     8
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## newY1     9    10    10    10    10    11    11    11    11    11    11
## newY2     9    10    11    12    13    14    15    16    17    18    19
##       [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## newY1    11    11    11    11    12    12    12    12    12    12    12
## newY2    20    21    22    23    24    25    26    27    28    29    30
##       [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## newY1    12    13    13    13    13    13    13    13    13    13    13
## newY2    31    32    33    34    35    36    37    38    39    40    41
##       [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57]
## newY1    13    14    15    16    17    18    19    20    21    22    23
## newY2    42    43    44    44    45    45    45    45    45    45    46
##       [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66] [,67] [,68]
## newY1    24    25    26    27    28    29    30    31    32    33    34
## newY2    46    47    47    47    47    47    47    47    47    47    48
##       [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78]
## newY1    35    36    37    38    39    40    41    42    43    44
## newY2    48    48    48    48    48    48    48    48    48    48
```

The second and third matrices are the projections from the original data to the low-dimensional representation. Instead of printing the matrices, we can visualize the matrices.

```
newX1 = output$Y1
newX2 = output$Y2
par(mfrow = c(1,2))
plot(newX1[, 1], newX1[, 2], type = 'l', height = 4)
```

```
## Warning in plot.window(...): "height" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "height" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "height" is
## not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "height" is
## not a graphical parameter
```

```
## Warning in box(...): "height" is not a graphical parameter
```

```
## Warning in title(...): "height" is not a graphical parameter
```

```
plot(newX2[, 1], newX2[, 2], type = 'l', height = 4)
```

```
## Warning in plot.window(...): "height" is not a graphical parameter
```

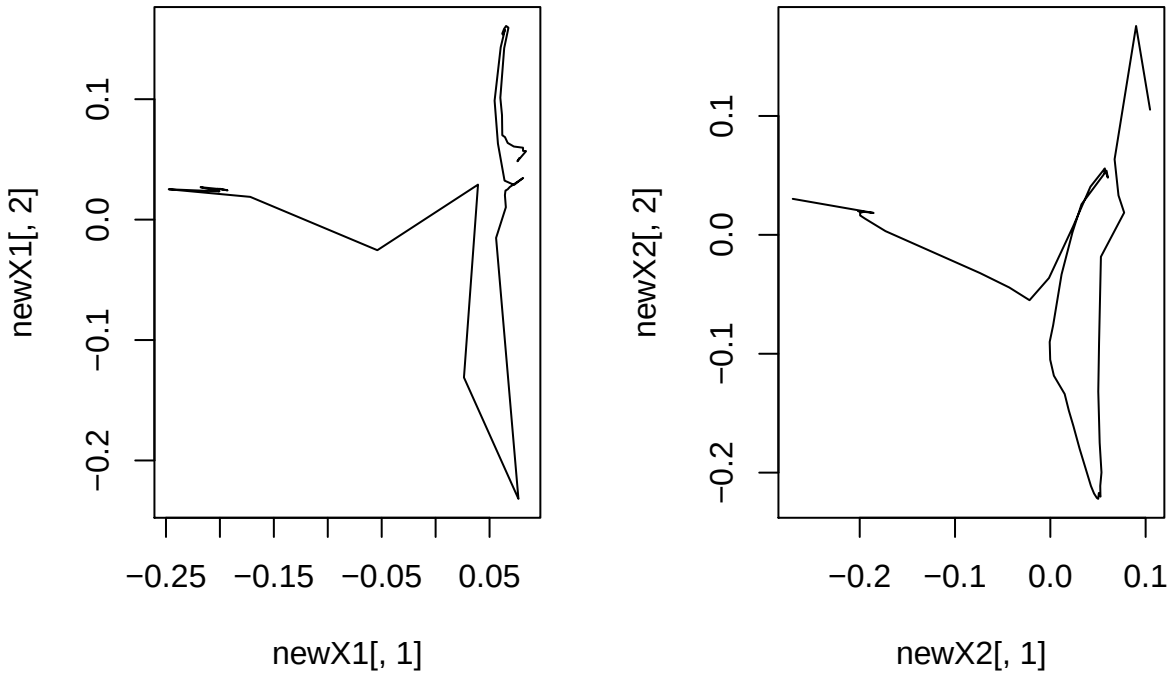
```
## Warning in plot.xy(xy, type, ...): "height" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "height" is
## not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "height" is
## not a graphical parameter
```

```
## Warning in box(...): "height" is not a graphical parameter
```

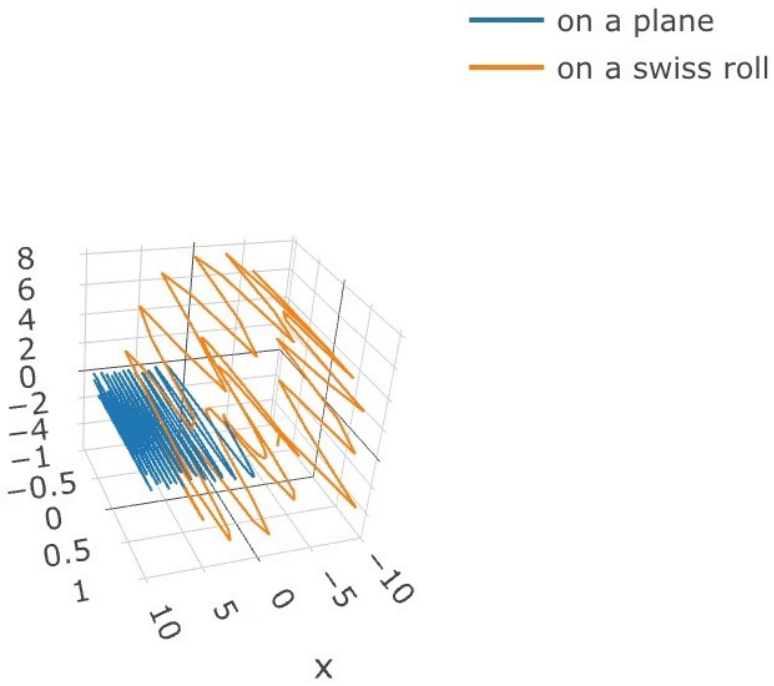
```
## Warning in title(...): "height" is not a graphical parameter
```



Sine function example

Next, We compare the performance of manifold warping by trying to align two $sin(x^2)$ curves: one is on the flat plane, the another is projected onto the Swiss roll as illustrated in the following figure. Some duplicate points are added along the curves to create many-to-one correspondences in the alignment.

```
X3 = dataset2()$X1
X4 = dataset2()$X2
p = dataset2()$p
```



As shown in the following figures, manifold warping produced similar embeddings for two curves based on their local geometry while embedding linearly collapsed the Swiss roll curve onto the plane. Here we use library 'plotly' for visualization.

```
output2 = manifold_warping(X3, X4, mode = 'embed', target_dim = NULL, k = 8,
                           mu = 0.3, thresh = 0.01, max_its = 100)
```

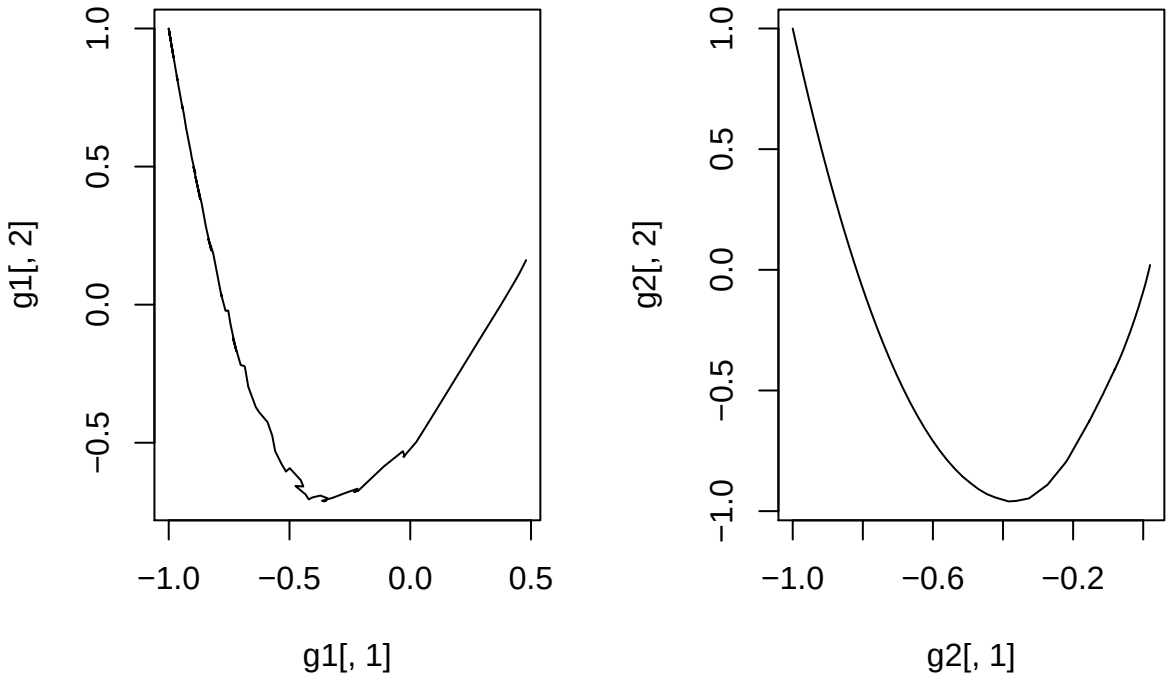
```
##
## Attaching package: 'igraph'
## The following object is masked from 'package:FNN':
##
##     knn
## The following object is masked from 'package:plotly':
##
##     groups
## The following objects are masked from 'package:stats':
##
```

```
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

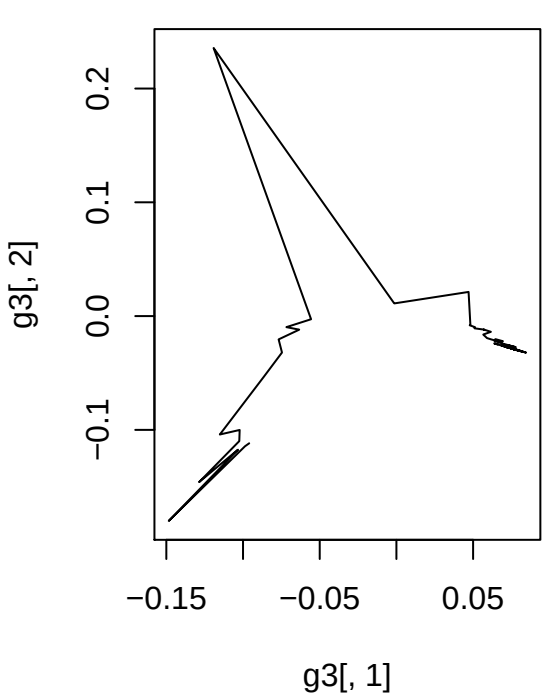
```
## [1] "Embedding didn't preserve all instances."
```

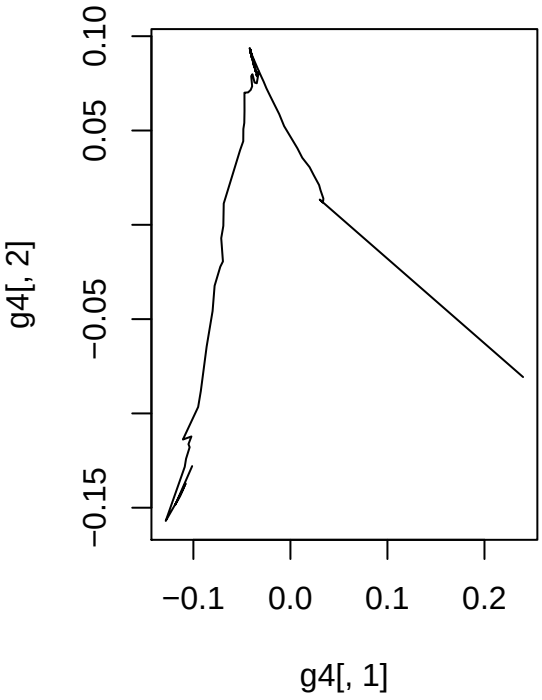
```
newX3 = output2$Y1
newX4 = output2$Y2
g1 = data.frame(x = newX3[, 1], y = newX3[, 2])
g2 = data.frame(x = newX4[, 1], y = newX4[, 2])
par(mfrow = c(1,2))
plot(g1[, 1], g1[, 2], type = 'l')
plot(g2[, 1], g2[, 2], type = 'l')
```



You may find the figures created by 'embedding' method look more similar. However, the scale is more important. There might be some inessential points ignored by manifold warping algorithm.

```
output3 = manifold_warping(X3, X4, mode = 'nonlinear', target_dim = NULL, k = 8,
                           mu = 0.3, thresh = 0.01, max_its = 100)
newX3 = output3$Y1
newX4 = output3$Y2
g3 = data.frame(x = newX3[, 1], y = newX3[, 2])
g4 = data.frame(x = newX4[, 1], y = newX4[, 2])
par(mfrow = c(1,2))
plot(g3[, 1], g3[, 2], type = 'l')
plot(g4[, 1], g4[, 2], type = 'l')
```


g3[, 2]
0.2
0.1
0.0
−0.1
−0.15
−0.05
0.05
g3[, 1]


g4[, 2]
0.10
0.05
−0.05
−0.15
−0.1
0.0
0.1
0.2
g4[, 1]