Title

stata.com

mds — Multidimensional scaling for two-way data

Description

`mds` performs multidimensional scaling (MDS) for dissimilarities between observations with respect to the specified variables. A wide selection of similarity and dissimilarity measures is available. `mds` performs classical metric MDS as well as modern metric and nonmetric MDS.

If your proximities are stored as variables in long format, see [MV] **mdslong**. For MDS with two-way proximity data in a matrix, see [MV] **mdsmat**.

Quick start

Classical metric multidimensional scaling of variables `v1`, `v2`, `v3`, and `v4` with observations identified by `idvar`

```
mds v1 v2 v3 v4, id(idvar)
```

Same as above, but use absolute-value distance and suppress the MDS configuration plot

```
mds v1 v2 v3 v4, id(idvar) measure(absolute) noplot
```

Same as above, but increase dimensions from 2 to 3 for approximating configuration

```
mds v1 v2 v3 v4, id(idvar) measure(absolute) noplot dimension(3)
```

Modern multidimensional scaling

```
mds v1 v2 v3 v4, id(idvar) method(modern)
```

Same as above, but Procrustes rotate toward classical solution instead of the default principal orientation

```
mds v1 v2 v3 v4, id(idvar) method(modern) normalize(classical)
```

Modern multidimensional scaling with Sammon mapping for the loss criterion

```
mds v1 v2 v3 v4, id(idvar) loss(sammon)
```

Modern nonmetric multidimensional scaling

```
mds v1 v2 v3 v4, id(idvar) method(nonmetric)
```

Menu

Statistics > Multivariate analysis > Multidimensional scaling (MDS) > MDS of data

Syntax

```
mds varlist [if] [in], id(varname) [options]
```

options	Description
Model	
* `id(`*varname*`)`	identify observations
`method(`*method*`)`	method for performing MDS
`loss(`*loss*`)`	loss function
`transform(`*tfunction*`)`	permitted transformations of dissimilarities
`normalize(`*norm*`)`	normalization method; default is `normalize(principal)`
`dimension(#)`	configuration dimensions; default is `dimension(2)`
`addconstant`	make distance matrix positive semidefinite
Model 2	
`unit[(`*varlist*$_2$`)]`	scale variables to min = 0 and max = 1
`std[(`*varlist*$_3$`)]`	scale variables to mean = 0 and sd = 1
`measure(`*measure*`)`	similarity or dissimilarity measure; default is `L2` (Euclidean)
`s2d(standard)`	convert similarity to dissimilarity: $\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}}$; the default
`s2d(oneminus)`	convert similarity to dissimilarity: $\text{dissim}_{ij} = 1 - \text{sim}_{ij}$
Reporting	
`neigen(#)`	maximum number of eigenvalues to display; default is `neigen(10)`
`config`	display table with configuration coordinates
`noplot`	suppress configuration plot
Minimization	
`initialize(`*initopt*`)`	start with configuration given in *initopt*
`tolerance(#)`	tolerance for configuration matrix; default is `tolerance(1e-4)`
`ltolerance(#)`	tolerance for loss criterion; default is `ltolerance(1e-8)`
`iterate(#)`	perform maximum # of iterations; default is `iterate(1000)`
`protect(#)`	perform # optimizations and report best solution; default is `protect(1)`
`[no]log`	display or suppress the iteration log; default is to display
`trace`	display current configuration in iteration log
`gradient`	display current gradient matrix in iteration log
`sdprotect(#)`	advanced; see *Options* below

* `id(`*varname*`)` is required.

`bootstrap`, `by`, `collect`, `jackknife`, `rolling`, `statsby`, and `xi` are allowed; see [U] **11.1.10 Prefix commands**.

The maximum number of observations allowed in `mds` is the maximum matrix size; see [R] **Limits**.

`sdprotect(#)` does not appear in the dialog box.

See [U] **20 Estimation and postestimation commands** for more capabilities of estimation commands.

method	Description
classical	classical MDS; default if neither loss() nor transform() is specified
modern	modern MDS; default if loss() or transform() is specified; except when loss(stress) and transform(monotonic) are specified
nonmetric	nonmetric (modern) MDS; default when loss(stress) and transform(monotonic) are specified

loss	Description
stress	stress criterion, normalized by distances; the default
nstress	stress criterion, normalized by disparities
sstress	squared stress criterion, normalized by distances
nsstress	squared stress criterion, normalized by disparities
strain	strain criterion (with transform(identity) is equivalent to classical MDS)
sammon	Sammon mapping

tfunction	Description
identity	no transformation; disparity = dissimilarity; the default
power	power α: disparity = dissimilarity$^{\alpha}$
monotonic	weakly monotonic increasing functions (nonmetric scaling); only with loss(stress)

norm	Description
principal	principal orientation; location = 0; the default
classical	Procrustes rotation toward classical solution
target(*matname*)[, copy]	Procrustes rotation toward *matname*; ignore naming conflicts if copy is specified

initopt	Description
classical	start with classical solution; the default
random[(#)]	start at random configuration, setting seed to #
from(*matname*)[, copy]	start from *matname*; ignore naming conflicts if copy is specified

Options

Model

id(*varname*) is required and specifies a variable that identifies observations. A warning message is displayed if *varname* has duplicate values.

method(*method*) specifies the method for MDS.

method(classical) specifies classical metric scaling, also known as "principal coordinates analysis" when used with Euclidean proximities. Classical MDS obtains equivalent results to modern

MDS with `loss(strain)` and `transform(identity)` without weights. The calculations for classical MDS are fast; consequently, classical MDS is generally used to obtain starting values for modern MDS. If the options `loss()` and `transform()` are not specified, `mds` computes the classical solution, likewise if `method(classical)` is specified `loss()` and `transform()` are not allowed.

`method(modern)` specifies modern scaling. If `method(modern)` is specified but not `loss()` or `transform()`, then `loss(stress)` and `transform(identity)` are assumed. All values of `loss()` and `transform()` are valid with `method(modern)`.

`method(nonmetric)` specifies nonmetric scaling, which is a type of modern scaling. If `method(nonmetric)` is specified, `loss(stress)` and `transform(monotonic)` are assumed. Other values of `loss()` and `transform()` are not allowed.

`loss(`*loss*`)` specifies the loss criterion.

`loss(stress)` specifies that the stress loss function be used, normalized by the squared Euclidean distances. This criterion is often called Kruskal's stress-1. Optimal configurations for `loss(stress)` and for `loss(nstress)` are equivalent up to a scale factor, but the iteration paths may differ. `loss(stress)` is the default.

`loss(nstress)` specifies that the stress loss function be used, normalized by the squared disparities, that is, transformed dissimilarities. Optimal configurations for `loss(stress)` and for `loss(nstress)` are equivalent up to a scale factor, but the iteration paths may differ.

`loss(sstress)` specifies that the squared stress loss function be used, normalized by the fourth power of the Euclidean distances.

`loss(nsstress)` specifies that the squared stress criterion, normalized by the fourth power of the disparities (transformed dissimilarities) be used.

`loss(strain)` specifies the strain loss criterion. Classical scaling is equivalent to `loss(strain)` and `transform(identity)` but is computed by a faster noniterative algorithm. Specifying `loss(strain)` still allows transformations.

`loss(sammon)` specifies the Sammon (1969) loss criterion.

`transform(`*tfunction*`)` specifies the class of allowed transformations of the dissimilarities; transformed dissimilarities are called disparities.

`transform(identity)` specifies that the only allowed transformation is the identity; that is, disparities are equal to dissimilarities. `transform(identity)` is the default.

`transform(power)` specifies that disparities are related to the dissimilarities by a power function,

$$\text{disparity} = \text{dissimilarity}^{\alpha}, \qquad \alpha > 0$$

`transform(monotonic)` specifies that the disparities are a weakly monotonic function of the dissimilarities. This is also known as nonmetric MDS. Tied dissimilarities are handled by the primary method; that is, ties may be broken but are not necessarily broken. `transform(monotonic)` is valid only with `loss(stress)`.

`normalize(`*norm*`)` specifies a normalization method for the configuration. Recall that the location and orientation of an MDS configuration is not defined ("identified"); an isometric transformation (that is, translation, reflection, or orthonormal rotation) of a configuration preserves interpoint Euclidean distances.

`normalize(principal)` performs a principal normalization, in which the configuration columns have zero mean and correspond to the principal components, with positive coefficient for the observation with lowest value of `id()`. `normalize(principal)` is the default.

`normalize(classical)` normalizes by a distance-preserving Procrustean transformation of the configuration toward the classical configuration in principal normalization; see [MV] **procrustes**. `normalize(classical)` is not valid if `method(classical)` is specified.

`normalize(target(`*matname*`)` [`, copy`]`)` normalizes by a distance-preserving Procrustean transformation toward *matname*; see [MV] **procrustes**. *matname* should be an $n \times p$ matrix, where n is the number of observations and p is the number of dimensions, and the rows of *matname* should be ordered with respect to `id()`. The rownames of *matname* should be set correctly but will be ignored if `copy` is also specified.

Note on `normalize(classical)` and `normalize(target())`: the Procrustes transformation comprises any combination of translation, reflection, and orthonormal rotation—these transformations preserve distance. Dilation (uniform scaling) would stretch distances and is not applied. However, the output reports the dilation factor, and the reported Procrustes statistic is for the dilated configuration.

`dimension(#)` specifies the dimension of the approximating configuration. The default # is 2 and should not exceed the number of observations; typically, # would be much smaller. With `method(classical)`, it should not exceed the number of positive eigenvalues of the centered distance matrix.

`addconstant` specifies that if the double-centered distance matrix is not positive semidefinite (psd), a constant should be added to the squared distances to make it psd and, hence, Euclidean. `addconstant` is allowed with classical MDS only.

Model 2

`unit`[`(`*varlist*$_2$`)`] specifies variables that are transformed to min $= 0$ and max $= 1$ before entering in the computation of similarities or dissimilarities. `unit` by itself, without an argument, is a shorthand for `unit(_all)`. Variables in `unit()` should not be included in `std()`.

`std`[`(`*varlist*$_3$`)`] specifies variables that are transformed to mean $= 0$ and sd $= 1$ before entering in the computation of similarities or dissimilarities. `std` by itself, without an argument, is a shorthand for `std(_all)`. Variables in `std()` should not be included in `unit()`.

`measure(`*measure*`)` specifies the similarity or dissimilarity measure. The default is `measure(L2)`, Euclidean distance. This option is not case sensitive. See [MV] ***measure_option*** for detailed descriptions of the supported measures.

If a similarity measure is selected, the computed similarities will first be transformed into dissimilarities, before proceeding with the scaling; see the `s2d()` option below.

Classical metric MDS with Euclidean distance is equivalent to principal component analysis (see [MV] **pca**); the MDS configuration coordinates are the principal components.

`s2d(standard`|`oneminus)` specifies how similarities are converted into dissimilarities. By default, the command assumes dissimilarity data. Specifying `s2d()` indicates that your proximity data are similarities.

Dissimilarity data should have zeros on the diagonal (that is, an object is identical to itself) and nonnegative off-diagonal values. Dissimilarities need not satisfy the triangular inequality, $D(i,j)^2 \leq D(i,h)^2 + D(h,j)^2$. Similarity data should have ones on the diagonal (that is, an object is identical to itself) and have off-diagonal values between zero and one. In either case, proximities should be symmetric.

The available `s2d()` options, `standard` and `oneminus`, are defined as follows:

`standard`	$\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} + \text{sim}_{jj} - 2\text{sim}_{ij}} = \sqrt{2(1 - \text{sim}_{ij})}$
`oneminus`	$\text{dissim}_{ij} = 1 - \text{sim}_{ij}$

`s2d(standard)` is the default.

`s2d()` should be specified only with measures in similarity form.

Reporting

`neigen(#)` specifies the number of eigenvalues to be included in the table. The default is `neigen(10)`. Specifying `neigen(0)` suppresses the table. This option is allowed with classical MDS only.

`config` displays the table with the coordinates of the approximating configuration. This table may also be displayed using the postestimation command `estat config`; see [MV] **mds postestimation**.

`noplot` suppresses the graph of the approximating configuration. The graph can still be produced later via `mdsconfig`, which also allows the standard graphics options for fine-tuning the plot; see [MV] **mds postestimation plots**.

Minimization

These options are available only with `method(modern)` or `method(nonmetric)`:

`initialize(`*initopt*`)` specifies the initial values of the criterion minimization process.

`initialize(classical)`, the default, uses the solution from classical metric scaling as initial values. With `protect()`, all but the first run start from random perturbations from the classical solution. These random perturbations are independent and normally distributed with standard error equal to the product of `sdprotect(#)` and the standard deviation of the dissimilarities. `initialize(classical)` is the default.

`initialize(random)` starts an optimization process from a random starting configuration. These random configurations are generated from independent normal distributions with standard error equal to the product of `sdprotect(#)` and the standard deviation of the dissimilarities. The means of the configuration are irrelevant in MDS.

`initialize(from(`*matname*`)[ , copy])` sets the initial value to *matname*. *matname* should be an $n \times p$ matrix, where n is the number of observations and p is the number of dimensions, and the rows of *matname* should be ordered with respect to `id()`. The rownames of *matname* should be set correctly but will be ignored if `copy` is specified. With `protect()`, the second-to-last runs start from random perturbations from *matname*. These random perturbations are independent normal distributed with standard error equal to the product of `sdprotect(#)` and the standard deviation of the dissimilarities.

`tolerance(#)` specifies the tolerance for the configuration matrix. When the relative change in the configuration from one iteration to the next is less than or equal to `tolerance()`, the `tolerance()` convergence criterion is satisfied. The default is `tolerance(1e-4)`.

`ltolerance(#)` specifies the tolerance for the fit criterion. When the relative change in the fit criterion from one iteration to the next is less than or equal to `ltolerance()`, the `ltolerance()` convergence is satisfied. The default is `ltolerance(1e-8)`.

Both the `tolerance()` and `ltolerance()` criteria must be satisfied for convergence.

`iterate(#)` specifies the maximum number of iterations. The default is `iterate(1000)`.

protect(#) requests that # optimizations be performed and that the best of the solutions be reported. The default is protect(1). See option initialize() on starting values of the runs. The output contains a table of the run, return code, iteration, and criterion value reached. Specifying a large number, such as protect(50), provides reasonable insight whether the solution found is a global minimum and not just a local minimum.

If any of the options log, trace, or gradient is also specified, iteration reports will be printed for each optimization run. Beware: this option will produce a lot of output.

log and nolog specify whether to display the iteration log. The iteration log is displayed by default unless you used set iterlog off to suppress it; see set iterlog in [R] *set iter*.

trace displays the configuration matrices in the iteration report. Beware: this option may produce a lot of output.

gradient displays the gradient matrices of the fit criterion in the iteration report. Beware: this option may produce a lot of output.

The following option is available with mds but is not shown in the dialog box:

sdprotect(#) sets a proportionality constant for the standard deviations of random configurations (init(random)) or random perturbations of given starting configurations (init(classical) or init(from())). The default is sdprotect(1).

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

Multidimensional scaling (MDS) is a dimension-reduction and visualization technique. Dissimilarities (for instance, Euclidean distances) between observations in a high-dimensional space are represented in a lower-dimensional space (typically two dimensions) so that the Euclidean distance in the lower-dimensional space approximates the dissimilarities in the higher-dimensional space. See Kruskal and Wish (1978) for a brief nontechnical introduction to MDS. Young (1987) and Borg and Groenen (2005) offer more advanced textbook-sized treatments.

If you already have the similarities or dissimilarities of the n objects, you should continue by reading [MV] **mdsmat**.

In many applications of MDS, however, the similarity or dissimilarity of objects is not measured but rather *defined* by the researcher in terms of variables ("attributes") $x_1, \ldots, x_k$ that are measured on the objects. The pairwise dissimilarity of objects can be expressed using a variety of similarity or dissimilarity measures in the attributes (for example, Mardia, Kent, and Bibby [1979, sec. 13.4]; Cox and Cox [2001, sec. 1.3]). A common measure is the Euclidean distance L2 between the attributes of the objects i and j:

$$\mathtt{L2}_{ij} = \left\{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ik} - x_{jk})^2\right\}^{1/2}$$

A popular alternative is the `L1` distance, also known as the `cityblock` or `Manhattan` distance. In comparison to `L2`, `L1` gives less influence to larger differences in attributes:

$$\texttt{L1}_{ij} = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ik} - x_{jk}|$$

In contrast, we may also define the extent of dissimilarity between 2 observations as the maximum absolute difference in the attributes and thus give a larger influence to larger differences:

$$\texttt{Linfinity}_{ij} = \max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \ldots, |x_{ik} - x_{jk}|)$$

These three measures are special cases of the Minkowski distance $L(q)$, for $q = 2$ (`L2`), $q = 1$ (`L1`), and $q = \infty$ (`Linfinity`), respectively. Minkowski distances with other values of q may be used as well. Stata supports a wide variety of other similarity and dissimilarity measures, both for continuous variables and for binary variables. See [MV] ***measure_option*** for details.

Multidimensional scaling constructs approximations for dissimilarities, not for similarities. Thus, if a similarity measure is specified, `mds` first transforms the similarities into dissimilarities. Two methods to do this are available. The default `standard` method,

$$\text{dissim}_{ij} = \sqrt{\text{sim}_{ii} - 2\text{sim}_{ij} + \text{sim}_{jj}}$$

has a useful property: if the similarity matrix is positive semidefinite, a property satisfied by most similarity measures, the standard dissimilarities are Euclidean.

Usually, the number of observations exceeds the number of variables on which the observations are compared, but this is not a requirement for MDS. MDS creates an $n \times n$ dissimilarity matrix $\mathbf{D}$ from the n observations on k variables. It then constructs an approximation of $\mathbf{D}$ by the Euclidean distances in a matching configuration $\mathbf{Y}$ of n points in p-dimensional space:

$$\text{dissimilarity}(x_i, x_j) \approx \texttt{L2}(y_i, y_j) \quad \text{for all } i, j$$

Typically, of course, $p << k$, and most often $p = 1$, 2, or 3.

A wide variety of MDS methods have been proposed. `mds` performs classical and modern scaling. Classical scaling has its roots in Young and Householder (1938) and Torgerson (1952). MDS requires complete and symmetric dissimilarity interval-level data. To explore modern scaling, see Borg and Groenen (2005). Classical scaling results in an eigen decomposition, whereas modern scaling is accomplished by the minimization of a loss function. Consequently, eigenvalues are not available after modern MDS.

Computing the classical solution is straightforward, but with modern MDS the minimization of the loss criteria over configurations is a high-dimensional problem that is easily beset by convergence to local minimums. `mds` provides options to control the minimization process 1) by allowing the user to select the starting configuration and 2) by selecting the best solution among multiple minimization runs from random starting configurations.

Euclidean distances

▷ Example 1

The most popular dissimilarity measure is Euclidean distance. We illustrate with data from table 7.1 of Yang and Trewn (2004, 182). This dataset consists of eight variables with nutrition data on 25 breakfast cereals.

```
. use https://www.stata-press.com/data/r18/cerealnut
(Cereal nutrition)
. describe
Contains data from https://www.stata-press.com/data/r18/cerealnut.dta
 Observations:            25                  Cereal nutrition
    Variables:             9                  24 Feb 2022 17:19
                                              (_dta has notes)
```

Variable name	Storage type	Display format	Value label	Variable label
brand	str25	%25s		Cereal Brand
calories	int	%9.0g		Calories (Cal/oz)
protein	byte	%9.0g		Protein (g)
fat	byte	%9.0g		Fat (g)
Na	int	%9.0g		Na (mg)
fiber	float	%9.0g		Fiber (g)
carbs	float	%9.0g		Carbs (g)
sugar	byte	%9.0g		Sugar (g)
K	int	%9.0g		K (mg)

```
Sorted by:
. summarize calories-K, sep(4)
```

Variable	Obs	Mean	Std. dev.	Min	Max
calories	25	109.6	21.30728	50	160
protein	25	2.68	1.314027	1	6
fat	25	.92	.7593857	0	2
Na	25	195.8	71.32204	0	320
fiber	25	1.7	2.056494	0	9
carbs	25	15.3	4.028544	7	22
sugar	25	7.4	4.609772	0	14
K	25	90.6	77.5043	15	320

```
. replace brand = subinstr(brand," ","_",.)
(20 real changes made)
```

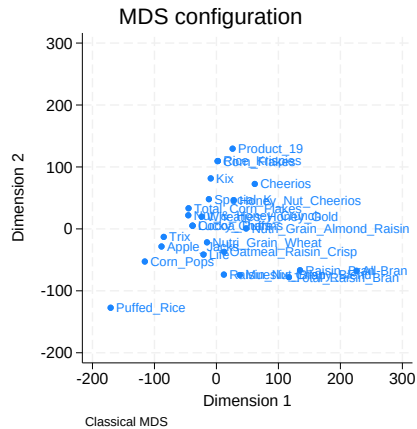
We replaced spaces in the cereal brand names with underscores to avoid confusing which words in the brand names are associated with which points in the graphs we are about to produce. Removing spaces is not required.

The default dissimilarity measure used by `mds` is the Euclidean distance `L2` computed on the raw data (unstandardized). The summary of the eight nutrition variables shows that `K`, `Na`, and `calories`—having much larger standard deviations—will largely determine the Euclidean distances.

```
. mds calories-K, id(brand)
Classical metric multidimensional scaling
Dissimilarity: L2, computed on 8 variables

                                      Number of obs        =         25
Eigenvalues > 0      =         8      Mardia fit measure 1 =     0.9603
Retained dimensions  =         2      Mardia fit measure 2 =     0.9970
```

Dimension	Eigenvalue	abs(eigenvalue) Percent	abs(eigenvalue) Cumul.	(eigenvalue)^2 Percent	(eigenvalue)^2 Cumul.
1	158437.92	56.95	56.95	67.78	67.78
2	108728.77	39.08	96.03	31.92	99.70
3	10562.645	3.80	99.83	0.30	100.00
4	382.67849	0.14	99.97	0.00	100.00
5	69.761715	0.03	99.99	0.00	100.00
6	12.520822	0.00	100.00	0.00	100.00
7	5.7559984	0.00	100.00	0.00	100.00
8	2.2243244	0.00	100.00	0.00	100.00

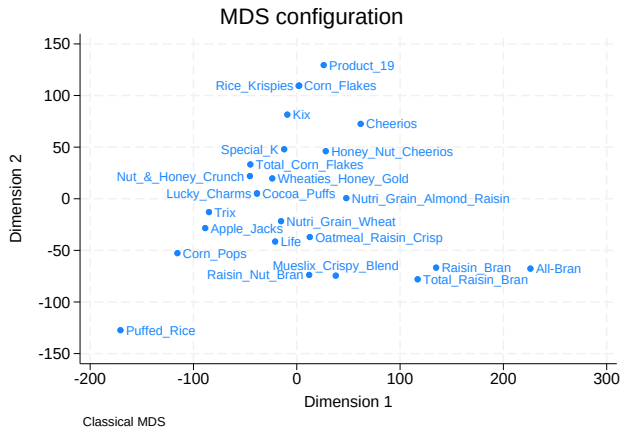


The default MDS configuration graph can be improved upon by using the `mdsconfig` postestimation command. We will demonstrate this in a moment. But first, we explain the output of `mds`.

`mds` has performed classical metric scaling and extracted two dimensions, which is the default action. To assess goodness of fit, the two statistics proposed by Mardia are reported (see Mardia, Kent, and Bibby [1979, sec. 14.4]). The statistics are defined in terms of the eigenvalues of the double-centered distance matrix. If the dissimilarities are truly Euclidean, all eigenvalues are nonnegative. Look at the eigenvalues. We may interpret these as the extent to which the dimensions account for dissimilarity between the cereals. Depending on whether you look at the eigenvalues or squared eigenvalues, it takes two or three dimensions to account for more than 99% of the dissimilarity.

We can produce a prettier configuration plot with the `mdsconfig` command; see [MV] **mds postestimation plots** for details. command; see [MV] **mds postestimation plots** for details.

```
. generate place = 3
. replace place = 9 if inlist(brand,"Rice_Krispies","Nut_&_Honey_Crunch",
> "Special_K","Raisin_Nut_Bran","Lucky_Charms")
(5 real changes made)
. replace place = 12 if inlist(brand,"Mueslix_Crispy_Blend")
(1 real change made)
. mdsconfig, autoaspect mlabvpos(place)
```



The *marker_label_option* `mlabvposition()` allowed fine control over the placement of the cereal brand names. We created a variable called `place` giving clock positions where the cereal names were to appear in relation to the plotted point. We set these to minimize overlap of the names. We also requested the `autoaspect` option to obtain better use of the graphing region while preserving the scale of the x and y axes.

MDS has placed the cereals so that all the brands fall within a triangle defined by Product 19, All-Bran, and Puffed Rice. You can examine the graph to see how close your favorite cereal is to the other cereals.

But, as we saw from the variable summary, three of the eight variables are controlling the distances. If we want to provide for a more equal footing for the eight variables, we can request that `mds` compute the Euclidean distances on standardized variables. Euclidean distance based on standardized variables is also known as the *Karl Pearson distance* (Pearson 1900). We obtain standardized measures with the option `std`.

```
. mds calories-K, id(brand) std noplot
Classical metric multidimensional scaling
Dissimilarity: L2, computed on 8 variables

                                      Number of obs        =         25
Eigenvalues > 0      =         8      Mardia fit measure 1 =     0.5987
Retained dimensions  =         2      Mardia fit measure 2 =     0.7697
```

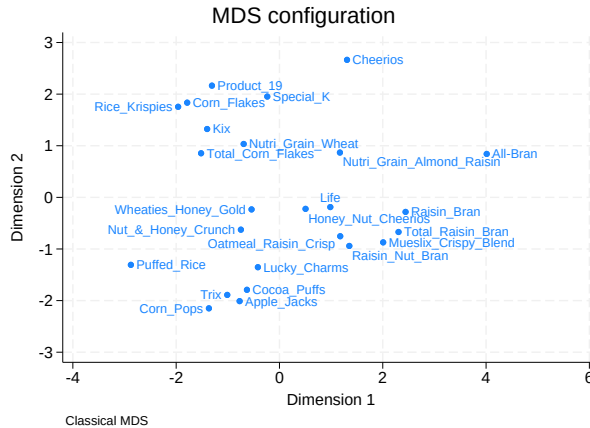
Dimension	Eigenvalue	abs(eigenvalue) Percent	abs(eigenvalue) Cumul.	(eigenvalue)^2 Percent	(eigenvalue)^2 Cumul.
1	65.645395	34.19	34.19	49.21	49.21
2	49.311416	25.68	59.87	27.77	76.97
3	38.826608	20.22	80.10	17.21	94.19
4	17.727805	9.23	89.33	3.59	97.78
5	11.230087	5.85	95.18	1.44	99.22
6	8.2386231	4.29	99.47	0.78	99.99
7	.77953426	0.41	99.87	0.01	100.00
8	.24053137	0.13	100.00	0.00	100.00

In this and the previous example, we did not specify a `method()` for `mds` and got classical metric scaling. Classical scaling is the default when `method()` is omitted and neither the `loss()` nor `transform()` option is specified.

Accounting for more than 99% of the underlying distances now takes more MDS-retained dimensions. For this example, we have still retained only two dimensions. We specified the `noplot` option because we wanted to exercise control over the configuration plot by using the `mdsconfig` command. We generate a variable named `pos` that will help minimize cereal brand name overlap.

```
. generate pos = 3
. replace pos = 5 if inlist(brand,"Honey_Nut_Cheerios","Raisin_Nut_Bran",
> "Nutri_Grain_Almond_Raisin")
(3 real changes made)
. replace pos = 8 if inlist(brand,"Oatmeal_Raisin_Crisp")
(1 real change made)
. replace pos = 9 if inlist(brand,"Corn_Pops","Trix","Nut_&_Honey_Crunch",
> "Rice_Krispies","Wheaties_Honey_Gold")
(5 real changes made)
. replace pos = 12 if inlist(brand,"Life")
(1 real change made)
```

```
. mdsconfig, autoaspect mlabvpos(pos)
```



This configuration plot, based on the standardized variables, better incorporates all the nutrition data. If you are familiar with these cereal brands, spotting groups of similar cereals appearing near each other is easy. The bottom-left corner has several of the most sweetened cereals. The brands containing the word "Bran" all appear to the right of center. Rice Krispies and Puffed Rice are the farthest to the left.

Classical multidimensional scaling based on standardized Euclidean distances is actually equivalent to a principal component analysis of the correlation matrix of the variables. See Mardia, Kent, and Bibby (1979, sec. 14.3) for details.

We now demonstrate this property by doing a principal component analysis extracting the leading two principal components. See [MV] **pca** for details.

```
. pca calories-K, comp(2)
Principal components/correlation                  Number of obs    =         25
                                                  Number of comp.  =          2
                                                  Trace            =          8
Rotation: (unrotated = principal)                 Rho              =     0.5987
```

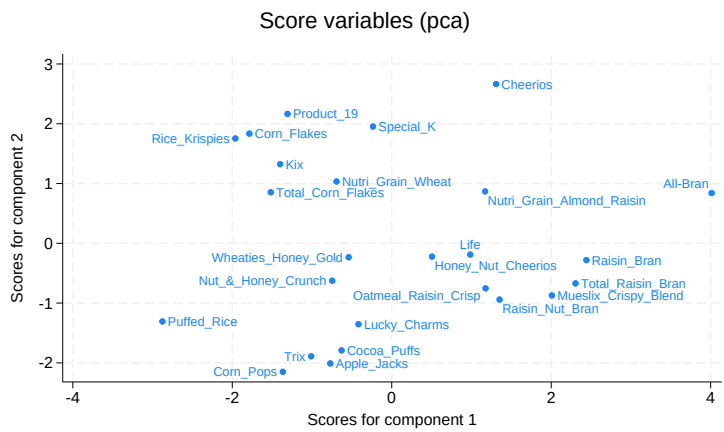
Component	Eigenvalue	Difference	Proportion	Cumulative
Comp1	2.73522	.680583	0.3419	0.3419
Comp2	2.05464	.436867	0.2568	0.5987
Comp3	1.61778	.879117	0.2022	0.8010
Comp4	.738659	.270738	0.0923	0.8933
Comp5	.46792	.124644	0.0585	0.9518
Comp6	.343276	.310795	0.0429	0.9947
Comp7	.0324806	.0224585	0.0041	0.9987
Comp8	.0100221	.	0.0013	1.0000

Principal components (eigenvectors)

Variable	Comp1	Comp2	Unexplained
calories	0.1992	-0.0632	.8832
protein	0.3376	0.4203	.3253
fat	0.3811	-0.0667	.5936
Na	0.0962	0.5554	.3408
fiber	0.5146	0.0913	.2586
carbs	-0.2574	0.4492	.4043
sugar	0.2081	-0.5426	.2765
K	0.5635	0.0430	.1278

The proportion and cumulative proportion of the eigenvalues in the PCA match the percentages from MDS. We will ignore the interpretation of the principal components but move directly to the principal coordinates, also known as the scores of the PCA. We make a plot of the first and second scores, using the `scoreplot` command; see [MV] **scoreplot**. We specify the `mlabel()` option to label the cereals and the `mlabvpos()` option for fine control over placement of the brand names.

```
. replace pos = 11 if inlist(brand,"All-Bran")
(1 real change made)
. scoreplot, mlabel(brand) mlabvpos(pos)
```



Compare this PCA score plot with the MDS configuration plot. Apart from some differences in how the graphs were rendered, they are the same.

◁

Non-Euclidean dissimilarity measures

With non-Euclidean dissimilarity measures, the parallel between PCA and MDS no longer holds.

▷ Example 2

To illustrate MDS with non-Euclidean distance measures, we will analyze books on multivariate statistics. Gifi (1990) reports on the number of pages devoted to seven topics in 20 textbooks on multivariate statistics. We added similar data on five more recent books.

```
. use https://www.stata-press.com/data/r18/mvstatsbooks, clear
. describe
Contains data from https://www.stata-press.com/data/r18/mvstatsbooks.dta
 Observations:            25
    Variables:             8                  15 Mar 2022 16:27
                                              (_dta has notes)
```

Variable name	Storage type	Display format	Value label	Variable label
author	str17	%17s		
math	int	%9.0g		Math other than statistics (e.g., linear algebra)
corr	int	%9.0g		Correlation and regression, including linear structural and functional equations
fact	byte	%9.0g		Factor analysis and principal component analysis
cano	byte	%9.0g		Canonical correlation analysis
disc	int	%9.0g		Discriminant analysis, classification, and cluster analysis
stat	int	%9.0g		Statistics, incl. dist. theory, hypothesis testing & est.; categorical data
mano	int	%9.0g		Manova and the general linear model

```
Sorted by:
```

A brief description of the topics is given in the variable labels. For more details, we refer to Gifi (1990, 15). Here are the data:

```
. list, noobs
```

author	math	corr	fact	cano	disc	stat	mano
Roy57	31	0	0	0	0	164	11
Kendall57	0	16	54	18	27	13	14
Kendall75	0	40	32	10	42	60	0
Anderson58	19	0	35	19	28	163	52
CooleyLohnes62	14	7	35	22	17	0	56
(output omitted)							
GreenCaroll76	290	10	6	0	8	0	2
CailliezPages76	184	48	82	42	134	0	0
Giri77	29	0	0	0	41	211	32
Gnanadesikan77	0	19	56	0	39	75	0
Kshirsagar78	0	22	45	42	60	230	59
Thorndike78	30	128	90	28	48	0	0
MardiaKentBibby79	34	28	68	19	67	131	55
Seber84	16	0	59	13	116	129	101
Stevens96	23	87	67	21	30	43	249
EverittDunn01	0	54	65	0	56	20	30
Rencher02	38	0	71	19	105	135	131

For instance, the 1979 book by Mardia, Kent, and Bibby has 34 pages on mathematics (mostly linear algebra); 28 pages on correlation, regression, and related topics (in this particular case, simultaneous equations); etc. In most of these books, some pages are not classified. Anyway, the number of pages

and the amount of information per page vary widely among the books. A Euclidean distance measure is not appropriate here. Standardization does not help us here—the problem is not differences in the scales of the variables but those in the observations. One possibility is to transform the data into *compositional data* by dividing the variables by the total number of classified pages. See Mardia, Kent, and Bibby (1979, 377–380) for a discussion of specialized dissimilarity measures for compositional data. However, we can also use the correlation between observations (not between variables) as the similarity measure. The higher the correlation between the attention given to the various topics, the more similar two textbooks are. We do a classical MDS, suppressing the plot to first assess the quality of a two-dimensional representation.

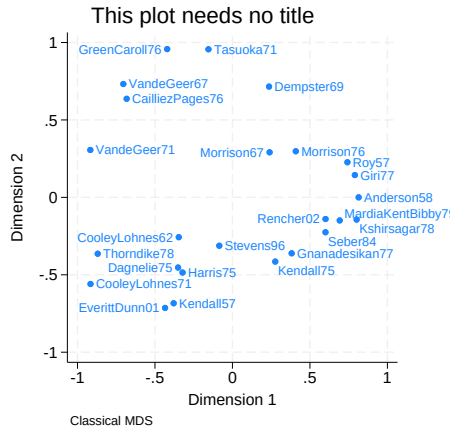
```
. mds math-mano, id(author) measure(corr) noplot
Classical metric multidimensional scaling
   Similarity: correlation, computed on 7 variables
Dissimilarity: sqrt(2(1-similarity))
                                        Number of obs        =         25
Eigenvalues > 0      =         6        Mardia fit measure 1 =     0.6680
Retained dimensions  =         2        Mardia fit measure 2 =     0.8496
```

Dimension	Eigenvalue	abs(eigenvalue) Percent	abs(eigenvalue) Cumul.	(eigenvalue)^2 Percent	(eigenvalue)^2 Cumul.
1	8.469821	38.92	38.92	56.15	56.15
2	6.0665813	27.88	66.80	28.81	84.96
3	3.8157101	17.53	84.33	11.40	96.35
4	1.6926956	7.78	92.11	2.24	98.60
5	1.2576053	5.78	97.89	1.24	99.83
6	.45929376	2.11	100.00	0.17	100.00

Again the quality of a two-dimensional approximation is somewhat unsatisfactory, with 67% and 85% of the variation accounted for according to the two Mardia criteria. Still, let's look at the plot, using a title that refers to the self-referential aspect of the analysis (Smullyan 1986). We reposition some of the author labels to enhance readability by using the `mlabvpos()` option.

```
. generate spot = 3
. replace spot = 5 if inlist(author,"Seber84","Kshirsagar78","Kendall75")
(3 real changes made)
. replace spot = 2 if author=="MardiaKentBibby79"
(1 real change made)
. replace spot = 9 if inlist(author, "Dagnelie75","Rencher02",
> "GreenCarroll76","EverittDunn01","CooleyLohnes62","Morrison67")
(6 real changes made)
```

```
. mdsconfig, mlabvpos(spot) title(This plot needs no title)
```



A striking characteristic of the plot is that the textbooks seem to be located on a circle. This is a phenomenon that is regularly encountered in multidimensional scaling and was labeled the "horseshoe effect" by Kendall (1971, 215–251). This phenomenon seems to occur especially in situations in which a one-dimensional representation of objects needs to be constructed, for example, in *seriation* applications, from data in which small dissimilarities were measured accurately but moderate and larger dissimilarities are "lumped together".

◁

❑ Technical note

These data could also be analyzed differently. A particularly interesting method is correspondence analysis (CA), which seeks a simultaneous geometric representation of the rows (textbooks) and columns (topics). We used `camat` to analyze these data. The results for the textbooks were not much different. Textbooks that were mapped as similar using MDS were also mapped this way by CA. The Green and Carroll book that appeared much different from the rest was also displayed away from the rest by CA. In the CA biplot, it was immediately clear that this book was so different because its pages were classified by Gifi (1990) as predominantly mathematical. But CA also located the topics in this space. The pattern was easy to interpret and was expected. The seven topics were mapped in three groups. `math` and `stat` appear as two groups by themselves, and the five applied topics were mapped close together. See [MV] **ca** for information on the `ca` command.

❑

Introduction to modern MDS

We return to the data on breakfast cereals explored above to introduce modern MDS. We repeat some steps taken previously and then perform estimation using options `loss(strain)` and `transform(identity)`, which we demonstrate are equivalent to classical MDS.

`mds` is an estimation or `eclass` command; see `program define` in [P] **program**. You can display its stored results using `ereturn list`. The configuration is stored as `e(Y)` and we will compare the configuration obtained from classical MDS with the equivalent one from modern MDS.

▷ Example 3

```
. use https://www.stata-press.com/data/r18/cerealnut, clear
(Cereal nutrition)
. replace brand = subinstr(brand," ","_",.)
(20 real changes made)
. quietly mds calories-K, id(brand) noplot
. mat Yclass = e(Y)
. mds calories-K, id(brand) meth(modern) loss(strain) trans(ident) noplot
Iteration 1:   strain =  594.12657
Iteration 2:   strain =  594.12657
Modern multidimensional scaling
Dissimilarity: L2, computed on 8 variables
    Loss criterion: strain = loss for classical MDS
    Transformation: identity (no transformation)
                                                Number of obs    =         25
                                                Dimensions       =          2
    Normalization: principal                    Loss criterion   =   594.1266
. mat Ymod = e(Y)
. assert mreldif(Yclass, Ymod) < 1e-6
```

Note the output differences between modern and classical MDS. In modern MDS we have an iteration log from the minimization of the loss function. The method, measure, observations, dimensions, and number of variables are reported as before, but we do not have or display eigenvalues. The normalization is always reported in modern MDS and with `normalize(target())` for classical MDS. The loss criterion is simply the value of the loss function at the minimum.

◁

Protecting from local minimums

Modern MDS can sometimes converge to a local rather than a global minimum. To protect against this, multiple runs can be made, giving the best of the runs as the final answer. The option for performing this is `protect(#)`, where # is the number of runs to be performed. The `nolog` option is of particular use with `protect()`, because the iteration logs from the runs will create a lot of output. Repeating the minimization can take some time, depending on the number of runs selected and the number of iterations it takes to converge.

▷ Example 4

We choose `loss(stress)`, and `transform(identity)` is assumed with modern MDS. We omit the iteration logs to avoid a large amount of output. The number of iterations is available after estimation in `e(ic)`. We first do a run without the `protect()` option, and then we use `protect(50)` and compare our results.

```
. mds calories-K, id(brand) method(modern) loss(stress) nolog noplot
(transform(identity) assumed)

Modern multidimensional scaling
Dissimilarity: L2, computed on 8 variables

    Loss criterion: stress = raw_stress/norm(distances)
    Transformation: identity (no transformation)

                                                Number of obs    =         25
                                                Dimensions       =          2
    Normalization: principal                    Loss criterion   =     0.0263

. di e(ic)
45

. mat Ystress = e(Y)

. set seed 123456789

. mds calories-K, id(brand) method(modern) loss(stress) nolog protect(50)
(transform(identity) assumed)
```

run	mrc	#iter	lossval
1	0	74	.02626681
2	0	101	.02626681
3	0	78	.02626681
4	0	75	.02626681
5	0	75	.02626681
6	0	57	.02626681
7	0	84	.02626681
8	0	75	.02626681
9	0	85	.02626681
10	0	60	.02626681
11	0	63	.02626681
12	0	45	.02626681
13	0	55	.02626681
14	0	57	.02626682
15	0	82	.02626682
16	0	63	.02626682
17	0	63	.02626682
18	0	66	.02626682
19	0	72	.02626682
20	0	71	.02626682
21	0	52	.02626682
22	0	66	.02626683
23	0	61	.02626683
24	0	59	.02626683
25	0	84	.02626684
26	0	138	.026303
27	0	100	.026303
28	0	74	.026303
29	0	55	.026303
30	0	56	.026303
31	0	67	.026303
32	0	67	.026303
33	0	75	.026303
34	0	58	.026303
35	0	60	.026303
36	0	59	.026303
37	0	53	.026303
38	0	52	.026303
39	0	87	.026303
40	0	63	.02630301
41	0	60	.02630301
42	0	60	.02630301
43	0	58	.02630301

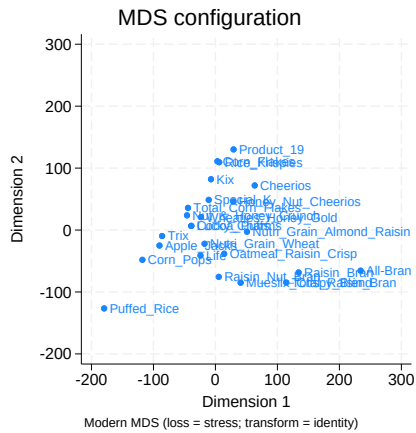
```
 44     0      66    .02630301
 45     0      63    .02630301
 46     0      56    .02630302
 47     0      53    .02630302
 48     0     131    .19899027
 49     0     140    .23020403
 50     0     170    .23794378
------------------------------

Modern multidimensional scaling
Dissimilarity: L2, computed on 8 variables
    Loss criterion: stress = raw_stress/norm(distances)
    Transformation: identity (no transformation)

                                                Number of obs     =         25
                                                Dimensions        =          2
    Normalization: principal                    Loss criterion    =     0.0263

. mat YstressP = e(Y)

. assert mreldif(Ystress, YstressP) < 2e-3
```



The output provided when `protect()` is specified includes a table with information on each run, sorted by the loss criterion value. The first column simply counts the runs. The second column gives the internal return code from modern MDS. This example only has values of 0, which indicate converged results. The column header `mrc` is clickable and opens a help file explaining the various MDS return codes. The number of iterations is in the third column. These runs converged in as few as 45 iterations to as many as 170. The loss criterion values are in the fourth column.

In this example, the results from our original run versus the protected run did not differ by much. However, looking at runs 48–50, we see loss criterion values that are much higher than the rest.

The graph from this protected modern MDS run may be compared with the first one produced. There are obvious similarities, though inspection indicates that the two are not the same.

◁

Stored results

mds stores the following in e():

Scalars

e(N)	number of observations
e(p)	number of dimensions in the approximating configuration
e(np)	number of strictly positive eigenvalues
e(addcons)	constant added to squared dissimilarities to force positive semidefiniteness
e(mardia1)	Mardia measure 1
e(mardia2)	Mardia measure 2
e(critval)	loss criterion value
e(alpha)	parameter of transform(power)
e(ic)	iteration count
e(rc)	return code
e(converged)	1 if converged, 0 otherwise

Macros

e(cmd)	mds
e(cmdline)	command as typed
e(method)	classical or modern MDS method
e(method2)	nonmetric, if method(nonmetric)
e(loss)	loss criterion
e(losstitle)	description loss criterion
e(tfunction)	identity, power, or monotonic, transformation function
e(transftitle)	description of transformation
e(id)	ID variable name (mds)
e(idtype)	int or str; type of id() variable
e(duplicates)	1 if duplicates in id(), 0 otherwise
e(labels)	labels for ID categories
e(strfmt)	format for category labels
e(varlist)	variables used in computing similarities or dissimilarities
e(dname)	similarity or dissimilarity measure name
e(dtype)	similarity or dissimilarity
e(s2d)	standard or oneminus (when e(dtype) is similarity)
e(unique)	1 if eigenvalues are distinct, 0 otherwise
e(init)	initialization method
e(irngstate)	initial random-number state used for init(random)
e(rngstate)	random-number state for solution
e(norm)	normalization method
e(targetmatrix)	name of target matrix for normalize(target)
e(properties)	nob noV for modern or nonmetric MDS; nob noV eigen for classical MDS
e(estat_cmd)	program used to implement estat
e(predict)	program used to implement predict
e(marginsnotok)	predictions disallowed by margins

Matrices

e(D)	dissimilarity matrix
e(Disparities)	disparity matrix for nonmetric MDS
e(Y)	approximating configuration coordinates
e(Ev)	eigenvalues
e(idcoding)	coding for integer identifier variable
e(coding)	variable standardization values; first column has value to subtract and second column has divisor
e(norm_stats)	normalization statistics
e(linearf)	two element vector defining the linear transformation; distance equals first element plus second element times dissimilarity

Functions

e(sample)	marks estimation sample

Methods and formulas

mds creates a dissimilarity matrix $\mathbf{D}$ according to the *measure* specified in option measure(). See [MV] ***measure_option*** for descriptions of these measures. Subsequently, mds uses the same subroutines as mdsmat to compute the MDS solution for $\mathbf{D}$. See *Methods and formulas* in [MV] **mdsmat** for information.

References

Borg, I., and P. J. F. Groenen. 2005. *Modern Multidimensional Scaling: Theory and Applications*. 2nd ed. New York: Springer.

Corten, R. 2011. Visualization of social networks in Stata using multidimensional scaling. *Stata Journal* 11: 52–63.

Cox, T. F., and M. A. A. Cox. 2001. *Multidimensional Scaling*. 2nd ed. Boca Raton, FL: Chapman and Hall/CRC.

Gifi, A. 1990. *Nonlinear Multivariate Analysis*. New York: Wiley.

Kendall, D. G. 1971. Seriation from abundance matrices. In *Mathematics in the Archaeological and Historical Sciences*. Edinburgh: Edinburgh University Press.

Kruskal, J. B., and M. Wish. 1978. *Multidimensional Scaling*. Newbury Park, CA: Sage.

Lingoes, J. C. 1971. Some boundary conditions for a monotone analysis of symmetric matrices. *Psychometrika* 36: 195–203. https://doi.org/10.1007/BF02291398.

Mardia, K. V., J. T. Kent, and J. M. Bibby. 1979. *Multivariate Analysis*. London: Academic Press.

Pearson, K. 1900. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50(302): 157–175. https://doi.org/10.1080/14786440009463897.

Sammon, J. W., Jr. 1969. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers* 18: 401–409. https://doi.org/10.1109/T-C.1969.222678.

Smullyan, R. M. 1986. *This Book Needs No Title: A Budget of Living Paradoxes*. New York: Touchstone.

Torgerson, W. S. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17: 401–419. https://doi.org/10.1007/BF02288916.

Yang, K., and J. Trewn. 2004. *Multivariate Statistical Methods in Quality Management*. New York: McGraw–Hill.

Young, F. W. 1987. *Multidimensional Scaling: History, Theory, and Applications*, ed. R. M. Hamer. Hillsdale, NJ: Erlbaum Associates.

Young, G., and A. S. Householder. 1938. Discussion of a set of points in terms of their mutual distances. *Psychometrika* 3: 19–22. https://doi.org/10.1007/BF02287916.

Also see *References* in [MV] **mdsmat**.

Joseph Bernard Kruskal (1928–2010) was born in New York. His brothers were statistician William Henry Kruskal (1919–2005) and mathematician and physicist Martin David Kruskal (1925–2006). He earned degrees in mathematics from Chicago and Princeton and worked at Bell Labs until his retirement in 1993. In statistics, Kruskal made major contributions to multidimensional scaling. In computer science, he devised an algorithm for computing the minimal spanning tree of a weighted graph. His other interests include clustering and statistical linguistics.

Also see

[MV] **mds postestimation** — Postestimation tools for mds, mdsmat, and mdslong

[MV] **mds postestimation plots** — Postestimation plots for mds, mdsmat, and mdslong

[MV] **biplot** — Biplots

[MV] **ca** — Simple correspondence analysis

[MV] **factor** — Factor analysis

[MV] **mdslong** — Multidimensional scaling of proximity data in long format

[MV] **mdsmat** — Multidimensional scaling of proximity data in a matrix

[MV] **pca** — Principal component analysis

[U] **20 Estimation and postestimation commands**